

Technical Design Document

CareConnect

University of Maryland Global Campus

SWEN 670 – Software Engineering Capstone

Dr. Mir Assadullah

Spring 2026

Contributors: Parthav Dani, Pemon Kouadio, Mawuko Kpelevi, Jordene Downer, Melissa Burgos

Contents

1. Introduction	8
1.1 Purpose	8
1.2 Intended Audience.....	9
1.3 Document Scope	9
1.4 Definitions, Acronyms, and Abbreviations.....	9
1.5 References.....	11
2. System Overview	11
2.1 System Description.....	11
2.2 System Objectives	11
2.3 System Users	12
2.4 Operating Environment.....	12
3. Architecture Diagram	13
3.1 Logical (Application) Components	13
3.2 Persistence and Infrastructure Components	15
3.3 AI Integration Component View	16
3.4 EVV System Architecture.....	17
3.5 Deployment View.....	18
4. Technology Stack.....	19
4.1 WebSocket Architecture	21
4.1.1 Why WebSocket Over Polling.....	21
4.1.2 WebSocket Connection Lifecycle	21
4.1.3 WebSocket Message Types	22
4.2 WebRTC and Video Calling Architecture	22
4.2.1 WebRTC Fundamentals	22
4.2.2 Why Twilio Over Self-Hosted Solutions.....	22
4.2.3 Call Flow Sequence	23
4.2.4 HIPAA Compliance for Video Calling	24
5. Functional Specifications	25

5.1 Electronic Visit Verification (EVV)	25
5.1.1. File Submission Workflow	27
5.1.2 Response Handling Workflow	28
5.1.3 Error Resolution Workflow (Rejected Files)	28
5.2 In-Home Residential Support.....	28
5.3 AI-Assisted Services.....	29
5.4 Communication Services.....	29
5.4.1 Chat Messaging.....	29
5.4.2 Audio and Video Calling	32
6. API Design Specification	34
6.1 API Conventions and Standards	34
6.1.1 Authentication and Authorization	35
6.1.2 Privacy, PHI, and Data Minimization	35
6.1.3 Permission Enforcement	35
6.2 Communication APIs (Chat + Audio/Video Calling)	36
6.2.1 REST Endpoints (Communication)	36
6.2.2 WebSocket Endpoints (Real-Time Channels)	37
6.2.3 WebSocket Message Types and Payloads.....	37
6.2.4 Communication Eligibility and Permission Rules (Enforcement Points).....	39
6.2.5 Error Handling and Status Codes (Communication)	39
6.3 API Design Specification – EVV Module	40
6.3.1 XML Processing service:	40
6.3.2 Netsmart API Integration	41
6.4 API Design Specification – In-Home Residential Support	41
6.4.1 REST Endpoints (In-Home Residential Support)	41
6.4.2 Validation and Error Handling (In-Home Residential Support).....	42
6.5 API Design Specification – AI-Assisted Services	42
6.6 API Design Specification – Administration, Audit, and Reporting	43
6.6.1. Scope and Responsibilities.....	43

6.6.2. REST Endpoints (Administration & Audit)	43
6.6.3 Audit Events and Logging Rules	45
6.6.4 Security and Access Control	45
6.7 API Versioning and Backward Compatibility (Placeholder)	45
6.7.1 Versioning Strategy	45
6.7.2 Backward Compatibility Principles.....	46
6.7.3 Deprecation Policy	46
6.7.4 Client Compatibility Considerations.....	46
6.7.5 Error Handling Across Versions	46
7. Data Model Specification	47
7.1 Data Modeling Approach and Assumptions	47
7.2 Communication Data Models (Chat + Audio/Video Calling).....	47
7.2.1 Entity Relationship Summary (Communication).....	47
7.2.2 Conversation	48
7.2.3 ConversationParticipant.....	48
7.2.4 Message	49
7.2.5 MessageReceipt	50
7.2.6 Attachment.....	51
7.2.7 CallSession.....	52
7.2.8 CallParticipant	52
7.2.9 CallSignalEvent (Optional / Recommended for Troubleshooting)	53
7.2.10 Indexing and Query Patterns	54
7.2.11 Data Retention and Compliance Considerations (Communication)	54
7.3 Data Models – EVV Module	54
7.3.1 Core Entities	55
7.4 Data Models – In-Home Residential Support Module	56
7.4.1 Entity Relationship Summary (In-Home Residential Support).....	56
7.4.2 Support Activity.....	57
7.5 Data Models – AI-Assisted Services	57

7.6 Data Models – Administration, Audit, and Reporting (Placeholder)	58
7.7 Data Dictionary Alignment (Placeholder)	58
8. UI / UX Design Specification.....	58
8.1 UI Design Approach and Principles	58
8.2 Accessibility Requirements (WCAG-Aligned)	58
8.3 Global Navigation and Information Architecture.....	59
8.4 Common UI Components and Design System Elements	59
8.5 Communication UI (Chat + Audio/Video Calling).....	60
8.5.1 Communication Entry Points	60
8.5.2 Permission-Based UX Messages	60
8.5.3 Message Thread Screen (Updated).....	60
8.5.4 Audio/Video Call UI (Twilio Integration)	61
8.5.5 Offline and Reconnection UX	61
8.6 EVV	62
8.6.1 Dashboard Overview.....	62
8.6.2 Upload Visit Data	63
8.6.3 Submission Status Interface	64
8.6.4 Error Resolution Interface	65
8.7 In-Home Residential Support UI	66
8.7.1 Activity Logging Experience	66
8.7.2 Incident Documentation	66
8.7.3 Read-Only and Review States	66
8.7.4 Accessibility Considerations	66
8.8 AI-Assisted Services UI	67
9. Security Considerations	68
9.1.1 Data Privacy & HIPAA Compliance	68
9.1.2 Authentication & Authorization	68
9.1.3 Secure Data Transmission	68
9.1.4 API Security	68

9.1.5 Data Storage & Backup	68
9.1.6 Mobile Application Security	68
9.1.7 AI-Powered Features.....	68
9.1.8 Emergency Response Features	69
9.1.9 Subscription Billing Security.....	69
9.1.10 Security Testing & Monitoring	69
9.1.11 EVV Data Security Measures.....	69
9.1.12 EVV 21st Century Cures Act Compliance	69
10. Error Handling	70
11. Performance & Scalability	70
11.1. Objectives.....	70
11.2 Scalability Strategy	71
11.2.1 Horizontal Scaling Architecture.....	71
11.2.2 WebSocket Scaling Considerations	71
11.2.3 Database Scaling	71
11.3 Performance Monitoring	72
11.3.1 Key Metrics to Monitor	72
11.3.2 Monitoring Tools	72
11.4 Bottleneck Mitigation.....	73
Potential Bottleneck	73
Mitigation Strategy	73
Implementation	73
12. Testing Strategy	73
12.1 Testing Overview	73
12.2 Testing Levels	73
12.2.1 Unit Testing	73
Layer	73
Tools	73
Focus Areas	73

12.2.2 Integration Testing	74
12.2.3 System Testing.....	74
12.2.4 Security Testing	74
12.3 Communication-Specific Test Cases.....	75
12.3.1 Chat Test Coverage.....	75
12.3.2 Call Test Coverage	75
12.4 Test Automation	75
13. Deployment Strategy	75
13.1 Deployment Environments	75
13.2 AWS Infrastructure.....	76
13.2.1 Production Architecture.....	76
13.2.2 Security Groups Configuration	76
13.2.3 ALB Configuration for WebSocket.....	77
13.3 CI/CD Pipeline	77
13.3.1 Backend Pipeline	77
13.3.2 Frontend Pipeline.....	77
13.4 Rollback Procedure	77
14. Known Issues & Limitations	78
14.1 Technical Limitations.....	78
14.2 Known Issues.....	78
14.3 Exclusions	79
14.4 Future Enhancements.....	79

1. Introduction

1.1 Purpose

This Technical Design Document (TDD) provides a comprehensive description of the technical architecture and implementation design for the CareConnect system. It translates the functional and non-functional requirements defined in the Software Requirements Specification (SRS) and the organizational and planning decisions documented in the Project Plan into a concrete and actionable system design. The purpose of this document is to serve as a blueprint for development and testing activities by clearly identifying system components, their responsibilities, and their interactions. Additionally, this document supports academic evaluation by demonstrating the rationale behind technology selection, architectural structure, and design trade-offs.

1.2 Intended Audience

This document is intended for multiple stakeholder groups, including software developers responsible for implementing the CareConnect system, system architects who require insight into design structure and integration strategies, quality assurance engineers who will design test cases based on system behavior, and academic reviewers who evaluate compliance with course requirements. It is assumed that readers have a general understanding of software engineering principles, cloud-based deployment models, and client-server architectures.

1.3 Document Scope

The scope of this document includes the technical design of both frontend and backend components of the CareConnect system. Specifically, it addresses the system architecture supporting real-time communication (chat and audio/video calling), Electronic Visit Verification (EVV), in-home residential support documentation, and AI-assisted service integration. User interface mockups, clinical workflows, and organizational policy enforcement are outside the scope of this document. This document focuses on how the system will be technically implemented rather than what the system must do, which is defined in the SRS.

1.4 Definitions, Acronyms, and Abbreviations

Definitions, acronyms, and abbreviations used throughout this document are defined in the Software Requirements Specification (SRS). These include but are not limited to:

Term	Definition
ADL	Activities of Daily Living
IADL	Instrumental Activities of Daily Living
WebRTC	Web Real-Time Communication - Browser/device built-in API for peer-to-peer audio/video/data
WebSocket	Protocol for persistent, bidirectional communication between client and server
SDP	Session Description Protocol - Format describing media capabilities for WebRTC negotiation

ICE	Interactive Connectivity Establishment - Process for finding optimal network path between peers
STUN	Session Traversal Utilities for NAT - Server that helps discover public IP addresses
TURN	Traversal Using Relays around NAT - Relay server for when direct P2P fails
Twilio Video	A managed video calling API service that handles WebRTC complexity, providing audio/video communication through Go Rooms (optimized for 1:1 calls) and Group Rooms. HIPAA-eligible with BAA available.
Go Room	Twilio's room type optimized for peer-to-peer 1:1 video calls, offering the lowest per-minute pricing (\$0.001/participant-minute).
Access Token	A JWT-based token generated by the backend that authenticates users to join specific Twilio video rooms.
ALB	Application Load Balancer - AWS service for distributing traffic
EC2	Elastic Compute Cloud - AWS virtual server instances
PHI	Protected Health Information
Term	Definition
HIPAA	Health Insurance Portability and Accountability Act
EVV	Electronic Visit Verification
RBAC	Role-Based Access Control
HTTP	Hypertext Transfer Protocol
AI	Artificial Intelligence

The consistent use of these terms ensures clarity and avoids ambiguity when describing system components and processes.

1.5 References

- CareConnect Software Requirements Specification (SRS)
- CareConnect Project Plan
- SWEN 670 Course Materials and IEEE 1016 Software Design Description guidelines

2. System Overview

2.1 System Description

CareConnect is a healthcare coordination platform designed to facilitate secure and efficient communication between patients and caregivers while supporting regulatory-compliant documentation of care delivery activities. The system enables real-time messaging and audio/video calling, structured visit verification through EVV workflows, and the recording of in-home residential support activities such as Activities of Daily Living (ADLs) and Instrumental Activities of Daily Living (IADLs). Additionally, the system integrates AI-assisted services to support summarization and workflow guidance without replacing clinical judgment.

CareConnect includes an AI-assisted service capably designed to support caregivers and patients through non-diagnostic, assistive functionality. These services are implemented through a dedicated AI Orchestration Service in the backend, which integrates with Amazon Web Services (AWS) Bedrock to generate natural-language summaries and guidance based on structured application data. All AI processing is performed on the server-side, ensuring consistent security controls, role-based access enforcement, and protection of sensitive healthcare information.

The system enables healthcare providers to capture, validate, and submit EVV records that seamlessly interface with Netsmart's API infrastructure, ensuring real-time data transmission in the required XML format while maintaining full compliance with federal 21st Century Cures Act mandates.

2.2 System Objectives

The primary objectives of the CareConnect system are to improve care coordination through timely communication, ensure regulatory compliance for visit documentation, and provide a

scalable technical foundation for future enhancements. Specific objectives include:

- Enabling secure, real-time communication between authorized users.
- Supporting accurate and auditable EVV data capture.
- Providing structured and consistent documentation of in-home support activities.
- Ensuring protection of PHI through layered security controls.
- Supporting cloud-based scalability and maintainability.
- AI is assistive only and explicitly excludes diagnosis, treatment, or care enforcement

2.3 System Users

Primary system users include patients who receive care and caregivers who provide care services. Secondary users include supervisors, compliance officers, and system administrators who review EVV records, manage user accounts, and monitor system activity. Each user category is associated with distinct permissions and access rights enforced through RBAC mechanisms.

2.4 Operating Environment

The frontend application operates on Android devices and modern web browsers using the Flutter framework built using the Dart Programming language. The backend services operate within a cloud-hosted environment using containerized Java Spring Boot services deployed on AWS infrastructure. PostgreSQL serves as the primary data store. Real-time communication uses WebSocket for messaging and signaling, with WebRTC (via Twilio) for audio/video calling.

AI services are accessed exclusively through AWS Bedrock using backend managed credentials and configuration. The frontend client does not directly interact with Bedrock or any external AI provider. Model selection, prompt configuration parameters, and runtime settings are externalized via AWS Parameter Store and Secrets Manager, allowing AI behavior to be modified without redeploying application code while maintaining centralized security and governance controls.

3. Architecture Diagram

This section enumerates the major architectural components of CareConnect and describes how each component contributes to system behavior. The tables below provide a structured view of responsibilities, key technologies, and primary interfaces between the frontend, backend services, persistence layer, and supporting cloud infrastructure.

3.1 Logical (Application) Components

Table 1- Architecture component

Component	Layer	Primary Responsibilities	Key Interfaces / Endpoints	Technology / Implementation Notes
Flutter Client (Mobile/Web)	Presentation	Renders UI; captures user input; manages local state; initiates REST requests; maintains WebSocket connection(s); initiates/receives WebRTC calls and renders media streams.	HTTPS REST APIs (e.g., /api/auth, /api/chat, /api/evv); WebSocket channels (/ws/chat, /ws/call); WebRTC peer connection	Flutter (Dart). Uses secure token storage; supports accessibility-first UI patterns; uses background connectivity handling.
API Gateway / Load Balancer (Edge)	Infrastructure /Edge	Routes inbound HTTP(S) and WebSocket traffic to backend services; enforces TLS termination; supports	Client → ALB/API Gateway → backend service endpoints	AWS ALB or API Gateway (depending on deployment topology). Encourages consistent URL base

		basic throttling and health routing.		and environment separation.
Authentication & Authorization Service	Backend (Security)	Authenticates users; issues/validates JWTs; enforces RBAC and relationship checks (patient–caregiver assignment); supports session expiration and token refresh strategy.	POST /api/auth/login; token validation filter applied across protected APIs	Spring Security + JWT. Optional integration with managed identity provider (e.g., Cognito) if selected.
Chat Service	Backend (Communication)	Validates chat eligibility rules; manages message send/receive; persists messages; delivers real-time notifications; supports attachments metadata and delivery status (sent/delivered/read where applicable).	WebSocket /ws/chat; REST: GET /api/chat/history, POST /api/chat/send (representative)	Spring Boot + Spring WebSocket. Stores message metadata in PostgreSQL; attachments stored in S3 with signed URLs.
Call Signaling Service	Backend (Communication)	Coordinates call session lifecycle (start/accept/decline/end); creates and manages Twilio Video rooms; issues time-limited access tokens; enforces permissions; records call session metadata.	Twilio Video (audio-only and audio/video); REST: POST /api/call/start, POST /api/call/end; Webhooks: Twilio room and participant events (representative).	Video Rooms used for both audio-only and video calls (audio tracks only when video disabled). Media transport handled by Twilio WebRTC infrastructure; backend mediates signaling for control, compliance, and audit.
EVV Service	Backend (Domain)	The backend service layer handles all business logic, data transformation, and external integrations		Spring Boot service layer. Uses PostgreSQL tables keyed by participant/visit identifiers; implements validation rules for required fields.

In-Home Residential Support Service	Backend (Domain)	Captures ADL/IADL activity logs associated with visits; supports supervisor review and compliance reporting; enforces role-based visibility rules.	POST /api/support/activity; GET /api/support/visit/{id}/activities (representative)	Stores structured activity logs linked to Visit records. Designed to evolve without destabilizing EVV core workflows.
AI Orchestration Service	Backend (Integration)	Receives AI requests from client; performs input validation and PHI minimization; calls AWS Bedrock; stores request metadata and returns summaries/assistance outputs to the client.	POST /api/ai/summarize; POST /api/ai/assist (representative)	Backend-only integration to prevent direct client access to AI services. Applies governance and non-clinical boundaries.
Audit Logging Service	Backend (Cross-cutting)	Captures security-relevant events (auth attempts, access to PHI, permission denials, administrative actions); supports immutable audit trails and reporting queries.	Internal service calls or interceptors; stored audit records query-able via admin APIs (if enabled)	Implemented via Spring interceptors/aspects and database append-only tables. Supports compliance review requirements.

3.2 Persistence and Infrastructure Components

Table 2- Infrastructure Component

Component	Purpose	Data / Artifacts Stored	Security Controls	Technology / Implementation Notes
PostgreSQL Database	Primary relational data store for CareConnect domain entities and audit records.	Users/roles; patient–caregiver assignments; messages metadata; EVV visits; ADL/IADL logs; call session metadata; audit logs.	Encryption at rest (managed service controls); least-privilege DB accounts; backups; row-level access patterns enforced by backend.	PostgreSQL (managed or containerized for dev). Indexing for recent messages/visits; pagination for large history queries.
Object Storage (S3)	Stores binary artifacts and large	Chat attachments (images/audio/video); optional call	Bucket policies; server-side encryption; signed URLs for time-	Amazon S3. Access mediated by backend;

	payloads to avoid database bloat.	recordings; exported reports.	bounded access; lifecycle rules for retention.	attachment metadata stored in PostgreSQL.
Compute / Container Runtime	Runs backend services as scalable, isolated units.	Docker images; runtime configuration; service logs.	Network security groups; IAM roles for service-to-service access; secrets management for credentials.	AWS ECS (Fargate) containers. Health checks and auto-scaling policies applied where supported.
Logging & Monitoring	Provides operational visibility and supports troubleshooting and audit readiness.	Application logs; metrics (CPU/memory/latency); alarms; request traces (if enabled).	Access controls on log groups; retention policies; anomaly detection/alerts.	Amazon CloudWatch (and optionally X-Ray). Enables cost-aware monitoring and incident response support.
Secrets & Configuration Management	Stores environment variables, credentials, and configuration parameters securely.	DB credentials; JWT signing keys; service endpoints; API keys.	Encryption at rest; IAM-restricted access; rotation policies where applicable.	AWS Secrets Manager / Parameter Store (or environment variables in dev). Avoids hardcoding secrets in code repositories.

Note: The component boundaries above reflect a modular design that supports iterative development. During implementation, some services may initially be deployed as a single backend application and later refactored into independently deployable services as scale, testing needs, and team capacity evolve.

3.3 AI Integration Component View

AI functionality within CareConnect follows a layered integration model designed to isolate external AI dependencies from core business logic. User initiated AI requests originate in the Flutter frontend and are routed through authenticated REST endpoints exposed by the backend. These requests are handled by the AI Orchestration Service, which performs authorization checks, contextual validation, and prompt construction before invoking AWS Bedrock via the Bedrock Runtime API.

Responses returned from Bedrock are evaluated for completeness and safety, optionally truncated or reformatted, and then returned to the frontend for user review. At no point does the AI system directly persist in generated content without explicit user action. This architecture ensures that AI services enhance user workflows without introducing hidden side effects or compromising system reliability.

3.4 EVV System Architecture

The EVV system follows a modern three-tier architecture comprising a presentation layer (Flutter application), application layer (backend services), and data layer (PostgreSQL database), integrated with the external Netsmart Tellus API.

Below is the architecture diagram for this system:

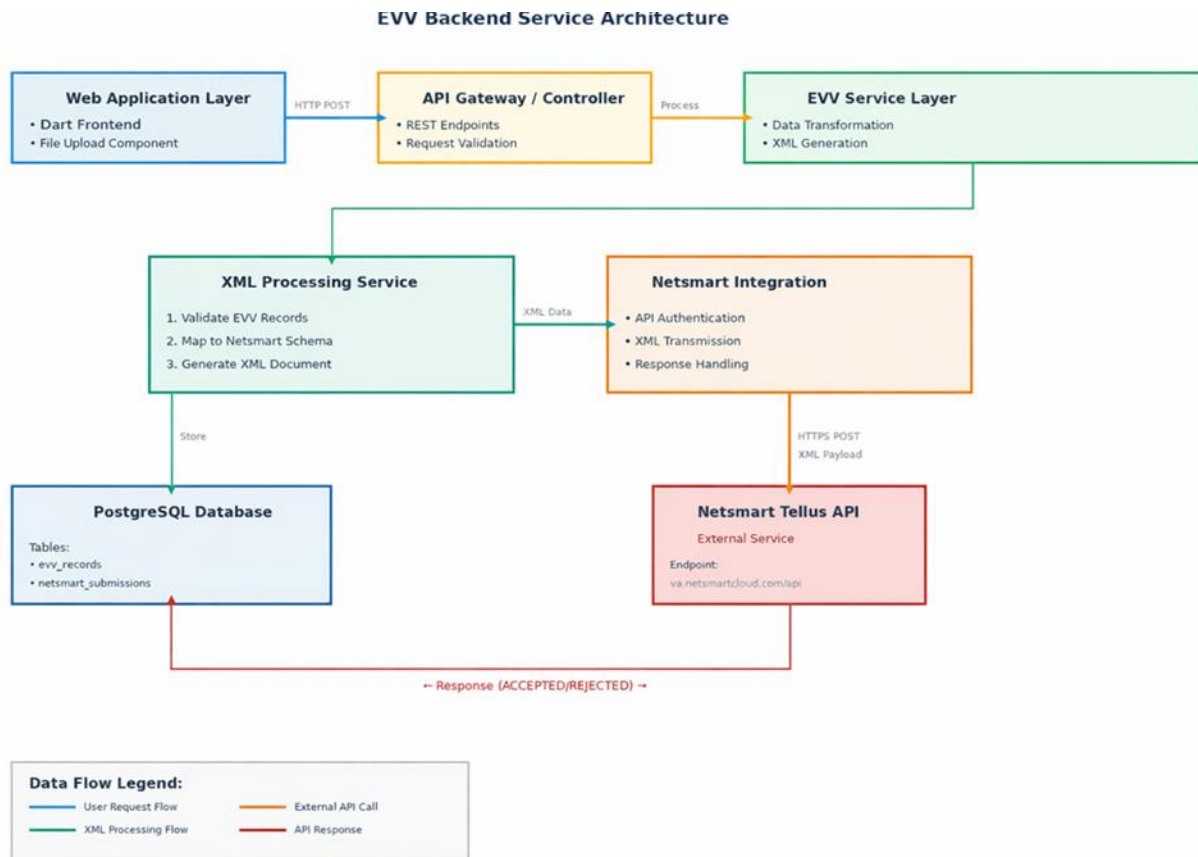


Figure 1: EVV Backend Service Architecture showing the complete data flow from Flutter application through service layers to Netsmart API

Presentation Layer:

The application is built using Flutter framework and Dart programming language, providing consistent, responsive experience across web, iOS, and Android platforms. Key features include:

- Cross-platform compatibility: Single codebase for web, iOS, and Android
- Material Design UI components with custom CareConnect branding
- File upload with drag-and-drop support for XML/JSON files
- Real-time validation feedback and status updates
- Responsive dashboard with submission tracking and analytics

- Error resolution interface with guided remediation
- Role-based access control (RBAC) implementation

Application Layer (Backend services):

The backend service layer handles all business logic, data transformation, and external integrations:

- API Gateway/Controller: RESTful endpoints for file submission, status queries, and error retrieval
- XML Processing Service: Validates EVV records, maps to Netsmart schema, generates compliant XML
- Netsmart Integration Service: Handles authentication, HTTPS transmission, response polling

Data Persistence Service: Manages database operations for EVV records and submission tracking.

Data Layer:

PostgreSQL database storing the tables below:

- *evv_records*: Individual visit records with 6 federal elements
- *netsmart_submissions*: Transmission metadata, status, response data
- *error_log*: Detailed error tracking for rejected submissions
- *audit_trail*: Complete history of data changes and user actions

3.5 Deployment View

Backend services are packaged as Docker containers and deployed on AWS compute services such as ECS. A load balancer distributes incoming traffic across multiple service instances to improve availability and fault tolerance. The database is hosted as a managed PostgreSQL service. Static content and attachments are stored on Amazon S3. AI services are accessed via

AWS Bedrock through secured backend integrations. Configuration parameters are externalized to support multiple deployment environments including development, testing, and production.

4. Technology Stack

This section summarizes the primary technologies selected for the CareConnect frontend and backend implementation. The stack emphasizes cross-platform delivery, real-time communication, security controls appropriate for healthcare data, and cloud-native deployment practices that support scalability and maintainability.

Table 3- Technology Stack

Layer	Technology Used	Purpose
Frontend (Mobile/Web)	Flutter (Dart)	Single cross-platform codebase for Android and Web UI; supports responsive layouts and accessibility-oriented UI patterns.
State Management	Provider / Riverpod (TBD by implementation)	Manages UI state, async data loading, and dependency injection to keep presentation logic maintainable and testable.
Backend Framework	Java + Spring Boot	Implements business logic, REST APIs, and WebSocket endpoints; integrates security, validation, and service-layer patterns.
API Layer (REST)	Spring MVC (REST controllers) + JSON	Provides request/response APIs for authentication, EVV workflows, in-home support documentation, and AI requests.
Real-Time Messaging	WebSocket (Spring WebSocket)	Enables low-latency bidirectional chat messaging and event delivery (presence, typing indicators, message status updates). Replaces HTTP polling for instant message delivery.

Call Signaling	WebSocket (Spring WebSocket) + Twilio Video	WebSocket handles call invitation, acceptance/decline, and session coordination. Twilio Video manages WebRTC session establishment (signaling, ICE negotiation) and room participant coordination via backend-issued access tokens.
Audio/Video Media	WebRTC via Twilio Video (managed service)	Twilio abstracts WebRTC complexity, providing real-time, encrypted audio-only and audio/video transport. Media is handled by Twilio infrastructure; backend retains control for authorization, auditing, and compliance.
Database	PostgreSQL	Stores users, roles, assignments, EVV records, in-home activity logs, message metadata, call session metadata, and audit trails; supports relational integrity and indexing.
Cache / Session (optional)	Redis (optional)	Improves responsiveness for presence/session state, rate limiting, and temporary signaling/session data in high-concurrency scenarios. Required if scaling to multiple backend instances.
File & Media Storage	Amazon S3	Stores chat attachments (images/audio/video), profile pictures, and exports; supports lifecycle policies, encryption at rest, and time-bounded signed URLs for secure access.
AI / LLM Integration	AWS Bedrock	Provides managed access to foundation models used for summarization and assistive workflows; requests routed via backend to control PHI exposure.

Load Balancing	AWS Application Load Balancer (ALB)	Routes HTTPS/WSS traffic to backend instances; handles SSL termination with ACM certificates; configured with sticky sessions and 3600s idle timeout for WebSocket support.
Logging & Monitoring	Amazon CloudWatch	Centralized logs, metrics, alarms, and operational visibility for troubleshooting and audit support.
Containerization	Docker	Packages backend services consistently across development/test environments and enable portable deployment configurations.

4.1 WebSocket Architecture

4.1.1 Why WebSocket Over Polling

The previous implementation used HTTP polling (checking for new messages every 2 seconds). WebSocket provides significant advantages:

- Instant message delivery (no 2-second delay)
- Reduced server load (no constant polling requests)
- Lower battery consumption on mobile devices
- Support for typing indicators, presence, and read receipts

4.1.2 WebSocket Connection Lifecycle

The WebSocket connection follows this lifecycle:

- Connection Establishment: Client connects to `wss://api.careconnect.com/ws/chat` with JWT token
- Authentication: Server validates JWT and binds session to user identity
- Heartbeat: Client sends ping every 30 seconds to maintain connection through NAT/firewalls
- Message Exchange: Bidirectional JSON messages for chat, typing indicators, and status updates
- Reconnection: On disconnect, client uses exponential backoff (1s, 2s, 4s, 8s...) up to 10 attempts

4.1.3 WebSocket Message Types

Type	Direction	Purpose	Key Fields
CHAT_SEND	Client→Server	Send a chat message	conversationId, text, messageType
CHAT_DELIVER	Server→Client	Deliver message to recipient	messageId, senderId, content, timestamp
CHAT_ACK	Client→Server	Acknowledge receipt/read	messageId, status (delivered read)
TYPING	Bidirectional	Typing indicator	conversationId, isTyping
PING/PONG	Bidirectional	Keep connection alive	timestamp

4.2 WebRTC and Video Calling Architecture

4.2.1 WebRTC Fundamentals

WebRTC (Web Real-Time Communication) is a browser/device built-in technology that enables peer-to-peer audio and video communication. Key concepts:

- Peer-to-Peer: After initial setup, media flows directly between devices without passing through the server
- Signaling: The exchange of connection information (SDP offers/answers, ICE candidates) through our WebSocket server
- STUN: Server that helps devices discover their public IP address for NAT traversal
- TURN: Relay server used when direct P2P connection fails (approximately 15% of cases)

4.2.2 Why Twilio Over Self-Hosted Solutions

CareConnect uses Twilio Video rather than self-hosted solutions (like Jitsi) or pure WebRTC for the following reasons:

Factor	Pure WebRTC	Self-Hosted Jitsi	Twilio Video
Implementation Time	3-6 months for production quality	1-2 weeks integration	1-2 weeks integration

Infrastructure Required	TURN Server (EC2)	EC2 Instance	None (managed)
Complexity	High - must handle all edge cases	Low - SDK handles complexity	Low
HIPAA Compliance	Must implement all controls	Self-hosted = full data control	BAA required (available)
Cost	Free + TURN server costs	Fixed (~\$30/mo EC2)	Usage based (~\$3- 27/mo)
1:1 Call Optimization	Manual	Generic	Go Rooms Optimized
Flutter SDK	Manual WebRTV	Jitsi_meet_flutter_sdk	Twilio_programmable_video (mature)
Global TURN	Build Yourself	Single Server	Included

4.2.3 Call Flow Sequence

The following sequence describes how a video call is established:

****Phase 1 - Signaling (via WebSocket through Spring Boot):****

1. Caller sends CALL_INVITE message via WebSocket
2. Server validates relationship/permissions and creates call session
3. Server sends CALL_RINGING to callee via WebSocket
4. Callee sends CALL_ACCEPT or CALL_DECLINE
5. Server generates Twilio access tokens for both users
6. Server sends CALL_READY with room name and access tokens to both participants

****Phase 2 - Media (via Twilio Infrastructure):****

7. Both clients connect to Twilio using twilio_programmable_video SDK with their access tokens
8. Twilio SDK handles WebRTC negotiation (ICE/STUN/TURN) automatically
9. Encrypted audio/video streams flow through Twilio (or P2P when possible)

10. Call metadata (duration, participants, outcome) logged to PostgreSQL

4.2.4 HIPAA Compliance for Video Calling

Twilio Video provides HIPAA compliance through their managed infrastructure:

- Business Associate Agreement (BAA): Required and available from Twilio. Must be signed before handling PHI.
- Encryption in Transit: Twilio uses DTLS for key exchange and SRTP for media encryption by default (automatic).
- Access Control: Access tokens authenticate users to specific rooms; tokens are short-lived and room-specific.
- Audit Logging: Backend logs call metadata (participants, start/end times, duration) for compliance. Twilio also provides webhooks for additional logging.
- Data at Rest: Calls are not recorded by default. If recording is enabled, Twilio stores encrypted and can be configured for your own S3 bucket.
- Shared Responsibility: Twilio manages infrastructure security; CareConnect manages access control, token generation, and application-level audit logging.

IMPORTANT: Before going to production, the BAA with Twilio must be executed. Contact Twilio sales to initiate the BAA process.

5. Functional Specifications

This section defines the functional behavior of the CareConnect system based on the Software Requirements Specification (SRS) and Project Plan. It describes the major functional modules and their expected system capabilities. Communication-related features (chat and audio/video calling) are included at a high level and will be further expanded in subsequent revisions.

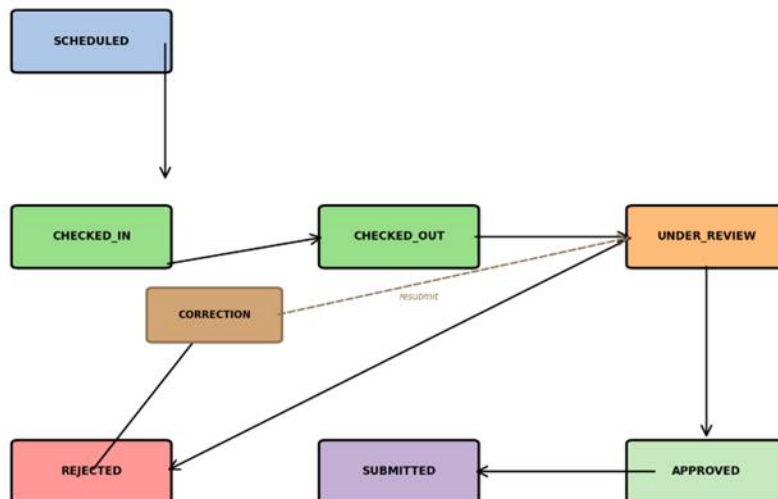
5.1 Electronic Visit Verification (EVV)

The Electronic Visit Verification (EVV) module is responsible for recording and validating caregiver visit events in a manner that supports regulatory compliance and audibility. The module captures visit check-in and check-out events, including caregiver identity, patient identity, service type timestamps, and location data where required. EVV serves as the authoritative record of when a visit occurred and provides the structural anchor for all in-home residential support documentation associated with that visit.

Complete EVV Record Lifecycle:

EVV records progress through a well-defined lifecycle: from initial scheduling through check-in/check-out verification, review and approval, and finally submission to the payer/aggregator. Corrections can be made with full audit trails.

EVV Record Lifecycle



6 Federal EVV Elements Captured:

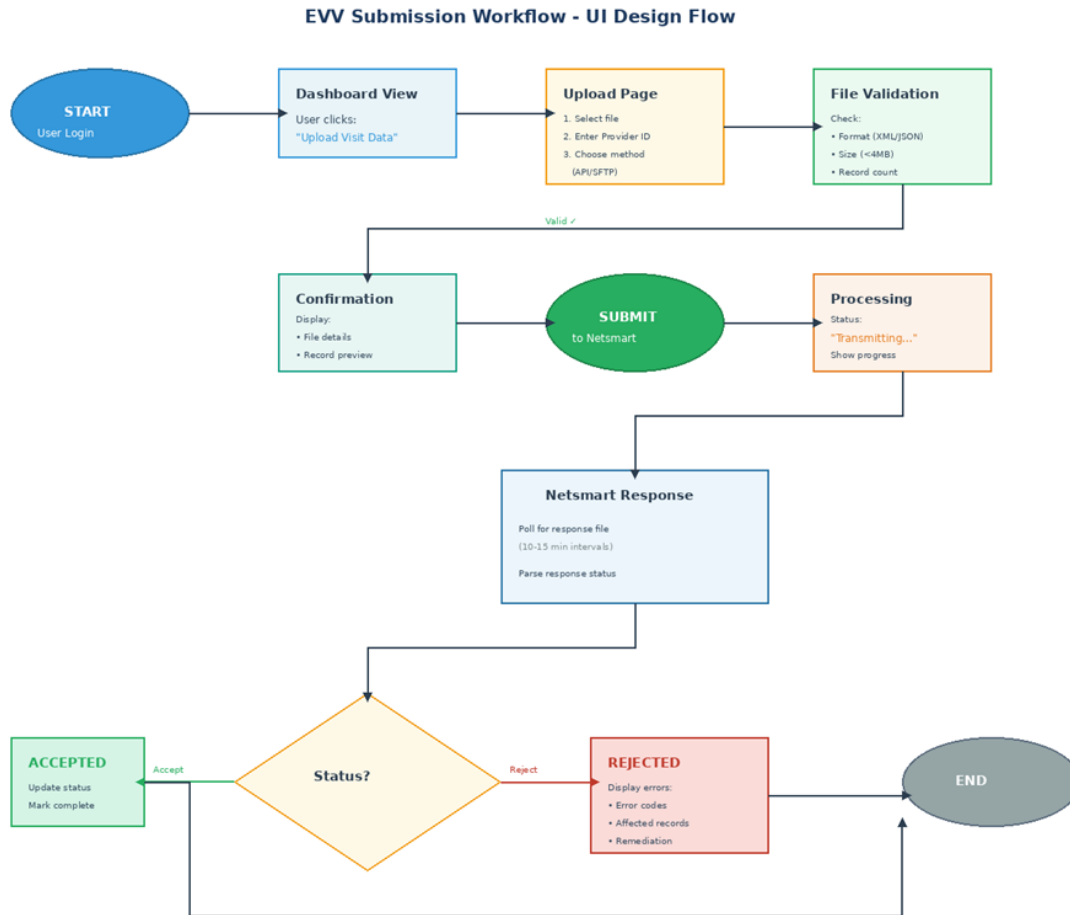
1. Service Type 2. Individual (Patient) 3. Caregiver 4. Date 5. Time In/Out 6. Location (GPS)

Role based access Control:

The EVV system implements fine-grained role-based access control with 6 distinct roles: Caregiver, Supervisor, EOR, Billing Specialist, Auditor, and Administrator. Each role has specific permissions across EVV operations.

Role	View Schedules	Create EVV	Correct EVV	Approve Claims	Submit Claims	View Reports	Admin
Caregiver	✓	✓	-	-	-	-	-
Supervisor	✓	✓	✓	✓	-	✓	-
Billing Admin	✓	-	-	-	✓	✓	-
EOR (Admin)	✓	✓	✓	✓	✓	✓	-
System Admin	✓	✓	✓	✓	✓	✓	✓
State (DMAS)	-	-	-	-	-	✓	-

Complete Submission workflow:



Workflow Steps Detail:

5.1.1. File Submission Workflow

1. User logs into CareConnect Flutter application with credentials
2. Dashboard displays with navigation drawer/menu
3. User navigates to 'Upload Visit Data' screen
4. Upload page loads with file selection area
5. User selects/drops XML or JSON file
6. System validates file format, size (<4MB), record count (<1,200)
7. User enters Provider ID and selects submission method (API/SFTP)
8. Confirmation dialog shows batch details and record preview
9. User confirms submission
10. System displays processing status: 'Transmitting to Netsmart...'

5.1.2 Response Handling Workflow

11. Backend polls Netsmart API for response file (10–15-minute intervals)
12. Response file detected and downloaded
13. System parses response: ACCEPTED or REJECTED status
14. Database updated with submission result
15. User receives notification (push notification on mobile, in-app alert on web)

5.1.3 Error Resolution Workflow (Rejected Files)

16. Flutter app displays error summary with affected record count
17. User taps 'View Details' to navigate to Error Resolution screen
18. Error resolution interface shows: Visit ID, Error Code, Description, Remediation Steps
19. User corrects data (e.g., updates Medicaid ID, adds missing GPS coordinates)
20. User taps 'Fix Now' button for individual fixes or 'Fix Selected' for batch updates
21. System tracks remediation history: attempt number, corrections made, new submission
22. Reports generated on resolution time and success rate

5.2 In-Home Residential Support

The In-Home Residential Support module enables caregivers to document care activities performed during an EVV-tracked visit. This module supports the structured recording of Activities of Daily Living (ADLs), Instrumental Activities of Daily Living (IADLs), observations, and incident notes associated with a specific visit. While EVV is responsible for verifying when a visit occurred, the In-Home Residential Support module focuses on capturing what care was delivered during that visit.

The in-home residential support records are implemented as domain entities that are linked to an existing EVV visit through a shared visit identifier. The backend service enforces this relationship by requiring a valid EVV visit reference for all submitted support activities. This ensures that care documentation cannot exist independently of a verified visit and maintains consistency between visit verification and care delivery records.

The separation between the EVV Service and the In-Home Residential Support Service allows the support documentation model to be extended independently of EVV visit lifecycle logic, improving maintainability while preserving a clear linkage between visit verification and care documentation. This design enables future enhancements, such as additional activity types or documentation fields, without introducing changes to the core EVV workflow.

Caregivers submit in-home residential support entries through REST-based backend APIs, which validate user authorization, visit association, and required fields before persisting records. Role-

based access control governs visibility of support documentation, allowing supervisory users to review completed records while restricting modification privileges once a visit is finalized. All in-home residential support actions generate audit log entries to support compliance review and traceability.

5.3 AI-Assisted Services

CareConnect provides AI assisted services to support caregivers and patients by reducing documentation burden and improving information clarity. These services are explicitly designed as assistive features and do not provide clinical diagnoses, treatment recommendations, or medical decision making.

Primary AI use cases include generating concise summaries of visit documentation, drafting narrative descriptions from structured ADL and IADL data, and providing nonclinical guidance related to system usage or recorded care activities. AI outputs are always presented to users as editable drafts, requiring explicit user review before incorporating official records.

Each AI request undergoes validation to ensure the requesting user is authorized and contextually permitted to access the referenced data. Sensitive data is minimized prior to model invocation, and all AI operations are designed to fail gracefully, allowing users to continue core workflows even when AI services are unavailable.

5.4 Communication Services

5.4.1 Chat Messaging

The chat messaging feature enables secure, real-time communication between authorized users (patients and caregivers). The system replaces the previous HTTP polling mechanism with persistent WebSocket connections to achieve instant message delivery.

Figure 1- WebSocket Chat Architecture



Figure 2 illustrates the real-time messaging architecture. Key components include:

- Flutter clients maintain persistent WebSocket connections to the Spring Boot server
- Messages are delivered instantly (<100ms) via server push, replacing 2-second polling
- All messages are persisted to PostgreSQL before delivery acknowledgment
- Attachments are stored in Amazon S3 with time-limited signed URLs

Technical Implementation

- Transport Protocol: WebSocket (WSS) over TLS for encrypted, bidirectional communication
- Connection Lifecycle: Persistent connection established on app launch; maintained via 30-second heartbeat (ping/pong)
- Reconnection Strategy: Exponential backoff (1s, 2s, 4s, 8s...) with maximum 10 retry attempts
- Message Delivery: Server pushes messages instantly to connected recipients; offline messages queued in PostgreSQL
- Attachment Handling: Files uploaded to Amazon S3 via pre-signed URLs; metadata stored in PostgreSQL

WebSocket Message Types

Type	Direction	Purpose
CHAT_SEND	Client → Server	Send a chat message (text or attachment reference)
CHAT_DELIVER	Server → Client	Deliver message to recipient in real-time
CHAT_ACK	Client → Server	Acknowledge delivery or read receipt
TYPING	Bidirectional	Typing indicator (isTyping: true/false)
PING/PONG	Bidirectional	Connection keepalive heartbeat (every 30 seconds)

Chat Permission Rules

Scenario	Result	Enforcement
Caregiver initiates chat with assigned patient	ALLOWED	Caregiver can start conversation anytime
Patient initiates chat (no prior conversation)	BLOCKED	Must wait for caregiver to send first message
Patient replies after caregiver initiates	ALLOWED	Patient can participate once conversation exists
Patient messages unassigned caregiver	BLOCKED	Relationship validation required
Patient messages another patient	BLOCKED	Patient-to-patient chat not permitted
Caregiver messages another caregiver	ALLOWED	Caregivers can freely communicate

Supported Message Types

- Text messages (UTF-8, up to 4000 characters)
- Image attachments (JPEG, PNG; max 10MB)
- Audio clips (MP3, AAC; max 5 minutes)
- Video clips (MP4; max 2 minutes, max 50MB)

Real-Time Features

- Typing indicators (TYPING message type)

- Delivery receipts (CHAT_ACK with status: delivered)
- Read receipts (CHAT_ACK with status: read)
- Online/offline presence (connection state)

Permission Rules

Chat messaging enforces strict rules to protect patient privacy:

- Caregiver to Patient: Caregivers may initiate conversations with their assigned patients at any time
- Patient to Caregiver: Patients may only reply to conversations after a caregiver has sent the first message
- Patient to Unassigned Caregiver: BLOCKED - Patients cannot message caregivers they are not assigned to
- Patient to Patient: BLOCKED - Patient-to-patient messaging is not permitted
- Caregiver to Caregiver: ALLOWED - Caregivers may freely communicate with each other

5.4.2 Audio and Video Calling

The audio and video calling feature enables synchronous communication between patients and caregivers using Twilio Video, a managed real-time communication platform built on WebRTC. Twilio Video is used for both audio-only and audio/video calls, with the backend mediating session creation, access control, and auditing. Media transport is handled by Twilio's managed infrastructure under a Business Associate Agreement (BAA).

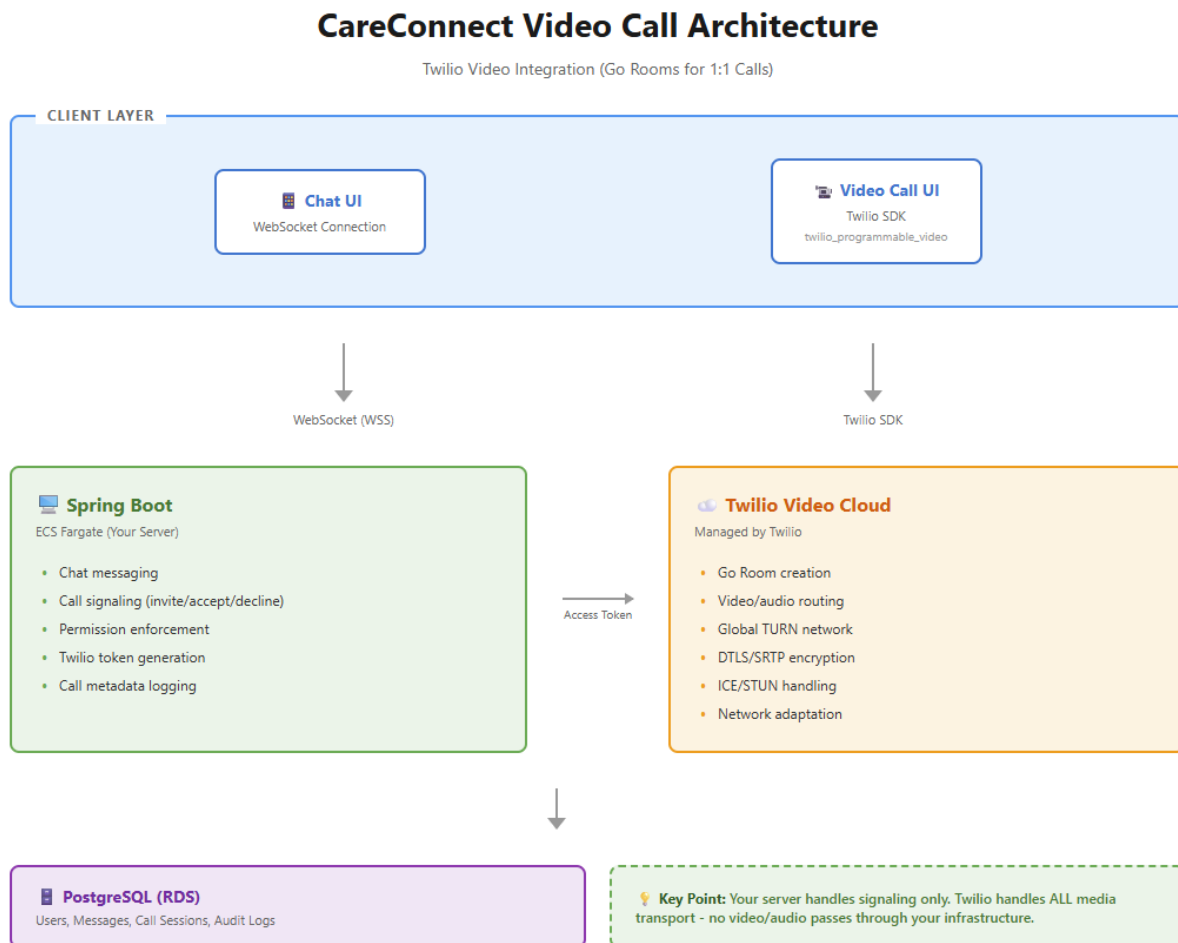
Why Twilio Video

CareConnect uses Twilio Video rather than implementing pure WebRTC or self-hosted solutions for the following reasons:

- No Infrastructure Required: Twilio is fully managed; no EC2 instances needed.
- Implementation Time: 1-2 weeks integration with mature Flutter SDK

- HIPAA Compliance: HIPAA-eligible with BAA available from Twilio
- Cost Efficiency: Go Rooms at \$0.001/participant-minute is optimal for 1:1 calls
- Global Infrastructure: Twilio's worldwide TURN network ensures reliable connectivity
- Flutter Support: `twilio_programmable_video` is a mature, well-documented SDK

Figure 2- Twilio Video Architecture



The diagram above illustrates the call establishment flow with two distinct phases:

Call Flow Architecture

The call establishment process involves two phases:

Phase 1 - Signaling (via WebSocket through Spring Boot):

1. Caller sends `CALL_INVITE` message via WebSocket
2. Server validates relationship/permissions and creates call session

3. Server sends CALL_RINGING to callee via WebSocket
4. Callee sends CALL_ACCEPT or CALL_DECLINE
5. Server generates Twilio access tokens for both users (using Twilio SDK)
6. Server sends room name and access tokens to both participants

Phase 2 - Media (Twilio Video / WebRTC):

7. Both clients connect to Twilio room using twilio_programmable_video SDK with access tokens
8. Twilio Video SDK handles WebRTC negotiation (ICE/STUN/TURN) automatically
9. Encrypted audio/video streams flow through Twilio's global infrastructure
10. Call metadata (duration, participants, outcome) logged to PostgreSQL

Permission Rules for Calling

Audio/video calling enforces relationship-based access controls:

- Caregiver to Assigned Patient: ALLOWED
- Patient to Assigned Caregiver: ALLOWED
- Patient to Unassigned Caregiver: BLOCKED - Patients can only call their assigned caregiver
- Patient to Patient: BLOCKED - Patient-to-patient calls are not permitted
- Caregiver to Caregiver: ALLOWED

Security and Compliance

- Encryption in Transit: WebRTC uses DTLS for key exchange and SRTP (AES-128) for media encryption
- Room Access Control: Backend-issued, time-limited Twilio access tokens (JWTs) authenticate users to specific Twilio Video Rooms; unauthorized users cannot join.
- Audit Logging: Call metadata (participants, timestamps, duration, outcome) logged for compliance
- No Recording by Default: Calls are not recorded unless explicitly enabled by policy

6. API Design Specification

This section specifies the external service interfaces that enable the CareConnect frontend to interact with backend services. It defines API conventions, authentication requirements, and module-specific endpoints. The Communication APIs (chat and audio/video calling) are drafted in detail in this revision. Placeholder headings are provided for other modules so additional teammates can complete those sections.

6.1 API Conventions and Standards

CareConnect APIs follow standard REST and WebSocket patterns:

- Transport: HTTPS for REST APIs; WSS (WebSocket over TLS) for real-time channels.

- Payload format: JSON for REST bodies and WebSocket messages.
- Versioning: All REST endpoints are versioned (e.g., /api/v1/...). WebSocket endpoints are versioned where feasible.
- Time format: ISO 8601 timestamps in UTC.
- Identifiers: UUIDs (recommended) for user, message, and session identifiers.
- Error format: Consistent JSON error objects for REST; typed error messages for WebSockets.

6.1.1 Authentication and Authorization

All protected endpoints require authentication. Token-based authentication is used (JWT recommended). Authorization is enforced using Role-Based Access Control (RBAC) and relationship checks (e.g., patient–caregiver assignment). For Communication features, the backend must validate eligibility before allowing a session to be established.

- REST APIs: Include Authorization: Bearer <access_token> on every request
- WebSocket: Token provided during connection handshake (preferred) or via an initial AUTH message (query parameter only if required)
- Token Expiration: Access tokens expire in 15 minutes; refresh tokens valid for 7 days
- Twilio Sessions: Backend issues time-limited Twilio Access Tokens (JWTs) scoped to the room/session (e.g., Video Room) for each call.

6.1.2 Privacy, PHI, and Data Minimization

APIs must minimize exposure of Protected Health Information (PHI). Only authorized participants can access message content and attachments. Audit logs must record blocked or unauthorized communication attempts. Attachment delivery should use time-bounded signed URLs, so the client never receives permanent public links.

6.1.3 Permission Enforcement

The backend enforces communication eligibility rules at multiple points:

- WebSocket Connection: Validate JWT; bind session to authenticated user identity
- Chat Initiation: Verify caregiver-patient assignment; enforce patient initiation restrictions
- Call Initiation: Verify relationship exists; block patient-to-patient and unassigned calls
- Message Routing: Validate sender is participant in conversation before delivery
- All blocked attempts are logged for audit purposes

6.2 Communication APIs (Chat + Audio/Video Calling)

The Communication module provides (1) person-to-person chat messaging and (2) audio/video calling. Chat relies on persistent bidirectional channels for near real-time delivery. Calling relies on WebRTC for media transport, with backend-mediated signaling over WebSockets. REST endpoints are used for history retrieval, metadata, attachment workflows, and audit-friendly session records.

6.2.1 REST Endpoints (Communication)

The following endpoints are representative and may be implemented within a single service (CommunicationService) or separated into ChatService and CallService. Endpoint naming should remain consistent across environments.

Table 4- REST API Endpoints

Endpoint	Method	Purpose	Permission Rules
/api/v1/chat/conversations	GET	List user's conversations	Return only conversations where user is authorized participant
/api/v1/chat/conversations/{id}/messages	GET	Retrieve message history	Verify participant membership; supports pagination
/api/v1/chat/messages	POST	Send message (REST fallback)	Enforce initiation rules; patients blocked if caregiver hasn't initiated
/api/v1/chat/attachments	POST	Request S3 upload URL	Only conversation participants may upload
/api/v1/chat/attachments/{id}	GET	Get download URL	Time-limited signed URL (1 hour); participants only
/api/v1/calls/sessions	POST	Initiate call session	Validate relationship; block patient-to-patient; block unassigned
/api/v1/calls/sessions/{id}	GET	Get call metadata	Only call participants may view

/api/v1/calls/sessions/{id}/end	POST	End call session	Only participants may end; records outcome
/api/v1/calls/sessions/{id}/twilio-token	GET	Get Twilio JWT token	Only accepted call participants; includes room name and access token

Implementation guidance:

- Message persistence: Store messages to support offline delivery.
- Attachments: Store binary content in object storage (e.g., S3); store metadata in PostgreSQL.
- Idempotency: Provide clientMessageId to prevent duplicates during retries.
- Auditing: Record message send attempts (success/blocked) and call session outcomes.

6.2.2 WebSocket Endpoints (Real-Time Channels)

WebSocket channels provide persistent bidirectional communication to support real-time chat delivery and call signaling. All WebSocket connections must be authenticated and must enforce authorization checks for every message routed.

Table 5-WebSocket Endpoints

Endpoint	Purpose	Notes
/ws/v1/chat	Real-time chat messaging, typing indicators, delivery receipts	JWT required at handshake; 30s heartbeat; auto-reconnect on disconnect
/ws/v1/call	Call signaling: invite, accept, decline, end notifications	JWT required; validates user relationship and permissions before routing; coordinates Twilio Video Room creation and access token issuance on accept

6.2.3 WebSocket Message Types and Payloads

WebSocket messages are JSON objects with a common envelope. Recommended fields:

- type: message type identifier
- requestId: client-generated id for correlation
- timestamp: ISO 8601 UTC
- payload: type-specific body

- auth (optional): only for initial AUTH message if not provided in headers

Table 6-Chat Messages Types

Type	Direction	Purpose	Key Payload Fields
CHAT_SEND	Client → Server	Send a chat message	conversationId, messageType, text/attachmentId, clientMessageId
CHAT_DELIVER	Server → Client	Deliver message to recipient	messageId, conversationId, senderId, content, timestamp
CHAT_ACK	Client → Server	Acknowledge delivery/read	messageId, status (delivered read), timestamp
TYPING	Bidirectional	Typing indicator	conversationId, isTyping (true false)
PING/PONG	Bidirectional	Connection keepalive	timestamp (sent every 30 seconds)

Table 7- Call Signaling Types

Type	Direction	Purpose	Key Payload Fields
CALL_INVITE	Client → Server	Initiate a call	calleeUserId, callType (audio video), clientCallId
CALL_RINGING	Server → Client	Notify callee of incoming call	callSessionId, callerUserId, callerName, callType
CALL_ACCEPT	Client → Server	Callee accepts call	callSessionId
CALL_DECLINE	Client → Server	Callee declines call	callSessionId, reason (busy decline)
CALL_READY	Server → Client	Call accepted; client may join Twilio Video Room	callSessionId, roomName, twilioAccessToken
CALL_END	Bidirectional	Call ended	callSessionId, reason (hangup timeout error), duration

6.2.4 Communication Eligibility and Permission Rules (Enforcement Points)

The backend must enforce eligibility and permission rules before establishing chat sessions or call sessions. Key enforcement points include:

- WebSocket connection establishment: validate token; bind connection to user identity.
- Session creation (chat/call): validate role, assignment relationship, and eligibility rules.
- Message routing: validate participant membership for every conversationId/callSessionId.
- Administrative controls: allow caregiver-controlled muting/unmuting per patient where required.

The SRS specifies that caregivers may initiate chat with assigned patients and that patients may be restricted from initiating chat unless the caregiver has explicitly initiated the conversation. The backend must validate these rules and log blocked/unauthorized attempts for audit purposes.

6.2.5 Error Handling and Status Codes (Communication)

REST APIs should return standard HTTP status codes. WebSocket errors should return typed error messages. Errors must avoid exposing sensitive information and must be logged internally for troubleshooting.

Table 8- REST API HTTP Status codes

Scenario	REST/WS Code	Client Action	Audit
Invalid/expired JWT	401 / AUTH_INVALID	Refresh token and retry	Log auth failure
Not authorized for conversation	403 / CHAT_FORBIDDEN	Show access denied	Log blocked attempt
Patient initiation restricted	403 / CHAT_INIT_RESTRICTED	Show 'Caregiver must initiate'	Log blocked initiation
Patient calling unassigned	403 / CALL_NOT_AUTHORIZED	Show 'Can only call assigned caregiver'	Log blocked call
Patient calling patient	403 / CALL_NOT_AUTHORIZED	Show 'Cannot call other patients'	Log blocked call
Recipient offline	200 / CHAT_QUEUED	Show queued status	Store for delivery

Call timeout (no answer)	408 / CALL_TIMEOUT	Show missed call	Record timeout outcome
WebSocket disconnect	WS_DISCONNECTED	Auto-reconnect (exp backoff)	Log disconnect event

6.3 API Design Specification – EVV Module

The backend service orchestrates all data processing, validation, XML generation, and communication with the Netsmart API.

6.3.1 XML Processing service:

Data Validation:

The service validates all EVV records against Netsmart's schema requirements:

- Required fields: Service Date, Start Time, End Time, Service Type, Patient ID, Provider ID
- Data type validation: Dates in ISO 8601 format, times in HH:MM format
- Business rule validation: End time after start time, service duration within limits.
- GPS coordinate validation: Valid latitude/longitude ranges

XML Document generation:

The service generates Netsmart-compliant XML with the following structure:

```
<RenderedServices>  <Metadata>      <ProviderID>12345</ProviderID>
<SourceSystem>PVCC</SourceSystem>    <SubmissionTimestamp>2026-02-
07T10:30:00Z</SubmissionTimestamp>  </Metadata>    <ServiceRecords>
<ServiceRecord>      <VisitID>VID-2026-001</VisitID>
<PatientMedicaidID>1234567890</PatientMedicaidID>      <ServiceDate>2026-02-
07</ServiceDate>      <StartTime>09:00:00</StartTime>
<EndTime>11:30:00</EndTime>      <ServiceType>Personal Care</ServiceType>
<ProviderNPI>9876543210</ProviderNPI>      <GPSCheckIn>
<Latitude>38.9072</Latitude>      <Longitude>-77.0369</Longitude>
<Timestamp>2026-02-07T09:00:15Z</Timestamp>      </GPSCheckIn>      <GPSCheckOut>
<Latitude>38.9075</Latitude>      <Longitude>-77.0365</Longitude>
<Timestamp>2026-02-07T11:30:22Z</Timestamp>      </GPSCheckOut>      </ServiceRecord>
</ServiceRecords> </RenderedServices>
```


6.3.2 Netsmart API Integration

API Authentication:

The service authenticates with Netsmart using OAuth 2.0:

1. Obtain access token from Netsmart authorization server
2. Include token in Authorization header: Bearer {access_token}
3. Refresh token before expiration (typically 1 hour)

Data Transmission:

XML data is transmitted via HTTPS POST to the Netsmart endpoint:

- Endpoint: <https://va.netsmartcloud.com/api/evv/rendered-services>
- Method: POST
- Content-Type: application/xml
- Headers: Authorization, X-Provider-ID, X-Source-System

Response processing:

The service polls for response files at 10–15-minute intervals:

- ACCEPTED response: Updates all records to NETSMART_ACCEPTED status
- REJECTED response: Parses error codes, creates remediation tasks
- PARTIAL response: Identifies specific failed records with error details

6.4 API Design Specification – In-Home Residential Support

The In-Home Residential Support API enables caregivers to submit structured documentation of care activities performed during an EVV-tracked visit. These APIs support the recording and retrieval of ADLs, IADLs, observations, and incident notes, all of which are linked to a verified EVV visit.

All in-home support APIs require authentication and enforce role-based authorization. Support documentation must reference a valid EVV visit identifier, ensuring that care records cannot exist independently of verified visits.

6.4.1 REST Endpoints (In-Home Residential Support)

Table 9 – In-Home Residential Support REST API Endpoints

Endpoint	Method	Purpose	Auth	Role/Eligibility Rules	Notes/ Key Fields

/api/v1/support/activities	POST	Submit an in-home support activity	YES	Caregiver must be associated with visit	Body: visitId, activityType (ADL/IADL), activityLabel, notes (optional), timestamp
/api/v1/support/visits/{visitId}/activities	GET	Retrieve support activities for a visit	Yes	Caregiver (own visits); Supervisor/Admin (authorized scope)	Returns ordered activity list
/api/v1/support/visits/{visitId}/activities	GET	Retrieve a specific support activity	YES	Authorized participant or supervisor	Read-only after visit finalization

6.4.2 Validation and Error Handling (In-Home Residential Support)

The backend validates that each submitted support activity references a valid EVV visit and that the submitting user is authorized for that visit. If the associated EVV visit is finalized, modification, or deletion of support records is restricted based on role and policy.

Invalid visit references, missing required fields, or unauthorized access attempts to result in appropriate error responses. Standard HTTP status codes are used to indicate failure conditions, and all submission attempts generate audit log entries to support compliance review and traceability.

6.5 API Design Specification – AI-Assisted Services

AI assisted services are exposed through versioned REST endpoints within the CareConnect backend. These endpoints are accessible only to authenticated users with appropriate role-based permissions and valid contextual relationships (i.e., assigned caregiver to visit).

All AI endpoints delegate processing to the AI Orchestration Service, which constructs and executes AWS Bedrock requests internally. External clients never interact directly with Bedrock or supply AI provider credentials.

Example endpoints include:

- “POST /api/v1/ai/visits/{visitId}/summary - Generates a summary of visit activities.”
- “POST /api/v1/ai/documentation-assist – Produces a draft from structured care data”
- “POST /api/v1/ai/faq -Returns nonclinical assistance related to the use of the application.”

Error handling follows standard HTTP semantics, with upstream AI failures mapped to controlled, user friendly responses that do not expose provider specific details.

6.6 API Design Specification – Administration, Audit, and Reporting

The Administration, Audit, and Reporting APIs support system oversight, compliance verification, and operational monitoring. These APIs are intended for authorized administrative and supervisory roles only and are not accessible to patients or standard caregivers unless explicitly permitted by policy. All endpoints in this module require elevated authorization and generating audit records.

6.6.1. Scope and Responsibilities

This module provides the following capabilities:

- User and role management (administrative view only; provisioning may be externalized).
- Audit trail retrieval for security- and compliance-relevant events.
- Reporting access to EVV records, communication, metadata, and system usage.
- System health and configuration visibility (read-only where possible).

The module is designed to support regulatory compliance, internal reviews, and academic evaluation without exposing sensitive PHI beyond the minimum necessary.

6.6.2. REST Endpoints (Administration & Audit)

Table 8 – Administration and Audit REST APIs

Endpoint	Method	Purpose	Auth	Role / Eligibility Rules	Notes
/api/v1/admin/users	GET	Retrieve list of users and roles	Yes	Admin or Supervisor only	Supports pagination and filtering by role/status
/api/v1/admin/users/{userId}	GET	Retrieve user profile and role assignments	Yes	Admin or Supervisor only	PHI-minimized view
/api/v1/admin/audit/logs	GET	Query audit logs	Yes	Admin only	Filters: eventType, date range, actorId
/api/v1/admin/audit/logs/{logId}	GET	Retrieve detailed audit record	Yes	Admin only	Read-only, immutable
/api/v1/admin/reports/evv	GET	Generate EVV compliance reports	Yes	Admin or Compliance role	Supports date range and export formats
/api/v1/admin/reports/communication	GET	Retrieve communication metadata reports	Yes	Admin only	No message content included
/api/v1/admin/system/health	GET	Retrieve system health summary	Yes	Admin only	Aggregated, non-sensitive metrics

Implementation guidance:

- Audit logs are append-only and cannot be modified or deleted through APIs.
- Report exports (PDF/CSV) should be generated asynchronously for large datasets.
- Sensitive content (message bodies, attachments) must never be exposed through reporting APIs

6.6.3 Audit Events and Logging Rules

The backend must generate audit events for:

- Authentication attempts (success and failure).
- Authorization failures (403 conditions).
- Access to PHI-related resources.
- Administrative actions (role changes, report access).
- Communication session lifecycle events (metadata only).

Each audit record should include:

- Actor user ID
- Action type
- Target resource identifier
- Timestamp (UTC)
- Outcome (success/failure)
- Source context (API, WebSocket, system)

Audit data is retained according to defined compliance and project policies.

6.6.4 Security and Access Control

- All Administration APIs require JWT authentication with elevated RBAC checks.
- Supervisor roles may have read-only access to limited reports.
- Administrative access attempts are logged regardless of success.
- APIs must enforce least-privilege access and data minimization principles.

6.7 API Versioning and Backward Compatibility (Placeholder)

CareConnect APIs are versioned to support iterative development, controlled evolution, and backward compatibility across frontend clients and integrated services.

6.7.1 Versioning Strategy

- REST APIs are versioned using a URI-based strategy:
 - Example: /api/v1/...
- WebSocket endpoints are versioned in the path where feasible:
 - Example: /ws/v1/chat, /ws/v1/call
- Major version increments (v1 → v2) indicate breaking changes.

6.7.2 Backward Compatibility Principles

The system follows these compatibility rules:

- Minor changes (adding optional fields, new endpoints) must not break existing clients.
- Existing fields must not change meaning or data type within the same major version.
- Deprecated fields or endpoints must remain supported for at least one academic release cycle unless security risks require immediate removal.

6.7.3 Deprecation Policy

When an API element is deprecated:

- It is clearly documented as deprecated.
- A replacement endpoint or field is provided.
- Deprecation warnings may be included in response headers.
- Removal occurs only in a subsequent major version.

6.7.4 Client Compatibility Considerations

- Frontend clients must declare the API version they target.
- Backend services may support multiple API versions concurrently when feasible.
- Feature flags may be used to selectively enable new behavior without version changes.

6.7.5 Error Handling Across Versions

- Error formats remain consistent across versions.
- Version mismatch errors return a clear, non-sensitive message indicating incompatibility.
- Clients encountering unsupported versions should fail gracefully and prompt for update.

7. Data Model Specification

This section defines the logical data models required to implement CareConnect features. The Communication data models (chat and audio/video calling) are specified in detail in this revision. Placeholder headings are provided for EVV, In-Home Residential Support, AI-Assisted Services, and Administration/Audit models so other team members can complete those areas.

7.1 Data Modeling Approach and Assumptions

CareConnect uses a relational data model implemented in PostgreSQL. Core entities use stable identifiers (UUID recommended), with explicit foreign-key relationships to enforce referential integrity. Time fields are stored as UTC timestamps. For Communication features, large binary payloads (attachments) are stored in object storage (e.g., Amazon S3), while the database stores metadata and access control references.

Design principles:

- Normalize core entities (users, conversations, messages, call sessions) to reduce duplication.
- Use append-friendly tables for auditability (e.g., message events, call state transitions).
- Support efficient retrieval (indexes for recent messages per conversation, active call sessions).
- Enforce access rules at the service layer (RBAC + relationship checks), while preserving strong integrity constraints in the schema.

7.2 Communication Data Models (Chat + Audio/Video Calling)

The Communication domain supports (1) chat conversations and message exchange and (2) call session coordination. Data models are designed to support offline delivery, delivery/read status, attachments, and audit-friendly call session records.

7.2.1 Entity Relationship Summary (Communication)

Key relationships:

- Users participate in one or more Conversations (many-to-many via ConversationParticipant).
- Conversation contains Messages (one-to-many).
- Message may reference an Attachment (optional).
- CallSession links two or more participants (via CallParticipant) and may reference Conversation context (optional).
- CallSession contains CallSignalEvents (one-to-many) to support troubleshooting/audit (optional but recommended).

7.2.2 Conversation

Table 9-Conversation Entity relationships

Field	Type	Null?	Key/Index	Description	Notes / Constraints
conversation_id	UUID	No	PK	Unique conversation identifier	Generated server-side
conversation_type	VARCHAR(32)	No	IDX	Conversation type	E.g., 'P2P' (patient-caregiver), 'GROUP' (future)
created_by_user_id	UUID	No	FK->User	User who created/initiated the conversation	Must be authorized to initiate per SRS rules
created_at	TIMESTAMP (UTC)	No		Conversation creation time	
status	VARCHAR(16)	No	IDX	Conversation lifecycle state	E.g., ACTIVE, ARCHIVED
subject	VARCHAR(120)	Yes		Optional display label	Optional for P2P; useful for future expansion

7.2.3 ConversationParticipant

Table 10- Conversation participants Entity relationships

Field	Type	Null?	Key/Index	Description	Notes / Constraints
conversation_id	UUID	No	PK(FK)	Conversation reference	FK->Conversation; composite PK with user_id

user_id	UUID	No	PK(FK)	Participant user reference	FK->User; composite PK with conversation_id
role_in_conversation	VARCHAR(32)	No	IDX	Role within conversation	E.g., PATIENT, CAREGIVER
can_initiate	BOOLEAN	No		Indicates if user can initiate messages	Supports caregiver-initiated rule enforcement
joined_at	TIMESTAMP (UTC)	No		When the participant joined	
left_at	TIMESTAMP (UTC)	Yes		When the participant left (if applicable)	Null when active
muted	BOOLEAN	No		Whether participant has muted notifications	Client preference; does not block message delivery

7.2.4 Message

Table 11- Messages Entity Relationships

Field	Type	Null ?	Key/Index	Description	Notes / Constraints
message_id	UUID	No	PK	Unique message identifier	Generated server-side
conversation_id	UUID	No	FK->Conversation , IDX	Conversation reference	Indexed for retrieval
sender_user_id	UUID	No	FK->User, IDX	Sender user reference	Sender must be a participant

message_type	VARCHAR(16)	No	IDX	Message content type	TEXT, IMAGE, AUDIO, VIDEO, SYSTEM
text_body	TEXT	Yes		Message text content	Required for TEXT; null for binary attachments
attachment_id	UUID	Yes	FK->Attachment	Reference to attachment metadata	Used when message_type is IMAGE/AUDIO/VIDEO
client_message_id	VARCHAR(64)	Yes	IDX	Client idempotency key	Used to prevent duplicates during retries
sent_at	TIMESTAMP (UTC)	No	IDX	Server receipt timestamp	
edited_at	TIMESTAMP (UTC)	Yes		Edit timestamp	If message edits are supported
is_deleted	BOOLEAN	No	IDX	Soft-delete flag	Prefer soft delete for auditability

7.2.5 MessageReceipt

Table 12- Message Receipt Entity Relationships

Field	Type	Null?	Key/Index	Description	Notes / Constraints
message_id	UUID	No	PK(FK)	Message reference	Composite PK with user_id
user_id	UUID	No	PK(FK)	Recipient user reference	FK->User
status	VARCHAR(16)	No	IDX	Delivery/read status	SENT, DELIVERED, READ
status_at	TIMESTAMP (UTC)	No		Timestamp of last status update	

delivery_channel	VARCHAR(16)	Yes		How message was delivered	WS, PUSH (future), POLL

7.2.6 Attachment

Table 13- Attachment Entity relationships

Field	Type	Null?	Key/Index	Description	Notes / Constraints
attachment_id	UUID	No	PK	Unique attachment identifier	
conversation_id	UUID	No	FK->Conversation, IDX	Conversation scope for access control	Enables participant-only access
uploaded_by_user_id	UUID	No	FK->User	Uploader reference	
storage_provider	VARCHAR(16)	No		Storage provider identifier	E.g., S3
storage_key	VARCHAR(512)	No		Provider object key/path	Never expose directly to client
mime_type	VARCHAR(128)	No		Attachment MIME type	
size_bytes	BIGINT	No		Attachment size	Validate max size per policy
sha256	VARCHAR(64)	Yes	IDX	Integrity hash	Optional; supports dedupe/integrity checks
created_at	TIMESTAMP (UTC)	No	IDX	Upload created time	

7.2.7 CallSession

Table 14- Call Session Entity Relationships

Field	Type	Null?	Key/Index	Description
call_session_id	UUID	No	PK	Unique call session identifier
call_type	VARCHAR(16)	No	IDX	AUDIO or VIDEO
initiator_user_id	UUID	No	FK->User, IDX	Caller reference
callee_user_id	UUID	No	FK->User, IDX	Callee reference
state	VARCHAR(16)	No	IDX	CREATED, RINGING, ACTIVE, ENDED, FAILED
twilio_room_name	VARCHAR(128)	Yes	IDX	Unique Twilio room name (generated on accept)
twilio_token_issued_at	TIMESTAMP	Yes		When access tokens were generated
created_at	TIMESTAMP	No		Call session creation time
started_at	TIMESTAMP	Yes		When call became ACTIVE (both joined Twilio Video room)
ended_at	TIMESTAMP	Yes		When call ended
end_reason	VARCHAR(32)	Yes		HANGUP, DECLINED, BUSY, TIMEOUT, ERROR

7.2.8 CallParticipant

Table 15-Call Participants Entity Relationships

Field	Type	Null?	Key/Index	Description	Notes / Constraints
call_session_id	UUID	No	PK(FK)	Call session reference	Composite PK with user_id

user_id	UUID	No	PK(FK)	Participant reference	FK->User
participant_role	VARCHAR(16)	No		Role in call	CALLER, CALLEE
joined_at	TIMESTAMP (UTC)	Yes		When participant joined media session	
left_at	TIMESTAMP (UTC)	Yes		When participant left	
last_signal_at	TIMESTAMP (UTC)	Yes	IDX	Last signaling activity time	Useful for timeout detection

7.2.9 CallSignalEvent (Optional / Recommended for Troubleshooting)

Table 16- Call Signal Event Entity Relationships

Field	Type	Null?	Key/Index	Description	Notes / Constraints
event_id	UUID	No	PK	Unique signaling event identifier	
call_session_id	UUID	No	FK->CallSession, IDX	Call session reference	
event_type	VARCHAR(16)	No	IDX	Signaling event type	INVITE, ACCEPT, SDP, ICE, END, ERROR
from_user_id	UUID	Yes	FK->User	Sender of signal	Null for system-generated events

to_user_id	UUID	Yes	FK->User	Recipient of signal	May be null for broadcast within session
payload_size_bytes	INTEGER	Yes		Size of signaling payload	Supports validation/audit without storing full SDP
created_at	TIMESTAMP (UTC)	No	IDX	Event timestamp	
notes	VARCHAR(256)	Yes		Non-sensitive notes	Avoid storing SDP/ICE contents if PHI risk or size concerns

7.2.10 Indexing and Query Patterns

- Recommended indexes and query patterns:
- Message(conversation_id, sent_at DESC) for efficient chat history retrieval.
- Message(sender_user_id, sent_at DESC) for user-level activity queries.
- MessageReceipt(user_id, status, status_at DESC) to retrieve unread messages efficiently.
- CallSession(state, created_at DESC) to identify active/ringing calls.
- Attachment(conversation_id, created_at DESC) to support attachment browsing.

All history endpoints should paginate results. For large conversations, cursor-based pagination (before/after timestamp) is recommended.

7.2.11 Data Retention and Compliance Considerations (Communication)

Communication data may contain sensitive information. The system should support retention policies aligned with project and compliance requirements. Soft-delete flags enable operational deletion while preserving auditability. Attachments should be managed with lifecycle policies in object storage (e.g., S3 lifecycle rules) to control retention and cost. Audit logs should record access to message content and attachment downloads where required.

7.3 Data Models – EVV Module

The core EVV data model consists of interconnected entities that capture all 6 federal EVV elements, support corrections with immutable audit trails, queue offline operations, and track locations separately

7.3.1 Core Entities

- EvvRecord - Contains all 6 federal EVV elements (service type, individual, caregiver, date, time in/out, location)
- EvvCorrection - Tracks corrections with reason codes and approval workflow
- EvvAuditEvent - Immutable audit log for all actions on EVV records
- EvvRecordLocation - Separate storage for check-in/check-out GPS coordinates
- EvvOfflineQueue - Queue for operations that occurred offline, with retry logic
- ScheduledVisit - Visit scheduling with status transitions

EVV Data Model (Core Entities)

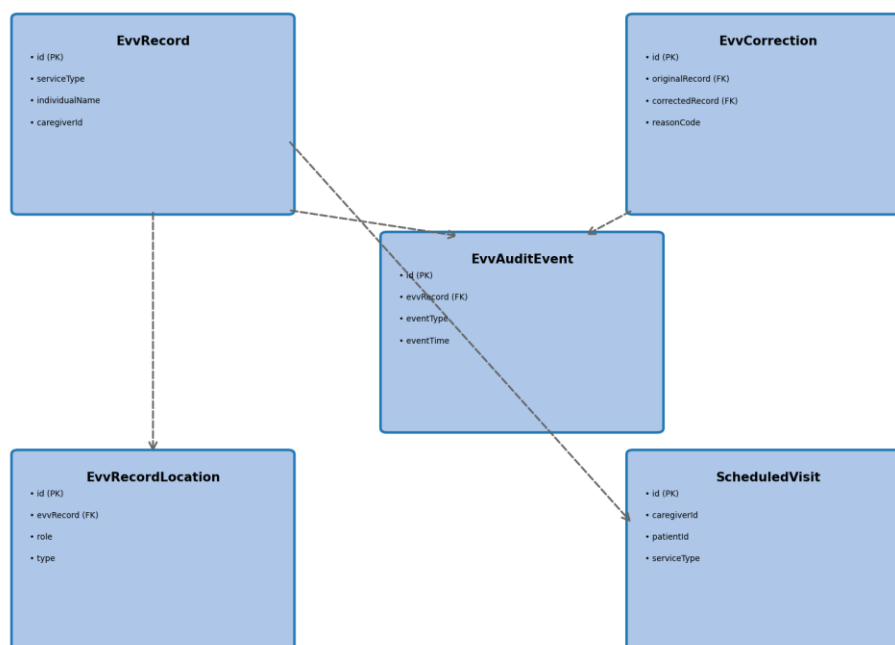


Figure: Core Data Model showing entity relationships and foreign keys for the EVV system.

The EVV data model supports the secure and auditable recording of caregiver visits. EVV entities capture visit lifecycle information, including check-in and check-out events, participant associations, and visit status. These models serve as the authoritative source of visit verification and provide the parent context for in-home residential support documentation.

Below are the core entities:

Field Name	Data Type	Description
visit_id	VARCHAR(50)	Unique identifier for visit
patient_medicaid_id	VARCHAR(20)	Patient Medicaid ID (required)
service_date	DATE	Federal Element 1: Service date
start_time	TIME	Federal Element 2: Service start time
end_time	TIME	Federal Element 3: Service end time
service_type	VARCHAR(100)	Federal Element 4: Type of service
gps_checkin_lat	DECIMAL(10,7)	Federal Element 5: Service location latitude
gps_checkin_lng	DECIMAL(10,7)	Federal Element 5: Service location longitude
provider_npi	VARCHAR(10)	Federal Element 6: Provider NPI

7.4 Data Models – In-Home Residential Support Module

The In-Home Residential Support data model captures structured documentation of care activities performed during an EVV-tracked visit. These entities store ADL and IADL activity logs, observations, and incident notes. All records are linked to an existing EVV visit, ensuring that care documentation remains contextually bound to verified visit data.

7.4.1 Entity Relationship Summary (In-Home Residential Support)

Key Relationships:

- An EVV visit contains zero or more Support Activities
- Each Support Activity is created by a caregiver and linked to exactly one visit.
- Supervisory roles may access support activities in a read-only capacity after visit finalization.

7.4.2 Support Activity

Table 18 – SupportActivity Entity Relationships

Field	Type	Null?	Key/Index	Description	Notes/Constraints
activity_id	UUID	No	PK	Unique activity identifier	
visit_id	UUID	No	FK -> EVVVisit, IDX	Associated EVV visit	Must reference valid visit
caregiver_user_id	UUID	No	FK -> User	Caregiver submitting activity	Must match visit caregiver
activity_category	VARCHAR (128)	No	IDX	Activity classification	ADL, IADL, OBSERVATION, INCIDENT
activity_label	VARCHAR (128)	No		Activity name	Example: “Meal Preparation”
activity_notes	TEXT	Yes		Free-text notes	Optional
performed_at	TIMESTAMP (UTC)	No	IDX	Time activity occurred	Within visit timeframe
created_at	TIMESTAMP (UTC)	No		Record creation time	
is_locked	BOOLEAN	No	IDX	Edit restriction flag	True after visit finalization

7.5 Data Models – AI-Assisted Services

AI assisted services introduce supplemental data models to support auditing, observability, and performance analysis. The AiRequest entity records metadata associated with each AI invocation, including the requesting user, associated visit context, request type, and timestamp.

The corresponding AiResponse entity captures execution metadata such as model identifier, token usage, response latency, and truncation indicators. Generated AI content does not persist as authoritative clinical data unless explicitly accepted and saved by the user through standard application workflows.

These models support compliance requirements, cost monitoring, and ongoing evaluation of AI system performance without storing unnecessary sensitive content.

7.6 Data Models – Administration, Audit, and Reporting (Placeholder)

7.7 Data Dictionary Alignment (Placeholder)

8. UI / UX Design Specification

This section describes the user interface (UI) and user experience (UX) design approach for CareConnect. At this stage of Milestone 2, the project has sufficient information to document UI goals, navigation structure, role-based screens, accessibility requirements, and the intended interaction patterns for major workflows. Detailed screen-by-screen pixel specifications and final visual assets (e.g., Figma component library exports) may be added in later iterations as prototypes mature.

In this revision, Communication-related UI (chat and audio/video calling) is drafted in detail. Placeholder headings are provided for EVV, In-Home Residential Support, AI-Assisted Services, and Administration screens so other team members can complete those areas.

8.1 UI Design Approach and Principles

CareConnect follows a mobile-first, accessibility-driven UI strategy suitable for healthcare contexts. The UI emphasizes clarity, safety, and low cognitive load. Design principles include:

- Consistent navigation and predictable screen layouts.
- Strong visual hierarchy (headings, grouping, and progressive disclosure).
- Minimal text density and plain-language labels.
- Large touch targets and adequate spacing for mobile usability.
- Error prevention and clear recovery pathways.
- Role-based experiences (patients see only relevant tasks; caregivers have additional workflow controls).

The frontend is implemented in Flutter, enabling consistent UI behavior across Android and web platforms.

8.2 Accessibility Requirements (WCAG-Aligned)

CareConnect UI is designed to align with accessibility best practices and course requirements. Key considerations include:

- Color contrast: maintain sufficient contrast between text and background.
- Typography: scalable fonts, readable sizes, and support for system font scaling.

- Screen reader support: semantic labeling of controls, logical focus order, descriptive buttons.
- Keyboard navigation (web): tab order consistency and visible focus indicators.
- Touch targets: minimum target size and spacing.
- Error messaging: descriptive, non-technical messages with guidance for recovery.
- Reduced cognitive load: avoid unnecessary multi-step complexity and provide orientation cues.

Accessibility checks should be integrated into UI testing (e.g., Flutter widget tests for semantics, manual checks using platform accessibility tools).

8.3 Global Navigation and Information Architecture

The CareConnect UI is organized around primary user tasks. Navigation is role-aware and may present different menu items based on the authenticated user's role.

Proposed primary navigation groups:

- Home/Dashboard: summary of actionable items, alerts, and next steps.
- Communication: chat conversations and calling entry points.
- Visits/EVV: visit check-in/check-out, visit history.
- Support Logs: ADL/IADL documentation tied to visits.
- Profile/Settings: account, accessibility preferences, and notifications.

Navigation patterns:

- Mobile: bottom navigation bar for top-level modules + nested screens.
- Web: responsive sidebar or top navigation with the same modules.
- Contextual navigation: deep links from a visit record to support logs; from a conversation to call controls.

8.4 Common UI Components and Design System Elements

The UI uses reusable components to ensure consistency and testability:

- Buttons: primary, secondary, destructive; disabled states clearly visible.
- Forms: validated fields with inline error text.
- Lists: conversation lists, visit lists, activity logs with clear item separation.
- Cards: grouped information (visit summary cards, patient summary cards).
- Modals/Dialogs: confirmations (e.g., end call confirmation), permission prompts.
- Notifications/Toasts: transient feedback for non-blocking actions.
- Loading/Empty states: skeleton loaders and informative empty messaging.

Flutter implementation should centralize styling (theme, typography, spacing) to enforce consistency and support future iteration.

8.5 Communication UI (Chat + Audio/Video Calling)

8.5.1 Communication Entry Points

Users access Communication features through dashboard shortcuts, the primary navigation 'Communication' module, and contextual actions from profile pages.

Entry points enforce permission rules at the UI level:

- Patients see 'Message Caregiver' button only if conversation already exists (caregiver initiated)
- Patients see 'Call Caregiver' button only for their assigned caregiver
- Caregivers see full communication options for all assigned patients
- Unavailable actions are hidden (not shown as disabled) to reduce confusion

8.5.2 Permission-Based UX Messages

When users attempt blocked actions, the UI displays clear, non-technical messages:

Scenario	User-Facing Message
Patient tries to initiate chat (no prior conversation)	"Your caregiver must start the conversation first. You'll be able to reply once they message you."
Patient tries to call unassigned caregiver	"You can only call your assigned caregiver."
Patient tries to call another patient	"Calls between patients are not available."
Callee is offline	"[Name] is currently unavailable. Try again later."
Call times out (no answer)	"[Name] didn't answer. You can try calling again or send a message."

8.5.3 Message Thread Screen (Updated)

The message thread screen displays real-time updates via WebSocket:

- Messages appear instantly without refresh (WebSocket CHAT_DELIVER)
- Typing indicator shows when other participant is typing
- Message status badges: ✓ Sent, ✓✓ Delivered, ✓✓ Read (blue)
- Connection status indicator: 'Connected' (green) or 'Reconnecting...' (yellow)

- Offline banner: 'You're offline. Messages will be sent when you reconnect.'

8.5.4 Audio/Video Call UI (Twilio Integration)

Outgoing Call Screen

- Shows callee name and profile photo
- 'Calling...' status with animated indicator
- Cancel button to abort before answer
- Timeout after 60 seconds with 'No Answer' message

Incoming Call Screen

- Full-screen modal overlay (cannot be dismissed accidentally)
- Shows caller name and profile photo
- 'Incoming [Audio/Video] Call' label
- Large Accept (green) and Decline (red) buttons
- Ringtone/vibration per platform conventions

Active Call Screen (Twilio Video)

Once both parties join the Twilio Video Room, the application renders the in-call UI using media tracks and state provided by the Twilio Video SDK:

- Remote video (primary view) and optional self-view (picture-in-picture)
- Controls: Mute microphone, Toggle camera, Switch camera, End call
- Connection quality indicator
- Call duration timer
- Prominent End Call button

Call End Summary

- Brief overlay showing call duration
- Option to 'Call Again' or 'Send Message'
- Auto-dismiss after 5 seconds

8.5.5 Offline and Reconnection UX

The UI handles connectivity changes gracefully:

- WebSocket disconnect: Yellow banner 'Reconnecting...' with spinner
- Reconnection success: Green toast 'Connected' (auto-dismiss)
- Messages sent while offline: Shown with clock icon, sent automatically on reconnect
- Failed messages after all retries: Red warning with 'Retry' button

8.6 EVV

The Flutter application provides an intuitive, material design-based interface optimized for desktop web, tablet, and mobile devices. All screens are responsive and adapt to various screen sizes.

8.6.1 Dashboard Overview

The dashboard serves as the central hub for users, providing at-a-glance metrics, recent activity, and quick access to key functions.

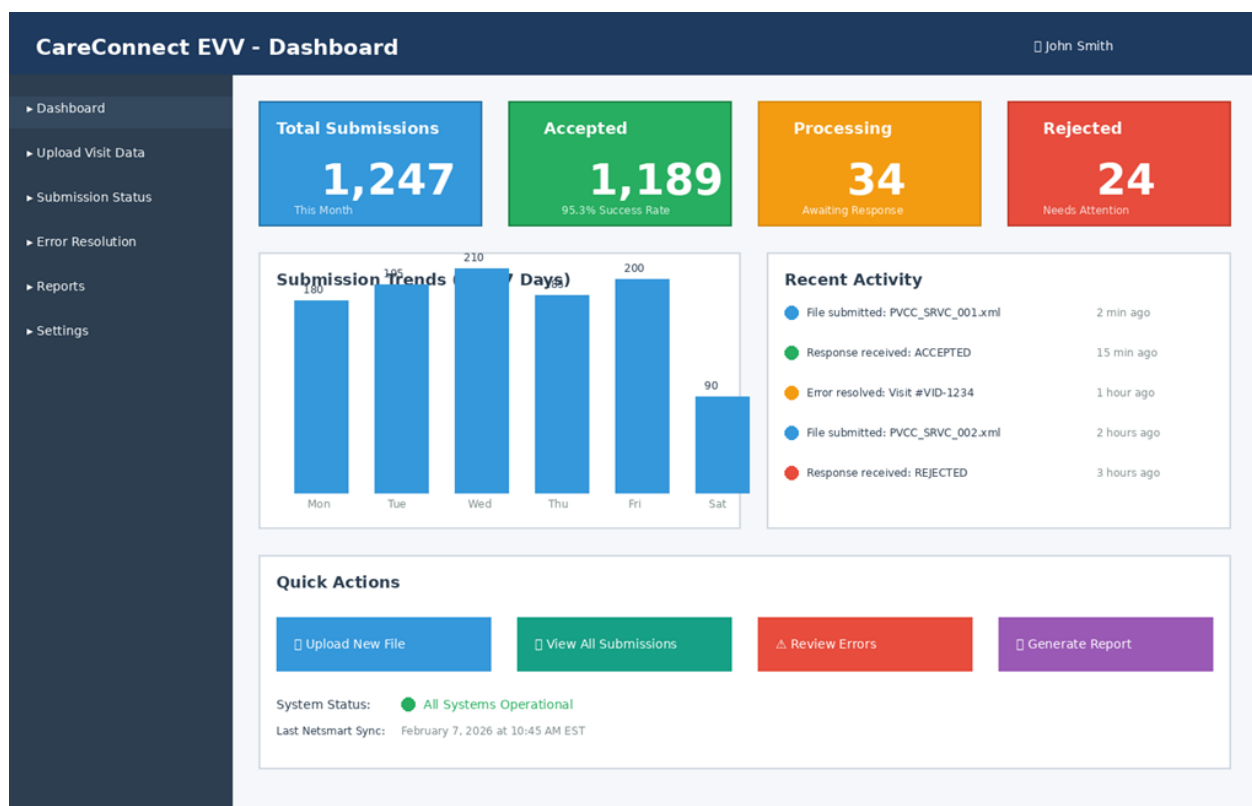


Figure: CareConnect EVV Dashboard showing submission statistics, trend charts, recent activity, and quick action buttons

Dashboard components:

Statistics Cards: Four metric cards displaying Total Submissions, accepted (with success rate), Processing, and Rejected counts. Color-coded for quick status recognition (blue, green, yellow, red).

Submission Trends Chart: Interactive bar chart showing daily submission volumes for the past 7 days, built with Charts Flutter package for smooth animations and touch/hover interactions.

Recent Activity Feed: Real-time activity stream showing file submissions, response receipts, and error resolutions with color-coded status indicators and timestamps.

Quick Actions: Four primary action buttons providing one-tap access to Upload New File, View All Submissions, Review Errors, and Generate Report.

System Status: Live status indicator showing system health and last Netsmart synchronization timestamp.

8.6.2 Upload Visit Data

The upload interface provides a streamlined workflow for submitting EVV data files to Netsmart.

The screenshot displays the 'CareConnect EVV - Netsmart Integration Portal'. On the left is a dark sidebar with navigation links: Dashboard, Upload Visit Data (selected), Submission Status, Error Resolution, Reports, and Settings. The main content area is titled 'Upload EVV Data to Netsmart'. It features a large light blue box with the text 'Drag & Drop XML/JSON file here or click to browse'. Below this are input fields for 'Provider ID' and 'Source System' (pre-filled with 'PVCC'). A 'Submission Method' section has two radio buttons: 'API' (selected) and 'SFTP'. A green 'Submit to Netsmart' button is positioned to the left of a yellow 'File Validation' box. The validation box lists: 'Max file size: 4MB', 'Max records: 1,200', and 'Format: XML/JSON'. Below the upload section is a 'Recent Submissions' table.

File Name	Submitted	Records	Status	Action
PVCC_SRVC_20260206_001.xml	02/06 10:45 AM	143	ACCEPTED	View Details
PVCC_SRVC_20260206_002.xml	02/06 11:23 AM	87	PROCESSING	View Details
PVCC_SRVC_20260206_003.xml	02/06 02:15 PM	215	REJECTED	View Details

Figure: File Upload Interface with drag-and-drop zone, configuration fields, validation info, and recent submissions table

Upload Features:

- Drag-and-drop file zone with visual feedback (built with file picker package)
- Real-time validation: Format (XML/JSON), size (<4MB), record count (<1,200)
- Provider ID input with autosuggestion from user profile
- Source system selection (pre-populated dropdown)

- Submission method toggle: API (default) or SFTP
- Submit button with loading indicator during transmission.
- Recent submissions table with expandable rows for details

8.6.3 Submission Status Interface

The submission status page provides comprehensive tracking and filtering capabilities for all EVV file submissions.

CareConnect EVV - Submission Status

Filter Submissions

Status: Date Range: Provider:

All Submissions Showing 1-10 of 247 submissions

Submission ID	File Name	Date/Time	Records	Status	Actions
SUB-2026-0247	PVCC_SRVC_20260207_001.xml	02/07 10:45 AM	143	ACCEPTED	Details
SUB-2026-0246	PVCC_SRVC_20260207_002.xml	02/07 09:30 AM	87	ACCEPTED	Details
SUB-2026-0245	PVCC_SRVC_20260207_003.xml	02/07 08:15 AM	215	PROCESSING	Details
SUB-2026-0244	PVCC_SRVC_20260206_015.xml	02/06 04:20 PM	156	REJECTED	Details Fix
SUB-2026-0243	PVCC_SRVC_20260206_014.xml	02/06 02:45 PM	98	ACCEPTED	Details
SUB-2026-0242	PVCC_SRVC_20260206_013.xml	02/06 01:10 PM	201	ACCEPTED	Details
SUB-2026-0241	PVCC_SRVC_20260206_012.xml	02/06 11:30 AM	176	PROCESSING	Details
SUB-2026-0240	PVCC_SRVC_20260206_011.xml	02/06 10:00 AM	122	REJECTED	Details Fix

◀ Previous 1 2 3 4 5 Next ▶

Figure: Submission Status page with advanced filtering, sortable table, status badges, and pagination controls

Status Page Features:

- Multi-criteria filters: Status (All/Accepted/Processing/Rejected), Date Range, Provider ID
- Search functionality across submission IDs and file names
- Export to CSV button for reporting and analysis
- Sortable table columns (Submission ID, File Name, Date/Time, Records, Status)
- Color-coded status badges: Green (Accepted), Yellow (Processing), Red (Rejected)
- Action buttons: Details (view full submission info), Fix (for rejected submissions)
- Pagination controls with page numbers and navigation arrows
- Record count display showing current range and total submissions

8.6.4 Error Resolution Interface

The error resolution page provides detailed error analysis and guided remediation for rejected submissions.

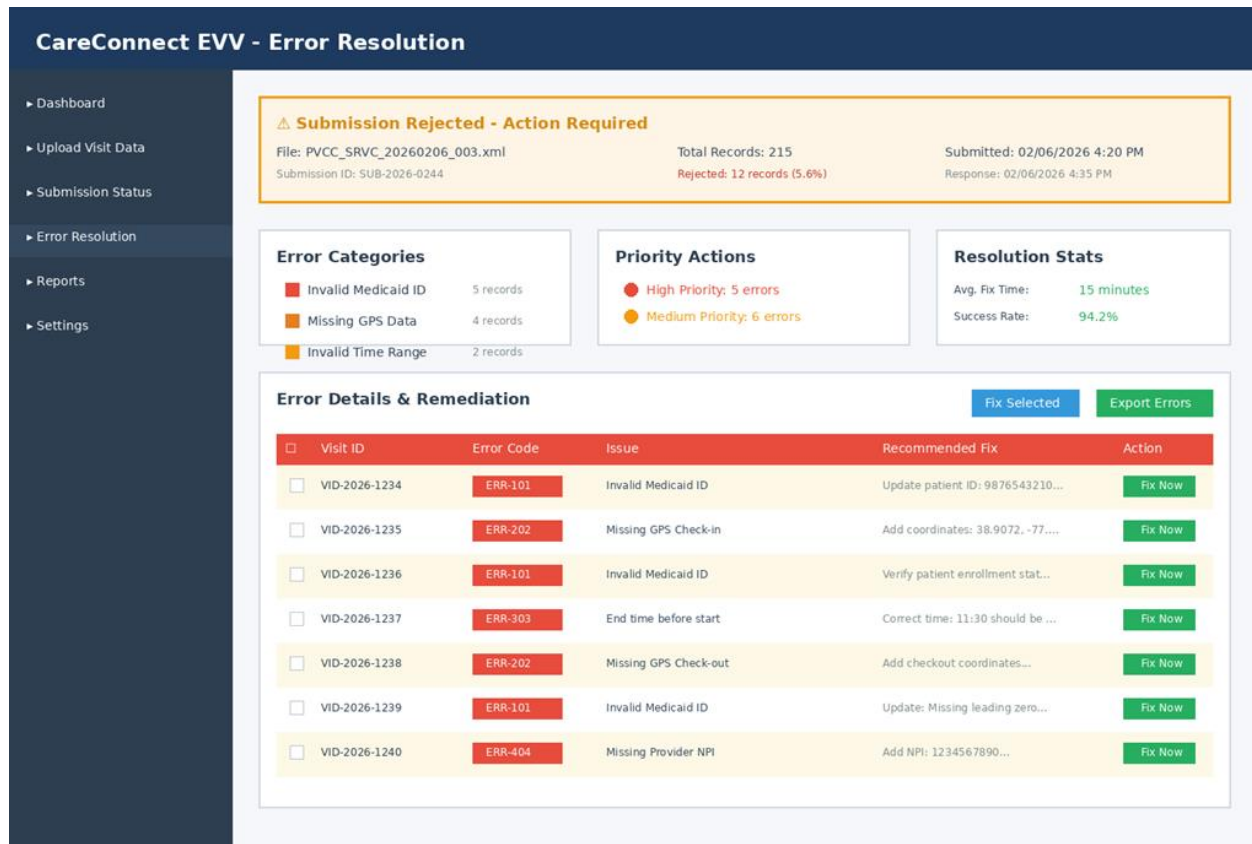


Figure: Error Resolution interface showing error summary, categorization, priority actions, and detailed remediation guidance

Error Resolution features:

- Error Summary Banner: High-visibility alert showing file details and rejection statistics
- Error Categories Widget: Breakdown by error type with color-coded indicators and record counts
- Priority Actions Panel: High/Medium priority classification for efficient triage
- Resolution Statistics: Average fix time and success rate metrics
- Detailed Error Table: Visit ID, Error Code, Issue Description, Recommended Fix
- Bulk Actions: Fix Selected checkbox selection and batch processing
- Export Errors button for offline analysis and documentation
- Fix Now buttons with modal dialogs for inline correction

8.7 In-Home Residential Support UI

The In-Home Residential Support UI enables caregivers to document care activities performed during an EVV-tracked visit in a structured, intuitive manner. The UI is designed to support quick entry during active visits while maintaining clarity and consistency across different activity types.

Support documentation screens are accessible only when a valid EVV visit is active or when viewing completed visit records. This contextual access ensures that care activities remain logically tied to verified visits.

8.7.1 Activity Logging Experience

Caregivers can add support activities through a simple form-based interface. Activity categories such as ADLs, IADLs, observations, and incidents are presented as selectable options to reduce free-text entry and improve data consistency. Optional notes fields allow caregivers to capture additional context when needed.

Each activity submission provides immediate visual feedback confirming successful entry. The UI allows caregivers to review a list of recorded activities associated with the visit, ordered chronologically for clarity.

8.7.2 Incident Documentation

When documenting incidents, the UI presents additional fields appropriate to the selected incident type, such as severity level or follow-up indicators. These fields are displayed conditionally to avoid overwhelming the user during routine documentation.

Clear visual cues differentiate incident entries from standard activity logs, supporting faster review by supervisory users.

8.7.3 Read-Only and Review States

Once an EVV visit is finalized, associated in-home residential support records transition to a read-only state. The UI visually distinguishes editable and locked records to prevent confusion. Supervisory users may access completed records for review, but modification controls are restricted based on role and policy.

8.7.4 Accessibility Considerations

Both EVV and in-home residential support screens follow the system's accessibility design principles, including:

- Clear visual hierarchy and consistent layouts
- Large touch targets for primary actions

- Descriptive labels compatible with screen readers
- Error messages that provide guidance without relying on color alone

These considerations ensure that caregivers with varying levels of technical proficiency can complete required workflows confidently and accurately.

8.8 AI-Assisted Services UI

AI assisted functionality is presented to users as optional, clearly labeled enhancements within existing CareConnect workflows. Entry points include contextual actions such as “Generate Visit Summary” or “Draft Documentation”, integrated into visit and care documentation screens.

While AI processing is underway, the interface provides visible progress indicators and allows users to cancel requests. Generated content is displayed in a distinct, editable format and is explicitly identified as AI generated. Users retain full control to accept, modify, or discard AI suggestions before saving any content.

9. Security Considerations

9.1.1 Data Privacy & HIPAA Compliance

CareConnect is designed to protect Protected Health Information (PHI) in accordance with healthcare privacy principles. The system enforces minimum-necessary data access and ensures that only authorized users can view sensitive information. Audit trails are maintained for access to PHI, supporting regulatory compliance and incident review. Data handling policies minimize exposure by avoiding unnecessary duplication or uncontrolled data sharing.

9.1.2 Authentication & Authorization

All users must authenticate before accessing protected system functions. Authentication is implemented using secure token-based mechanisms. Authorization is enforced using Role-Based Access Control (RBAC) combined with relationship-based rules (e.g., caregiver-patient assignment). Unauthorized attempts are logged for audit and security monitoring.

9.1.3 Secure Data Transmission

All client-server communication is encrypted using TLS. WebSocket channels use WSS to ensure confidentiality and integrity of real-time messaging and call signaling. No sensitive data is transmitted in clear text.

9.1.4 API Security

APIs validate authentication tokens and enforce authorization on every request. Input validation prevents injection and malformed payload attacks. Rate limiting and request size limits protect against denial-of-service conditions.

9.1.5 Data Storage & Backup

Persistent data is encrypted at rest. Regular backups are performed and retained according to defined policies. Backups are stored separately from production systems to reduce the risk of total data loss.

9.1.6 Mobile Application Security

The mobile application uses secure storage for tokens and does not persist plaintext credentials. Session timeouts and inactivity limits reduce exposure in case of device compromise.

9.1.7 AI-Powered Features

AI powered features within CareConnect are implemented using AWS Bedrock and are subject to the same security, auditing, and access control mechanisms as all other backend services. Prior to model invocation, inputs are minimized to reduce exposure to protected health information, and all communication with Bedrock occurs over encrypted channels. AI usage is logged for

traceability, and generated content is never automatically committed to persistent records without explicit user action.

9.1.8 Emergency Response Features

Emergency communication features prioritize availability and reliability. Redundant communication paths and fallback behaviors ensure that critical alerts can be delivered under degraded network conditions.

9.1.9 Subscription Billing Security

Billing-related data is isolated from clinical data. Payment processing uses secure third-party services and does not store raw payment credentials within the CareConnect system.

9.1.10 Security Testing & Monitoring

Security testing includes vulnerability scanning and penetration-style assessments. Continuous monitoring and log analysis detect anomalous behavior and potential breaches.

9.1.11 EVV Data Security Measures

- TLS 1.3 encryption for all data in transit
- AES-256 encryption for data at rest
- OAuth 2.0 authentication with Netsmart API
- Flutter Secure Storage for credential management on client devices
- Role-Based Access Control (RBAC) for application users
- Multi-factor authentication (MFA) for administrative access

9.1.12 EVV 21st Century Cures Act Compliance

- Capture of 6 federal EVV data elements as mandated
- Support for multiple verification methods (GPS, Phone, Video, Facility, Employer)
- Interoperability with state aggregator (Netsmart Tellus)
- Compliance with Virginia DMAS EVV requirements.

10. Error Handling

CareConnect includes robust error handling to prevent data corruption and ensure safe recovery. User-facing errors provide clear guidance without exposing internal details. System-level errors are logged with sufficient context to support troubleshooting and root cause analysis.

11. Performance & Scalability

11.1. Objectives

The CareConnect system is designed to meet the following performance targets for a healthcare communication application:

Metric	Target	Measurement Method
Chat message delivery latency	< 100ms (P95)	Time from CHAT_SEND to CHAT_DELIVER receipt
Call setup time	< 3 seconds	Time from CALL_ACCEPT to media flowing
REST API response time	< 500ms (P95)	Server processing time excluding obackwork
WebSocket connection establishment	< 1 second	Time to complete handshake and authentication
System uptime	99.5%	Excluding scheduled maintenance windows
Concurrent WebSocket connections	~500 per task (depends on CPU/memory + tuning)	With ECS Fargate (CPU/memory sizing), autoscaling, and proper tuning
Concurrent video calls	Unlimited	Managed by Twilio infrastructure (scales automatically)

11.2 Scalability Strategy

11.2.1 Horizontal Scaling Architecture

The backend is designed for horizontal scaling using container orchestration:

- Stateless Services: Spring Boot services maintain no local state; all session data in Redis or database
- Container Deployment: Docker containers enable rapid scaling and consistent deployment
- Load Balancing: AWS ALB distributes traffic across multiple backend instances
- Sticky Sessions: ALB configured with sticky sessions for WebSocket connection affinity

11.2.2 WebSocket Scaling Considerations

WebSocket connections require special handling for horizontal scaling:

Challenge	Solution
Connection Affinity	Application Load Balancer (ALB) sticky sessions ensure the same client remains connected to the same ECS Fargate task for the lifetime of the WebSocket connection.
Cross-Task Messaging	Redis (ElastiCache) pub/sub enables message routing when sender and recipient are connected to different ECS Fargate tasks
Connection Limits	Each ECS Fargate task can handle ~500 concurrent WebSocket connections depending on CPU/memory allocation and tuning; scale tasks horizontally to support additional users.
ALB Idle Timeout	Configure ALB idle timeout to 3600 seconds (1 hour) to prevent premature WebSocket disconnections.
Graceful Shutdown	Enable ECS task draining and handle SIGTERM to close WebSocket connections gracefully; clients automatically reconnect to healthy task

11.2.3 Database Scaling

- Read Replicas: PostgreSQL read replicas for message history queries (future)
- Connection Pooling: HikariCP connection pool sized appropriately for concurrent load
- Indexing Strategy: Indexes on conversation_id, sent_at for efficient message retrieval
- Archival Policy: Old messages archived to cold storage after retention period

11.3 Performance Monitoring

11.3.1 Key Metrics to Monitor

Metric	Alert Threshold	Response Action
EC2 CPU Utilization	> 80% for 5 minutes	Scale out (add instance) or investigate spike
Memory Utilization	> 85%	Check for memory leaks; consider larger instance
WebSocket Connection Count	> 400 per instance	Prepare to scale out before hitting limit
API Response Time (P95)	> 1 second	Investigate slow queries; check database load
Database Connections	> 80% of pool	Increase pool size or add read replica
Error Rate (5xx)	> 1%	Immediate investigation; potential incident

Note: Video call scaling is handled entirely by Twilio's managed infrastructure.

CareConnect only needs to scale the signaling capacity (WebSocket connections), not media processing capacity.

11.3.2 Monitoring Tools

- Amazon CloudWatch: Infrastructure metrics, logs, and alarms
- Application Logs: Structured JSON logging with correlation IDs
- Spring Boot Actuator: /actuator/health, /actuator/metrics endpoints
- CloudWatch Dashboard: Custom dashboard for real-time visibility

11.4 Bottleneck Mitigation

Potential Bottleneck	Mitigation Strategy	Implementation
Database Queries	Proper indexing and query optimization	Composite index on (conversation_id, sent_at DESC)
WebSocket Broadcasts	Redis pub/sub for cross-instance messaging	Publish to channel; subscribers push to local connections
Attachment Uploads	Direct S3 upload with pre-signed URLs	Client uploads directly to S3; bypasses backend
JWT Validation	In-memory token validation	Validate signature locally; no database lookup
Message History Queries	Cursor-based pagination	Use before/after timestamps instead of offset

12. Testing Strategy

12.1 Testing Overview

CareConnect employs a comprehensive testing strategy covering unit, integration, system, and security testing. The communication features (chat and video calling) require special attention due to their real-time nature and permission complexity.

12.2 Testing Levels

12.2.1 Unit Testing

Layer	Tools	Focus Areas
Backend Services	JUnit 5, Mockito	Business logic, permission validation, JWT generation, message routing logic
WebSocket Handlers	JUnit 5, Spring Test	Message parsing, authorization checks, error handling

Flutter Widgets	Flutter Test	UI component rendering, user interaction, state management
Flutter Services	Flutter Test, Mockito	WebSocket client, API client, local storage

12.2.2 Integration Testing

- WebSocket Integration: End-to-end message flow from client through server to recipient
- Database Integration: Message persistence, retrieval, and status updates
- S3 Integration: Attachment upload/download with pre-signed URLs
- Twilio Integration: Video Room creation, backend-issued Twilio access token generation, and client SDK connection
- Permission Integration: Full permission enforcement across chat and call flows

12.2.3 System Testing

- End-to-End Chat: Complete conversation flow including offline delivery
- End-to-End Video Call: Call initiation through media connection to call end
- Permission Scenarios: All blocked/allowed combinations verified
- Reconnection Scenarios: WebSocket disconnect and automatic recovery
- Load Testing: Multiple concurrent users, messages, and calls

12.2.4 Security Testing

Test Category	Specific Tests
Authentication Bypass	Invalid JWT, expired JWT, missing token, malformed token
Authorization Bypass	Access other users' conversations, attempt to join unauthorized Twilio Video Rooms, send messages or signaling events to non-participants
Permission Enforcement	Patient initiation restrictions, relationship validation, role-based access
Data Exposure	Error messages don't leak sensitive data, attachment URLs are time-limited
Encryption Verification	WSS traffic is encrypted (TLS); WebRTC media streams are encrypted using SRTP via Twilio Video

12.3 Communication-Specific Test Cases

12.3.1 Chat Test Coverage

- CHAT-001: Caregiver initiates conversation with assigned patient → SUCCESS
- CHAT-002: Patient initiates conversation (no prior contact) → BLOCKED
- CHAT-015: Patient replies after caregiver initiates → SUCCESS
- CHAT-016: Patient messages unassigned caregiver → BLOCKED
- CHAT-017: Patient messages another patient → BLOCKED
- CHAT-003: Caregiver messages another caregiver → SUCCESS
- WS-001 to WS-007: WebSocket connection lifecycle, heartbeat, reconnection tests

12.3.2 Call Test Coverage

- CALL-001: Caregiver calls assigned patient → SUCCESS
- CALL-018: Patient calls assigned caregiver → SUCCESS
- CALL-016: Patient calls unassigned caregiver → BLOCKED
- CALL-017: Patient calls another patient → BLOCKED
- CALL-019: Caregiver calls another caregiver → SUCCESS
- TWILIO-001 to TWILIO-009: Twilio Video creation, JWT auth, media encryption tests

12.4 Test Automation

- CI Pipeline: All unit and integration tests run on every commit
- Coverage Target: 80% code coverage for critical business logic
- Automated Regression: Full test suite runs nightly
- Manual Testing: UI/UX validation, accessibility checks, exploratory testing

13. Deployment Strategy

13.1 Deployment Environments

Environment	Purpose	Configuration

Development	Local developer testing	Docker Compose with local PostgreSQL, mock S3 (LocalStack)
Testing/QA	Integration and system testing	AWS ECS Fargate (non-production), test database, Twilio test credentials
Staging	Pre-production validation	Production-like config with synthetic data
Production	Live user traffic	Full HA configuration, monitoring, alerting

13.2 AWS Infrastructure

13.2.1 Production Architecture

Component	AWS Service	Est. Cost/Mo	Configuration
Spring Boot Backend	ECS Fargate	\$20-30	1 vCPU, 2GB RAM, Docker
Twilio Video	Usage-based	\$3-27	Go Rooms at \$0.001/participant-min; estimate based on 5-30 calls/day
Load Balancer	ALB	\$22	HTTPS/WSS termination, sticky sessions
Database	RDS PostgreSQL	\$15-20	20GB storage, automated backups
File Storage	S3 Standard	\$5	Attachments, lifecycle policies
DNS	Route 53	\$1	Domain routing
SSL Certificates	ACM	Free	Auto-renewed certificates
TOTAL		~65-80	<i>(Twilio costs vary by usage)</i>

13.2.2 Security Groups Configuration

Security Group	Port	Source	Purpose
ALB-SG	443/TCP	0.0.0.0/0	HTTPS from internet

ALB-SG	80/TCP	0.0.0.0/0	HTTP redirect to HTTPS
Backend-SG	8080/TCP	ALB-SG only	Spring Boot from ALB only
RDS-SG	5432/TCP	Backend-SG only	PostgreSQL from backend only

13.2.3 ALB Configuration for WebSocket

- Idle Timeout: 3600 seconds (critical - default 60s kills WebSocket connections)
- Sticky Sessions: Enabled with application cookie for connection affinity
- Health Check: GET /actuator/health every 30 seconds
- SSL Policy: ELBSecurityPolicy-TLS-1-2-2017-01 or newer
- Target Group: Instance-based, HTTP protocol, port 8080

13.3 CI/CD Pipeline

13.3.1 Backend Pipeline

1. Code Push: Developer pushes to feature branch
2. Build: Maven/Gradle builds Java application
3. Unit Tests: JUnit tests execute; fail build if tests fail
4. Static Analysis: SonarQube or similar for code quality
5. Docker Build: Create container image with build number tag
6. Push Image: Push to Amazon ECR
7. Deploy to Test: Automatic deployment to test environment
8. Integration Tests: Automated integration test suite
9. Manual Approval: Required for production deployment
10. Deploy to Production: Blue/green or rolling deployment

13.3.2 Frontend Pipeline

1. Code Push: Developer pushes Flutter code
2. Flutter Test: Widget and unit tests
3. Build APK/Web: flutter build apk, flutter build web
4. Test Distribution: Upload to Firebase App Distribution (Android)
5. Web Deployment: Deploy to S3 + CloudFront (web version)

13.4 Rollback Procedure

- Keep previous 3 Docker image versions in ECR
- Database migrations must be backward compatible
- Rollback command: Update ECS service to previous task definition
- Estimated rollback time: < 5 minutes

14. Known Issues & Limitations

14.1 Technical Limitations

Limitation	Impact	Mitigation / Future Work
Network Dependency	Real-time features require active internet connection; no true offline messaging	Messages queued locally when offline; synced on reconnect
Twilio Service Dependency	Audio/video calls depend on Twilio Video availability	Audio/video calls depend on Twilio Video availability
No Group Calls (MVP)	Only 1:1 audio/video calls supported in the initial release	Only 1:1 audio/video calls supported in the initial release
No Call Recording	Calls cannot be recorded for review or compliance	Twilio Video supports optional recording; future enablement requires explicit consent handling and storage/retention planning.
WebSocket Single-Region	Users far from AWS region may experience higher latency	Future: Multi-region deployment
No Message Search	Users cannot search through message history	Future: Elasticsearch integration

14.2 Known Issues

Issue ID	Description	Workaround	Status
KNOWN-001	Large attachment uploads may timeout on slow connections	Client shows progress; retry available	To be improved
KNOWN-002	Message history pagination can be slow for very old conversations	Limit initial load to recent 50 messages	Optimization planned

14.3 Exclusions

The following features are explicitly out of scope for the current implementation:

- Production billing integrations (Stripe, payment processing)
- External government EVV submission systems
- Third-party identity providers (OAuth with Google, Apple, etc.)
- SMS/email notification fallbacks
- Message translation or language detection
- AI-powered message suggestions or auto-responses
- End-to-end encryption (E2EE) for chat and video - transport encryption only (Twilio uses DTLS/SRTP)
- Self-hosted video infrastructure (using Twilio managed service)

14.4 Future Enhancements

The following improvements are planned for future iterations:

- Group video calls for care team meetings
- Message search with full-text indexing
- Push notifications for iOS and Android
- Call recording with consent management
- Multi-region deployment for global availability
- Enhanced analytics and reporting dashboard
- Integration with electronic health record (EHR) systems
- As a future enhancement, CareConnect may integrate PSTN calling using Twilio Programmable Voice to support communication with external phone numbers or as a fallback when in-app connectivity is unavailable. This capability is intentionally deferred to prevent scope creep in the current release.

Technical Design Document

CareConnect Team B

Version 2.0

University of Maryland Global Campus

SWEN 670 - Software Engineering Capstone

Dr. Mir Assadullah

February 7th, 2026

Contributors: Eduardo Estrada, Gary Jurado, Yismaw Tilaye, Joseph Wojcik, Chastity Sapp

Table of Contents

1. Introduction	6
1.1 Purpose.....	6
1.2 Intended Audience	6
1.3 Project Documents	6
1.4 Acronyms, Definitions, and Abbreviations.....	7
1.5 References	9
2. System Overview	10
2.1 Summary.....	10
2.2 System Users and Stakeholders.....	11
2.3 System Features and Capabilities	11
2.4 System Architecture Overview	13
2.5 System Architecture Design Rationale	15
2.6 Scope	17
3. Architecture	19
3.1 Architectural Components.....	19
4. Technology Stack – Team B Additions	24
4.1 Mobile Application Platform	24
4.2 Local Persistence and Offline Support (BNS 5)	24
4.3 Connectivity Detection	24
4.4 Backend Services and Observability	25
4.5 Infrastructure and Deployment (BNS 6).....	25
4.6 Monitoring and Logging	26
4.7 Analytics and Feature Telemetry (BNS 7).....	26
4.8 Security and Access Control.....	26
5. Functional Specifications – Team B Scope.....	26
5.1 Offline Data Capture and Synchronization.....	27
5.2 Infrastructure Management via CloudFormation.....	27
5.3 Observability and Feature Analytics	28
6. API Design Specification for CareConnect – Team B Extensions.....	28
6.1 Offline Synchronization API Interaction Pattern	28
7. Data Models	29

CareConnect Technical Design Document

7.1 Scope	30
7.2 Conceptual Data Model.....	30
7.3 Logical Data Model.....	32
7.4 Data Dictionary	33
7.4 Physical Data Model.....	35
7.6 Physical Design Considerations	36
7.7 AWS PostgreSQL Deployment	37
7.8 Security & Compliance	38
7.9 Backup & Recovery	40
7.10 Entity-Relationship Diagram.....	41
8. UI/UX Design	41
8.1 Legacy Design System Alignment.....	42
8.2 Offline Mode UI Behaviors (BNS 5)	42
8.3 User-Controlled Persistence Toggle (BNS 5).....	43
8.4 Observability and Anonymous Feature Analytics (BNS 7)	44
9. Security Considerations.....	47
10. Error Handling.....	48
10.1 Error Classification	48
10.2 Offline and Synchronization Error Handling.....	48
10.3 Authentication and Authorization Errors	49
10.4 Infrastructure and Deployment Errors	49
10.5 Observability and Error Reporting	49
10.6 Recovery-Oriented System Design.....	49
11. Performance & Scalability.....	50
11.1 System Architecture Considerations	50
11.2 Backend Performance Strategy	50
11.3 Infrastructure Scalability	50
11.4 Offline Performance Characteristics	51
11.5 Observability and Performance Monitoring.....	51
11.6 Scalability Boundaries for This Milestone	51
12. Testing Strategy.....	51
12.1 Validation Categories	52
12.2 Functional Validation Scenarios.....	52

CareConnect Technical Design Document

12.3 Failure and Recovery Scenarios	52
12.4 Observability and Privacy Validation	53
12.5 Accessibility and UI Considerations (Out of Scope)	53
13. Deployment Strategy	53
13.1 Environment Strategy.....	53
13.2 Infrastructure Provisioning (BNS 6).....	53
13.3 CI/CD Pipeline	54
13.4 Offline Synchronization Deployment Considerations (BNS 5).....	54
13.5 Observability Deployment Requirements (BNS 7).....	54
13.6 Rollback Strategy	55
14. Known Issues & Limitations.....	55
14.1 Offline Feature Scope Limitation (BNS 5)	55
14.2 Conflict Resolution Simplicity	55
14.3 Observability Scope Limitation (BNS 7).....	55
14.4 AWS Dependency	56
14.5 Third-Party Integration Constraints	56
14.6 AI Feature Limitations	56
14.7 Performance Boundaries.....	57

CareConnect Technical Design Document

Revision History

Team Name	Date	Reason for Changes	Version
Team B	02/07/2026	Initial team B draft	1.0
	2/17/2026	Modified aws architecture	2.0

Sign-off Sheet

Role	Name	Signature	Date
Team Lead	Eduardo Estrada	Eduardo Estrada	02/07/2026
Team Member	Joseph Wojcik	Joseph Wojcik	02/07/2026
Team Member	Yismaw Tilaye	Yismaw Tilaye	02/07/2026
Team Member	Gary Jurado	Gary Jurado	02/07/2026
Team Member	Chastity Sapp	Chastity Sapp	02/07/2026

1. Introduction

1.1 Purpose

The purpose of this document is to detail the technical design of the CareConnect application. This document provides an explanation of the system architecture, components, data flows, and integration points. This will align development of the application with requirements, quality standards, and maintainability. This document serves as a reference for developers, testers, and stakeholders. This document provides a blueprint for the engineering of team B's three core requirements:

- Local-First Offline Persistence (BNS 5),
- AWS-Native Infrastructure-as-Code (BNS 6), and
- Privacy-Preserving Telemetry (BNS 7).

1.2 Intended Audience

This document is intended for the stakeholder who requires information for the CareConnect application's architecture, and design decisions. This includes but is not limited to

- **Project Leadership:** Including the Project Manager and Technical Architects, to ensure architectural consistency and adherence to the project's high-level roadmap.
- **Development Team:** Backend and frontend engineers, and cloud specialists, who will implement the local-first synchronization patterns (BNS 5) and AWS CloudFormation stacks (BNS 6).
- **Quality Assurance (QA) Team:** Test personnel who must understand system interfaces, state transitions during offline modes, and telemetry data flows (BNS 7) to perform accurate validation.
- **Business and Analysis Team:** Business and Product Owners to ensure that the design facilitates requirements and business goals.
- **Clients and Stakeholders:** This document allows insight into and about technical decisions that affect maintainability, scalability and the overall system quality.

This document will provide an understanding of the technical foundation of the CareConnect application through development, testing, deployment, and maintenance phases of the application.

1.3 Project Documents

The Technical Design Document is just one artifact in a suite of documents comprising the software development life cycle for CareConnect. These documents contain important information regarding the project planning, implementation, testing, and more about the CareConnect product. A list of documents, their current versions, and delivery dates is provided in Table 1.3.1 below.

Table 1.3.1*List of Project Documents for CareConnect*

Document	Version	Date
Project Plan	2.0	2/07/2026
Software Requirements Document	2.0	2/07/2026
Technical Design Document	1.0	2/7/2026
Software Test Plan	1.0	2/7/2026
Deployment and Operations Guide	1.0	3/10/2026
Programmer Guide	1.0	3/10/2026
User Guide	1.0	3/31/2026
Test Report	1.0	3/31/2026

1.4 Acronyms, Definitions, and Abbreviations

A comprehensive list of acronyms, definitions, and abbreviations used in this document is provided in Tables 1.4.1 and 1.4.2.

Table 1.4.1.*Glossary Table*

Term	Definition
AI (Artificial Intelligence)	The capability of a system to perform tasks that typically require human intelligence, such as recognizing speech, making decisions, or learning from data.
Android OS	An open-source mobile operating system developed by Google, used for running applications on smartphones, tablets, and other devices.
CareConnect	Application that will be created to help manage the care receiver healthcare needs
Care Giver	A person (e.g., family member, nurse, or aide) who provides assistance, monitoring, or support to a Care Receiver through the app.
Care Receiver	The individual who requires support, assistance, or monitoring, is the primary beneficiary of the app's features.
Dart Tool	The primary programming language and toolset used to develop Flutter applications.
Diarization	The process of partitioning an audio stream containing human speech into homogeneous segments according to the identity of each speaker.

CareConnect Technical Design Document

Flutter	An open-source UI software development kit (SDK) created by Google, used for building natively compiled applications for mobile (Android, iOS), web, and desktop from a single codebase.
HIPAA (Health Insurance Portability and Accountability Act)	U.S. legislation that provides data privacy and security provisions for safeguarding medical information.
iOS	A mobile operating system created by Apple Inc. for iPhones and iPads, used to run applications developed specifically for Apple devices.
Machine Learning (ML)	A subset of AI that uses algorithms and statistical models to enable systems to improve performance on tasks through experience.
NLP (Natural Language Processing)	A branch of AI that enables computers to understand, interpret, and generate human language.
User Interface (UI)	The visual and interactive components of the application which allows users (care givers and receivers) to interact with the system.
User Experience (UX)	The overall experience and satisfaction a user has when interacting with the application, including ease of use, efficiency, and accessibility.

Table 1.4.2*Acronyms*

Acronyms	Definitions
AC-###	Acceptance criterion tested for a requirement/use Case
AI	Artificial Intelligence
API	Application Programming Interface
App	Application (mobile or web-based software)
BNS	Business Need Statement
BFF	Backend for Frontend
DB	Database
EHR	Electronic Health Record
GUI	Graphical User Interface
HIPAA/GDPR	U.S. healthcare privacy law / EU privacy regulation
IoT	Internet of Things
IaC	Infrastructure as code
ML	Machine Learning
NLP	Natural Language Processing

CareConnect Technical Design Document

PHI/PII	Protected Health Information
PII	Personally Identifiable Information
QA	Quality Assurance
RBAC	Role-Based Access Control
REQ-###	Requirement ID
REST	Representational State Transfer (API standard
RTO/RPO	Recovery Time/Point Objective
SRS	Software Requirements Specification
TC-###	Test case that verifies an AC
UC-###	Use case
UI	User Interface
UX	User Experience
SLA/SLO	Service Level Agreement/Objectives

1.5 References

1. Angeles, D., Bias, T., Gaucin, J., Chen, F., Curran, L., Grbreegziabhere, A., Harding, A. M., Malla-Paudel, A., Ramirez, M., Raphael, E., Truong, D., Vecchioni, A., & Yawn, C. (2025). Software requirements document: CareConnect. University of Maryland Global Campus. <https://umgc-cappms.azurewebsites.net/previousprojects>
2. Brown, M. (2020). SWEN 670 capstone project guide. University of Maryland Global Campus. <https://learn.umgc.edu/d21/le/content/1452226/viewContent/35360399/View>
3. CareConnect Developer Team. (2025). CareConnect project documentation and programmer guides. GitHub Repository. https://github.com/umgc/2025_fall/tree/developer/careconnect2025/docs/project-docs
4. Gemini 3 Flash. (2026). *CareConnect System Architecture Diagram* [AI-generated image]. Google AI. <https://gemini.google.com/>
5. Mir, A. (2025). CareConnect high-level requirements. University of Maryland Global Campus.
6. Mir, A. (2026). SWEN 670 fall Team B BNS distribution. University of Maryland Global Campus.
7. OpenAI. (2025). ChatGPT (June 11 version) [Large language model]. <https://chat.openai.com/>
8. Software Requirements Specification (SRS). (2025, July). CaPPMS. <https://umgc-cappms.azurewebsites.net/previousprojects>
9. van der Meij, J., & van der Meij, H. (2014). A comparison of paper-based and video tutorials for software learning. *Computers & Education*, 78, 150–159. <https://doi.org/10.1016/j.compedu.2014.06.003>
10. World Wide Web Consortium. (2025). Web content accessibility guidelines (WCAG) 3.0. <https://www.w3.org/TR/wcag-3.0/>

2. System Overview

2.1 Summary

The CareConnect system is an integrated health management platform designed to bridge the gap between home-based care and remote monitoring. While the original system architecture (CareConnect TDD v.4.1) [3, 8] utilized containerized Spring Boot microservices and third-party integrations like Stripe and USPS Digest, this current iteration evolves that foundation into a local-first, container-based client-server architecture that prioritizes system availability and data integrity in environments with intermittent or unavailable network connectivity. These third-party integrations are referenced solely as part of the legacy CareConnect architecture and are not included in the current MVP scope or Team B deliverables.

Client-Side Evolution

The mobile client, built on Flutter, utilizes an abstraction layer to decouple feature logic from the data persistence layer. Moving beyond the connectivity-dependent model of the previous summary, high-value modules—specifically Mood, Symptom, and Medication Tracking, EVV, and the Shared Calendar—now utilize a local Drift/SQLite database to persist data during offline states. A specialized Synchronization Engine (Team B) manages the queued transition of data to the cloud, employing a "Last-Write-Wins" conflict resolution strategy to maintain record consistency across the distributed environment.

Infrastructure and Backend

The backend architecture remains a containerized Spring Boot-based service model, consistent with the original CareConnect foundation. Backend services are deployed using managed container platforms with AWS ECS Fargate and are provisioned using AWS CloudFormation (BNS 6). This approach enables repeatable, environment-consistent deployments while avoiding manual configuration drift.

CloudFormation is used to define infrastructure resources declaratively, including compute services, networking components, storage, and observability integrations. Standardized health-related data is persisted in Amazon RDS (PostgreSQL), while semi-structured data and operational artifacts are stored in Amazon S3.

Observability and Compliance (BNS 7)

Team B is responsible for the telemetry capture layer. This ensures that all system activity and feature interactions generate an automated, traceable data record without manual intervention. The scope of this work focuses on defining standardized mechanisms for data emission, collection, and inspection rather than implementing full analytics pipelines.

2.2 System Users and Stakeholders

The CareConnect architecture is designed to support three primary user actors and several technical stakeholders involved in the system's deployment and maintenance.

2.2.1 Primary User Actors

These actors interact directly with the mobile and web interfaces and are the primary sources of data for the synchronization engine.

- **Caregivers:** Includes both professional and family caregivers. They are the primary administrative actors who manage schedules, review health telemetry, and receive system alerts. In the TDD context, they are the primary users of the web dashboard and high-privilege mobile functions.
- **Patients:** The central subjects of care. Their interactions are characterized by high-frequency, low-latency data entry (mood, symptoms, task completion). The system design prioritizes their user experience during offline states.
- **Family Members:** Secondary actors with restricted, read-only access. Their technical interaction is limited to viewing filtered health summaries and motivational milestones, requiring strict data-masking and permission controls.

2.2.2 Technical and Project Stakeholders

These stakeholders are responsible for the infrastructure, requirements validation, and long-term viability of the CareConnect ecosystem.

- **Product Owners and Sponsors:** Provide the high-level vision and business requirements. They validate that the technical implementation meets the project's strategic goals and budgetary constraints.
- **System Architects and Team Leads:** Responsible for the technical integrity of the Team B scope. They oversee the transition to CloudFormation (BNS 6) and the implementation of the sync framework (BNS 5).
- **Development and QA Teams:** Engineers and analysts responsible for implementing the Dart/Flutter logic, provisioning AWS resources, and performing end-to-end validation of the synchronization and observability pipelines.
- **Regulatory Compliance Entities:** External or internal parties responsible for auditing the application's adherence to HIPAA and GDPR standards, specifically regarding the scrubbing of PII in telemetry (BNS 7).
- **Infrastructure and API Providers:** Third-party entities providing the AWS backbone, mapping APIs, and database services essential for the system's operational uptime.

2.3 System Features and Capabilities

The CareConnect mobile and web ecosystem provides a resilient suite of caregiving tools. Features are designed to support high availability, even in offline states, through the implementation of local-first data patterns.

2.3.1 Core Care Management

This section defines the core MVP functional modules originally established in the CareConnect TDD v.4.1 [3, 8].

- **Onboarding & Identity:** Secure authentication via AWS Cognito (Email, SSO) with role-based routing to Patient or Caregiver dashboards.
- **Task & Schedule Management:** Caregivers can schedule clinical and wellness tasks (medication, meals, exercise). Data is synchronized via FCM for push notifications and stored in Amazon RDS.
- **Health Data Tracking:** Manual entry of mood, symptoms, and vitals. Data is visualized through time-series charts (PostgreSQL) to monitor patient wellness trends.
- **Messaging & Emergency SOS:** Secure text communication between linked users and a high-priority Emergency SOS system with location broadcasting. These capabilities are inherited from the legacy CareConnect platform and are not modified, extended, or made offline-capable as part of the current MVP. Team B involvement is limited to backend observability instrumentation where applicable.

2.3.2 Data Storage, Offline Persistence, and Synchronization (BNS 5)

- **Local Persistence:** Critical modules—including Mood Tracking, Symptom Tracking, Medication Tracking, EVV, and Calendar—persist data locally using a Flutter-compatible engine (Drift/SQLite).
- **Sync Logic:** An abstraction layer decouples UI features from storage logic, enabling deferred synchronization with a "Last-Write-Wins" conflict resolution strategy.
- **Data Security:** All patient health data (PHI) is encrypted at rest using AWS KMS and in transit via TLS 1.3, maintaining HIPAA compliance.

2.3.3 Electronic Visit Verification (EVV)

As established in the legacy CareConnect TDD [3, 8], the EVV module remains a critical regulatory requirement for home-based care. Team B's implementation focuses on the instrumentation of the capture mechanics for these events:

- **Compliance Logging:** Capture of caregiver clock-in/out events. Following the original specification, this includes the instrumentation to collect the required location telemetry (GPS coordinates) and associate them with the specific visit_id in the public.ev_v_record table.
- **Offline Resilience:** Per the legacy requirement for reliability in varied environments, this module supports the offline capture of EVV timestamps. These events are stored in the local Drift/SQLite database and queued for submission, ensuring compliance is maintained even in low-signal or no-signal home environments.

CareConnect Technical Design Document

2.3.4 Infrastructure Provisioning and Deployment (BNS 6)

- **AWS-Native Standardization:** Infrastructure is defined as IaC using **AWS CloudFormation** templates to ensure reproducible, modular deployments.
- **Scalable Back-end:** Backend services are deployed using managed container-based execution environments in AWS ECS Fargate.

2.3.5 Observability and Anonymous Feature Analytics (BNS 7)

- **Operational Monitoring:** Centralized logging (CloudWatch), metrics collection, and distributed tracing (X-Ray) for real-time system health diagnostics.
- **Privacy-Preserving Analytics:** Implementation of a PII-scrubbing middleware to record anonymous feature usage trends without capturing identifiable patient data.

2.4 System Architecture Overview

CareConnect utilizes a client-server architecture hosted on Amazon Web Services (AWS). Following the requirements established in the SRS and the evolution from the TDD v4.1 [3, 8], the system separates the frontend, backend compute, and data layers to ensure scalability and maintainability.

2.4.1 Frontend (BNS 5 Integration)

- **Flutter Client:** A unified codebase for mobile (iOS/Android) and web interfaces, emphasizing reactive UI state management.
- **Local Persistence Layer (Drift/SQLite):** A secure, encrypted local database engine used to store **Mood, Symptom, Medication, EVV, and Calendar** data while the device is offline.
- **Sync & Abstraction Layer:** A service-based architecture that intercepts API calls. When offline, it queues operations; when online, it executes a background synchronization protocol with the cloud.

2.4.2 API Gateway and Routing

As defined in the CareConnect baseline architecture, client requests are routed through the existing API access layer to backend application services hosted in managed container environments. This API access layer is inherited from the baseline CareConnect design and remains unchanged within Team B's scope. Team B responsibilities are explicitly limited to offline synchronization patterns (BNS 5), infrastructure standardization for Team B-owned resources using AWS CloudFormation (BNS 6), and observability and feature analytics (BNS 7).

- **Request Management:** RESTful API endpoints are consumed by the Flutter client and implemented by backend services deployed on Containerized Services (AWS ECS Fargate as the primary deployment target). The inherited API access layer forwards requests to these services without requiring Team B to modify routing behavior or endpoint exposure.

- **Security & Identity:** JWT-based authentication is enforced using existing CareConnect identity controls and validated by backend services via Spring Security. Tokens issued by the authentication service are validated on each request to ensure role-aware access control prior to business logic execution.
- **Routing & Protection:** Request distribution and ingress protection are handled by the existing platform routing and ingress configuration (e.g., Application Load Balancer or platform-managed ingress). These mechanisms are assumed to remain consistent with the legacy CareConnect deployment model and are not modified by Team B.
- **Instrumentation (Team B Extension):** In accordance with BNS 7, backend services emit structured access logs, error metrics, and request latency data to Amazon CloudWatch. Distributed tracing is implemented using OpenTelemetry instrumentation at the application service level, ensuring consistent observability independent of the inherited API access layer implementation.

Team B's offline synchronization logic reuses existing backend endpoints and does not introduce new domain APIs, gateway-level behaviors, or access-layer modifications. Minor refactoring of existing endpoints or request/response models may occur to support correctness, consistency, and offline synchronization reliability, without introducing new domain functionality.

2.4.3 Core Backend Services

Following the functional requirements established in the SRS and the legacy CareConnect microservices architecture [3, 8], the backend is implemented as a set of containerized application services. These services are deployed using managed container platforms and provisioned via infrastructure-as-code to ensure consistency, scalability, and maintainability. The backend services expose RESTful APIs consumed by the Flutter client and are responsible for enforcing business logic, security constraints, and data persistence. Team B's involvement focuses on enabling compatibility with offline synchronization and observability requirements, not re-architecting core business services.

- **User & Role Service:** Derived from the original User Service [3, 8], this service manages identity, authentication, and role-based access control (RBAC) for Patients, Caregivers, and Family members. Authentication tokens are validated at the service layer using Spring Security, and role enforcement is applied through application-level authorization rules.
- **Care Coordination Service:** This service evolves the legacy Scheduling Service [3, 8] and manages scheduling, task assignments, EVV workflows, and calendar event processing. It serves as the primary backend integration point for Team B's Offline Synchronization Engine by reconciling batched client-side updates originating from the local Drift/SQLite database once connectivity is restored.
- **Emergency & Alert Service:** Derived from the legacy Communication Service [3, 8], this module manages emergency signaling and notification dispatch mechanisms. Team B's responsibility is limited to telemetry instrumentation and logging support; alert delivery

CareConnect Technical Design Document

logic, user interaction flows, and availability guarantees remain unchanged from the legacy implementation.

2.4.4 Analytics and Telemetry Services (BNS 7)

- **Privacy-Preserving Analytics:** A dedicated pipeline that scrubs PII/PHI before recording feature usage events (e.g., "Medication_Check_Complete") for product health insights.
- **Observability Telemetry:** Integration with **AWS CloudWatch** and **AWS X-Ray** to capture logs, metrics, and distributed traces, enabling rapid troubleshooting of the synchronization lifecycle.

2.4.5 Infrastructure Provisioning and Deployment (BNS 6)

- **AWS CloudFormation:** The CareConnect infrastructure is defined using IaC through AWS CloudFormation templates. These templates provision and configure core infrastructure components, including networking resources (VPC, subnets, security groups), container execution environments, and data storage services (Amazon RDS and Amazon S3). This approach ensures infrastructure consistency and repeatability.
- **Repeatable Environments:** CloudFormation ensures that the environment can be reproduced identically for development, testing, and production phases, ensuring long-term system maintainability.

2.5 System Architecture Design Rationale

The architecture of CareConnect is engineered for high availability, cost-efficiency, and resilience. The design prioritizes Local-First reliability for clinical workflows, standardized infrastructure management via IaC, and privacy-centric observability. The architecture emphasizes modularity and separation of concerns between client, backend services, and infrastructure layers, allowing teams to evolve implementation strategies independently while preserving consistent data flows, security enforcement, and operational visibility.

2.5.1 Client–Server Model

CareConnect utilizes a decoupled client–server model where Flutter-based mobile and web clients interact with backend services through RESTful HTTP APIs exposed by the backend service layer.

To mitigate the risk of intermittent network connectivity in home-care settings, the architecture incorporates a local data persistence layer. This allows users to perform critical actions while offline, with synchronization handled automatically once connectivity is restored.

2.5.2 Rationale for Key Technology Choices

- **Flutter (Frontend Framework):** Selected to maintain a unified codebase across iOS, Android, and Web. This ensures that the complex **BNS 5** synchronization logic and abstraction layers are implemented once and behave consistently across all user devices.
- **Drift & SQLite (Local Persistence):** Drift was chosen for its reactive persistence capabilities and seamless integration with Flutter. It provides the structured, encrypted local storage required for offline-first behavior in the **Mood, Symptom, and Medication** modules.
- **Containerized Backend Services (Compute):** Containerized backend services were selected to provide a balance between scalability, operational control, and architectural clarity. Running Spring Boot services in managed container environments AWS ECS Fargate (primary deployment target) allows the system to scale based on demand while maintaining a clear execution model.
- **Backend Service Routing & Security:** Request routing and security enforcement are handled directly by the backend service layer. Authentication and authorization are implemented using JWT-based security controls, enabling consistent identity validation, role enforcement, and request handling without introducing infrastructure-level coupling.
- **AWS CloudFormation (BNS 6):** Selected as the primary IaC tool to ensure AWS-native standardization. Unlike third-party tools, CloudFormation provides deep integration with AWS resource states, ensuring that Team B's infrastructure is reproducible, modular, and easily maintained.
- **PostgreSQL via Amazon RDS:** Used as the primary relational datastore. Its strong **ACID compliance** is critical for maintaining the integrity of health logs, medication schedules, and audit trails during the synchronization process.
- **Amazon S3:** Selected for durable, low-cost storage of unstructured data, including telemetry logs and media assets, providing high durability and HIPAA-compliant encryption options.

2.5.3 Security and Compliance Considerations

The architecture enforces **HIPAA-aligned** data handling through:

- **End-to-End Encryption:** Enforcing TLS 1.3 for data in transit and AES-256 (via AWS KMS) for data at rest in RDS and S3.
- **Identity & Access Management (IAM):** Utilizing Cognito for JWT-based authentication and fine-grained RBAC (Role-Based Access Control) to ensure users only access data relevant to their specific role (Patient vs. Caregiver).

2.5.4 Observability and Analytics Rationale (BNS 7)

Operational observability and anonymous analytics are architecturally separated to protect Patient Privacy (PHI/PII):

- **CloudWatch & X-Ray:** Provide real-time health monitoring and distributed tracing to troubleshoot the synchronization lifecycle without exposing sensitive data.

- **Privacy-Middleware:** A custom Lambda-based filter that scrubs PII/PHI from events before they are recorded in the feature usage analytics dashboard, ensuring compliance with healthcare privacy constraints.

Note: System health monitoring (error logs and performance metrics) is active by default to ensure backend stability. However, tracking specific user activities and feature usage is strictly opt-in to protect user privacy.

2.6 Scope

This Technical Design Document defines the architectural boundaries for the CareConnect MVP. As established in the legacy TDD v4.1 [3, 8], the platform is designed for future extensibility; however, the current design responsibility is strictly limited to the core features required for a functional health-coordination MVP and the technical requirements assigned to Team B.

2.6.1 In-Scope Features (MVP Delivery)

The following features are included in the current architectural design and implementation plan:

Core Application Functionality [3, 8]:

- **Authentication & RBAC:** Secure login/registration via AWS Cognito with role-based dashboards for Patients and Caregivers.
- **Care Coordination:** Task creation, medication scheduling, and push notifications for reminders.
- **Health Tracking:** Manual entry of mood, symptoms, and vitals with time-series data visualization.
- **Communication:** Secure private messaging and a one-touch Emergency SOS system with GPS broadcasting.
- **Electronic Visit Verification (EVV):** Clock-in/out functionality with location capture for compliance.

Team B New Technical Requirements (BNS 5, 6, & 7):

- **Offline-First Persistence (BNS 5):** Local data storage for Mood, Symptoms, Medications, EVV, and Calendar using Drift/SQLite.
- **Synchronization Mechanism (BNS 5):** Deferred data synchronization logic that reconciles locally queued records with backend services upon connectivity restoration, using deterministic conflict resolution (Last-Write-Wins) and retry handling.
- **Infrastructure-as-Code (IaC) (BNS 6):** Standardized provisioning of AWS infrastructure resources, including networking, backend compute environments, storage, and observability components, using AWS CloudFormation to ensure reproducibility and environment parity.
- **Unified Observability (BNS 7):** Implementation of centralized logging, metrics, and distributed tracing using OpenTelemetry for instrumentation, with telemetry exported to AWS CloudWatch for system monitoring, diagnostics, and validation.

CareConnect Technical Design Document

- **Privacy-Preserving Analytics (BNS 7):** Application-level telemetry capture that records anonymized feature usage events, with PII/PHI scrubbing enforced prior to persistence, supporting system validation without exposing sensitive user data.

2.6.2 Out-of-Scope Features (Excluded from MVP)

To ensure delivery within the 12-week schedule and maintain focus on the Team B Technical requirements (BNS 5, 6, & 7), the following features are explicitly excluded from the current design scope. Many of these represent secondary features identified in the legacy SRS or TDD v4.1 [3, 8] that are not required for the immediate functional health-coordination MVP.

Third-Party & Hardware Integrations:

- **Billing & Payments:** Stripe integration, subscription plan management, and invoice generation [3, 8].
- **Wearables & IoT:** Fitbit, Apple Health, and advanced home automation (Nest/Ring/Smart Home).
- **USPS Integration:** USPS Digest API and mail-tracking functionality [3, 8].
- **Hardware & Medical Devices:** Custom medical device development, biometric scanners, and hardware-level integrations.
- **External Systems:** Insurance adjudication systems and public social media integration.

Advanced AI & Social Features:

- **ASL & Speech Translation:** AI-driven ASL interpretation and real-time speech-to-text telemedicine assistants.
- **Clinical & Diagnostic Logic:** Automated clinical diagnosis, medical advice, or ML-driven diagnostic recommendations.
- **Social & Community:** Peer-to-peer caregiver support communities, community forums, and group messaging/shared milestone posting.
- **Gamification:** Dynamic gamification features including levels, badges, and ranks.
- **Multilingual Support:** Localization for languages other than English (e.g., Arabic, Farsi, Spanish).

Administrative & Technical Exclusions:

- **Advanced Analytics:** Advanced executive analytics dashboards or predictive ML recommendations.
- **Management Overlays:** Role-based admin management or internal audit log visualization (outside of raw telemetry).
- **Accessibility:** Complex accessibility enhancements beyond the standard Flutter framework capabilities.
- **Legacy Tooling:** Infrastructure management using Terraform or non-AWS-native IaC tools.
- **Full Data Mirroring:** Local caching of the entire global PostgreSQL dataset on mobile devices (only "Subset" data is cached per BNS 5).

- **Custom Observability:** Development of proprietary monitoring tools outside the AWS native ecosystem (CloudWatch/X-Ray).

Note: Legacy integrations such as Stripe billing and USPS mail services are referenced solely for historical and architectural context and are explicitly excluded from the Team B MVP scope for Milestone 2.”

3. Architecture

This section describes the high-level architectural structure of the CareConnect system, focusing on the structural organization, interaction boundaries, and operational responsibilities of frontend clients, backend services, data stores, and supporting infrastructure. The architecture is derived from the legacy CareConnect design baseline and refined to support Team B’s assigned responsibilities, including offline-first data persistence (BNS 5), infrastructure standardization via AWS CloudFormation (BNS 6), and privacy-preserving observability (BNS 7).

3.1 Architectural Components

CareConnect is architected to handle sensitive PHI while operating within the practical constraints of home-care environments, including intermittent connectivity and varied device capabilities. The system design adheres to four primary architectural pillars: Security-by-Design, Offline-First Resilience, Operational Observability, and Controlled Scalability.

To support these objectives, the system employs a container-based client–server architecture hosted on Amazon Web Services (AWS). Backend services are implemented as containerized Spring Boot applications deployed on Containerized Services (AWS ECS Fargate as the primary deployment target). This approach provides predictable execution behavior and clearly defined service boundaries suitable for authorization, synchronization, and auditing workflows.

Team B’s scope focuses on standardizing infrastructure provisioning using AWS CloudFormation (IaC), enabling repeatable, secure, and environment-consistent deployments. In addition, Team B introduces offline-first data handling mechanisms and observability instrumentation while preserving the core business logic and service responsibilities defined in the legacy CareConnect architecture [3, 8].

3.1.1 Elasticity and Scale-to-Zero

Backend scalability in CareConnect is achieved through controlled horizontal scaling of container-based services rather than serverless execution. Application services are deployed as stateless Spring Boot containers on a managed container platform, AWS ECS Fargate (primary deployment target), enabling capacity to scale predictably in response to demand while maintaining consistent runtime behavior.

During peak operational periods, such as morning medication logging or Electronic Visit Verification (EVV) check-in windows, additional container instances may be provisioned automatically through platform-managed scaling policies. During periods of reduced activity,

CareConnect Technical Design Document

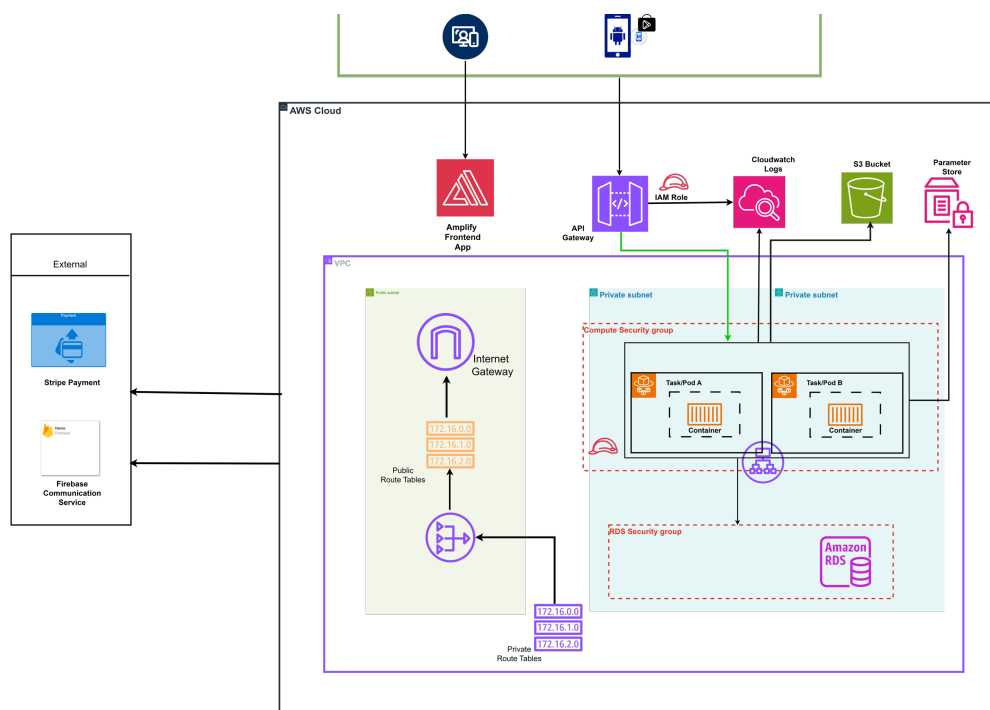
container counts may be scaled down to a defined minimum baseline capacity, balancing cost efficiency with continuous service availability.

This approach provides elasticity while maintaining a defined minimum service footprint. Unlike serverless execution models, container-based scaling preserves stable authorization flows, predictable database connection management, and well-defined observability boundaries. These characteristics are essential for healthcare workflows that involve offline synchronization, auditability, and regulatory compliance.

All scalability behavior is governed by infrastructure-as-code definitions (BNS 6), ensuring consistent configuration across environments and preventing ad-hoc scaling adjustments that could introduce instability or compliance risk.

Figure 3.1.1.1

CareConnect System Architecture.



3.1.2 Standardized Provisioning (BNS 6)

Infrastructure resources—including networking, container-based compute services, relational storage, and observability components—are defined and managed using AWS CloudFormation. This infrastructure-as-code (IaC) approach establishes a single authoritative source of truth for the system state, eliminating manual configuration drift and ensuring consistent deployments across development, testing, and production environments.

CloudFormation templates version infrastructure definitions alongside application code, enabling controlled updates, reproducible environments, and predictable rollback behavior. This

CareConnect Technical Design Document

standardization is essential for long-term system maintainability, collaborative development across teams, and rapid recovery from deployment or infrastructure failure

3.1.3 Resilient Observability (BNS 7)

To support future auditing and behavioral review, the system is instrumented to capture high-fidelity feature usage data. Team B is responsible for the mechanics of this capture, ensuring that specific user actions—such as logging a symptom or updating a care plan—are recorded as they pass through the synchronization layer.

These events are routed to AWS CloudWatch Logs, which serves as the durable, centralized storage for all captured usage data. To ensure this capture is consistent across the environment, the following mechanisms are implemented:

- Backend services are configured to emit structured logs whenever a key feature is accessed.
- Each feature interaction is assigned a unique Trace ID. This allows the system to track the "path" a feature usage event takes from the API Gateway through to final database persistence.
- Logging channels and retention policies are defined within AWS CloudFormation (BNS 6) templates, ensuring that feature capture is "always-on" by default in every deployment.

To maintain HIPAA compliance, the design utilizes a tiered data capture approach:

- **Operational Telemetry Capture:** Full tracing and logging of system health and execution, with all Protected Health Information (PHI) scrubbed at the source.
- **Anonymous Usage Logging:** Implementation of a capture mechanism for event-tracking that records activity patterns without capturing or associating data with identifiable patient behavior.

3.1.4 Offline-First Robustness (BNS 5)

CareConnect is designed with a Local-First architecture to ensure that caregivers can document essential health interactions in environments without reliable internet access. The system maintains a subset of critical features in an offline state, using the cloud primarily for long-term persistence and cross-device coordination.

Scope of Offline-Enabled Features

To balance device performance with clinical utility, only the following data types persist locally:

- **Mood & Symptom Tracking:** Quantitative and qualitative patient wellness logs.
- **Medication Logging:** Confirmation of dosage administration and timing.
- **EVV (Electronic Visit Verification):** Capture of high-precision **timestamps** for visit start/end (Note: full verification occurs upon reconnection).

- **Calendar:** Creation and viewing manual schedule entries.

Implementation Mechanics

- **Encrypted Local Database:** Offline data is committed to an **encrypted SQLite database** (via the **Drift** framework). This ensures data remains protected on the device if it is lost or stolen.
- **State-Based Synchronization:** Every record is assigned a sync_status metadata field (e.g., PENDING, SYNCED, or CONFLICT).
- **Reconciliation Logic:** Upon detecting a network connection, the Sync Coordinator performs a batch synchronization. For Calendar and Health Logs, a "Last-Write-Wins" resolution is applied based on client-side timestamps to ensure the most recent entry is preserved in the Amazon RDS global state.

Local Persistence (SQLite via Drift)

To support offline functionality, every table in the local store includes synchronization metadata to manage the state between the device and AWS RDS. The local_id acts as the primary key on the device to prevent collisions before a server_id is assigned by the Amazon RDS instance.

Key Synchronization Columns:

- local_id: (Primary Key) UUID generated on the device.
- sync_status: Enum (PENDING, SYNCED, CONFLICT).
- last_modified: ISO-8601 timestamp used for "Last-Write-Wins" resolution.
- server_id: Nullable UUID assigned by the cloud after the first successful sync.

Table 3.1.4.1

Standardized Offline Schema Definitions

Table Name	Feature-Specific Columns	Metadata Columns (BNS 5)
Mood_Logs	mood_score (int), note (text), patient_id (bigint), pain_value (int)	local_id, server_id, sync_status, last_modified
Symptom_Logs	symptom_key (text), symptom_value (text), severity (int), patient_id (bigint)	local_id, server_id, sync_status, last_modified
Medication_Entries	medication_name (text), dosage (text), is_active (bool)	local_id, server_id, sync_status, last_modified
EVV_Timestamps	visit_id (bigint), time_in (ISO), time_out (ISO), is_offline (bool)	local_id, server_id, sync_status, last_modified

Calendar_Events	scheduled_date (date), service_type (text), status (text)	local_id, server_id, sync_status, last_modified

Note. Represents the primary data structures for the five offline-enabled features.

Data Type Specifications

- **UUIDs:** All ID fields use Version 4 UUIDs to ensure uniqueness across distributed mobile devices.
- **Enums:** sync_status is restricted to 0 (PENDING), 1 (SYNCED), or 2 (CONFLICT).
- **Timestamps:** last_modified and feature timestamps use **UTC ISO-8601** format to prevent time-zone drift issues during cloud reconciliation.

The following table details the logical components of the CareConnect system. As established in the legacy CareConnect TDD [3, 8], the architecture is divided into clear functional tiers. Team B has refined these components to support Offline-First resilience (BNS 5), standardized infrastructure provisioning (BNS 6), and system-wide observability (BNS 7), while decoupling architectural responsibilities from specific backend compute implementations.

Table 3.1.4.2

Architectural Components

Tier	Logical Component	Responsibilities	Technology Choice
API Layer	Load Balancer	Request routing; SSL termination; entry point for sync engine.	Application Load Balancer (ALB)
Compute	Containerized Logic	Persistent business logic for User Management, Care Coordination, and Emergency Alerts.	AWS ECS (Fargate)
Sync Tier	Sync Coordinator (Team B)	Reconciliation of local client data; retry handling; conflict resolution; transactional integrity.	AWS ECS (Fargate)
Observability	Telemetry (Team B)	Centralized logging; capture of execution traces; container health monitoring.	CloudWatch / AWS X-Ray
Analytics	Feature Usage (Team B)	Mechanics for capturing non-PII usage events; scrubbing of PHI/PII via service middleware.	AWS ECS (Fargate) / CloudWatch

4. Technology Stack – Team B Additions

This section documents the technologies introduced or standardized by Team B to support offline-first data persistence (BNS 5), AWS-native infrastructure provisioning (BNS 6), and privacy-preserving observability (BNS 7). All technologies listed are additive and do not replace existing CareConnect components unless explicitly stated.

4.1 Mobile Application Platform

Flutter (Dart)

CareConnect continues to use Flutter as the unified client framework for mobile (iOS and Android) and web deployments. Flutter enables consistent behavior across platforms and allows offline synchronization logic to be implemented once and reused across device types.

4.2 Local Persistence and Offline Support (BNS 5)

Drift with SQLite (Selected Implementation)

Team B standardizes on **Drift (SQLite)** as the local persistence layer for all offline-capable modules, including Mood Tracking, Symptom Tracking, Medication Tracking, EVV timestamps, and Calendar entries.

This implementation provides:

- Structured relational storage suitable for queued offline events
- Reactive data access aligned with Flutter state management
- Explicit synchronization metadata (sync_status, last_modified, local_id)
- Compatibility with encrypted SQLite storage for data-at-rest protection

Offline data is persisted locally and synchronized with backend services when network connectivity is restored, using deterministic conflict resolution strategies documented elsewhere in this TDD.

Encryption at Rest

Local data is encrypted using platform-supported secure storage primitives and encrypted SQLite implementations where supported by the operating system.

4.3 Connectivity Detection

Existing network status utilities are reused to detect online and offline transitions. Connectivity changes trigger synchronization workflows without requiring user intervention. No custom networking frameworks are introduced as part of Team B's scope.

4.4 Backend Services and Observability

Spring Boot (Java)

Existing legacy backend services continue to use Spring Boot where applicable; Team B does not refactor business logic but standardizes infrastructure provisioning and observability across both legacy and serverless components.

Spring Boot Actuator

Used to expose standardized health and metrics endpoints for operational monitoring.

Micrometer Metrics

Provides metrics instrumentation integrated with Actuator and exported to AWS-native monitoring services.

OpenTelemetry (Backend)

Serves as the standard abstraction for distributed tracing and metrics collection. Instrumentation is limited to request-level and operational telemetry.

4.5 Infrastructure and Deployment (BNS 6)

AWS CloudFormation

AWS CloudFormation is the exclusive IaC mechanism for Team B-owned resources. All infrastructure is defined declaratively and deployed through CI/CD pipelines. AWS CloudFormation templates constitute the authoritative source of truth for all Team B-owned infrastructure. No infrastructure resources may be created, modified, or deleted outside of CloudFormation-managed stacks.

CloudFormation templates provision:

- **Networking and Access Components:** Security Groups, Target Groups, and Load Balancer listener rules required to support Team B-owned services.
- **Containerized Compute Infrastructure:** AWS ECS Fargate (primary deployment target), including Cluster and Service definitions.
- **Task Definitions:** Declarative configuration for container resource limits (CPU/RAM), environment variables, and Docker image references.
- **Amazon RDS (PostgreSQL):** Relational data persistence for normalized health records.
- **Amazon S3:** Scalable object storage for binary data and telemetry archives.
- **Identity & Access:** IAM roles and policies tailored for Fargate Task Execution and least-privilege resource access.
- **Observability Stack:** CloudWatch log groups, metric filters, and configuration supporting **OpenTelemetry-based** tracing and metrics export.

CareConnect Technical Design Document

Inherited infrastructure components, such as API ingress layers or platform-managed routing, are referenced for architectural context but are not redefined or modified as part of Team B–owned CloudFormation stacks.

All infrastructure changes are version-controlled, peer-reviewed, and deployed exclusively through CI/CD pipelines. Manual infrastructure modifications via the AWS console are explicitly prohibited to maintain environment parity, traceability, and long-term maintainability.

4.6 Monitoring and Logging

Amazon CloudWatch

CloudWatch serves as the centralized destination for logs and metrics. All operational telemetry emitted by backend services and infrastructure components is captured here.

No custom dashboards are required at the MVP stage.

4.7 Analytics and Feature Telemetry (BNS 7)

Anonymous Feature Analytics

Feature usage events are captured using event-based telemetry without collecting PII or PHI. Events include feature identifiers, timestamps, and platform metadata only.

OpenTelemetry-Compatible Events

OpenTelemetry-compatible client and server events are preferred. Firebase Analytics may be used only if OpenTelemetry introduces unacceptable delivery risk. Only feature usage and operational health events are collected.

4.8 Security and Access Control

OS-Level Encryption

Offline data stored on client devices is encrypted at rest using operating system–supported encryption mechanisms and encrypted persistence libraries.

AWS Identity and Access Management (IAM)

Existing IAM roles are reused wherever possible. No new privilege scopes are introduced without explicit review. Access follows the principle of least privilege across all Team B–owned components.

5. Functional Specifications – Team B Scope

This section defines functional behavior introduced or standardized by Team B. All functionality described here operates within the constraints defined in the Software Requirements Specification (SRS).

5.1 Offline Data Capture and Synchronization

The system allows selected CareConnect features to function during periods of limited or no network connectivity by persisting user actions locally and synchronizing them once connectivity is restored.

Supported Modules

- Mood Tracking
- Symptom Tracking
- Medication Tracking
- Electronic Visit Verification (EVV – timestamps only)
- Calendar

Functional Behavior

- User actions performed while offline are stored locally as queued events.
- A shared **OfflineService** abstraction coordinates:
 - Queue persistence
 - Sync triggering
 - Retry logic
- Each queued event includes:
 - Action type
 - Minimal payload
 - Timestamp
 - Sync status
- Queued events persist across application restarts.
- When connectivity is restored:
 - Synchronization is triggered automatically or manually.
 - Events are uploaded in FIFO order.
 - Successfully synced events are removed from local storage.

Exclusions

- Messaging, billing, AI features, media uploads, and administrative workflows do not support offline operation.

5.2 Infrastructure Management via CloudFormation

The Sync Coordinator is a logical abstraction representing synchronization responsibilities integrated within the backend application services. Team B standardizes the provisioning of its owned infrastructure using **AWS CloudFormation**, ensuring that stack definitions are strictly aligned with the **containerized deployment topology (AWS Fargate)**.

Functional Behavior

- **Infrastructure Alignment:** CloudFormation templates define and manage Team B–owned resources (such as specific RDS tables, S3 buckets, and IAM policies) in direct alignment with the current **service-based** deployment topology.
- **Modular & Version-Controlled:** Templates are managed in Git, ensuring infrastructure changes are peer-reviewed.
- **Automated Deployment:** Infrastructure updates occur via CI pipelines to prevent configuration drift.
- **Zero Manual Intervention:** Manual AWS Console changes are strictly prohibited.
- **Security & Continuity:** Resource names, IAM boundaries, and network structures are preserved to ensure secure service-to-service communication.

5.3 Observability and Feature Analytics

The system provides operational visibility and anonymous feature usage insights while maintaining healthcare privacy constraints.

Functional Behavior

- Backend exposes health and metrics endpoints via Spring Boot Actuator.
- Metrics and logs are exported to Amazon CloudWatch.
- OpenTelemetry is used as the standard telemetry abstraction.
- Anonymous feature usage events are emitted from the mobile application.
- No personally identifiable or health-related data is collected.

6. API Design Specification for CareConnect – Team B Extensions

This section documents API interactions required to support offline synchronization, observability, and analytics. Existing APIs are reused wherever possible.

6.1 Offline Synchronization API Interaction Pattern

Purpose: Support upload of locally queued offline events once connectivity is restored.

General Characteristics

- Authenticated requests
- Idempotent where applicable
- Minimal payloads
- Server-side validation enforced

Interaction Pattern

- Offline events are synchronized by invoking **existing feature APIs** for each supported module (mood, symptom, medication, EVV timestamps, calendar).
- If batching is required, it will be documented as a **Team B recommendation** and implemented only if confirmed by backend ownership during Milestone 3.

Conflict Handling

- MVP conflict handling is deterministic and minimal (server-wins or last-write-wins).
- Conflicts are logged for troubleshooting.
- Advanced conflict resolution workflows are deferred.

Observability Endpoints

- /actuator/health
- /actuator/metrics

These endpoints:

- Are non-public
- Are protected within trusted network boundaries
- Are configured to avoid emitting sensitive data

Analytics Event Submission

- Events transmitted asynchronously.
- Events include feature name, timestamp, and platform.
- No identifiers tied to users or devices are transmitted.

7. Data Models

This section details the data architecture and database design for CareConnect. To meet the performance and scalability requirements of a containerized environment, the system employs a PostgreSQL-based polyglot storage approach. This architecture ensures that all inherited data entities are optimized for AWS-native services and reliable offline synchronization via the Drift framework

The system utilizes specialized storage engines to handle the diverse data requirements defined in the baseline documentation [3, 8]:

- **PostgreSQL (Amazon RDS):** Serves as the primary relational engine for core entities including Users, Roles, Billing, Scheduling, and Health Records [3, 8]. Team B utilizes PostgreSQL to support the complex JSONB querying and indexing required for BNS 5 (Offline Sync). Billing entities are included for schema continuity only and are not implemented, tested, or exercised in the MVP.
- **Semi-Structured Storage:** Reserved for metadata-heavy modules such as OCR outputs, AI insights, and chat history [3, 8].
- **Time-Series Data:** Designed to handle high-velocity logging for EVV visit logs and symptom tracking as part of the current MVP. Support for wearable-derived metrics is inherited from the legacy CareConnect design [3, 8] and is not implemented, extended, or evaluated in the current Team B scope
- **Amazon S3:** Employed for unstructured media storage, including documents, images, and audio/video clips [3, 8].

7.1 Scope

The scope of this section is to define the data modeling and database design for the CareConnect platform [3, 8]. This design is presented progressively, advancing from inherited abstract models toward Team B's specific AWS-native implementation and operational considerations. This ensures the system's data layer remains conceptually sound with the legacy baseline while technically aligned with team B's requirements.

- **Conceptual Model:** Identifies high-level entities and relationships inherited from the initial prototype [3, 8]. It provides a broad understanding of how data domains (Users, Billing, Health Tracking, and Communications) interconnect.
- **Logical Model:** Refines the conceptual design into normalized tables and constraints [3, 8]. Team B has extended this model to include metadata for BNS 5 (Offline Synchronization), such as conflict-resolution timestamps and sync-status flags.
- **Data Dictionary:** Serves as the authoritative reference for attributes, keys, and traceability back to requirements [3, 8]. It has been updated to reflect PostgreSQL-specific data types (e.g., UUIDs for offline-safe primary keys).
- **Physical Model:** Translates the logical design into actual database structures. This includes schemas, table definitions, and constraints optimized for the Amazon RDS PostgreSQL engine.
- **Physical Design Considerations:** Outlines normalization decisions and indexing strategies inherited from the baseline [3, 8].
- **AWS Deployment (Polyglot Strategy):** Describes the deployment of data models using a polyglot approach:
 - Amazon RDS PostgreSQL for core relational data [3, 8].
 - Semi-structured/Time-series storage for high-volume logs and AI insights.
 - Amazon S3 for unstructured media (images/audio) [3, 8].
 - BNS 6 Note: All infrastructure is provisioned via AWS CloudFormation for automated environment lifecycle management.
- **Security & Compliance:** Details how HIPAA and GDPR requirements are met through Role-Based Access Control (RBAC) [3, 8], audit logging, and PHI/PII tagging—specifically supporting the privacy-preserving telemetry goals of BNS 7.
- **Backup & Recovery:** Defines strategies for snapshotting and disaster recovery to meet recovery objectives (RTO/RPO) inherited from the project baseline [3, 8].

7.2 Conceptual Data Model

The conceptual model represents the high-level business entities and their relationships within the CareConnect ecosystem. This model supports a polyglot persistence strategy where relational data resides in PostgreSQL [3, 8], while high-velocity telemetry and offline-first metadata are optimized for synchronization [BNS 5, 6].

7.2.1 BNS 5 Key Entities

The following entities are designated for offline persistence to ensure continuity during intermittent connectivity.

Mood Tracking [3, 8]

- **mood_entry**: Stores the primary mood score (1-5) and the time the entry was recorded.
- **mood_pain_log**: An extended log for recording both mood and pain levels (1-10) with additional text notes for context.

Symptom Tracking [3, 8]

- **symptom_entry**: Records specific health symptoms, their severity level (1-5), and qualitative notes provided by the patient or caregiver.

Medication Tracking [3, 8]

- **patient_medication**: Provides a local reference for medication names, dosages, and frequencies so users can log administration even without a signal.

Electronic Visit Verification (EVV) [3, 8]

- **scheduled_visits**: Acts as the local source of truth for the caregiver's daily itinerary.
- **visit** (and associated audit tables): Records the immutable check-in/check-out timestamps and GPS coordinates captured while the device is offline.

Calendar (Manual Entries) [3, 8]

- **tasks**: Stores daily care tasks and reminders that must remain accessible and "markable as complete" regardless of internet status.

7.2.2 High-Level Relationships (Offline Focus)

The relationships listed below are constrained to the entities essential for the BNS 5 (Offline Use) modules. For a comprehensive overview of the full system architecture—including Financials, Social Support, and Telemedicine—refer to the legacy TDD [3, 8]. These relationships govern how health logs and visit data are associated with the correct user during offline synchronization.

- **User to Profile (One-to-One)**: A User is the core identity; the Patient or Caregiver tables extend this identity with specific clinical or professional data needed for offline identification.
- **Patient to Caregiver (Many-to-Many)**: Managed via the `patient_caregiver` link. This allows a caregiver to view the offline-synced schedules and medication lists for multiple patients during their shift.
- **Patient to Health Logs (One-to-Many)**: A single Patient profile is associated with multiple `mood_entry`, `symptom_entry`, and `meal_entry` records. This relationship ensures that logs created offline are pinned to the correct patient record upon reconnection.
- **Patient to Medication (One-to-Many)**: A Patient is linked to multiple `patient_medication` entries. This relationship allows the local device to "know" which prescriptions to display in the offline medication tracker.

- **Scheduled Visits to Tasks (One-to-Many):** A *scheduled_visit* (the caregiver's appointment) acts as a parent container for specific tasks that must be completed and logged during that specific offline window.
- **Tasks to Notifications (One-to-Many):** Local tasks trigger *scheduled_notification* alerts directly on the device, ensuring reminders function without an active internet connection.

7.3 Logical Data Model

The tables listed below are the "Logical" implementation of the MVP requirements. This model adheres to the third Normal Form (3NF). For the complete logical model covering telehealth, home monitoring, and diarization, refer to Legacy TDD Section 7.3 [3, 8].

7.3.1 Core & Identity Tables

These tables provide the foundational identity and access control (RBAC) required for both online and offline sessions [3, 8].

Table	Description
users	Core identity record for all actors (patients, caregivers, family, and admins).
patient	One-to-one extension of the users table containing clinical and demographic details.
patient_caregiver	Junction table managing the many-to-many relationship between patients and their assigned caregivers.

7.3.2 Health Tracking (BNS 5: Offline Optimized)

Per the **BNS 5** requirement, these tables are explicitly scoped for local storage on the mobile device to support offline health logging [3, 8].

Table	Description
mood_entry	Primary diary for mood scores (1-5) and capture timestamps.
mood_pain_log	Granular logs tracking the intersection of emotional state and physical pain levels.
symptom_entry	Record of specific physical symptoms, including severity and qualitative notes.
patient_medication	Local registry of prescriptions, dosages, and active/inactive status.
meal_entry	Logs of nutritional intake and dietary responses.

7.3.3 Electronic Visit Verification (EVV) & Scheduling

These tables support the state-mandated verification of caregiver visits and the daily management of care tasks [3, 8].

Table	Description

CareConnect Technical Design Document

scheduled_visits	Itinerary of planned caregiver visits, including service types and priority.
tasks	Daily care activities that must be marked as complete during a visit.
participant	The EVV-specific record for the patient receiving services.
visit	Immutable record of visit events including time in/out, GPS coordinates, and caregiver ID.
visit_correction	Supervisor-governed audit trail for modifications to original visit data.

7.3.4 AI Support & Configuration [BNS 6]

These tables manage the behavior and output of the integrated AI models for patient support [3, 8]. The following tables represent architectural readiness only and are inherited from the legacy CareConnect design [3, 8]. Team B does not implement, train, evaluate, or operationalize AI-driven clinical logic as part of the current MVP.

Table	Description
patient_ai_config	Hardened configuration table for model parameters (OpenAI/DeepSeek) and system prompts.
patient_note	Storage for AI-generated summaries and structured visit observations.

7.3.5 Logical Design Constraints

The following logical rules are applied to the tables above:

1. **Immutability:** The visit and audit_log tables are "append-only." Once a record is synced to the cloud, it cannot be deleted; any modifications are recorded as new entries in the visit_correction table.
2. **Data Persistence:** Categorical data (e.g., severity or status) is enforced via PostgreSQL CHECK constraints (visible in the DDL) to ensure that even offline-generated data remains valid upon synchronization.
3. **Soft Deletes:** Clinical data such as *patient_medication* utilizes an *is_active* flag rather than hard deletion to preserve historical data for AI-driven longitudinal analysis [3, 8].

7.4 Data Dictionary

The following tables represent the detailed schema for the modules prioritized for Team B requirements. For the complete data dictionary covering all legacy tables (including Billing, Social, and Telehealth), please refer to Legacy TDD Section 7.4. [3, 8]

The attributes below are derived from the Legacy TDD Section 7.4. [3, 8] and align with the BNS 5 (Offline Use) and BNS 7 (Privacy/Audit) requirements

7.4.1 Health Tracking (Offline Optimized)

These tables are synchronized locally to the mobile device to support patient care in low-connectivity environments [3, 8].

Table: mood_entry

CareConnect Technical Design Document

- **id** (bigint, PK): Unique identifier for the mood record.
- **patient_user_id** (bigint, FK): Reference to the patient (users.id).
- **mood_score** (integer): Score from 1-5 (Enforced by PostgreSQL CHECK constraint).
- **taken_at** (timestamp): The user-reported time of the mood entry.
- **created_at** (timestamp): System timestamp when the record was initially saved.

Table: symptom_entry

- **id** (bigint, PK): Unique identifier for the symptom log.
- **patient_user_id** (bigint, FK): Reference to the patient (users.id).
- **symptom_key** (varchar 255): Categorical identifier for the symptom (e.g., "Nausea").
- **severity** (integer): Intensity score from 1-5 (Enforced by CHECK constraint).
- **notes** (text): Qualitative details provided by the caregiver or patient.
- **taken_at** (timestamp): Local time the symptom was observed.

7.4.2 Electronic Visit Verification (EVV) & Tasks

These entities provide the immutable evidence required for state-mandated visit verification and task management [3, 8].

Table: scheduled_visits

- **id** (bigint, PK): Unique identifier for the scheduled visit.
- **caregiver_id** (bigint, FK): Reference to the assigned caregiver (users.id).
- **patient_id** (bigint, FK): Reference to the assigned patient (patient.id).
- **scheduled_date** (date): Expected date of service delivery.
- **status** (varchar 50): Current state of the visit (e.g., PENDING, COMPLETED, CANCELLED).
- **priority** (varchar 20): Urgency level assigned to the visit.

Table: tasks

- **id** (bigint, PK): Unique identifier for the care task.
- **patient_id** (bigint, FK): Reference to the associated patient (patient.id).
- **name** (varchar 255): Short description of the task (e.g., "Administer Medication").
- **iscompleted** (boolean): Completion status; defaults to false.
- **task_type** (varchar 255): Category used for filtering and reporting.

7.4.3 AI Configuration & Clinical Notes [BNS 6]

Hardened tables supporting the integration of Large Language Models (LLMs) for patient support and summarization [3, 8]. The following tables represent architectural readiness only and are inherited from the legacy CareConnect design [3, 8]. Team B does not implement, train, evaluate, or operationalize AI-driven clinical logic as part of the current MVP.

Table: patient_ai_config

- **id** (bigint, PK): Unique identifier for the configuration profile.
- **patient_id** (bigint, FK): Reference to the specific patient being supported.
- **ai_provider** (varchar 20): Selected model provider (OPENAI or DEEPSEEK).
- **temperature** (numeric 3,2): AI creativity/randomness setting (0.0 to 2.0).
- **system_prompt** (text): Hardened behavior instructions and constraints for the AI assistant.

Table: patient_note

- **id** (bigint, PK): Unique identifier for the note.
- **patient_id** (bigint, FK): Reference to the patient record.
- **note** (text): The raw unstructured note content.
- **ai_summary** (text): The machine-generated summary derived from the raw note.

7.4.4 Data Integrity Implementation

- **Check Constraints:** As defined in the *schema_dump.sql*, fields such as *mood_score* and *severity* include PostgreSQL CHECK constraints to ensure that data captured offline remains within valid ranges (1-5) upon synchronization.
- **JSONB Utilization:** For fields requiring high flexibility—such as AI trigger keywords or device metadata—the schema utilizes the JSONB type to support efficient indexing of semi-structured data.
- **Immutability:** Following **BNS 7**, the visit and *audit_log* tables are designed as append-only structures to prevent the deletion of sensitive verification data.

7.4 Physical Data Model

This section outlines the physical implementation of the database for Team B MVP. For the comprehensive physical design conventions, including legacy storage engines and global charset standards, refer to the Legacy TDD Section 7.4 [3, 8].

7.4.1 Storage Conventions and Standards

The physical implementation of the PostgreSQL database follows the core standards inherited from the baseline [3, 8]:

- **Engine and Transactions:** All tables are managed via the default PostgreSQL storage engine to ensure full ACID compliance and robust Foreign Key (FK) support.
- **Primary and Foreign Keys:** Primary keys use BIGINT with GENERATED BY DEFAULT AS IDENTITY (as seen in the *schema_dump.sql*). Foreign keys are strictly typed as BIGINT to ensure referential integrity and performance across joins.
- **Timezone-Aware Timestamps:** Timestamps utilize timestamp(6) without time zone or timestamp with time zone, defaulting to CURRENT_TIMESTAMP or now() to ensure standardized UTC storage across the globally distributed care system.

- **JSONB for Flexibility:** Unlike the legacy MySQL JSON format, this implementation utilizes PostgreSQL **JSONB** for any semi-structured data (e.g., trigger_keywords in patient_note_taker_config). JSONB offers superior performance through binary storage and supports GIN indexing for efficient querying.
- **Hardened Type Safety:** Enumerations are primarily implemented via VARCHAR combined with explicit **CHECK constraints**. For example, the mood_score is restricted to (mood_score >= 1) AND (mood_score <= 5), and ai_provider is restricted to 'OPENAI' or 'DEEPSEEK'. This ensures strict type safety at the database level, preventing invalid data entry from offline clients.

7.4.2 Offline Persistence Strategy (BNS 5)

To support the requirement for a lightweight, offline-first application, the physical model incorporates the following:

- **Partial Syncing:** The physical database supports high-performance querying so that the mobile client can fetch only "Active" or "Recent" records (e.g., is_active = true), minimizing the local storage footprint.
- **Conflict Resolution Fields:** Tables such as mood_entry and symptom_entry include updated_at and created_at fields to facilitate "last-write-wins" synchronization when mobile devices reconnect to the RDS instance.

7.4.3 Privacy and Audit Hardening (BNS 7)

- **Immutable Records:** Tables associated with EVV (e.g., visit) and high-level health tracking are treated as append-only.
- **Audit Logging:** The audit_log table captures the actor_id and entity_id to ensure that every change made—whether online or synced later—is fully traceable and HIPAA-compliant.

7.6 Physical Design Considerations

This section details the performance and scalability optimizations for the Team B MVP database. These considerations prioritize BNS 5 (Offline Synchronization) and BNS 7 (Privacy-Preserving Telemetry). For general database standards not covered here, refer to Legacy TDD Section 7.6 [3, 8].

7.6.1 Indexing Strategy

To ensure high-performance data retrieval for health tracking and EVV verification, the following indexing standards are applied:

- **FK Column Indexing:** All foreign key columns (e.g., patient_id, caregiver_id) are indexed to optimize relational joins between the core identity tables and the offline-synced health logs.
- **Multi-Column Indexes:** Composite indexes are implemented on high-velocity tables to speed up time-based queries. Key examples include:

CareConnect Technical Design Document

- **symptom_entry**: (patient_user_id, taken_at)
- **meal_entry**: (patient_user_id, created_at)
- **mood_entry**: (patient_user_id, taken_at)

7.6.2 Partitioning (BNS 5 & 7)

To handle the growth of longitudinal health data and maintain audit compliance, range-partitioning is utilized:

- **Time-Series Partitioning**: High-volume tables, including mood_entry, and audit_log, are range-partitioned by month. This strategy allows the system to scan only relevant monthly shards when generating AI summaries or clinical reports, significantly reducing I/O overhead as the dataset grows over time.

7.6.3 VACUUM & Autovacuum Tuning

PostgreSQL's autovacuum process is hardened for high-write tables:

- **Bloat Management**: Autovacuum thresholds are specifically tuned for the visit and symptom_entry tables. Because these tables receive frequent inserts from offline clients syncing in batches, aggressive vacuuming ensures that table bloat is minimized and index performance remains consistent.

7.6.4 Connection Pooling

To manage the spike in connections when multiple mobile clients attempt to sync data simultaneously, connection management is offloaded:

- **PgBouncer / RDS Proxy**: The architecture utilizes connection pooling to reduce the overhead of establishing new database sessions. This ensures that the system remains responsive even during peak periods of EVV check-ins and health log synchronization.

7.7 AWS PostgreSQL Deployment

This section focuses on the AWS hosting environment for the core CareConnect database, specifically optimized for BNS 5 (Offline Support) and the high-integrity requirements of EVV. For the legacy MySQL deployment details and the full range of companion stores, refer to Legacy TDD Section 7.7 [3,8]

For the Hardening Phase, the deployment centers on **Amazon RDS for PostgreSQL (v15+)** to leverage modern features like the pgvector extension for AI support and robust JSONB indexing.

7.7.1 Managed Database Infrastructure

The database is deployed using **Amazon RDS for PostgreSQL**, configured with the following hardening standards:

- **Engine:** RDS PostgreSQL (v15+) is utilized to ensure compatibility with modern SQL standards and AI-specific extensions.
- **High Availability:** A **Multi-AZ** deployment is implemented across two AWS Availability Zones. This ensures automated failover to a standby instance in the event of an infrastructure failure, maintaining the "Always-On" requirement for critical health services.
- **Read Scaling:** Read replicas are deployed to offload heavy analytical workloads—such as generating AI health summaries and administrative reports—from the primary write instance, preserving performance for real-time offline synchronization.
- **Encryption:**
 - **At-Rest:** Data is encrypted using **AWS Key Management Service (KMS)** with AES-256 encryption.
 - **In-Transit:** All connections are enforced via **SSL/TLS 1.2+** to ensure data security between the API and the RDS instance.
- **Backups:** Automated daily snapshots are captured with a 35-day retention period. **Point-In-Time Recovery (PITR)** is enabled, allowing for recovery to any specific second, which is critical for maintaining the audit integrity of EVV records.
- **Monitoring:** CloudWatch Metrics and **Enhanced Monitoring** are active to provide 1-second granularity into system health. Performance Insights are enabled to identify and tune slow-running queries in the health-tracking modules.

7.7.2 Companion Stores (MVP Scoped)

To maintain a lightweight and high-performance primary database, secondary artifacts are offloaded to specialized AWS services:

- **Amazon S3:** Used for the storage of media and large artifacts (e.g., Clinical Note transcripts, EVV location snapshots). All data is KMS-encrypted and accessed strictly via **Private Signed URLs** with a short TTL (Time-To-Live).
- **Amazon RDS (High-Volume Logs):** Append-heavy artifacts such as raw OCR data or telemetry logs are stored in a dedicated high-performance RDS instance to prevent bloat in the primary clinical health database.

7.7.3 Offline-First Sync Optimization (BNS 5)

The AWS deployment incorporates **Amazon RDS Proxy** to manage the burst of connections expected when multiple mobile devices reconnect to the internet. The proxy handles connection pooling and preserves application performance during the "Sync Storm" that occurs when offline-captured Mood, Symptom, and EVV data are uploaded to the cloud.

7.8 Security & Compliance

This section details the security hardening measures for the Team B MVP, focusing on the transition from the legacy MySQL baseline to a hardened PostgreSQL environment. For full details on the legacy global security policy, refer to Legacy TDD Section 7.8 [3, 8].

The following security controls are implemented to protect sensitive health data and ensure compliance with HIPAA and **BNS 7 (Privacy-Preserving Telemetry)**.

CareConnect Technical Design Document

7.8.1 Network Security

- **Private Subnet Isolation:** The PostgreSQL RDS instance is deployed within isolated private subnets across multiple Availability Zones. It has no direct route to the public internet.
- **Controlled Access:** Access is strictly limited to the application layer (API) and authorized administrative personnel. Admins must connect via a **Bastion Host** or a secure **Site-to-Site VPN**, ensuring that database ports are never exposed to the open web.

7.8.2 Identity and Access Management (IAM)

- **IAM Database Authentication:** In place of traditional static passwords, the system leverages AWS IAM authentication. This allows the application to use short-lived authentication tokens, reducing the risk of credential leakage.
- **Principle of Least Privilege:** Access levels are strictly segmented into dedicated roles:
 - **App_Role:** Permissioned only for CRUD operations on MVP-specific tables (Health Tracking, EVV, Tasks).
 - **Reporting_Role:** Read-only access to specific views for AI summarization and administrative reporting, with no permission to modify records.
 - **Admin_Role:** Restricted to schema migrations and maintenance, used only via secure administrative tunnels.

7.8.3 Data Protection (Encryption)

- **Encryption At-Rest:** All data stored in the RDS instance, including automated backups and read replicas, is encrypted using **AWS KMS-managed keys (AES-256)**.
- **Encryption In-Transit:** The database configuration mandates **SSL/TLS 1.2+** for all connections. Any connection attempt made without encryption is automatically rejected by the RDS security group.

7.8.4 Audit Logging & Compliance (BNS 7)

- **Hardened Auditing:** While the legacy system utilized MySQL audit extensions, this implementation uses **pgAudit** for PostgreSQL.
- **Activity Monitoring:** The system maintains detailed logs of all database activities, including session starts, DDL changes, and access to sensitive tables (e.g., `patient_medication` and `visit`).
- **Non-Repudiation:** Per **BNS 7**, the `audit_log` table captures every state change for EVV records, providing a permanent, immutable trail for compliance monitoring and clinical safety audits.

7.8.5 Privacy-Preserving Telemetry

- **Anonymization for AI:** When data is processed for AI summaries (as stored in `patient_note`), the system utilizes hardened configuration parameters to ensure that only the minimum necessary clinical data is transmitted to the LLM providers, adhering to privacy-by-design principles.

7.9 Backup & Recovery

This section details the recovery and data durability strategies for Team B MVP, transitioning from legacy requirements to a hardened PostgreSQL environment. For the full legacy backup policy, refer to legacy TDD Section 7.9 [3, 8]

The backup and recovery strategy is designed to ensure HIPAA compliance and data integrity for critical modules like **EVV** and **Health Tracking**, which may involve large batches of data synced from offline states.

7.9.1 Automated Data Durability

To provide a reliable recovery point in the event of hardware or infrastructure failure, the following automated mechanisms are implemented:

- **Daily Snapshots:** Automated full-database snapshots are taken daily and retained for a period of **35 days** (hardened from the legacy 7–30 day window).
- **Point-in-Time Recovery (PITR):** The system is configured with a **35-day retention window**, allowing the database to be restored to any specific second within that timeframe. This is critical for reversing accidental batch-sync errors from offline clients.
- **Storage-Level Resilience:** Leveraging **RDS PostgreSQL v15.15**, the physical data is stored across multiple Availability Zones with 6-way replication (if using Aurora) or Multi-AZ synchronous replication.

7.9.2 Manual Safeguards and Maintenance

Before significant system changes, the following manual protocols are enforced:

- **Pre-Migration Snapshots:** Manual snapshots are created immediately before any schema updates or Flyway migrations to ensure immediate recoverability if issues are introduced.
- **Schema Integrity:** The use of **Flyway schema history** ensures that every version of the database is tracked and can be reconstructed alongside a backup.

7.9.3 Disaster Recovery and Business Continuity

In the event of a regional AWS failure, the system employs a cross-region strategy:

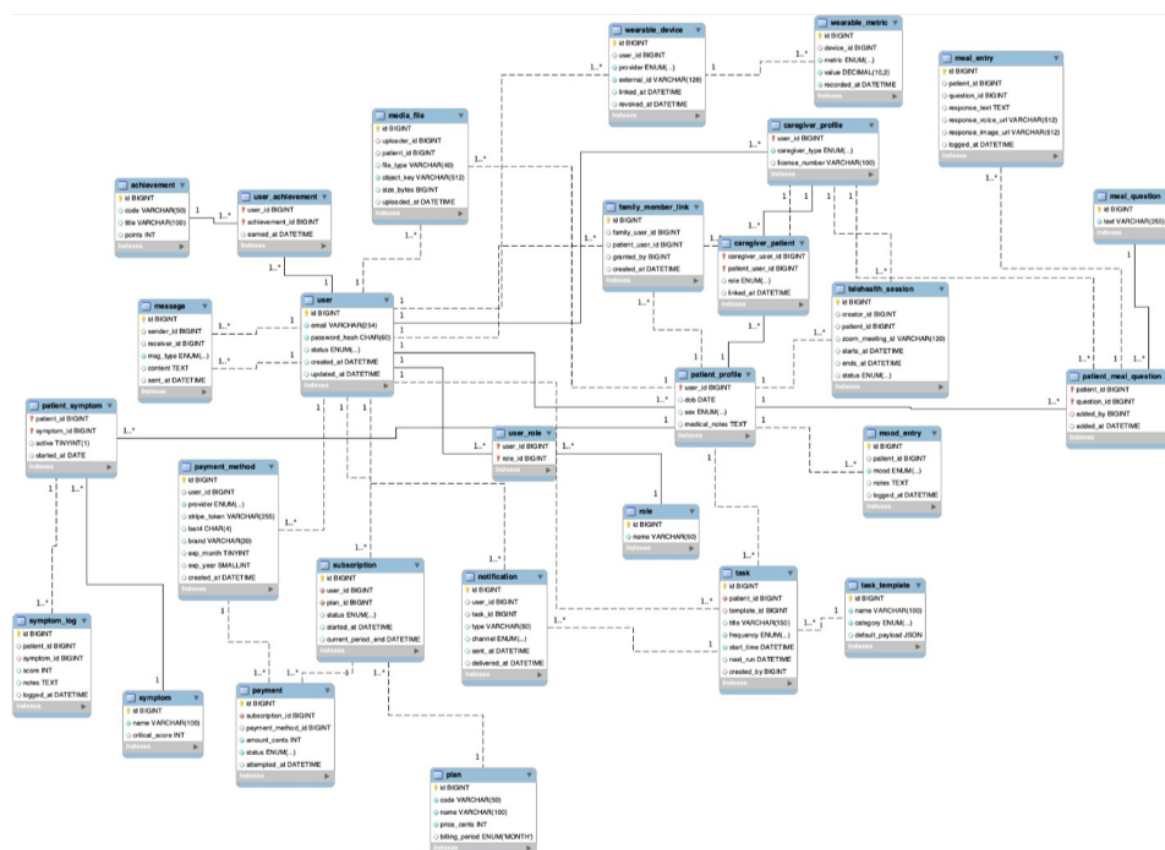
- **Cross-Region Read Replica Promotion:** A read replica is maintained in a secondary AWS region. In a disaster scenario, this replica is promoted to primary status, minimizing downtime and ensuring caregivers can continue verifying visits and patients can log symptoms.

7.9.4 Data Integrity Implementation

- **Transaction Logs:** PostgreSQL **Write-Ahead Logs (WAL)** are continuously backed up to Amazon S3, supporting the PITR mechanism and ensuring no data is lost between daily snapshots.

- ## 7.10 Entity-Relationship Diagram

Figure 7.10.1
Entity-Relationship Diagram (ERD)



CareConnect Technical Design Document

defined in the Legacy Technical Design Document (TDD) [3], Section 8. This ensures visual and behavioral consistency with existing CareConnect functionality.

When operating in offline mode, only the features explicitly identified in the Software Requirements Specification (SRS) remain available to the user. These offline-capable modules include Mood Tracking, Symptom Tracking, Medication Management, EVV (Electronic Visit Verification), and Calendar functionality. All other application features are disabled and visually grayed out to prevent unsupported interactions and reduce user confusion.

The user interface clearly communicates connectivity state through visible indicators such as an offline banner or status notification. Data entered while offline is persisted locally and labeled as “pending synchronization.” Upon restoration of network connectivity, the application provides clear feedback indicating synchronization progress and successful reconciliation with backend services. Sync status may be communicated through icons, progress indicators, or status messages consistent with the CareConnect design system.

In accordance with Business Needs Statement (BNS) requirement 5, the application includes a user-controlled persistence toggle located within the Application Settings menu. This toggle allows users to enable or disable local data persistence for offline-capable modules. The state of the toggle is clearly labeled and accompanied by explanatory text to ensure users understand its impact on offline data storage and synchronization behavior.

To preserve usability and minimize cognitive load, error messages and system notifications are designed to be clear, actionable, and non-technical. The UI avoids exposing implementation or system-level details while ensuring users are informed of offline limitations, synchronization outcomes, and feature availability. Accessibility and usability best practices are applied consistently across all UI elements affected by offline operation.

8.1 Legacy Design System Alignment

All UI components, layouts, color palettes, typography, and interaction patterns used by Team B enhancements adhere to the global CareConnect design system defined in legacy Technical Design Document Section 8 [3, 8].

Team B does not introduce new visual themes, color schemes, or typography variants. Offline indicators, synchronization status elements, and settings controls are implemented using existing CareConnect design tokens and UI patterns to ensure visual and behavioral consistency across the application.

This alignment ensures that Team B–owned enhancements (Offline Persistence and EVV behavior) integrate seamlessly into the existing CareConnect brand identity without introducing divergent UI conventions.

8.2 Offline Mode UI Behaviors (BNS 5)

When the application detects loss of network connectivity, the UI transitions into a defined offline state with the following behaviors:

Offline State Indicators

CareConnect Technical Design Document

- A persistent offline banner or status indicator is displayed to inform the user that the application is operating without network connectivity.
- Offline indicators use existing CareConnect notification styling as defined in the legacy design system.

Sync Status Indicators

- Data created or modified while offline is visually marked as “Pending Sync.”
- Upon reconnection, the UI displays synchronization progress and completion states using non-intrusive indicators (e.g., icons or progress markers).
- Successfully synchronized items transition automatically to a “Synced” state without requiring user action.

Gray-Out Logic for Non-Scoped Modules

- Features not supported during offline operation are visually disabled.
- Disabled modules remain visible but are non-interactive and visually muted.
- Gray-out behavior applies to, but is not limited to, messaging, administrative workflows, billing-related features, and non-MVP integrations.
- Attempts to interact with disabled modules do not trigger errors; instead, the UI maintains the disabled state without user disruption.

8.3 User-Controlled Persistence Toggle (BNS 5)

To support user control over offline behavior, the application includes a **User-Controlled Persistence Toggle** with the following characteristics:

Location

- The toggle is located within the **Application Settings** menu under data or connectivity preferences.

Default Behavior

- Offline persistence is enabled by default to ensure continuity of care workflows in low-connectivity environments.

Toggle Disabled Behavior

- When disabled, the application prevents new data from being stored locally for offline use.
- Features requiring offline persistence remain inaccessible while offline.
- Existing locally stored data is retained but not expanded or modified until persistence is re-enabled or connectivity is restored.

Scope of Effect

CareConnect Technical Design Document

- The toggle applies only to BNS 5–scoped modules (Mood, Symptoms, Medications, EVV timestamps, Calendar).
- It does not affect authentication, core navigation, or backend synchronization logic once connectivity is available.

Figure 8.3.1

Offline Toggle & Banner Display



8.4 Observability and Anonymous Feature Analytics (BNS 7)

The Observability and Anonymous Feature Analytics feature set establishes the telemetry framework necessary to monitor system health and understand feature adoption patterns while preserving user privacy.

This feature set enables operational monitoring and product insight without collecting personally identifiable information (PII).

Observability Framework

The application implements structured observability controls to monitor runtime health and diagnose system failures.

Application Logging

Frontend and backend services generate structured logs to capture:

- Application errors
- Warning conditions
- Performance anomalies
- Synchronization failures

Logs exclude personal identifiers, authentication tokens, request payload bodies, and user-generated content.

CareConnect Technical Design Document

Structured logging follows a consistent schema to support automated monitoring and troubleshooting.

Request and Trace Correlation

Each inbound request is assigned a unique request identifier for traceability.

Request identifiers:

- Are randomly generated
 - Are not derived from user identity
 - Persist only for the lifecycle of the request
 - Enable cross-service debugging without exposing personal data
- Distributed tracing supports technical diagnostics without behavioral profiling.

Performance and Health Metrics

System metrics are collected to monitor:

- API response times
 - Error rates per endpoint
 - WebSocket connection health
 - Offline synchronization performance
 - Database query duration
- These metrics support service-level monitoring and operational reliability.

Anonymous Feature Analytics

Anonymous analytics collect high-level feature usage data to inform product improvement decisions.

Event Collection Scope

Analytics events may include:

- Feature accessed (e.g., AI Assistant opened)
- Page navigation events
- Offline Mode toggled
- Sync initiated or completed
- Login success counts (aggregate only)

Events do not include:

- Names
- Email addresses
- User IDs
- Device identifiers tied to identity
- Message contents
- Clinical data

Only feature-level and aggregate behavioral signals are captured.

Data Minimization and Privacy Controls

The analytics implementation follows privacy-by-design principles.

Default State

CareConnect Technical Design Document

Analytics collection is disabled by default.

Explicit User Opt-In

Users must explicitly enable analytics within the Application Settings menu.

Enabling analytics requires confirmation through an informed consent dialog that:

- Explains the type of data collected
- States that no personal information is captured
- Clarifies that the setting can be changed at any time

User Revocation

Users may disable analytics at any time.

When disabled:

- Event collection ceases immediately
- No new telemetry is transmitted
- Existing aggregated analytics remain anonymized

Anonymous Session Handling

To enable non-identifiable usage analysis:

- A random session identifier may be generated locally
- The identifier is not linked to authentication credentials
- It is not transmitted alongside personal identifiers
- It may rotate periodically

This allows session-level flow analysis without establishing user identity.

Security Controls

Telemetry transmission adheres to the following controls:

- All analytics data is transmitted over secure HTTPS connections
 - Authentication tokens are never logged
 - Sensitive request data is excluded from logs
 - Rate limiting prevents analytics endpoint abuse
- Observability systems are isolated from user-facing application logic.

Scope of Effect

This feature applies to:

- Frontend application telemetry
- Backend service logging
- Feature usage analytics
- Performance monitoring

It does not:

- Modify authentication workflows
- Store personal data for analytics
- Track user identity across sessions
- Enable behavioral profiling

Observability data is strictly technical and aggregate in nature.

Compliance and Governance

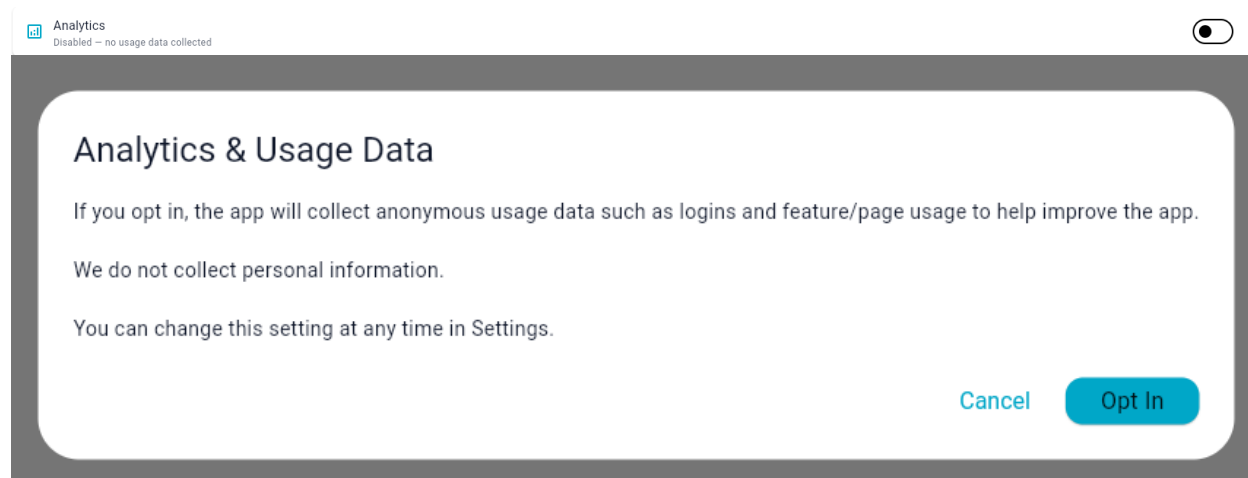
The Observability and Anonymous Feature Analytics implementation aligns with:

- Data minimization principles
- Privacy-by-design standards
- User-controlled consent requirements

Because no personally identifiable information is collected, the analytics framework operates as anonymous telemetry rather than personal data processing.

Figure 8.4.1

Analytics opt-in and toggle



9. Security Considerations

Security considerations for Team B–owned features are designed to align with the CareConnect Software Requirements Specification (SRS) and applicable healthcare privacy constraints. All data stored locally to support offline functionality is encrypted at rest using platform-supported secure storage mechanisms and encrypted local persistence (Drift/SQLite), ensuring protection of sensitive data in offline and at-rest states.

Data synchronization between client applications and backend services is performed exclusively over secure channels using HTTPS with TLS encryption. Authentication and authorization are enforced through centralized identity management and role-based access control (RBAC), ensuring that only authorized users and services can access or modify protected resources. Authentication tokens and credentials are handled securely and are never persisted in plaintext on client devices.

Observability and analytics features are implemented with strict privacy controls in accordance with BNS 7. Telemetry data collected for diagnostics and monitoring explicitly excludes personally identifiable information (PII) and protected health information (PHI). Logs, metrics, and traces are limited to operational and performance data required for troubleshooting and system health monitoring, and are logically separated from application user data paths.

Infrastructure security follows AWS best practices and is standardized through Infrastructure as Code (IaC). AWS CloudFormation templates provision Team B–owned infrastructure using least-privilege IAM roles, secure parameter handling, and environment isolation. Secrets and sensitive configuration values are managed through AWS-native secure services (e.g., IAM, KMS, Parameter Store) and are not embedded in source code or configuration files.

10. Error Handling

The CareConnect system implements a resilient, recoverable error-handling strategy designed to ensure system stability, data integrity, and operational continuity under partial failure conditions. Errors are treated as recoverable states whenever possible, with explicit handling paths defined at the service, synchronization, and infrastructure layers.

Error handling is implemented consistently across backend services, offline synchronization workflows, and infrastructure components, with all failures logged and observable through centralized monitoring.

10.1 Error Classification

Errors are categorized to ensure consistent handling and response:

- **Transient Errors:** Network interruptions, temporary service unavailability, timeout conditions.
- **Recoverable Errors:** Validation failures, synchronization conflicts, partial API failures.
- **Non-Recoverable Errors:** Data corruption, authorization violations, infrastructure misconfiguration.

Each category has defined retry, escalation, or fail-safe behavior.

10.2 Offline and Synchronization Error Handling

Offline-related errors are treated as expected operational conditions:

- **Network Loss:**
When connectivity is lost, the system transitions to offline mode and queues supported operations locally. No user data is discarded due to network failure.
- **Synchronization Failures:**
If synchronization fails due to backend unavailability or partial errors:
 - Failed items remain in the local queue.
 - Automatic retry logic is applied using exponential backoff.
 - Duplicate record creation is prevented through idempotent request handling.
- **Conflict Resolution Errors:**

CareConnect Technical Design Document

Conflicts detected during offline-to-online reconciliation are handled using the conflict resolution strategy defined in the SRS. Conflicts are logged and resolved deterministically without data loss.

10.3 Authentication and Authorization Errors

- **Authentication Failures:**
Failed authentication attempts result in controlled rejection without exposing internal system details. Authentication errors do not affect system availability or data integrity.
- **Authorization Errors:**
Requests lacking proper authorization are rejected with appropriate HTTP status codes and logged for audit purposes.

10.4 Infrastructure and Deployment Errors

Infrastructure-related errors are handled through automated controls:

- **Deployment Failures:**
AWS CloudFormation deployments use rollback mechanisms to restore the last known good state in the event of failure.
- **Configuration Errors:**
Misconfigurations are detected during deployment validation and surfaced through CI/CD pipeline checks.

No manual intervention is required for recovery from standard deployment failures.

10.5 Observability and Error Reporting

All error conditions are observable and traceable:

- Errors are logged with structured metadata.
- Metrics capture error rates and retry behavior.
- Traces link failures across service boundaries.
- Telemetry and analytics explicitly exclude PII and PHI.

Error data supports operational troubleshooting without compromising privacy or compliance requirements.

10.6 Recovery-Oriented System Design

Across all components, the system ensures:

- No single failure causes data loss.
- Partial failures do not cascade across services.
- Recovery paths are automated and repeatable.
- System state remains consistent after recovery.

This approach ensures CareConnect remains reliable, resilient, and production-ready under real-world operating conditions.

11. Performance & Scalability

This section defines performance and scalability considerations for the CareConnect system, with emphasis on backend services, infrastructure design, offline operation, and production readiness. Performance requirements are addressed through architectural choices, infrastructure automation, and operational monitoring rather than UI-level optimizations.

11.1 System Architecture Considerations

CareConnect follows a **service-oriented, backend-driven architecture** with clear separation between client applications, backend services, and supporting infrastructure. Performance-sensitive operations are handled server-side where possible, while offline-capable features are executed locally to reduce dependency on continuous network connectivity.

Key architectural principles include:

- Stateless backend services to support horizontal scaling
- Asynchronous processing for synchronization and telemetry workflows
- Local data persistence to minimize network dependency during offline operation

11.2 Backend Performance Strategy

Backend services are designed to support predictable performance under variable load:

- **API Performance:**
Backend APIs are designed to respond within acceptable latency bounds under normal operating conditions, with timeout and retry mechanisms applied where appropriate.
- **Asynchronous Processing:**
Synchronization tasks, analytics emission, and non-blocking workflows are processed asynchronously to avoid blocking user-facing operations.
- **Idempotent Operations:**
Idempotent request handling ensures retries do not introduce duplicate records or data inconsistencies during transient failures.

11.3 Infrastructure Scalability

Scalability is achieved through infrastructure-as-code and cloud-native design:

- **Infrastructure as Code (IaC):**
All infrastructure is provisioned and managed using AWS CloudFormation, enabling repeatable and scalable deployments.
- **Horizontal Scaling:**

CareConnect Technical Design Document

Backend services are designed to scale horizontally through load-balanced deployments as demand increases.

- **Resource Isolation:**

Logging, analytics, and synchronization services are logically separated to prevent performance degradation in one area from impacting others.

11.4 Offline Performance Characteristics

Offline-first functionality contributes directly to performance and resilience:

- Local data operations execute without backend latency.
- Synchronization is deferred until connectivity is restored.
- Offline queues prevent blocking behavior during network instability.

This approach ensures consistent application behavior even under poor or intermittent connectivity conditions.

11.5 Observability and Performance Monitoring

Performance and scalability are continuously monitored through observability tooling:

- Metrics track request latency, error rates, and synchronization throughput.
- Logs capture performance-related failures and retry behavior.
- Traces enable end-to-end visibility across backend services.

Operational telemetry is used for performance monitoring only and does not include PII or PHI.

11.6 Scalability Boundaries for This Milestone

The following are explicitly **out of scope** for this milestone:

- Predictive or adaptive UI behavior
- Machine learning-driven recommendations
- Wearable device integrations
- Web-based caregiver portals beyond existing functionality

Scalability efforts in this milestone focus on **infrastructure readiness, backend reliability, and offline synchronization behavior**, rather than expansion of feature scope.

12. Testing Strategy

This section describes the system-level validation strategy used to verify correctness, reliability, and resilience of CareConnect backend services and Team B-owned enhancements. Testing focuses on functional correctness, failure handling, and operational behavior rather than UI usability or interaction design.

Each Team B requirement defined in the SRS is associated with one or more acceptance criteria (AC-SB-XXX) that define measurable success conditions. These acceptance criteria are validated through corresponding test cases (TC-XXX) documented in the Software Test Plan. This structure ensures full traceability from requirements to design and verification, and supports

auditable validation of offline functionality, infrastructure provisioning, and observability requirements.

12.1 Validation Categories

System validation activities are grouped into the following categories:

- **Functional Validation:**
Confirms that backend services and offline-capable features behave according to documented requirements.
- **Failure and Recovery Validation:**
Ensures the system handles partial failures, retries, and recovery scenarios without data loss or corruption.
- **Integration Validation:**
Verifies correct interaction between client applications, backend services, synchronization mechanisms, and infrastructure components.
- **Operational Validation:**
Confirms observability, logging, and telemetry function correctly in deployed environments.

12.2 Functional Validation Scenarios

Functional validation focuses on Team B–owned capabilities and selected inherited workflows:

- Offline data creation, update, and persistence
- Synchronization on reconnect
- Conflict resolution during concurrent updates
- Infrastructure provisioning via CloudFormation
- Telemetry and analytics emission without PII/PHI

Validation confirms that each functional requirement defined in the SRS is exercised by at least one test case, as documented in the Requirements Traceability Matrix. Functional validation focuses on Team B–owned capabilities and selected inherited workflows necessary to validate offline synchronization and infrastructure behavior:

12.3 Failure and Recovery Scenarios

Failure scenarios are explicitly tested to ensure system resilience:

- Network interruption during synchronization
- Backend service unavailability
- Partial synchronization failures
- Retry logic under unstable connectivity
- CloudFormation deployment failure and rollback

Expected behavior includes safe retries, preservation of local data, clear logging, and absence of duplicate or corrupted records.

12.4 Observability and Privacy Validation

Operational validation ensures system visibility and compliance:

- Logs, metrics, and traces are generated and accessible through monitoring tools
- Telemetry reflects system health and performance
- Analytics events are anonymous and contain no PII or PHI
- Observability data is logically separated from user data

Privacy validation is treated as a non-functional requirement and enforced consistently across all tested workflows, in alignment with HIPAA-aligned privacy constraints defined in the SRS.

12.5 Accessibility and UI Considerations (Out of Scope)

Detailed usability, cognitive load analysis, biometric authentication flows, SOS handling, and WCAG compliance testing are **out of scope** for this milestone and are addressed in prior coursework or future UI-focused efforts.

For this milestone, accessibility validation is limited to ensuring that backend and synchronization changes do not introduce regressions that would impair existing client-side accessibility behavior.

13. Deployment Strategy

The CareConnect deployment strategy follows a controlled, repeatable, and infrastructure-as-code–driven approach to ensure environment consistency, security, and scalability across development, testing, and production environments.

13.1 Environment Strategy

CareConnect maintains separate environments to support structured development and release management:

- **Development Environment** – Used for feature development and integration testing.
- **Testing/QA Environment** – Used for regression testing, validation of offline synchronization, and observability verification.
- **Production Environment** – Live environment serving end users.

Each environment is provisioned using AWS CloudFormation templates to ensure configuration parity and prevent infrastructure drift.

13.2 Infrastructure Provisioning (BNS 6)

All AWS infrastructure is defined and deployed using AWS CloudFormation. This includes:

- **Application Load Balancer (ALB)** and listener rules.
- **Amazon ECS (Fargate)** Clusters, Services, and Task Definitions.
- **Amazon ECR** (Elastic Container Registry) repositories.

CareConnect Technical Design Document

- **Amazon RDS (PostgreSQL)** instances and Proxy.
- **Amazon S3** buckets for object storage.
- **IAM roles and policies** (Task Execution and Service roles).
- **CloudWatch log groups** and retention policies.

CloudFormation templates are:

- Version-controlled in the project repository.
- Deployed through CI/CD pipelines.
- Reviewed before execution.
- Restricted from manual console modification.

No infrastructure resources may be provisioned or modified outside of CloudFormation templates.

13.3 CI/CD Pipeline

Deployment is automated through a continuous integration and continuous deployment (CI/CD) pipeline that performs:

- **Container Image Build:** Packaging the backend application into Docker images.
- **Image Scanning:** Vulnerability scanning of images prior to push.
- **ECR Registry Update:** Pushing versioned images to the Amazon Elastic Container Registry.
- **CloudFormation Template Validation.**
- **Controlled Stack Deployment:** Updating ECS Task Definitions to trigger rolling deployments of new container tasks.

Infrastructure changes are deployed using CloudFormation Change Sets prior to execution to allow review of proposed modifications.

13.4 Offline Synchronization Deployment Considerations (BNS 5)

Deployment must ensure:

- Backward compatibility of API endpoints used by the OfflineService abstraction
- Idempotent request handling to prevent duplicate data creation
- Logging instrumentation enabled by default
- Sync status fields preserved in database schema

Database schema changes must be applied through controlled migration scripts prior to application deployment.

13.5 Observability Deployment Requirements (BNS 7)

Observability configuration is provisioned automatically during CloudFormation stack deployment, including:

CareConnect Technical Design Document

- CloudWatch log groups
- Metric filters
- X-Ray tracing
- Log retention policies

Telemetry must be enabled in all environments to ensure traceability and debugging capability. PII/PHI scrubbing middleware must be validated during deployment testing.

13.6 Rollback Strategy

In the event of deployment failure:

- **CloudFormation** automatically rolls back to the last stable stack state.
- **ECS Service Scheduler** maintains the previous "Last Known Good" container tasks if new tasks fail health checks (Liveness/Readiness probes).
- **Database schema rollbacks** are executed via controlled migration scripts.

Rollback procedures are tested in the QA environment prior to production release.

14. Known Issues & Limitations

The following limitations are acknowledged within the current MVP scope.

14.1 Offline Feature Scope Limitation (BNS 5)

Offline functionality is limited to the following modules:

- Mood Tracking
- Symptom Tracking
- Medication Tracking
- EVV timestamps
- Calendar events

Messaging, AI features, billing, media uploads, and administrative workflows require active network connectivity. Full dataset mirroring is intentionally excluded to preserve device performance and security.

14.2 Conflict Resolution Simplicity

The MVP implements a deterministic conflict resolution strategy (Last-Write-Wins or server-wins). Advanced reconciliation workflows, manual merge interfaces, and audit-based resolution tools are deferred.

14.3 Observability Scope Limitation (BNS 7)

Telemetry focuses on:

CareConnect Technical Design Document

- Operational logs
- Execution traces
- Anonymous feature usage events

The system does not include:

- Executive analytics dashboards
- Predictive analytics
- ML-driven behavioral recommendations

Telemetry retention and visualization remain AWS-native and do not include custom monitoring tools.

14.4 AWS Dependency

The architecture is AWS-native and leverages a persistent, containerized service model to ensure high availability and reliability. The system is not designed for multi-cloud portability; migration to another cloud provider would require architectural redesign.

Key AWS Dependencies:

- **AWS ECS Fargate:** Serves as the primary compute engine for containerized backend services.
- **Application Load Balancer (ALB):** Manages request routing, SSL termination, and health-based traffic distribution.
- **Amazon RDS (PostgreSQL):** Provides the authoritative relational data store for the platform.
- **Amazon S3:** Utilized for unstructured object storage and telemetry archives.
- **AWS CloudFormation:** The exclusive Infrastructure as Code (IaC) mechanism for resource provisioning.
- **Amazon CloudWatch:** Provides centralized observability, logging, and performance metrics.

14.5 Third-Party Integration Constraints

Integrations with external systems (e.g., wearables, IoT devices, USPS, billing systems) are excluded from the MVP scope and may introduce additional risk, cost, or compliance complexity if implemented in future iterations.

14.6 AI Feature Limitations

AI-driven features are limited to architectural readiness and do not include:

- Clinical diagnosis

CareConnect Technical Design Document

- Automated medical recommendations
- Real-time ASL translation
- Advanced mood detection

All AI outputs must be considered advisory and not medical-grade guidance.

14.7 Performance Boundaries

The system's performance is dependent on:

- AWS service availability
- Mobile device resource constraints
- Network reliability

While serverless scaling supports elasticity, extremely high concurrency beyond projected MVP user volume has not been load-tested at enterprise scale

Technical Design Document

CareConnect

University of Maryland Global Campus SWEN 670 - Software

Engineering Capstone Dr. Mir Assadullah

February 7th, 2026

Contributors: Colyn Mahoney

Zechariah Hillman

Henry Stewart

Obeng Buo

Ja'Lisa Hawkins

Revision History

Date	Version	Description	Author
02/07/2026	1.0	Initial Release	CareConnect Team C

Sign-off Sheet

Role	Name	Signature	Date
Project Manager	Colyn Mahoney		
Team Lead	Colyn Mahoney		
Technical Lead/Architect	Obeng Buo		
Client	Dr. Mir Assadullah		

Table of Contents

Introduction.....	10
1.1 Purpose.....	10
1.2 Intended Audience	10
1.3 Project Documents	10
1.4 Acronyms, Definitions, and Abbreviations	10
1.5 References.....	13
2. System Overview	14
2.1 Summary	14
2.2 System Users and Stakeholders	14
2.3 System Features and Capabilities	15
2.4 System Architecture Overview	17
2.5 System Architecture Design Rationale	20
2.6 In-scope and Out-of-Scope Features.....	23
3. Architecture.....	24
3.1 System Architecture Diagram.....	24
4. Technology Stack.....	28
The Technology Stack for CareConnect is provided below:	28
5. Functional Specifications.....	29
5.1 Onboarding & Authentication.....	29
6. API Design Specification for CareConnect	49
6.1 Overview	49
6.1.2 Architecture Implementation.....	49
6.2 Base Configuration	51
6.2.1 Base URL	51
6.2.2 Request Headers	51
6.2.3 Response Format	52
6.2.4 HTTP Status Codes	53
6.3 Authentication & Authorization APIs	53
6.3.1 User Registration	53
6.3.3 Password Recovery Request.....	54

6.3.4 Password Reset	55
6.3.5 Token Refresh	55
6.3.6 Logout.....	56
6.4 User Management APIs	56
6.4.1 Get User Profile	56
6.4.2 Update User Profile	57
6.4.3 Get User Permissions	58
6.5 Billing & Subscription APIs	59
6.5.1 Get Subscription Plans	59
6.5.2 Create Subscription	60
6.5.3 Get Subscription Status	61
6.5.4 Get Billing History	61
6.5.5 Cancel Subscription.....	62
6.6 Dashboard APIs	63
6.6.1 Get Caregiver Dashboard	63
6.6.2 Get Patient Dashboard.....	64
6.7 Task & Schedule Management APIs	66
6.7.1 Create Task.....	66
6.7.2 Get Tasks	67
6.7.3 Update Task Status	68
6.7.4 Delete Task.....	69
6.8 Health Data APIs	69
6.8.1 Record Health Metric	69
6.8.2 Get Health Metrics	71
6.8.3 Get Health Data Report	72
6.9 AI Integration APIs.....	73
6.9.1 Get AI Health Insights.....	73
6.9.2 Ask AI Query	74
6.9.3 Transcribe Care Notes	74
6.10 Communication APIs.....	80
6.10.1 Send Message	80

6.10.2 Get Conversations	81
6.10.3 Get Messages in Conversation	83
6.10.4 Initiate Audio/Video Call	84
6.10.5 Upload Media	85
6.11 Device Integration APIs.....	86
6.11.1 Connect Wearable Device	86
6.11.2 Complete Wearable Connection.....	87
6.11.3 Get Connected Devices	88
6.11.4 Disconnect Device.....	88
6.12 Notification APIs	88
6.12.1 Get Notifications	88
6.12.2 Mark Notification as Read.....	89
6.12.3 Update Notification Preferences.....	90
6.12.4 Send Notification.....	91
6.12.5 Test Notification Delivery	97
6.13 Gamification APIs	98
6.13.1 Get User Achievements	98
6.13.2 Get Leaderboard	99
6.14 Analytics & Reporting APIs	100
6.14.1 Get Care Analytics	100
6.14.2 Get QuickSight Dashboard URL.....	101
6.14.3 Refresh Analytics Data.....	105
6.15 Social Networking APIs	106
6.15.1 Create Activity Post.....	106
6.15.2 Get Activity Feed	107
6.16 Admin APIs (Role-Based Access Control).....	108
6.16.1 Get All Users (Admin Only)	108
6.16.2 Update User Role (Admin Only).....	109
6.17 Webhook Configuration APIs.....	110
6.17.1 Register Webhook	110
6.18 API Security & Compliance	111

6.18.1 Authentication Flow	111
6.18.3 Data Encryption.....	112
6.18.5 HIPAA Compliance.....	112
6.19 API Versioning & Deprecation.....	113
6.19.1 Versioning Strategy	113
6.19.2 Deprecation Process	113
6.20 Error Codes Reference.....	113
6.21 API Testing Recommendations	113
6.21.1 Unit Testing.....	113
6.22 API Documentation & Developer Resources	114
6.22.1 Interactive API Documentation.....	114
6.23 WebSocket APIs (Real-Time Communication).....	115
6.23.1 WebSocket Connection	115
6.23.2 Supported Actions	118
6.23.3 Supported Channels and Events	119
6.23.4 Connection Management.....	122
6.23.5 Error Handling.....	123
6.23.6 Performance and Scalability.....	124
6.23.7 Security Considerations.....	124
6.23.8 Testing Criteria.....	125
6.23.9 AWS Architecture Notes	126
7. Data Models.....	127
7.1 Scope.....	127
7.2 Conceptual Data Model	127
7.2.1 Key Entities	127
7.2.2 High-Level Relationships.....	128
7.3 Logical Data Model	128
7.3.1 Core Tables.....	129
7.3.2 RBAC & Tenancy	129
7.3.3 Billing.....	129
7.3.4 Scheduling & Notifications	129

7.3.5 Care Management (Appointments & Medications).....	130
7.3.6 Health Tracking	130
7.3.7 Wearables	130
7.3.8 Communication & Media	130
7.3.9 Audit & Transcriptions	131
7.3.10 Gamification	131
7.4 Non-Relational Stores	131
7.4.1 DynamoDB Collections.....	131
7.4.2 Timestream	131
7.4.3 Redis (ElastiCache)	131
7.5 Data Dictionary	131
7.5.1 Tenant	132
7.5.2 User.....	132
7.5.3 Role.....	133
7.5.4 Permission	133
7.5.5 Role Permission.....	134
7.5.6 User Role	134
7.5.7 Patient Profile	134
7.5.8 Caregiver Profile	135
7.5.9 Caregiver Patient	135
7.5.10 Plan	136
7.5.11 Subscription.....	136
7.5.12 Payment Method.....	137
7.5.13 Payment	137
7.5.14 Task Template	138
7.5.15 Task	139
7.5.16 Notification.....	139
7.5.17 Appointment	140
7.5.18 Medication	140
7.5.19 Patient Medication.....	141
7.5.20 Symptom.....	141

7.5.21 Patient Symptom	142
7.5.22 Symptom Log	142
7.5.23 Meal Question	142
7.5.24 Patient Meal Question	143
7.5.25 Meal Entry	143
7.5.26 Mood Entry	144
7.5.27 Wearable Device.....	144
7.5.28 Wearable Metric	144
7.5.29 Message	145
7.5.30 Media File.....	145
7.5.31 Webhook Subscription	146
7.5.32 Audit Log.....	146
7.5.33 Transcription.....	147
7.5.34 Telehealth Session	148
7.5.35 Achievement.....	148
7.5.36 User Achievement	148
7.5.37 Location.....	149
7.6 Physical Data Model	149
7.6.1 Schemas & Convention	149
7.6.2 Enum Types.....	150
7.6.3 Table DDL.....	150
7.7 Physical Design Considerations.....	153
7.8 AWS PostgreSQL Deployment	153
7.9 Security & Compliance.....	154
7.10 Backup & Recovery	154
8. UI/UX Design	154
8.1 User Roles & Management (Inherited from CareConnect TDD V3)	154
8.1.1 Caregiver Profile	155
8.1.2 Patient Profile	156
8.1.3 Multi-Caregiver Support	158
8.1.4 Profile Linking.....	158

8.1.5 Access Control & Permissions	161
8.1.6 Family “Read-Only” Access	162
9. Security Considerations	164
9.1 Data Privacy & HIPAA/GDPR Compliance	164
9.2 Authentication & Authorization (RBAC)	164
9.3 Secure Data Transmission	164
9.4 API & Application Security	164
9.5 Infrastructure / Hosting Security (Fargate)	164
9.6 Monitoring, Logging, and Audit Trails	164
9.7 Mobile / Client Security	164
9.8 AI / Third-Party Integrations	165
9.9 Security Testing	165
10. Error Handling	165
10.1 Client / UI	165
10.2 Server / API	165
10.3 Monitoring, Logging, and Audit	165
10.4 Testing	166
11. Performance & Scalability	166
11.1 Objectives	166
11.2 Key Metrics	166
11.3 Scalability Strategy	167
11.4 Monitoring and Capacity Management	167
11.5 Bottleneck Mitigation	168
11.6 Review Cadence	168
12. 169	
Testing Strategy	169
13. Deployment Strategy	171
13.1 Deployment Preparation	171
13.2 CI/CD Flows	171
14. Known Issues & Limitations	172

1. Introduction

1.1 Purpose

The Purpose of this document is to detail the technical design of the tree new requirements Team C has been given to update the CareConnect application.

1.2 Intended Audience

The intended audience for this Technical Design Document is all current and future stakeholders engaged in the maintenance and modification of the CareConnect application. These groups include the clients, project manager, development team C, SWEN 670 Professor and volunteer Mentors.

1.3 Project Documents

Table 1

List of Project Documents for CareConnect

Table 1 Documents

Document	Version	Date
Project Plan	1.0	01/24/2026
Software Requirements Document	1.0	01/24/2026
Technical Design Document	1.0	02/07/2026
Software Test Plan	1.0	02/07/2026
Deployment and Operations Guide	1.0	TBD
Programmer Guide	1.0	TBD
User Guide	1.0	TBD
Test Report	1.0	TBD

Note. Documents with date labeled TBD have not been delivered yet.

1.4 Acronyms, Definitions, and Abbreviations

Table 2 Definitions

Term	Definition
AI (Artificial Intelligence)	A system capable of doing human like tasks involving pattern recognition, speech recognition, generating content

Amazon Web Services (AWS)	A suite of different cloud services offered by amazon including different database and server services needed for CareConnect.
Android OS	An open-source mobile operating system developed by Google.
App Runner	A serverless environment to run applications offered by AWS
CareConnect	A web/mobile application developed by SWEN 670 cohorts to assist caregivers and patients with healthcare management.
Caregiver	An individual who assists with the care of an individual with some extraordinary needs.
Care Receiver	An individual with some extraordinary healthcare needs.
Dart Tool	Programming language and toolset used to create Flutter applications.
Fargate	A serverless environment to run applications offered by AWS
Flutter	An open-source UI software development kit (SDK) created by Google, capable of building natively compiled applications for mobile, web, and desktop from a single codebase.
HIPAA (Health Insurance Portability and Accountability Act)	U.S. regulation intended to protect an individual's healthcare related information
Roll based Access Control	A way of controlling which screens and data a user has access to according to the roll they are assigned in the system.
iOS	A mobile operating system developed by Apple for iPhone
Unit testing	Refers to automated testing used in software development to help catch errors in compile time.

User Interface (UI)	The visual and interactive components of the application which allows users (Caregivers and receivers) to interact with the system.
User Experience (UX)	The overall experience and satisfaction a user has when interacting with the application, including ease of use, efficiency, and accessibility.

(As cited in the CareConnect SRS, Aug 30, 2025)

Table 3 Acronyms

Acronyms	Definitions
AC-###	Acceptance criterion tested for a requirement/use case
AI	Artificial Intelligence
API	Application Programming Interface
App	Application (mobile or web-based software)
AWS	Amazon Web Services
BNS	Business Need Statement
DB	Database
GUI	Graphical User Interface
HIPAA/GDPR	U.S. healthcare privacy law / EU privacy regulation
IoT	Internet of Things
PHI/PII	Protected Health Information / Personally Identifiable Information
PMP	Project Management Plan
QA	Quality Assurance

RAM	Random Access Memory
RBAC	Role-Based Access Control
RDS	Relational Database Service
REQ-###	Requirement ID
REST	Representational State Transfer (API standard)
SRS	Software Requirements Specification
TC-###	Test case that verifies an AC
UC-###	Use case
UI	User Interface
UX	User Experience
SLA/SLO	Service Level Agreement/Objectives
MA	Medicaid Number
EOR	Employer of Record

(As cited in the CareConnect SRS, Aug 30, 2025)

1.5 References

- Requirements document provided by stakeholder mentors.
- Software Requirements Specification document from Summer/Fall 2025 cohort https://umgc-cappms.azurewebsites.net/download/d342ee6d-bbe6-49ee-aa03-75a359b26bf8----SWEN670_CareConnect_Fall2025_SRS.pdf
- AWS Fargate <https://aws.amazon.com/fargate/>
- AWS App Runner <https://aws.amazon.com/apprunner/>
- An introduction to unit testing <https://docs.flutter.dev/cookbook/testing/unit/introduction>
- JUnit User Guide <https://docs.junit.org/6.0.2/overview.html>
- Technical Design Document from Summer/Fall 2025 cohort https://umgc-cappms.azurewebsites.net/download/1c45811b-6d19-415a-8b17-02205cf5c942----SWEN670_CareConnect_TechnicalDesignDocument_v3.0.pdf

2. System Overview

2.1 Summary

CareConnect is a caregiving coordination platform that enables patients and caregivers to securely manage care routines, communication, and care-related information through a cross-platform application. The system supports a separation of concerns between the presentation layer (client applications), the service layer (API and backend services), and the data layer (persistent storage). This architecture enables the platform to evolve while maintaining service reliability, security boundaries, and predictable operational behavior.

For the Spring 2026 enhancement effort, CareConnect's primary focus is **technical modernization and operational hardening** rather than introducing new end-user features. The technical design emphasizes the following scope items aligned to the current SRS and PMP:

- **Cloud hosting and deployment modernization** by deploying the application using a managed AWS container runtime (e.g., AWS Fargate or an equivalent managed option), improving scalability, availability, and operational consistency.
- **Role-Based Access Control (RBAC) framework** introduction so that authorization is explicit, testable, and enforceable across API endpoints and application workflows.
- **Improved automated testing coverage** through the addition and expansion of unit and integration testing (including JUnit for the backend and Dart test coverage where applicable), increasing confidence in releases and reducing regression risk.

2.2 System Users and Stakeholders

This section defines the actors that interact with CareConnect and maps them to access expectations that will be enforced by the RBAC framework.

2.2.1 Primary Users (End Users)

Patient

- Receives caregiving support and completes assigned care activities.
- Interacts with patient-facing features relevant to care routines and personal status tracking.
- Authorization expectations: patient access is limited to the patient's own data and permitted workflows; patients cannot access caregiver-only or administrative capabilities.

Caregiver

- Coordinates care activities and monitors patient progress.
 - Creates and manages care routines, assigns tasks, and reviews patient-reported
-

updates.

- Authorization expectations: caregiver access includes patient-specific resources only for patients they are assigned to; caregivers cannot perform system-wide administrative actions unless explicitly granted.

2.2.2 Administrative / Operational Users

System Administrator (or Operational Admin)

- Manages operational aspects of the platform such as user administration, configuration, and monitoring.
- May be responsible for environment configuration and support workflows.
- Authorization expectations: admin privileges are tightly controlled and audited; “admin” is not an implicit capability and must be granted via RBAC.

2.2.3 Supporting Stakeholders

Product Owner / Client Stakeholder

- Defines scope priorities and approves design direction for the semester deliverables.

Development Team (Engineering + QA)

- Implements RBAC, testing, and AWS deployment changes.
- Validates quality via automated tests and controlled releases.

Course Stakeholders (Professor / Mentors)

- Review deliverables for completeness, feasibility, and alignment to documented requirements and scope constraints.

2.3 System Features and Capabilities

This section describes the capabilities that are in-scope for Spring 2026 and explicitly ties them to the technical design goals. The intent is to show what the system must be capable of doing technically (deploy, authorize, test, secure), rather than re-listing all end-user functional modules from earlier versions.

2.3.1 Cloud Hosting and Deployment Capability (Managed AWS Runtime)

CareConnect shall support a production-ready hosting model using a managed AWS container environment. The system design must enable:

- **Containerized application deployment** with consistent runtime configuration across environments.
 - **High availability** through multiple task instances and health-checked service routing.
 - **Scalability** through horizontal scaling policies based on utilization metrics.
-

- **Environment separation** (at minimum: dev/test and production), with configuration managed via environment variables and secure parameter storage.
- **Repeatable deployments** via an automated delivery mechanism (CI/CD) that promotes tested artifacts across environments.

This capability is directly aligned to the product owner requirement to migrate the system to AWS managed hosting service Fargate and to the PMP's focus on deployment/infrastructure deliverables.

2.3.2 Role-Based Access Control (RBAC) Capability

CareConnect shall implement an RBAC framework that ensures authorization is:

- **Explicit:** access decisions are based on defined roles and permissions rather than implicit UI behavior.
- **Centralized and consistent:** enforced at the API/service layer so that all clients (web/mobile) receive the same authorization behavior.
- **Least-privilege:** users only receive permissions required for their role and assigned relationships (e.g., caregiver-to-patient assignment).
- **Auditable:** authorization events and denied access attempts can be logged for troubleshooting and security review.

2.3.3 Automated Testing and Quality Capability (JUnit + Dart)

CareConnect shall support a test strategy that increases confidence in system stability and reduces regression risk. The design must enable:

- **Backend unit testing** (JUnit) for service and business logic components.
- **Testable authorization** (RBAC) with repeatable test cases for role/permission evaluation and access boundary checks.
- **Client-side testing** (Dart where applicable) for critical UI logic and client behaviors that are part of the maintained codebase.
- **CI test execution** so tests run automatically on pull requests/merges and prior to deployment promotion.

This capability maps to the semester scope item "Fix/Add JUnit tests + Add Dart tests for existing code base."

2.3.4 Security and Operational Hardening Capability

To support production hosting and RBAC enforcement, CareConnect shall implement baseline operational security controls, including:

- Secure secret handling (no hardcoded secrets; use managed secret/config storage),
 - Least-privilege access for services and deployment roles,
-

- Encrypted transport for external traffic (TLS),
- Centralized logging sufficient to diagnose deployment failures, authorization denials, and runtime errors.

2.4 System Architecture Overview

2.5.1 Architectural Summary

CareConnect is designed as a multi-tier application with clear separation between:

- **Client Layer** (web/mobile UI)
- **Service Layer** (API + backend services)
- **Data Layer** (persistent storage)
- **Supporting Platform Services** (authentication/authorization, secrets, logs/metrics, CI/CD)

2.5.2 Logical Architecture

At a logical level, CareConnect is composed of the following components:

Client Layer

Responsibilities

- Presents application UI and collects user input.
- Communicates with backend via HTTPS API.
- Includes client-side logic that is covered by Dart tests (where applicable to maintained code).

Interfaces

- HTTPS requests to CareConnect API endpoints.
- Authentication token (e.g., JWT or session token) attached to requests.

Service Layer (API + Application Services)

Responsibilities

- Exposes REST endpoints for CareConnect workflows.
- Enforces authentication and RBAC authorization for every protected operation.
- Implements business logic and orchestration of data access.
- Emits logs/metrics for operational observability.

Internal Modules (design-time view)

- **AuthN module:** validates identity (token/session verification).
 - **RBAC module:** evaluates permissions and authorization rules.
-

- **Application services:** domain logic (care routines, tasks, profiles, etc. existing functionality).
- **Data access layer:** repositories/DAOs for DB access.

Data Layer

Responsibilities

- Stores user profiles, role assignments, permissions mappings, application data.
- Supports transactional consistency requirements for role assignment and authorization-critical operations.

Key data concerns

- Access control tables and relationships must be designed to support RBAC efficiently (e.g., user-role, role-permission, user-resource relationships).

2.5.3 Platform Services

Responsibilities

- Secrets/config management.
- Audit logging and request tracing.
- Monitoring, alerting, and operational dashboards.
- CI/CD to build, test, and deploy.

2.5.4 Physical Deployment Architecture (AWS)

This section documents the Spring 2026 target deployment on AWS using a managed container runtime.

Core AWS Components

Compute / Container Runtime

- ECS on AWS Fargate runs containerized CareConnect services without managing EC2 hosts.
- Services run as Tasks in one or more ECS Services to support scaling and self-healing.

Container Registry

- Amazon ECR stores versioned container images produced by the CI pipeline.

Networking

- VPC with segmented subnets:
 - Public subnets for internet-facing components (e.g., ALB)
 - Private subnets for application tasks and databases
 - Security Groups restrict inbound/outbound traffic by least privilege.
-

Ingress

- Application Load Balancer (ALB) routes HTTPS traffic to backend services.
- ALB performs health checks and supports multi-AZ routing.

Database

- Managed relational DB (Amazon RDS) hosts application data and RBAC tables.
- Database is deployed in private subnets; inbound only from application service security group.

Secrets & Configuration

- AWS Secrets Manager (or SSM Parameter Store) stores DB credentials, API secrets, and sensitive configuration.
- Applications fetch secrets at startup (or via task role access).

Logging & Monitoring

- Amazon CloudWatch Logs collects application logs per service/task.
- CloudWatch Metrics/Alarms monitor CPU/memory utilization, error rates, and ALB target health.
- Optional: distributed tracing integration if supported by current stack.

IAM

- Task execution role (pull image, write logs)
- Task role (read secrets/config, access AWS services as needed)
- CI role (push images, deploy service updates)

2.5.5 Deployment Topology

Environment Segmentation

CareConnect deployment supports at minimum:

- **Development/Test environment**
 - Frequent deployments
 - Lower-cost scaling settings
 - Used for integration testing and validation
- **Production environment**
 - Higher availability configuration
 - Stricter access controls and auditing
 - Separate secrets and database instance(s)

Availability and Scaling

Availability

- Services run across multiple Availability Zones by deploying tasks across subnets in at least two AZs.
- ALB routes traffic only to healthy targets.
- ECS Service ensures failed tasks are replaced automatically.

Scaling

- Horizontal scaling via ECS Service desired count and optional autoscaling:
 - CPU utilization thresholds
 - Memory utilization thresholds
 - Request-based scaling (optional, via ALB request metrics)

2.5.6 CI/CD and Release Flow (High-Level)

CareConnect will follow a release flow that ensures tests gate deployments:

1. Code commit / pull request
2. Automated build
3. Automated unit tests
 - a. JUnit for backend
 - b. Dart tests where applicable
4. Container image build
5. Push image to ECR
6. Deploy to dev/test
7. Smoke checks / integration checks
8. Promote to production

2.5.7 Architecture Boundaries

To stay aligned with the current semester scope, this TDD does not redesign end-user feature modules from prior semesters (e.g., messaging/calling/SOS). Those may exist in the legacy codebase, but Spring 2026 design effort focuses on:

- The deployment platform (AWS managed hosting)
- RBAC enforcement across the application stack
- Automated testing improvements
- Operational security and observability required to safely run the system

2.5 System Architecture Design Rationale

2.5.1 Architectural Objectives

The CareConnect system architecture for Spring 2026 was selected to satisfy the following non-

functional objectives derived from the Software Requirements Specification (SRS) and Project Management Plan (PMP):

- Secure and centralized enforcement of authentication and role-based access control (RBAC)
- Deployment within a controlled network boundary with private application and data tiers
- High availability and fault tolerance using managed cloud services
- Predictable and cost-optimized scaling behavior
- Support for automated testing, deployment, and monitoring

To meet these objectives, a managed AWS container-based architecture was selected, with AWS Fargate chosen as the primary runtime.

2.5.2 Managed Container Hosting Decision

CareConnect requires a hosting model that supports a traditional three-tier architecture consisting of:

- Internet-facing ingress
- Private application services enforcing AuthN and RBAC
- Private persistent data storage

Two AWS managed container options were evaluated:

- **Amazon ECS on AWS Fargate**
- **AWS App Runner**

Both services remove the need for server management; however, they differ significantly in networking control, scaling behavior, and cost structure.

2.5.3 Technical Rationale: Fargate vs. App Runner

Networking and Security Control

AWS Fargate allows CareConnect services to run entirely within private VPC subnets, protected by security groups and fronted by an Application Load Balancer (ALB). This enables:

- Explicit isolation between public and private tiers
- Least-privilege network access to the database
- Clear enforcement boundaries for RBAC and administrative operations

App Runner, while capable of VPC connectivity, is optimized for simplified web services and offers less granular control over classic multi-tier network segmentation. Given CareConnect's RBAC-heavy backend and private database requirements, Fargate better aligns with the security and architectural constraints of the Spring 2026 design.

Scaling Model Considerations

App Runner scales based on **request concurrency per container**, requiring configuration of:

- Maximum concurrent requests per container
- Minimum provisioned container instances

This model introduces complexity and potential inefficiency for workloads that involve RBAC evaluation and database access, where CPU and memory utilization—not raw request count—are the primary scaling drivers.

Fargate scales based on **resource utilization (CPU and memory)** and service health, which more closely matches the runtime characteristics of CareConnect’s backend services.

2.5.4 Cost Optimization Analysis

To support an evidence-based decision, a cost comparison was performed using a conservative and defensible workload profile appropriate for a Spring 2026 academic deployment.

Assumptions (Compute Only)

- 2 running instances for high availability
- Each instance: **0.5 vCPU, 1 GB RAM**
- Always-on backend API
- 486.68 hours per month
- Costs exclude database, load balancer, and data transfer (common to both options)

Table 2.5-1. Monthly Compute Cost Comparison

Service	Workload Model Used	Total Runtime (hrs/month)	Hourly Compute Rate	Monthly Compute Cost
AWS Fargate	2 tasks/day × 4 hrs/task	243.33	~\$0.04048/hr	\$6.00 (<i>AWS calc</i>)
AWS App Runner	Equivalent active runtime	243.33	\$0.07800/hr	\$13.20

2.5.5 Cost Optimization Capabilities

In addition to lower baseline compute costs, Fargate provides several optimization mechanisms aligned with Spring 2026 requirements:

- Compute Savings Plans for predictable workloads (up to ~50% reduction)
-

- Fargate Spot for non-production or test environments (up to ~70% reduction)
- Resource-based autoscaling tuned to actual CPU/memory demand
- Independent scaling of environments (dev/test vs production)

App Runner requires payment for minimum provisioned containers, even during idle periods, and relies on concurrency-based scaling that may over-provision for RBAC- and database-intensive workloads.

Table 2.5-2. Architectural Trade-off Summary

Category	AWS Fargate	AWS App Runner
Network isolation	Full VPC control	Limited / abstracted
RBAC enforcement fit	Strong (private backend)	Moderate
Scaling trigger	CPU / memory	Request concurrency
Baseline idle cost	Lower	Higher (min containers)
Cost optimization options	Savings Plans, Spot	Limited
Suitability for CareConnect	High	Moderate

2.5.6 Final Decision

AWS Fargate was selected as the managed container runtime for CareConnect because it:

- Better supports private, RBAC-enforced backend services
- Aligns with standard multi-tier AWS architectures
- Provides more predictable and cost-effective scaling behavior
- Offers superior long-term cost optimization mechanisms
- Reduces operational risk while maintaining architectural clarity

This decision ensures that CareConnect meets its Spring 2026 non-functional requirements for security, scalability, maintainability, and cost efficiency without introducing unnecessary platform constraints.

2.6 In-scope and Out-of-Scope Features

2.6.1 In-Scope Features

The following features are within the scope of the Spring 2026 CareConnect system design:

- Deployment of the existing CareConnect application using a managed AWS container
-

hosting solution.

- Container-based execution of backend services using AWS Fargate.
- Use of an Application Load Balancer to route external traffic to backend services.
- Deployment of application services and data storage within a Virtual Private Cloud (VPC).
- Separation of public and private network subnets.
- Implementation of Role-Based Access Control (RBAC) to manage authorization.
- Centralized enforcement of authentication and authorization at the backend service layer.
- Secure storage and retrieval of application secrets and configuration values.
- Integration of automated backend testing using JUnit.
- Limited client-side testing related to authentication and authorization behavior.
- Automated build and deployment using a CI/CD pipeline.
- Centralized logging and basic monitoring of application services.

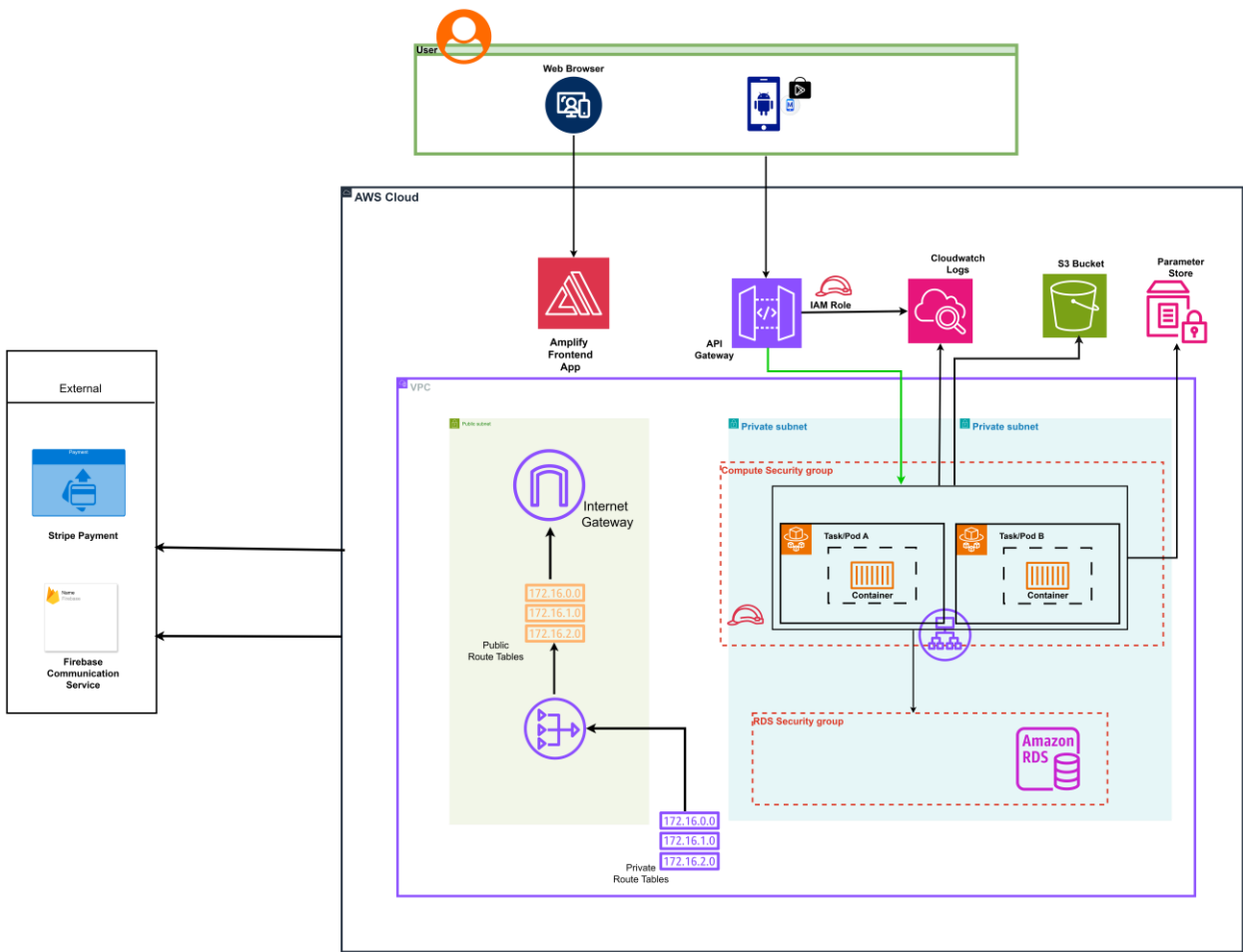
2.6.2 Out-of-Scope Features

The following features are explicitly out of scope for the Spring 2026 system design:

- Addition of new functional features or user-facing capabilities.
- Modification of existing caregiving workflows or business logic.
- User interface redesign or visual enhancements.
- Development of new client platforms.
- Migration to Kubernetes or container orchestration platforms beyond AWS Fargate.
- Multi-region deployment or disaster recovery implementations.
- Performance tuning beyond baseline operational requirements.
- Load testing or stress testing for production-scale traffic.
- Implementation of authorization models beyond RBAC.
- Integration with external identity providers or single sign-on solutions.
- Formal security compliance certifications or audits.

3. Architecture

3.1 System Architecture Diagram



3.2 Architectural Components A list of the architectural components for CareConnect is provided in Table 3.

Table 3 Architectural Components

Component	Description	Primary Responsibilities	Design Rationale
Client Applications	Web and mobile clients used by end users to interact with the CareConnect system.	Capture user input, initiate requests, display responses.	Clients remain thin and do not enforce authorization logic, reducing attack surface and ensuring centralized control.
AWS Amplify Frontend	Managed service hosting the frontend application.	Serve static frontend assets and manage frontend deployment.	Offloads frontend hosting concerns and integrates cleanly with

			AWS-managed services.
API Gateway	Managed API entry point for backend services.	Route client requests, provide controlled ingress, integrate with backend services.	Prevents direct access to backend services and provides a consistent API access layer.
Application Load Balancer (ALB)	Layer 7 load balancer routing traffic to backend containers.	Distribute traffic across healthy backend tasks and perform health checks.	Enables high availability and supports horizontal scaling of backend services.
Amazon ECS on AWS Fargate	Managed container runtime for backend services.	Execute backend application containers, scale tasks, and enforce runtime isolation.	Removes server management overhead while supporting private subnet deployment and security controls.
Backend Application Services	Containerized CareConnect backend services.	Implement business logic, authentication, and RBAC authorization checks.	Centralizes RBAC enforcement and ensures consistent authorization across all endpoints.
Amazon VPC	Isolated virtual network hosting backend infrastructure.	Provide network isolation and controlled routing.	Ensures backend and database resources are not exposed to the public internet.
Public Subnet	Subnet containing internet-facing components.	Host routing components such as the internet gateway.	Limits exposure to only required entry points.
Private	Subnets hosting	Run application	Prevents direct

Subnets	backend services and database.	containers and database services securely.	public access to sensitive system components.
Security Groups	Virtual firewalls controlling inbound and outbound traffic.	Enforce least-privilege network access between components.	Adds an additional layer of protection beyond application-level RBAC.
Amazon RDS	Managed relational database service.	Store application data, including user roles and permissions.	Provides durable, managed data storage while remaining isolated in private subnets.
IAM Roles	AWS identity roles assigned to backend services.	Grant controlled access to AWS services such as S3 and CloudWatch.	Separates infrastructure access control from application-level RBAC.
AWS Parameter Store	Managed configuration and secrets service.	Store and retrieve sensitive configuration values.	Eliminates hard-coded credentials and improves security posture.
Amazon S3	Object storage service.	Store application artifacts or files.	Provides scalable, durable storage with controlled access via IAM.
Amazon CloudWatch	Monitoring and logging service.	Collect logs, metrics, and operational data.	Enables observability, troubleshooting, and system monitoring.
External Services	Third-party services such as payment and notification providers.	Provide external functionality via controlled integrations.	External interactions occur through managed interfaces without exposing

internal
infrastructure.

4. Technology Stack

The Technology Stack for CareConnect is provided below:

Layer	Technology used	Purpose
Frontend	Flutter (Dart)	Cross-platform client for mobile/web; supports Dart-based testing for maintained UI/client logic.
Backend	Java + Spring Boot	RESTful services implementing business logic; central enforcement point for RBAC (authorization) and AuthN validation.
Ingress / Routing	Application Load Balancer (ALB)	Internet-facing HTTPS entry point; health checks and traffic routing to containers.
Compute / Runtime	Amazon ECS on AWS Fargate	Managed container runtime to deploy CareConnect without managing servers; supports scaling and task self-healing.
Container Registry	Amazon ECR	Stores versioned container images built by CI.
Networking	Amazon VPC (public/private subnets) + Security Groups	Network isolation and least-privileged traffic controls (ALB → service; service → DB).
Database	Managed RDBMS (Amazon RDS)	Persistent application data + RBAC tables (users/roles/permissions mappings).
Authentication	Spring Security (JWT/session) or Amazon Cognito (TBD)	Establishes identity and session/token validation for protected API access.
Authorization (RBAC)	RBAC Policy Module (in-service)	Centralized role/permission evaluation for all protected operations; supports auditable allow/deny decisions.

Secrets & Config	AWS Secrets Manager or SSM Parameter Store	Secure storage for DB credentials and sensitive configuration; retrieved at runtime via IAM task role.
Logging	Amazon CloudWatch Logs	Centralized collection of application/service logs for troubleshooting and audit support.
Monitoring / Alerts	Amazon CloudWatch Metrics + Alarms	Health visibility (CPU/memory, target health, error rates); alerting for failures and anomalies.
CI/CD	GitHub Actions	Automates build, unit tests (JUnit/Dart), container build, image push to ECR, and deployment updates.
Infrastructure as Code	Cloudformation	Reproducible provisioning of AWS resources (ECS, ALB, VPC, IAM, etc.).
Backend Testing	JUnit	Automated unit/integration testing for service logic and RBAC enforcement.
Client Testing	Dart test framework	Automated testing for client-side logic and request/response handling where applicable.
Security Compliance	HIPAA & GDPR Compliant	Ensures every component is secure and compliant
File & Media Storage	Amazon S3	Stores images, videos, PDFs

5. Functional Specifications

5.1 Onboarding & Authentication

5.1.1 User Registration **Description:**

The user registration process enables new users (caregivers, patients, or family members) to create an account in the CareConnect system. Registration is performed using an email address and password. Upon successful registration, the system creates a user profile, assigns an initial role, and establishes credentials for future authentication.

Actors:

Primary: End User (Caregiver, Patient, or Family Member)

Secondary: CareConnect Authentication Service

Flow:

The user launches the CareConnect application and navigates to the registration option from

the login screen.

The system displays the registration form requesting an email address and password.

The user enters a valid email address and a password that meets defined security requirements.

The user submits the registration form.

The system validates the input (email format, password complexity, and uniqueness of the email).

Upon successful validation, the system creates a new user account in the database and assigns a default role based on registration context.

The system establishes an authenticated session for the user.

The user is redirected to the appropriate dashboard based on their assigned role.

The system may send a confirmation email to verify the user's email address.

Inputs/Outputs:

Inputs: Email address, password

Outputs: New user record stored in the database, active authentication session/token, optional email confirmation message

Edge Cases:

- Duplicate Email: If the email address is already registered, the system displays an error indicating that the email is in use.
- Invalid Email Format: If the email format is invalid, the system prompts the user to correct it.
- Weak Password: If the password does not meet complexity requirements, the system displays a validation error.
- System or Network Failure: If a network or server error occurs during registration, the system displays an error message and does not create a partial account.

5.1.2 User LoginDescription:

The user login process allows registered users to authenticate and access the CareConnect system using their email address and password.

Actors:

Primary: End User

Secondary: CareConnect Authentication ServiceFlow:

1. The user navigates to the login screen.
 2. The system displays fields for email address and password.
 3. The user enters their credentials and submits the form.
 4. The system validates the credentials against stored user records.
 5. Upon successful authentication, the system creates a secure session.
 6. The user is redirected to the appropriate role-based dashboard.
-

Inputs/Outputs:

Inputs: Email address, password

Outputs: Authenticated user session/token, access to role-based dashboard **Edge Cases:**

- Invalid Credentials: Incorrect email or password results in an error message.
- Account Lockout: Repeated failed login attempts may temporarily lock the account.
- Inactive Account: If the account is disabled, access is denied with an appropriate message.
- Connectivity Issues: Network errors prevent login and prompt the user to retry.

UI/UX:

- Password input shall be masked.
- Error messages shall not reveal whether the email or password was incorrect.

5.1.3 Password Recovery

Description:

The password recovery process allows users to securely reset their password if they forget their login credentials.

Actors:

Primary: End User

Secondary: CareConnect Authentication Service, Email Service

Flow:

1. The user selects the “Forgot Password” option on the login screen.
2. The system prompts the user to enter their registered email address.
3. The user submits the email address.
4. The system verifies that the email exists in the system.
5. The system sends a password reset link to the user’s email address.
6. The user opens the link and enters a new password.
7. The system validates and updates the password.
8. The user is notified that the password reset was successful.

Inputs/Outputs:

Inputs: Email address, new password

Outputs: Updated password stored securely, Confirmation message

Edge Cases:

Unregistered Email: If the email does not exist, the system displays a generic error

message.

Expired Reset Link: The reset link has expired and must be reissued.

Weak New Password: The new password does not meet complexity requirements.

Email Delivery Failure: Reset email fails to send; user is prompted to retry.

UI/UX:

- Password reset links shall be time-limited for security.
- Confirmation messages shall clearly indicate success or failure

5.2 Billing & Subscription Management

5.2.1 Subscription Selection & Activation

Description:

The subscription selection and activation process allows caregivers to choose a pricing plan and activate access to subscription-based CareConnect features. Payments are processed securely through a third-party payment provider (Stripe), and subscription status determines feature availability.

Actors:

Primary: Caregiver

Secondary: CareConnect Billing Service, Payment API

Flow:

1. The caregiver logs into the CareConnect application.
2. The caregiver navigates to the subscription or billing section.
3. The system displays available subscription plans (e.g., monthly, per-patient).
4. The caregiver selects a subscription plan.
5. The system redirects the caregiver to a secure payment form.
6. The caregiver enters payment details and submits the payment.
7. The system sends the payment request to Stripe for processing.
8. Stripe returns a success or failure response.
9. Upon successful payment, the system activates the subscription and enables subscription-based features.
10. The caregiver receives a confirmation notification.

Inputs/Outputs:

Inputs: Selected subscription plan, payment details

Outputs:

- Active subscription record
- Payment confirmation
- Updated access permissions

Edge Cases:

- **Payment Failure:** Payment is declined or fails; the system displays an error and does not activate the subscription.
- **Invalid Payment Information:** Incorrect or incomplete payment details prompt the user to correct the input.
- **Network Failure:** Connectivity issues during payment processing result in an error message and no subscription change.

UI/UX:

- Pricing information shall be clearly displayed before payment.
- Payment forms shall comply with security and privacy standards.
- Success and failure messages shall be clear and actionable.

5.2.2 Billing Notifications & History**Description:**

This process allows caregivers to view billing history and receive notifications related to subscription status and billing events.

Actors:

Primary: Caregiver

Secondary: CareConnect Billing Service

Flow:

1. The caregiver navigates to the billing history section.
2. The system retrieves and displays past invoices and payment records.
3. The system automatically sends notifications for upcoming charges, successful payments, or failed transactions.

Inputs/Outputs:

Inputs: Caregiver billing request

Outputs: Billing history display, billing notifications

Edge Cases:

- **Missing Records:** Billing history is unavailable due to a system error.
 - **Notification Delivery Failure:** Notification fails to send; the system logs the failure.
-

UI/UX:

- Billing records shall be presented in a clear, chronological format.

5.4 Dashboards

5.4.1 Caregiver Dashboard

Description

The caregiver dashboard provides a centralized view of assigned patients, tasks, schedules, notifications, and summarized health data to support daily care management.

Actors:

Primary: Caregiver

Secondary: CareConnect Dashboard Service

Flow:

1. The caregiver logs into the CareConnect system.
2. The system authenticates the caregiver and identifies the caregiver role.
3. The system loads the caregiver dashboard.
4. The dashboard displays patient lists, assigned tasks, upcoming schedules, alerts, and summarized health metrics.
5. The caregiver selects a patient or task to view additional details or perform actions.

Inputs/Outputs:

Inputs: Caregiver dashboard request

Outputs: Display of patient information, Task lists and alerts, Health data summaries

Edge Cases:

- No Assigned Patients: The dashboard displays an empty state with guidance.
- Data Sync Delay: Health data is temporarily unavailable or outdated.
- System Error: Dashboard fails to load due to backend issues.

UI/UX:

- Critical alerts shall be visually prioritized.
- Dashboard layout shall be responsive and accessible.

5.4.2 Patient Dashboard

Description:

The patient dashboard provides patients with a personalized view of their tasks, schedules, notifications, and personal health data.

Actors:

Primary: Patient

Secondary: CareConnect Dashboard Service

Flow:

1. The patient logs into the CareConnect system.
2. The system authenticates the patient and identifies the patient role.
3. The system loads the patient dashboard.
4. The dashboard displays scheduled tasks, reminders, notifications, and personal health metrics.
5. The patient interacts with tasks (e.g., marks tasks as completed).

Inputs/Outputs:

Inputs: Patient dashboard request

Outputs: Task and schedule view, Health metric visualizations, Notification indicators

Edge Cases:

- No Scheduled Tasks: The Dashboard displays an informational empty state.
- Incomplete Data: Health data is partially unavailable.

UI/UX:

- Health data visualizations shall be simple and easy to interpret.
- Task completion of actions shall be clearly visible.

5.5 Scheduling & Notifications

5.5.1 Task Scheduling

Description:

The task scheduling process allows caregivers to create, modify, and manage scheduled care-related tasks for patients. Tasks may include medication reminders, appointments, or daily activities and can be configured with specific times and recurrence patterns.

Actors:

- Primary: Caregiver
- Secondary: CareConnect Scheduling Service

Flow:

1. The caregiver logs into the CareConnect system.
-

2. The caregiver navigates to the scheduling or task management section.
3. The system displays a task creation form.
4. The caregiver enters task details, including task name, description, scheduled time, and recurrence (if applicable).
5. The caregiver submits the task.
6. The system validates the task information and stores the task in the database.
7. The task becomes visible on the patient's dashboard and calendar view.

Inputs/Outputs:

- Inputs: Task name, description, scheduled date/time, recurrence settings, assigned patient
- Outputs: Scheduled task record stored in the system, Updated task and calendar views

Edge Cases:

- Invalid Time or Date: Scheduled time is in the past or incorrectly formatted.
- Missing Required Fields: Required task information is incomplete.
- System Failure: Task cannot be saved due to a backend error.

UI/UX:

- Task creation forms shall provide clear validation feedback.
- Recurring task options shall be easy to configure.

5.5.2 Notifications & Reminders

Description:

The notifications and reminders process delivers alerts to patients and caregivers regarding scheduled tasks, task completion, and missed tasks using configurable notification channels.

Actors:

- Primary: Patient, Caregiver
- Secondary: CareConnect Notification Service

Flow:

1. A scheduled task reaches its reminder time.
 2. The system determines the user's notification preferences.
 3. The system sends a notification via the configured channel (push notification, email, or SMS).
 4. The patient receives the notification and may complete the task.
 5. The system updates the task status and notifies the caregiver of task completion or missed tasks.
-

Inputs/Outputs:

- Inputs: Scheduled task, notification preferences
- Outputs: Notification delivery (push, email, SMS), Updated task status

Edge Cases:

- Notification Delivery Failure: Notification fails to send; the system logs the failure and may retry.
- Muted Notifications: User has disabled notifications for the task.
- Time Zone Differences: Notifications are sent at incorrect times due to misconfigured time zones.

UI/UX:

- Notification preferences shall be configurable per user.
- Alerts shall clearly indicate task details and urgency.

5.6 Health Data Tracking

5.6.1 Manual Health Data Entry

Description:

The manual health data entry process allows caregivers and patients to record health-related information such as vital signs and symptoms for tracking and monitoring purposes.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect Health Data Service

Flow:

1. The user logs into the CareConnect system.
2. The user navigates to the health data entry section.
3. The system displays a form for entering health metrics.
4. The user enters health data (e.g., blood pressure, weight, symptoms).
5. The user submits the data.
6. The system validates and stores the data.
7. The system updates visualizations and health summaries.

Inputs/Outputs:

- Inputs: Health metric type, value, timestamp, optional notes
 - Outputs: Stored health data record, Updated graphs and summaries
-

Edge Cases:

- Out-of-Range Values: Entered values exceed acceptable thresholds and trigger a warning.
- Invalid Input Format: Non-numeric or improperly formatted values are entered.
- Duplicate Entries: Multiple entries for the same metric and time period.

UI/UX:

- Data entry forms shall clearly label units of measurement.
- Warnings shall be displayed for unusual values without blocking entry.

5.6.2 Health Data Visualization & Reporting**Description:**

This process enables users to view historical health data through charts and generate reports for informational and sharing purposes.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect Reporting Service

Flow:

1. The user navigates to the health data visualization section.
2. The system retrieves historical health data.
3. The system displays charts and trend visualizations.
4. The user requests a report export.
5. The system generates the report in the selected format.

Inputs/Outputs:

- Inputs: Date range, metric selection, export format
- Outputs: On-screen charts, Downloadable CSV or PDF report

Edge Cases:

- No Data Available: No health data exists for the selected period.
- Report Generation Failure: Export fails due to system error.

UI/UX:

- Charts shall be simple and easy to interpret.
 - Export options shall be clearly visible.
-

5.7 AI Integration

5.7.1 AI-Assisted Health Insights

Description:

The AI-assisted health insights feature provides non-diagnostic, decision-support information by analyzing historical health data and identifying patterns or anomalies for caregiver review.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect AI Service

Flow:

1. The user navigates to the AI insights or dashboard section.
2. The system retrieves relevant historical health data.
3. The AI service analyzes the data for trends or anomalies.
4. The system displays AI-generated insights to the user.
5. The caregiver reviews the insights and determines follow-up actions.

Inputs/Outputs:

Inputs: Historical health data, user context

Outputs: AI-generated insights and alerts

Edge Cases:

- Insufficient Data: Not enough data is available for meaningful analysis.
- AI Service Unavailable: AI processing fails; the system displays a fallback message.

UI/UX:

- AI insights shall include clear disclaimers indicating non-diagnostic use.
- Insights shall be presented in plain, understandable language.

5.7 AI Integration

5.7.1 AI-Assisted Health Insights

Description:

The AI-assisted health insights feature provides non-diagnostic, decision-support information by analyzing historical health data and identifying patterns or anomalies for caregiver review.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect AI Service

Flow:

1. The user navigates to the AI insights or dashboard section.
2. The system retrieves relevant historical health data.
3. The AI service analyzes the data for trends or anomalies.
4. The system displays AI-generated insights to the user.
5. The caregiver reviews the insights and determines follow-up actions.

Inputs/Outputs:

Inputs: Historical health data, user context

Outputs: AI-generated insights and alerts

Edge Cases:

- Insufficient Data: Not enough data is available for meaningful analysis.
- AI Service Unavailable: AI processing fails; the system displays a fallback message.

UI/UX:

- AI insights shall include clear disclaimers indicating non-diagnostic use.
- Insights shall be presented in plain, understandable language.

5.7.2 “Ask AI” Query Feature

Description:

The “Ask AI” feature allows users to submit natural-language questions related to care tasks or health data and receive AI-generated informational responses.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect AI Service

Flow:

1. The user opens the “Ask AI” interface.
2. The user enters a question related to care or health data.
3. The system sends the query and relevant context to the AI service.
4. The AI service generates a response.
5. The system displays the response to the user.

Inputs/Outputs:

- Inputs: Natural-language query
- Outputs: AI-generated response

Edge Cases:

- Unsupported Query: The AI cannot answer the question; a fallback response is displayed.
- Ambiguous Query: The AI requests clarification.

UI/UX:

- The input interface shall resemble a simple chat interaction.
- Responses shall include disclaimers and encourage professional consultation when appropriate.

5.8 Communication & Media

5.8.1 In-App Messaging

Description:

The in-app messaging process enables secure, real-time communication between caregivers, patients, and authorized family members.

Actors:

- Primary: Caregiver, Patient, Family Member
- Secondary: CareConnect Messaging Service

Flow:

1. The user logs into the CareConnect system.
 2. The user navigates to the messaging section.
 3. The user selects a conversation or starts a new one.
 4. The user enters a message and sends it.
 5. The system delivers the message to the recipient.
 6. The recipient receives a notification of the new message.
-

Inputs/Outputs:

Inputs: Message content, recipient identifier

Outputs: Delivered message, Message notification

Edge Cases:

- Recipient Unavailable: Recipient is offline; message is delivered when available.
- Message Delivery Failure: Message fails to send due to network issues.

UI/UX:

- Messages shall display delivery timestamps.
- Unread messages shall be visually indicated.

5.8.2 Audio and Video Communication

Description:

This process supports audio and video calls for remote communication and telemedicine interactions.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect Communication Service

Flow:

1. The user initiates an audio or video call.
2. The system sends a call request to the recipient.
3. The recipient accepts or declines the call.
4. Upon acceptance, a real-time communication session is established.
5. The call ends when either party disconnects.

Inputs/Outputs:

- Inputs: Call request, media permissions
- Outputs: Live audio/video stream, Call status updates

Edge Cases:

- Permission Denied: Camera or microphone access is denied.
 - Poor Network Quality: Call quality degrades or disconnects.
-

UI/UX:

- Call controls (mute, end call) shall be easily accessible.
- Connection status shall be clearly indicated.

5.8.3 Media Upload & Sharing**Description:**

The media upload process allows users to share images or documents related to care activities.

Actors:

- Primary: Caregiver, Patient
- Secondary: CareConnect Media Service

Flow:

1. The user selects the option to upload media.
2. The user selects a file from their device.
3. The system validates the file type and size.
4. The system uploads and stores the media securely.
5. The media becomes accessible to authorized users.

Inputs/Outputs:

- Inputs: Media file (image or document)
- Outputs: Stored media file, Media access link or preview

Edge Cases:

- Unsupported File Type: File format is not allowed.
- File Too Large: File exceeds size limits.
- Upload Failure: Network interruption during upload.

UI/UX:

- Upload progress shall be clearly indicated.
- Errors shall explain why an upload failed.

5.9 Device & Third-Party Integrations**5.9.1 Wearable Device Integration****Description:**

The wearable device integration process allows CareConnect to collect health data automatically

from supported wearable platforms and display the data within the application.

Actors:

- Primary: Patient
- Secondary: CareConnect Integration Service, Wearable Platform API

Flow:

1. The patient logs into the CareConnect system.
2. The patient navigates to the device integration settings.
3. The system displays available wearable platforms.
4. The patient selects a wearable platform to connect.
5. The system redirects the patient to the wearable platform's authorization flow.
6. The patient grants permission for data sharing.
7. The system receives authorization confirmation and access credentials.
8. The system retrieves health data from the wearable platform.
9. The system stores and displays the data on the patient dashboard.

Inputs/Outputs:

- Inputs: User consent, authorization token
- Outputs: Synchronized wearable health data, Updated dashboard visualizations

Edge Cases:

- Authorization Denied: User declines permission to share data.
- Token Expiration: Authorization token expires and requires reauthorization.
- Sync Failure: Data synchronization fails due to API or network issues.

UI/UX:

- Integration status shall be clearly displayed.
- Users shall be able to disconnect devices at any time.

5.9.2 Third-Party Service Integration

Description:

This process enables integration with third-party services such as payment processors and smart home devices to extend system functionality.

Actors:

- Primary: Caregiver
- Secondary: CareConnect Integration Service, Third-Party APIs

Flow:

1. The caregiver navigates to the integrations or settings section.
2. The system displays available third-party services.
3. The caregiver selects a service to integrate.
4. The system initiates the authorization or API connection process.
5. Upon successful integration, the system enables related features.

Inputs/Outputs:

- Inputs: Authorization credentials, service configuration settings
- Outputs: Active third-party integration, Enabled related features

Edge Cases:

- Service Unavailable: Third-party API is temporarily unavailable.
- Invalid Credentials: Authorization fails due to incorrect credentials.

UI/UX:

- Integration setup steps shall be clearly guided.
- Error messages shall indicate integration status.

5.10 Gamification

5.10.1 Rewards and Achievements

Description:

The rewards and achievements process encourages user engagement by awarding points, badges, or achievements for completing care-related tasks and health goals.

Actors:

- Primary: Patient, Caregiver
- Secondary: CareConnect Gamification Service

Flow:

1. The user completes a task or achieves a defined goal.
-

2. The system detects task completion.
3. The system awards points or badges based on predefined rules.
4. The system updates the user's achievements and progress indicators.
5. The user receives a notification or visual confirmation.

Inputs/Outputs:

- Inputs: Task completion event
- Outputs: Awarded points or badges, Updated achievement status

Edge Cases:

- Duplicate Awards: Same task completion triggers multiple awards.
- Rule Misconfiguration: Achievement rules are incorrectly defined.

UI/UX:

- Achievements shall be visually engaging but non-intrusive.
- Rewards shall not block critical care workflows.

5.11 Social Networking

5.11.1 Activity Feed

Description:

The activity feed process allows users to post and view care-related updates shared with authorized participants.

Actors:

- Primary: Caregiver, Patient, Family Member
- Secondary: CareConnect Social Service

Flow:

1. The user navigates to the activity feed.
2. The system displays recent care updates.
3. The user creates a new post.
4. The system validates the post content.
5. The system publishes the post to authorized users.

Inputs/Outputs:

- Input: Post content
-

- Outputs: Published activity feed post, Updated feed for authorized users

Edge Cases:

- Unauthorized Content: User attempts to post restricted content.
- Feed Load Failure: Feed cannot be loaded due to system error.

UI/UX:

- Feed entries shall display timestamps and author information.

5.11.2 Group Discussions

Description:

Group discussions enable communication among caregivers, patients, or family members within defined care groups or support communities.

Actors:

- Primary: Caregiver, Patient, Family Member
- Secondary: CareConnect Social Service

Flow:

1. The user joins or is added to a group.
2. The user navigates to the group discussion area.
3. The user posts a message.
4. The system displays the message to group members.

Inputs/Outputs:

- Inputs: Group message content
- Outputs: Posted group message

Edge Cases:

- Unauthorized Access: User attempts to access a group they do not belong to.

UI/UX:

- Group membership shall be clearly indicated.

5.12 Analytics & Reporting

5.12.1 Health Analytics Dashboard

Description:

The health analytics dashboard provides caregivers with aggregated insights and trends derived from patient health and care data.

Actors:

- Primary: Caregiver
- Secondary: CareConnect Analytics Service

Flow:

1. The caregiver navigates to the analytics section.
2. The system retrieves relevant health and task data.
3. The system displays charts and trend visualizations.

Inputs/Outputs:

- Inputs: Analytics request
- Outputs: Visual analytics dashboards

Edge Cases:

- Insufficient Data: Not enough data is available for meaningful analysis.

UI/UX:

- Visualizations shall be clear and easy to interpret.

5.12.2 Report Generation and Export

Description:

The report generation process allows authorized users to generate and export health and care reports for external review or record keeping.

Actors:

- Primary: Caregiver
- Secondary: CareConnect Reporting Service

Flow:

1. The caregiver selects a report type and date range.
2. The system generates the report.
3. The caregiver downloads the report in the selected format.

Inputs/Outputs:

- Inputs: Report parameters (date range, format)
- Outputs: Downloadable CSV or PDF report

Edge Cases:

- Export Failure: Report generation fails due to system error.

UI/UX:

- Export options shall be clearly labeled and accessible.

6. API Design Specification for CareConnect

6.1 Overview

This section defines the Application Programming Interface (API) design for the CareConnect system. The API follows RESTful principles and provides secure, role-based access to system functionality. All endpoints enforce authentication, authorization, and comply with HIPAA/GDPR requirements through encrypted transmission and audit logging.

6.1.1 API Architecture Principles**RESTful Design:** All endpoints follow REST conventions with standard HTTP methods (GET, POST, PUT, PATCH, DELETE).

Versioning: API versioning is implemented via URL path (`/api/v1/`) to ensure backward compatibility.

Authentication: All endpoints require valid JWT (JSON Web Token) authentication except for public authentication endpoints.

Role-Based Access Control (RBAC): Each endpoint enforces role-based permissions (Caregiver, Patient, Family Member, Admin) as defined in the streamlined RBAC system.

Data Encryption: All API traffic is transmitted over TLS 1.3. Sensitive data is encrypted at rest using AES-256.

Rate Limiting: API requests are rate-limited per user/IP to prevent abuse (100 requests/minute for authenticated users, 10 requests/minute for unauthenticated).

Error Handling: Consistent error response format with appropriate HTTP status codes and descriptive messages.

6.1.2 Architecture Implementation

CareConnect uses a hybrid serverless architecture that leverages both AWS Lambda and AWS Fargate to optimize performance, cost, and scalability. The selection of runtime environment

depends on the specific characteristics of each API endpoint.

AWS Lambda (Node.js 18): Lambda functions handle lightweight, stateless operations that benefit from rapid scaling and pay-per-request pricing:

- Authentication Services: Login, registration, password recovery, token refresh, and logout operations
- Lightweight CRUD Operations: Simple data retrieval and updates with minimal processing
- Event Processors: Webhook handlers (Stripe, wearable devices), notification delivery, scheduled tasks
- API Proxies: Lightweight request routing and data transformation
- File Operations: Image resizing, PDF generation, media upload processing

Lambda Configuration:

- Runtime: Node.js 18.x
- Memory: 256MB - 1GB (varies by function)
- Timeout: 30 seconds (standard), 5 minutes (file processing)

Concurrency: Auto-scaling with reserved capacity for critical functionsAWS Fargate (Spring Boot 3 / Java 17): Fargate containers host the core business logic microservices that require stateful processing, complex algorithms, or longer execution times:

- User Service: Complex user management, role assignment, permissions calculations
- Scheduling Service: Appointment algorithms, conflict detection, recurring event generation
- Health Tracking Service: Medical data processing, trend analysis, anomaly detection
- Communication Service: Message threading, media attachment handling, conversation management
- Medication Service: Drug interaction checking, dosage calculations, adherence analytics
- Billing Service: Complex subscription logic, proration calculations, invoice generation
- Analytics Service: Data aggregation, statistical analysis, report generation

Fargate Configuration:

- Container Image: Spring Boot 3 application with Java 17
- CPU: 0.5 vCPU - 2 vCPU (varies by service)
- Memory: 1GB - 4GB (varies by service)
- Auto-scaling: Target tracking based on CPU and request count
- Health Checks: HTTP health endpoint monitoring with automatic replacement

API Gateway Integration: Amazon API Gateway serves as the unified entry point for all client requests, intelligently routing to the appropriate backend:

API Gateway provides:

- Request Routing: Based on endpoint path and method
- Authentication: JWT validation before routing to backend
- Throttling: Rate limiting per user/IP address
- CORS Handling: Cross-origin resource sharing for web clients
- Request/Response Transformation: Data format conversion when needed

Monitoring: CloudWatch metrics for all API calls
Polyglot Persistence: Different database technologies are used based on data access patterns:

- Amazon RDS PostgreSQL: Relational data (users, appointments, medications, audit logs)
- Amazon DynamoDB: Session data, user preferences, notification queues
- Amazon Timestream: Time-series health metrics from wearable devices
- Amazon ElastiCache Redis: Caching layer for frequently accessed data

Event-Driven Architecture: While not directly exposed via REST APIs, CareConnect uses Amazon MSK (Managed Streaming for Kafka) for internal event streaming between microservices. Events such as task completion, health metric updates, and subscription changes are published to Kafka topics for asynchronous processing.

Benefits of event-driven architecture:

- Loose coupling between services
- Reliable message delivery with at-least-once semantics
- Scalable event processing
- Real-time data synchronization
- Audit trail of all system events

This event-driven architecture is transparent to API consumers but ensures system reliability and scalability.

6.2 Base Configuration

6.2.1 Base URL

Development: <https://dev-api.careconnect.com/api/v1> Staging: <https://staging-api.careconnect.com/api/v1> Production: <https://api.careconnect.com/api/v1>

6.2.2 Request Headers

All API requests must include the following headers:

```
http
Content-Type: application/json
Authorization: Bearer {jwt_token}
X-Request-ID: {unique_request_identifier}
```

6.2.3 Response Format

All API responses follow a consistent JSON structure:

Success Response:

```
json
{
  "status": "success",
  "data": {
    // Response payload
  },
  "metadata": {
    "timestamp": "2026-01-31T10:30:00Z",
    "requestId": "req-12345-abcde"
  }
}
```

Error Response:

```
json
{
  "status": "error",
  "error": {
    "code": "ERROR_CODE",
    "message": "Human-readable error message",
    "details": [
      {
        "field": "email",
        "issue": "Invalid email format"
      }
    ]
  },
  "metadata": {
    "timestamp": "2026-01-31T10:30:00Z",
    "requestId": "req-12345-abcde"
  }
}
```

6.2.4 HTTP Status Codes

Table # HTTP Status Codes

Status Code	Meaning	Usage
200	OK	Successful GET, PUT, PATCH requests
201	Created	Successful POST request creating a resource
204	No Content	Successful DELETE request
400	Bad Request	Invalid request format or parameters
401	Unauthorized	Missing or invalid authentication token
403	Forbidden	Authenticated but insufficient permissions
404	Not Found	Requested resource does not exist
409	Conflict	Resource conflict (e.g., duplicate email)
422	Unprocessable Entity	Validation errors
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Server-side error
503	Service Unavailable	Service temporarily unavailable

6.3 Authentication & Authorization APIs

6.3.1 User Registration

Endpoint: POST /auth/register

Description: Creates a new user account and assigns default role.

Access: Public (unauthenticated)

Request Body:

```
json
{
  "email": "user@example.com",
  "password": "SecurePassword123!",
  "firstName": "John",
  "lastName": "Doe",
  "role": "patient",
  "phoneNumber": "+1234567890"
}
```

Validation Rules:

- Email: Valid format, unique in system
-

- Password: Minimum 8 characters, 1 uppercase, 1 lowercase, 1 number, 1 special character
- Role: One of [caregiver, patient, family_member]
- Phone number: Optional, E.164 format

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "userId": "usr_1a2b3c4d5e",
    "email": "user@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "role": "patient",
    "emailVerificationSent": true,
    "createdAt": "2026-01-31T10:30:00Z"
  }
}
```

Error Responses:

- 409: Email already registered
- 422: Validation errors (weak password, invalid email format)

6.3.3 Password Recovery Request

Endpoint: POST /auth/password-recovery

Description: Initiates password recovery process by sending reset link to email.

Access: Public (unauthenticated)

Request Body:

```
json
{
  "email": "user@example.com"
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "message": "If an account exists with this email, a password reset link has been sent."
  }
}
```

```
    "emailSent": true
  }
}
```

Note: Returns success regardless of email existence to prevent user enumeration.

6.3.4 Password Reset

Endpoint: POST /auth/password-reset

Description: Completes password reset using valid reset token.

Access: Public (unauthenticated)

Request Body:

```
json
{
  "resetToken": "reset_token_from_email",
  "newPassword": "NewSecurePassword456!"
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "message": "Password successfully reset. You can now log in with your new password.",
    "passwordUpdated": true
  }
}
```

Error Responses:

- 400: Invalid or expired reset token
- 422: New password does not meet requirements

6.3.5 Token Refresh

Endpoint: POST /auth/refresh

Description: Obtains new access token using refresh token.

Access: Requires valid refresh token

Request Body:

```
json
{
  "refreshToken": "refresh_token_here"
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "tokenType": "Bearer",
    "expiresIn": 3600
  }
}
```

6.3.6 Logout

Endpoint: POST /auth/logout

Description: Invalidates current access and refresh tokens.

Access: Authenticated users

Success Response (204): No content

6.4 User Management APIs

6.4.1 Get User Profile

Endpoint: GET /users/{userId}

Description: Retrieves user profile information.

Access:

- Caregivers: Can view patients they manage
- Patients: Can view own profile
- Family Members: Can view assigned patient profile
- Admin: Can view all profiles

Path Parameters:

- **userId:** User identifier (use "me" for current user)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
```

```
"userId": "usr_1a2b3c4d5e",
"email": "user@example.com",
"firstName": "John",
"lastName": "Doe",
"role": "patient",
"phoneNumber": "+1234567890",
"dateOfBirth": "1980-05-15",
"profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr_1a2b3c4d5e.jpg",
"createdAt": "2026-01-15T08:00:00Z",
"lastLoginAt": "2026-01-31T09:45:00Z",
"preferences": {
  "notifications": {
    "email": true,
    "push": true,
    "sms": false
  },
  "language": "en",
  "timezone": "America/New_York"
}
}
```

Error Responses:

- 403: Insufficient permissions to view this user
- 404: User not found

6.4.2 Update User Profile

Endpoint: PATCH /users/{userId}

Description: Updates user profile information.

Access:

- Users can update own profile
- Caregivers can update limited patient fields
- Admin can update all fields

Request Body:

```
json
{
  "firstName": "John",
  "lastName": "Smith",
  "phoneNumber": "+1234567890",
  "profilePictureUrl":
```

```
"https://cdn.careconnect.com/profiles/new_image.jpg",
  "preferences": {
    "notifications": {
      "email": true,
      "push": true,
      "sms": true
    }
  }
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "userId": "usr_1a2b3c4d5e",
    "updatedFields": ["firstName", "lastName", "phoneNumber"],
    "updatedAt": "2026-01-31T10:30:00Z"
  }
}
```

6.4.3 Get User Permissions

Endpoint: GET /users/{userId}/permissions

Description: Retrieves current user's role-based permissions.

Access: Authenticated users (own permissions only, unless admin)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "userId": "usr_1a2b3c4d5e",
    "role": "caregiver",
    "permissions": [
      "view_patients",
      "create_tasks",
      "update_schedules",
      "view_health_data",
      "send_messages",
      "manage_subscriptions"
    ],
  },
}
```

```
    "restrictions": [  
      "cannot_delete_patients",  
      "cannot_modify_billing"  
    ]  
  }  
}
```

6.5 Billing & Subscription APIs

6.5.1 Get Subscription Plans

Endpoint: GET /subscriptions/plans

Description: Retrieves available subscription plans and pricing.

Access: Authenticated caregivers

Success Response (200):

```
json  
{  
  "status": "success",  
  "data": {  
    "plans": [  
      {  
        "planId": "plan_monthly_basic",  
        "name": "Monthly Basic",  
        "description": "Single patient management",  
        "price": 29.99,  
        "currency": "USD",  
        "billingPeriod": "monthly",  
        "features": [  
          "Manage 1 patient",  
          "Task scheduling",  
          "Health data tracking",  
          "Basic notifications"  
        ]  
      },  
      {  
        "planId": "plan_monthly_premium",  
        "name": "Monthly Premium",  
        "description": "Up to 5 patients with advanced features",  
        "price": 79.99,  
        "currency": "USD",  
        "billingPeriod": "monthly",  
        "features": [  
          "Manage up to 5 patients",  
          "Task scheduling",  
          "Health data tracking",  
          "Basic notifications",  
          "Advanced patient management"  
        ]  
      }  
    ]  
  }  
}
```

```
        "Advanced task scheduling",
        "Health data tracking & analytics",
        "Priority notifications",
        "AI-assisted insights",
        "Video calling"
    ]
}
]
```

6.5.2 Create Subscription

Endpoint: POST /subscriptions

Description: Activates a new subscription using Stripe payment.

Access: Authenticated caregivers

Request Body:

```
json
{
  "planId": "plan_monthly_premium",
  "paymentMethodId": "pm_card_visa_12345",
  "billingDetails": {
    "name": "John Doe",
    "email": "john@example.com",
    "address": {
      "line1": "123 Main St",
      "city": "New York",
      "state": "NY",
      "postalCode": "10001",
      "country": "US"
    }
  }
}
```

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "subscriptionId": "sub_1a2b3c4d5e",
  }
}
```

```
    "planId": "plan_monthly_premium",
    "status": "active",
    "currentPeriodStart": "2026-01-31T10:30:00Z",
    "currentPeriodEnd": "2026-02-28T10:30:00Z",
    "nextBillingDate": "2026-02-28T10:30:00Z",
    "amount": 79.99,
    "currency": "USD"
  }
}
```

Error Responses:

- 400: Invalid payment method
- 402: Payment failed
- 409: Active subscription already exists

6.5.3 Get Subscription Status

Endpoint: GET /subscriptions/current

Description: Retrieves current subscription details for authenticated caregiver.

Access: Authenticated caregivers

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "subscriptionId": "sub_1a2b3c4d5e",
    "planId": "plan_monthly_premium",
    "planName": "Monthly Premium",
    "status": "active",
    "currentPeriodStart": "2026-01-31T10:30:00Z",
    "currentPeriodEnd": "2026-02-28T10:30:00Z",
    "cancelAtPeriodEnd": false,
    "trialEnd": null
  }
}
```

6.5.4 Get Billing History

Endpoint: GET /billing/history

Description: Retrieves paginated billing history with invoices.

Access: Authenticated caregivers

Query Parameters:

- **page:** Page number (default: 1)
- **limit:** Items per page (default: 10, max: 50)
- **startDate:** Filter start date (ISO 8601)
- **endDate:** Filter end date (ISO 8601)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "invoices": [
      {
        "invoiceId": "inv_123456",
        "subscriptionId": "sub_1a2b3c4d5e",
        "amount": 79.99,
        "currency": "USD",
        "status": "paid",
        "invoiceDate": "2026-01-31T10:30:00Z",
        "paidDate": "2026-01-31T10:35:00Z",
        "invoicePdfUrl": "https://stripe.com/invoices/inv_123456/pdf",
        "description": "Monthly Premium - February 2026"
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 5,
      "totalItems": 48,
      "itemsPerPage": 10
    }
  }
}
```

6.5.5 Cancel Subscription

Endpoint: DELETE /subscriptions/{subscriptionId}

Description: Cancels subscription at end of current billing period.

Access: Authenticated caregivers (own subscription only)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "subscriptionId": "sub_1a2b3c4d5e",
  }
}
```

```
    "status": "active",
    "cancelAtPeriodEnd": true,
    "currentPeriodEnd": "2026-02-28T10:30:00Z",
    "message": "Your subscription will remain active until 2026-02-28"
  }
}
```

6.6 Dashboard APIs

6.6.1 Get Caregiver Dashboard

Endpoint: GET /dashboard/caregiver

Description: Retrieves caregiver dashboard with patient summaries, tasks, and alerts.

Access: Authenticated caregivers

Query Parameters:

- **includeAlerts:** Include urgent alerts (default: true)
- **includeMetrics:** Include health metrics summary (default: true)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "caregiverId": "usr_caregiver_123",
    "patientCount": 3,
    "patients": [
      {
        "patientId": "usr_patient_456",
        "firstName": "Jane",
        "lastName": "Smith",
        "age": 72,
        "profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr\_patient\_456.jpg",
        "recentActivity": "Completed morning medication - 2 hours
ago",
        "upcomingTasks": 2,
        "urgentAlerts": 0
      }
    ],
    "upcomingTasks": [
      {
        "taskId": "task_789",
```

```
    "patientId": "usr_patient_456",
    "patientName": "Jane Smith",
    "title": "Afternoon medication",
    "scheduledTime": "2026-01-31T14:00:00Z",
    "priority": "high",
    "status": "pending"
  }
],
"alerts": [
  {
    "alertId": "alert_012",
    "type": "health_anomaly",
    "severity": "medium",
    "patientId": "usr_patient_456",
    "patientName": "Jane Smith",
    "message": "Blood pressure reading above normal range",
    "timestamp": "2026-01-31T09:15:00Z",
    "acknowledged": false
  }
],
"metricsOverview": {
  "tasksCompletedToday": 8,
  "tasksScheduledToday": 12,
  "missedTasks": 1,
  "averageCompletionRate": 92.5
}
}
```

6.6.2 Get Patient Dashboard

Endpoint: GET /dashboard/patient

Description: Retrieves patient dashboard with tasks, health data, and notifications.

Access: Authenticated patients

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "patientId": "usr_patient_456",
    "todaysTasks": [
      {
        "taskId": "task_789",
        "title": "Afternoon medication",

```

```
    "description": "Take blood pressure medication",
    "scheduledTime": "2026-01-31T14:00:00Z",
    "status": "pending",
    "priority": "high",
    "completedAt": null
  }
],
"upcomingAppointments": [
  {
    "appointmentId": "appt_345",
    "title": "Cardiology Follow-up",
    "scheduledTime": "2026-02-05T10:00:00Z",
    "provider": "Dr. Sarah Johnson",
    "location": "Main Street Clinic"
  }
],
"recentHealthMetrics": {
  "bloodPressure": {
    "systolic": 128,
    "diastolic": 82,
    "unit": "mmHg",
    "timestamp": "2026-01-31T08:00:00Z",
    "status": "normal"
  },
  "heartRate": {
    "value": 72,
    "unit": "bpm",
    "timestamp": "2026-01-31T08:00:00Z",
    "status": "normal"
  }
},
"notifications": [
  {
    "notificationId": "notif_567",
    "type": "reminder",
    "message": "Your afternoon medication is due in 30 minutes",
    "timestamp": "2026-01-31T13:30:00Z",
    "read": false
  }
],
"gamification": {
  "pointsEarned": 450,
  "level": 5,
  "recentAchievements": [
    {
      "achievementId": "ach_001",
```

```
        "name": "7-Day Streak",
        "description": "Completed all tasks for 7 consecutive days",
        "earnedDate": "2026-01-30T23:59:59Z",
        "iconUrl":
"https://cdn.careconnect.com/achievements/7day\_streak.png"
      }
    ]
  }
}
```

6.7 Task & Schedule Management APIs

6.7.1 Create Task

Endpoint: POST /tasks

Description: Creates a new care task for a patient.

Access: Authenticated caregivers

Request Body:

```
json
{
  "patientId": "usr_patient_456",
  "title": "Evening medication",
  "description": "Take diabetes medication with dinner",
  "scheduledTime": "2026-01-31T18:00:00Z",
  "recurrence": {
    "type": "daily",
    "interval": 1,
    "endDate": "2026-03-31T18:00:00Z"
  },
  "priority": "high",
  "category": "medication",
  "notificationSettings": {
    "reminderMinutesBefore": 30,
    "channels": ["push", "email"]
  }
}
```

Recurrence Types:

- **once:** One-time task
 - **daily:** Repeats every N days
 - **weekly:** Repeats every N weeks on specified days
-

- `monthly`: Repeats every N months

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "taskId": "task_890",
    "patientId": "usr_patient_456",
    "title": "Evening medication",
    "scheduledTime": "2026-01-31T18:00:00Z",
    "recurrence": {
      "type": "daily",
      "interval": 1,
      "endDate": "2026-03-31T18:00:00Z"
    },
    "status": "scheduled",
    "createdAt": "2026-01-31T10:30:00Z",
    "createdBy": "usr_caregiver_123"
  }
}
```

Error Responses:

- 400: Invalid recurrence pattern
- 403: Cannot create task for this patient
- 422: Validation errors

6.7.2 Get Tasks

Endpoint: GET /tasks

Description: Retrieves filtered list of tasks.

Access:

- Caregivers: Can view tasks for managed patients
- Patients: Can view own tasks
- Family Members: Can view tasks for assigned patient

Query Parameters:

- `patientId`: Filter by patient (required for caregivers)
 - `status`: Filter by status [pending, completed, missed, cancelled]
 - `startDate`: Filter start date (ISO 8601)
 - `endDate`: Filter end date (ISO 8601)
 - `category`: Filter by category [medication, appointment, exercise, meal, other]
-

- **page:** Page number (default: 1)
- **limit:** Items per page (default: 20, max: 100)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "tasks": [
      {
        "taskId": "task_890",
        "patientId": "usr_patient_456",
        "title": "Evening medication",
        "description": "Take diabetes medication with dinner",
        "scheduledTime": "2026-01-31T18:00:00Z",
        "status": "pending",
        "priority": "high",
        "category": "medication",
        "completedAt": null,
        "createdBy": "usr_caregiver_123"
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 3,
      "totalItems": 45,
      "itemsPerPage": 20
    }
  }
}
```

6.7.3 Update Task Status

Endpoint: PATCH /tasks/{taskId}/status

Description: Updates task completion status.

Access:

- Patients: Can mark own tasks complete
- Caregivers: Can update status for managed patient tasks

Request Body:

```
json
{
  "status": "completed",
  "completedAt": "2026-01-31T18:05:00Z",
  "notes": "Medication taken as scheduled"
```

```
}
```

Status Values: pending, completed, missed, cancelled, in_progress

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "taskId": "task_890",
    "status": "completed",
    "completedAt": "2026-01-31T18:05:00Z",
    "updatedBy": "usr_patient_456"
  }
}
```

6.7.4 Delete Task

Endpoint: DELETE /tasks/{taskId}

Description: Deletes a task (or single occurrence in recurring series).

Access: Authenticated caregivers (tasks for managed patients only)

Query Parameters:

- **deleteRecurring:** If true, deletes entire recurring series (default: false)

Success Response (204): No content

6.8 Health Data APIs

6.8.1 Record Health Metric

Endpoint: POST /health/metrics

Description: Manually records a health data point.

Access:

- Patients: Can record own health data
- Caregivers: Can record data for managed patients

Request Body:

```
json
{
  "patientId": "usr_patient_456",
  "metricType": "blood_pressure",
  "value": {
    "systolic": 128,
    "diastolic": 82
  },
  "unit": "mmHg",
  "timestamp": "2026-01-31T08:00:00Z",
  "notes": "Measured after morning walk",
  "source": "manual"
}
```

Supported Metric Types:

- blood_pressure
- heart_rate
- blood_glucose
- weight
- temperature
- oxygen_saturation
- steps
- sleep_hours

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "metricId": "metric_123",
    "patientId": "usr_patient_456",
    "metricType": "blood_pressure",
    "value": {
      "systolic": 128,
      "diastolic": 82
    },
    "unit": "mmHg",
    "timestamp": "2026-01-31T08:00:00Z",
    "source": "manual",
    "recordedBy": "usr_caregiver_123",
    "createdAt": "2026-01-31T10:30:00Z"
  }
}
```

6.8.2 Get Health Metrics

Endpoint: GET /health/metrics

Description: Retrieves historical health metrics with filtering. **Access:**

- Patients: Can view own health data
- Caregivers: Can view data for managed patients
- Family Members: Can view data for assigned patient

Query Parameters:

- **patientId:** Patient identifier (required for caregivers)
- **metricType:** Filter by metric type
- **startDate:** Start of date range (ISO 8601)
- **endDate:** End of date range (ISO 8601)
- **source:** Filter by source [manual, wearable, device]
- **page:** Page number (default: 1)
- **limit:** Items per page (default: 50, max: 200)

Success Response (200):

json

```
{
  "status": "success",
  "data": {
    "metrics": [
      {
        "metricId": "metric_123",
        "metricType": "blood_pressure",
        "value": {
          "systolic": 128,
          "diastolic": 82
        },
        "unit": "mmHg",
        "timestamp": "2026-01-31T08:00:00Z",
        "source": "manual",
        "status": "normal",
        "recordedBy": "usr_caregiver_123"
      }
    ],
    "aggregations": {
      "average": {
        "systolic": 126,
        "diastolic": 80
      },
      "min": {
        "systolic": 118,
        "diastolic": 75
      }
    }
  }
}
```

```

    },
    "max": {
      "systolic": 135,
      "diastolic": 88
    }
  },
  "pagination": {
    "currentPage": 1,
    "totalPages": 8,
    "totalItems": 385,
    "itemsPerPage": 50
  }
}

```

6.8.3 Get Health Data Report

Endpoint: GET /health/reports/{patientId}

Description: Generates exportable health data report.

Access:

- Patients: Can generate own reports
- Caregivers: Can generate reports for managed patients

Query Parameters:

- startDate: Report start date (ISO 8601)
- endDate: Report end date (ISO 8601)
- format: Export format [json, csv, pdf] (default: json)

metricTypes: Comma-separated metric types to include **Success Response (200):**

```

json
{
  "status": "success",
  "data": {
    "reportId": "report_456",
    "patientId": "usr_patient_456",
    "reportPeriod": {
      "startDate": "2026-01-01T00:00:00Z",
      "endDate": "2026-01-31T23:59:59Z"
    },
    "format": "pdf",
    "downloadUrl":
"https://cdn.careconnect.com/reports/report\_456.pdf",
    "expiresAt": "2026-02-07T10:30:00Z",
    "generatedAt": "2026-01-31T10:30:00Z"
  }
}

```

```
}  
}
```

6.9 AI Integration APIs

6.9.1 Get AI Health Insights

Endpoint: GET /ai/insights/{patientId}

Description: Retrieves AI-generated health insights based on historical data.

Access:

- Caregivers: Can view insights for managed patients
- Patients: Can view own insights

Query Parameters:

- **timeframe:** Analysis timeframe [7d, 30d, 90d] (default: 30d)

Success Response (200):

```
json  
{  
  "status": "success",  
  "data": {  
    "patientId": "usr_patient_456",  
    "analysisTimeframe": "30d",  
    "insights": [  
      {  
        "insightId": "insight_789",  
        "category": "blood_pressure",  
        "severity": "medium",  
        "title": "Blood pressure trending upward",  
        "description": "Analysis shows systolic blood pressure has  
increased by an average of 5 mmHg over the past 30 days.",  
        "recommendation": "Consider reviewing medication adherence and  
consulting with healthcare provider.",  
        "confidence": 0.87,  
        "relatedMetrics": ["blood_pressure"],  
        "generatedAt": "2026-01-31T10:30:00Z"  
      }  
    ],  
    "disclaimer": "These insights are for informational purposes only  
and do not constitute medical advice. Consult a healthcare  
professional for medical decisions."  
  }  
}
```

6.9.2 Ask AI Query

Endpoint: POST /ai/query

Description: Submits natural language query to AI assistant.

Access: Authenticated caregivers and patients

Request Body:

```
json
{
  "query": "What is the normal range for blood pressure in elderly patients?",
  "context": {
    "patientId": "usr_patient_456"
  }
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "queryId": "query_012",
    "query": "What is the normal range for blood pressure in elderly patients?",
    "response": "For elderly patients (65+), normal blood pressure is generally considered to be less than 130/80 mmHg. However, individual target ranges may vary based on health conditions. Always consult with a healthcare provider for personalized guidance.",
    "sources": [
      {
        "title": "American Heart Association Guidelines",
        "url": "https://www.heart.org/guidelines"
      }
    ],
    "confidence": 0.92,
    "disclaimer": "This information is for educational purposes only and does not constitute medical advice.",
    "timestamp": "2026-01-31T10:30:00Z"
  }
}
```

6.9.3 Transcribe Care Notes

Endpoint: POST /api/v1/ai/transcribe

Description: Transcribes audio recordings of care notes using Amazon Transcribe Medical, which is specifically trained on medical terminology. The AI automatically structures the transcription into standard care note formats, extracting key information such as vital signs, symptoms, and care activities.

Access: Authenticated caregivers

RBAC Enforcement:

- Required Role: Caregiver
- Permissions: can_create_notes, can_use_ai_features
- Backend Validation: User must have assigned patient relationship

Request Format: Multipart form data

Form Fields:

audioFile: File (required)

- Supported formats: mp3, wav, m4a, ogg
- Maximum size: 100MB
- Maximum duration: 30 minutes

patientId: String (required)

- UUID of patient for the visit note

visitDate: String (required)

- ISO 8601 date of visit
- Example: "2026-01-31"

noteType: String (optional)

- Type of note: visit_note, phone_call, assessment
- Default: visit_note

language: String (optional)

- Audio language code (ISO 639-1)
- Default: en-US
- Supported: en-US, es-US

Processing Flow:

1. Upload audio file to S3 temporary storage (encrypted)
 2. Initiate Transcribe Medical job asynchronously
 3. Return job ID to client for status polling
 4. Process transcription when complete:
 - Structure into care note format
-

- Extract medical entities (medications, symptoms, vital signs)
- Generate suggested note template
- 5. Store structured note in database
- 6. Send completion notification to caregiver
- 7. Delete temporary audio file after 24 hours

Success Response (202 Accepted):

```
json
{
  "status": "success",
  "data": {
    "transcriptionJobId": "trans_1a2b3c4d",
    "status": "processing",
    "estimatedCompletionTime": "2026-01-31T10:35:00Z",
    "statusCheckUrl": "/api/v1/ai/transcribe/trans_1a2b3c4d"
  }
}
```

Get Transcription Status:

Endpoint: GET /api/v1/ai/transcribe/{jobId}

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "transcriptionJobId": "trans_1a2b3c4d",
    "status": "completed",
    "transcription": "Patient reports feeling much better today. Blood pressure measured at 128 over 82. Heart rate is 72 beats per minute. Patient has been taking prescribed medication regularly. No side effects reported. Energy levels have improved significantly. Patient is walking 30 minutes daily as recommended. Follow-up scheduled in two weeks.",
    "structuredNote": {
      "vitalSigns": {
        "bloodPressure": {
          "systolic": 128,
          "diastolic": 82,
          "unit": "mmHg"
        },
        "heartRate": {
          "value": 72,
          "unit": "bpm"
        }
      }
    }
  }
}
```

```

    }
  },
  "symptoms": [
    {
      "name": "improved energy",
      "severity": "mild",
      "trend": "improving"
    }
  ],
  "medications": [
    {
      "compliance": "regular",
      "sideEffects": "none"
    }
  ],
  "activities": [
    {
      "type": "walking",
      "duration": "30 minutes",
      "frequency": "daily"
    }
  ],
  "followUp": {
    "scheduled": true,
    "timeframe": "two weeks"
  }
},
"confidence": 0.94,
"suggestedTemplate": "***Vital Signs:**\n- Blood Pressure: 128/82 mmHg\n- Heart Rate: 72 bpm\n\n**Subjective:**\nPatient reports feeling much better with improved energy levels.\n\n**Objective:**\nVital signs within normal range. Patient ambulatory and alert.\n\n**Assessment:**\nImprovement noted. Medication compliance excellent with no adverse effects.\n\n**Plan:**\nContinue current medication regimen. Maintain daily walking routine. Follow-up in two weeks.",
"processingTime": 23.4,
"wordCount": 67,
"completedAt": "2026-01-31T10:32:14Z"
}
}

```

Processing Status Values:

- **queued:** Job submitted, waiting to start
 - **processing:** Transcription in progress
-

- completed: Transcription finished successfully
- failed: Transcription failed (see error details)

Error Responses:*Invalid Audio Format:*

```
json
{
  "status": "error",
  "error": {
    "code": "INVALID_AUDIO_FORMAT",
    "message": "Audio file must be in mp3, wav, m4a, or ogg format.",
    "supportedFormats": ["mp3", "wav", "m4a", "ogg"]
  }
}
```

Audio File Too Large:

```
json
{
  "status": "error",
  "error": {
    "code": "FILE_TOO_LARGE",
    "message": "Audio file exceeds maximum size of 100MB.",
    "fileSize": 125000000,
    "maxSize": 100000000
  }
}
```

Transcription Failed:

```
json
{
  "status": "error",
  "error": {
    "code": "TRANSCRIPTION_FAILED",
    "message": "Unable to transcribe audio. The audio quality may be too poor or contains unsupported language.",
    "details": "Insufficient speech detected in audio file"
  }
}
```

Performance Requirements:

- Upload time: < 10 seconds for 100MB file
 - Processing time: ~1 minute per minute of audio
-

- Status check: < 200ms
- Concurrent transcriptions: 50 simultaneous jobs

Security Considerations:

- Audio files encrypted in S3 (AES-256)
- Temporary storage with 24-hour auto-deletion
- Transcribe Medical operates in HIPAA-eligible mode
- Access logs maintained for audit compliance
- Only authorized caregiver can retrieve transcription
- Patient identifier linked to transcription for access control

Testing Criteria:*AC-001: Successful audio transcription*

- Given: Caregiver has 2-minute audio recording of visit note
- When: Caregiver uploads audio via transcription API
- Then:
 - File uploaded to S3 successfully
 - Transcription job created
 - Job ID returned to client
 - Status check shows "processing"
 - Transcription completes within 3 minutes
 - Structured note extracted with vital signs
 - Suggested template generated

AC-002: Medical entity extraction accuracy

- Given: Audio contains "blood pressure 120 over 80"
- When: Transcription completes
- Then:
 - Structured note includes bloodPressure object
 - systolic = 120, diastolic = 80
 - unit = "mmHg"
 - Confidence score > 0.85

AC-003: Invalid audio format rejected

- Given: Caregiver uploads .avi video file
 - When: File validation occurs
 - Then:
 - Response status is 400 Bad Request
 - Error indicates invalid format
 - Supported formats listed
 - No S3 upload occurs
-

UI/UX:

- Upload progress indicator with percentage
- Processing status updates every 5 seconds
- Audio preview/playback before upload
- Ability to edit transcription before saving
- Clear disclaimer that AI output requires review
- Save as draft or publish to patient record
- Option to re-transcribe if quality poor

AWS Architecture Notes:

- Lambda Function: ai-transcribe-upload-handler (Node.js 18, 512MB, 30s timeout)
- Lambda Function: ai-transcribe-processor (Node.js 18, 1GB, 5min timeout)
- API Gateway: POST /api/v1/ai/transcribe (multipart/form-data)
- Storage: S3 bucket (ai-transcriptions-temp) with lifecycle policy (24h deletion)
- AI Service: Amazon Transcribe Medical (HIPAA-eligible)
- AI Service: Amazon Comprehend Medical (entity extraction)
- Database: RDS PostgreSQL (transcriptions table)
- Queue: SQS for async processing
- Notifications: SNS for completion alerts
- Monitoring: CloudWatch metrics (job success rate, processing time)

6.10 Communication APIs

6.10.1 Send Message

Endpoint: POST /messages

Description: Sends in-app message to another user.

Access: Authenticated users (can message connected users only)

Request Body:

```
json
{
  "recipientId": "usr_patient_456",
  "messageText": "How are you feeling today?",
  "attachments": [
```

```
{
  "type": "image",
  "url": "https://cdn.careconnect.com/media/image123.jpg"
}
]
```

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "messageId": "msg_345",
    "conversationId": "conv_678",
    "senderId": "usr_caregiver_123",
    "recipientId": "usr_patient_456",
    "messageText": "How are you feeling today?",
    "sentAt": "2026-01-31T10:30:00Z",
    "deliveryStatus": "delivered"
  }
}
```

6.10.2 Get Conversations

Endpoint: GET /messages/conversations

Description: Retrieves list of message conversations.

Access: Authenticated users

Query Parameters:

- **page:** Page number (default: 1)
- **limit:** Items per page (default: 20, max: 50)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "conversations": [
      {
        "conversationId": "conv_678",
        "participants": [
```

```
{
  {
    "userId": "usr_caregiver_123",
    "firstName": "John",
    "lastName": "Doe",
    "profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr\_caregiver\_123.jpg"
  },
  {
    "userId": "usr_patient_456",
    "firstName": "Jane",
    "lastName": "Smith",
    "profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr\_patient\_456.jpg"
  }
],
"lastMessage": {
  "messageText": "How are you feeling today?",
  "sentAt": "2026-01-31T10:30:00Z",
  "senderId": "usr_caregiver_123"
},
"unreadCount": 0,
"updatedAt": "2026-01-31T10:30:00Z"
}
],
"pagination": {
  "currentPage": 1,
  "totalPages": 2,
  "totalItems": 25,
  "itemsPerPage": 20
}
}
```

Endpoint: GET /messages/conversations

Description: Retrieves list of message conversations.

Access: Authenticated users

Query Parameters:

- **page:** Page number (default: 1)
- **limit:** Items per page (default: 20, max: 50)

Success Response (200):

json

```
{
  "status": "success",
  "data": {
    "conversations": [
      {
        "conversationId": "conv_678",
        "participants": [
          {
            "userId": "usr_caregiver_123",
            "firstName": "John",
            "lastName": "Doe",
            "profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr\_caregiver\_123.jpg"
          },
          {
            "userId": "usr_patient_456",
            "firstName": "Jane",
            "lastName": "Smith",
            "profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr\_patient\_456.jpg"
          }
        ],
        "lastMessage": {
          "messageText": "How are you feeling today?",
          "sentAt": "2026-01-31T10:30:00Z",
          "senderId": "usr_caregiver_123"
        },
        "unreadCount": 0,
        "updatedAt": "2026-01-31T10:30:00Z"
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 2,
      "totalItems": 25,
      "itemsPerPage": 20
    }
  }
}
```

6.10.3 Get Messages in Conversation

Endpoint: GET /messages/conversations/{conversationId}

Description: Retrieves messages in a specific conversation.

Access: Authenticated users (conversation participants only)

Query Parameters:

- **page:** Page number (default: 1)
- **limit:** Items per page (default: 50, max: 100)
- **before:** Get messages before this message ID

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "conversationId": "conv_678",
    "messages": [
      {
        "messageId": "msg_345",
        "senderId": "usr_caregiver_123",
        "messageText": "How are you feeling today?",
        "sentAt": "2026-01-31T10:30:00Z",
        "readAt": "2026-01-31T10:32:00Z",
        "attachments": []
      }
    ],
    "pagination": {
      "currentPage": 1,
      "hasMore": false
    }
  }
}
```

6.10.4 Initiate Audio/Video Call

Endpoint: POST /calls/initiate

Description: Initiates audio or video call request.

Access: Authenticated users (can call connected users only)

Request Body:

```
json
{
  "recipientId": "usr_patient_456",
  "callType": "video",
  "metadata": {
    "purpose": "daily_checkup"
  }
}
```

```
}  
}
```

Call Types: audio, video

Success Response (201):

```
json  
{  
  "status": "success",  
  "data": {  
    "callId": "call_901",  
    "callType": "video",  
    "initiatorId": "usr_caregiver_123",  
    "recipientId": "usr_patient_456",  
    "status": "ringing",  
    "webrtcConfig": {  
      "iceServers": [  
        {  
          "urls": "stun:stun.careconnect.com:3478"  
        },  
        {  
          "urls": "turn:turn.careconnect.com:3478",  
          "username": "temp_user_123",  
          "credential": "temp_credential_456"  
        }  
      ]  
    },  
    "initiatedAt": "2026-01-31T10:30:00Z"  
  }  
}
```

6.10.5 Upload Media

Endpoint: POST /media/upload

Description: Uploads image or document for sharing.

Access: Authenticated users

Request: Multipart form data

Form Fields:

- **file:** File to upload (max 10MB)
 - **type:** Media type [image, document]
 - **description:** Optional description
-

Supported Formats:

- Images: jpg, png, gif (max 10MB)
- Documents: pdf, docx (max 10MB)

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "mediaId": "media_234",
    "type": "image",
    "fileName": "checkup_photo.jpg",
    "fileSize": 2458392,
    "mimeType": "image/jpeg",
    "url": "https://cdn.careconnect.com/media/media_234.jpg",
    "thumbnailUrl":
      "https://cdn.careconnect.com/media/media_234_thumb.jpg",
    "uploadedBy": "usr_caregiver_123",
    "uploadedAt": "2026-01-31T10:30:00Z"
  }
}
```

Error Responses:

- 400: Invalid file format
- 413: File size exceeds limit

6.11 Device Integration APIs

6.11.1 Connect Wearable Device

Endpoint: POST /devices/wearables/connect

Description: Initiates OAuth flow to connect wearable device.

Access: Authenticated patients

Request Body:

```
json
{
  "platform": "fitbit",
  "redirectUri": "https://app.careconnect.com/devices/callback"
```

```
}
```

Supported Platforms: fitbit, apple_health, google_fit

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "authorizationUrl":
"https://www.fitbit.com/oauth2/authorize?client\_id=...",
    "state": "random_state_token_123",
    "expiresAt": "2026-01-31T10:40:00Z"
  }
}
```

6.11.2 Complete Wearable Connection

Endpoint: POST /devices/wearables/callback

Description: Completes wearable device connection after OAuth.

Access: Authenticated patients

Request Body:

```
json
{
  "platform": "fitbit",
  "authorizationCode": "auth_code_from_provider",
  "state": "random_state_token_123"
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "connectionId": "conn_567",
    "platform": "fitbit",
    "status": "active",
    "connectedAt": "2026-01-31T10:30:00Z",
    "syncSettings": {
      "autoSync": true,
      "syncInterval": "hourly",
      "metrics": ["steps", "heart_rate", "sleep"]
    }
  }
}
```

```
}  
}
```

6.11.3 Get Connected Devices

Endpoint: GET /devices/connected

Description: Retrieves list of connected devices and their sync status.

Access: Authenticated patients

Success Response (200):

```
json  
{  
  "status": "success",  
  "data": {  
    "devices": [  
      {  
        "connectionId": "conn_567",  
        "platform": "fitbit",  
        "deviceName": "Fitbit Charge 5",  
        "status": "active",  
        "connectedAt": "2026-01-31T10:30:00Z",  
        "lastSyncAt": "2026-01-31T11:00:00Z",  
        "syncedMetrics": ["steps", "heart_rate", "sleep"],  
        "dataPointsCollected": 1247  
      }  
    ]  
  }  
}
```

6.11.4 Disconnect Device

Endpoint: DELETE /devices/{connectionId}

Description: Disconnects wearable device and revokes access.

Access: Authenticated patients (own devices only)

Success Response (204): No content

6.12 Notification APIs

6.12.1 Get Notifications

Endpoint: GET /notifications

Description: Retrieves user notifications with filtering.

Access: Authenticated users

Query Parameters:

- **type:** Filter by type [reminder, alert, message, system]
- **read:** Filter by read status [true, false]
- **page:** Page number (default: 1)
- **limit:** Items per page (default: 20, max: 50)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "notifications": [
      {
        "notificationId": "notif_567",
        "type": "reminder",
        "title": "Medication Reminder",
        "message": "Your afternoon medication is due in 30 minutes",
        "read": false,
        "actionUrl": "/tasks/task_890",
        "createdAt": "2026-01-31T13:30:00Z"
      }
    ],
    "unreadCount": 5,
    "pagination": {
      "currentPage": 1,
      "totalPages": 2,
      "totalItems": 35,
      "itemsPerPage": 20
    }
  }
}
```

6.12.2 Mark Notification as Read

Endpoint: PATCH /notifications/{notificationId}/read

Description: Marks a notification as read.

Access: Authenticated users (own notifications only)

Success Response (200):

json

```
{
  "status": "success",
  "data": {
    "notificationId": "notif_567",
    "read": true,
    "readAt": "2026-01-31T10:30:00Z"
  }
}
```

6.12.3 Update Notification Preferences

Endpoint: PATCH /notifications/preferences

Description: Updates user notification preferences.

Access: Authenticated users

Request Body:

```
json
{
  "channels": {
    "email": true,
    "push": true,
    "sms": false
  },
  "types": {
    "reminders": true,
    "alerts": true,
    "messages": true,
    "system": false
  },
  "quietHours": {
    "enabled": true,
    "startTime": "22:00",
    "endTime": "07:00",
    "timezone": "America/New_York"
  }
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "preferences": {
      "channels": {
```

```
    "email": true,
    "push": true,
    "sms": false
  },
  "types": {
    "reminders": true,
    "alerts": true,
    "messages": true,
    "system": false
  },
  "quietHours": {
    "enabled": true,
    "startTime": "22:00",
    "endTime": "07:00",
    "timezone": "America/New_York"
  }
},
"updatedAt": "2026-01-31T10:30:00Z"
}
```

6.12.4 Send Notification

Endpoint: POST /api/v1/notifications/send

Description: Sends notifications to specified users via configured channels (push, email, SMS). Caregivers can send notifications to assigned patients, and admins can send to any user. Supports immediate delivery and scheduled notifications.

Access:

- Caregivers: Can send to assigned patients only
- Admins: Can send to any users
- Patients: Cannot send notifications

RBAC Enforcement:

- Required Role: Caregiver or Admin
- Permissions: can_send_notifications
- Backend Validation:
 - Caregivers can only send to assigned patients
 - Admins can send to any user
 - Verify recipient IDs exist and are active
 -

Request Body:

json

```
{
  "recipientIds": ["usr_patient_456", "usr_patient_789"],
  "type": "reminder",
  "title": "Medication Reminder",
  "message": "Time to take your afternoon blood pressure medication (Lisinopril 10mg)",
  "channels": ["push", "email"],
  "priority": "high",
  "actionUrl": "/tasks/task_890",
  "scheduledFor": null,
  "expiresAt": "2026-01-31T15:00:00Z",
  "metadata": {
    "taskId": "task_890",
    "medicationName": "Lisinopril"
  }
}
```

Table # Request Fields

Field	Type	Required	Description
recipientIds	Array[String]	☒	User IDs to notify (max 100, must be accessible by sender)
type	Enum	☒	Notification type: reminder, alert, message, system, announcement
title	String	☒	Subject line (max 100 chars, plain text only)
message	String	☒	Body content (max 500 chars, plain text or markdown)
channels	Array[String]	☒	Delivery channels: push, email, sms (at least one required)
priority	Enum	☐	Priority level: low, medium, high, urgent (default: medium)
actionUrl	String	☐	Deep link URL (e.g., /tasks/task_890)
scheduledFor	ISO 8601	☐	Future delivery time (null = immediate, max 30 days)
expiresAt	ISO 8601	☐	Expiration time (default: 7 days from creation)
metadata	Object	☐	Custom key-value pairs for analytics (not displayed to user)

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "notificationId": "notif_batch_abc123",
```

```
"recipientCount": 2,
"deliveryStatus": {
  "queued": 2,
  "sent": 0,
  "failed": 0
},
"scheduledFor": null,
"estimatedDeliveryTime": "2026-01-31T10:30:30Z",
"deliveryDetails": [
  {
    "recipientId": "usr_patient_456",
    "channels": ["push", "email"],
    "status": "queued"
  },
  {
    "recipientId": "usr_patient_789",
    "channels": ["push", "email"],
    "status": "queued"
  }
]
}
```

Delivery Status Tracking:

Endpoint: GET /api/v1/notifications/send/{notificationId}/status

Response:

```
json
{
  "status": "success",
  "data": {
    "notificationId": "notif_batch_abc123",
    "overallStatus": "partially_delivered",
    "deliveryDetails": [
      {
        "recipientId": "usr_patient_456",
        "channels": {
          "push": {
            "status": "delivered",
            "deliveredAt": "2026-01-31T10:30:35Z"
          },
          "email": {
            "status": "sent",
            "sentAt": "2026-01-31T10:30:32Z"
          }
        }
      }
    ]
  }
}
```

```

    }
  }
},
{
  "recipientId": "usr_patient_789",
  "channels": {
    "push": {
      "status": "failed",
      "error": "Device token expired",
      "failedAt": "2026-01-31T10:30:40Z"
    },
    "email": {
      "status": "delivered",
      "deliveredAt": "2026-01-31T10:31:15Z"
    }
  }
},
],
"metrics": {
  "totalRecipients": 2,
  "successfulDeliveries": 3,
  "failedDeliveries": 1,
  "successRate": 75.0
}
}
}

```

Channel Status Values:

- **queued:** Queued for delivery
- **sent:** Sent but delivery not confirmed
- **delivered:** Successfully delivered
- **failed:** Delivery failed
- **filtered:** Blocked by user preferences
- **bounced:** Email bounced or SMS rejected

Error Responses:

Unauthorized Recipients:

```

json
{
  "status": "error",
  "error": {
    "code": "UNAUTHORIZED_RECIPIENTS",
    "message": "You do not have permission to send notifications to

```

```
some recipients.",
  "unauthorizedIds": ["usr_patient_999"],
  "details": "Caregivers can only send to assigned patients"
}
}
```

Too Many Recipients:

```
json
{
  "status": "error",
  "error": {
    "code": "TOO_MANY_RECIPIENTS",
    "message": "Maximum 100 recipients per notification request.",
    "recipientCount": 150,
    "maxRecipients": 100
  }
}
```

Invalid Channel:

```
json
{
  "status": "error",
  "error": {
    "code": "INVALID_CHANNEL",
    "message": "Invalid notification channel specified.",
    "invalidChannels": ["telegram"],
    "supportedChannels": ["push", "email", "sms"]
  }
}
```

Performance Requirements:

- API response time: < 500ms
- Queuing time: < 1 second
- Push notification delivery: < 10 seconds
- Email delivery: < 30 seconds
- SMS delivery: < 60 seconds
- Concurrent send requests: 200 requests/second

Rate Limiting:

- Caregivers: 100 notifications per hour
 - Admins: 1000 notifications per hour
-

- Per-recipient: Maximum 50 notifications per day (prevents spam)

Security Considerations:

- Notification content encrypted in transit and at rest
- Sensitive PHI redacted from push notifications (visible only in-app)
- Delivery logs maintained for audit compliance
- Recipients verified against sender permissions
- Rate limiting prevents abuse
- Opt-out preferences respected

Testing Criteria:*AC-001: Successful notification delivery to multiple recipients*

- Given: Caregiver has 2 assigned patients
- When: Caregiver sends notification to both via push and email
- Then:
 - API returns 201 Created
 - Both recipients receive push notifications
 - Both recipients receive emails
 - Delivery status shows "delivered" for all channels
 - Notification appears in recipients' notification lists

AC-002: Unauthorized recipient blocked

- Given: Caregiver attempts to send to unassigned patient
- When: Notification sent
- Then:
 - Response status is 403 Forbidden
 - Error indicates unauthorized recipients
 - No notifications delivered
 - Attempt logged for security monitoring

AC-003: User preferences respected

- Given: Patient has disabled email notifications
- When: Caregiver sends notification via email and push
- Then:
 - Push notification delivered
 - Email filtered (not sent)
 - Delivery status shows email as "filtered"
 - Patient receives only push notification

UI/UX:

- Send notification form with recipient selection
-

- Channel selection checkboxes
- Priority indicator
- Schedule delivery option
- Preview before sending
- Delivery status dashboard
- Failed delivery alerts with retry option

AWS Architecture Notes:

- Lambda Function: `notification-send-handler` (Node.js 18, 512MB, 30s)
- Lambda Function: `notification-delivery-worker` (Node.js 18, 256MB, 2min)
- API Gateway: POST `/api/v1/notifications/send`
- Queue: SQS (`notification-delivery-queue`) with FIFO for ordered delivery
- Push: SNS for mobile push notifications
- Email: SES for email delivery
- SMS: SNS for SMS delivery
- Database: DynamoDB (`notification_deliveries` table)
- Monitoring: CloudWatch metrics (delivery rates, failures)
- Alarms: Alert if delivery failure rate > 10%

6.12.5 Test Notification Delivery

Endpoint: POST `/api/v1/notifications/test`

Description: Sends test notification to verify delivery channels are configured correctly. Useful for troubleshooting notification issues and validating user settings.

Access: Authenticated users (tests own notification settings)

Request Body:

```
json
{
  "channels": ["push", "email", "sms"],
  "message": "This is a test notification from CareConnect"
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "testId": "test_notif_xyz",
    "results": {
      "push": {
```

```
    "status": "delivered",
    "deviceTokens": 2,
    "deliveryTime": 1.2
  },
  "email": {
    "status": "sent",
    "recipientEmail": "user@example.com",
    "deliveryTime": 0.8
  },
  "sms": {
    "status": "failed",
    "error": "No phone number configured",
    "suggestion": "Add phone number in profile settings"
  }
},
"overallStatus": "partial_success"
}
```

AWS Architecture Notes:

- Lambda Function: notification-test-handler (Node.js 18, 256MB, 30s)
- Same delivery infrastructure as production notifications
- Tagged as test for analytics filtering

6.13 Gamification APIs

6.13.1 Get User Achievements

Endpoint: GET /gamification/achievements

Description: Retrieves user's earned achievements and progress.

Access: Authenticated users (own achievements only)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "userId": "usr_patient_456",
    "totalPoints": 450,
    "level": 5,
    "nextLevelPoints": 500,
    "achievements": [
      {
```

```
    "achievementId": "ach_001",
    "name": "7-Day Streak",
    "description": "Completed all tasks for 7 consecutive days",
    "category": "consistency",
    "iconUrl":
"https://cdn.careconnect.com/achievements/7day\_streak.png",
    "earnedDate": "2026-01-30T23:59:59Z",
    "points": 50
  }
],
"inProgress": [
  {
    "achievementId": "ach_002",
    "name": "30-Day Streak",
    "description": "Complete all tasks for 30 consecutive days",
    "category": "consistency",
    "progress": 7,
    "target": 30,
    "percentComplete": 23.33
  }
]
}
```

6.13.2 Get Leaderboard

Endpoint: GET /gamification/leaderboard

Description: Retrieves leaderboard rankings (anonymized for privacy).

Access: Authenticated patients

Query Parameters:

- **timeframe:** Leaderboard timeframe [week, month, all_time] (default: week)
- **category:** Filter by achievement category

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "timeframe": "week",
    "userRank": 15,
    "userPoints": 120,
```

```
    "topRankings": [
      {
        "rank": 1,
        "displayName": "Patient #142",
        "points": 250,
        "level": 8
      },
      {
        "rank": 2,
        "displayName": "Patient #89",
        "points": 230,
        "level": 7
      }
    ]
  }
}
```

Note: Display names are anonymized to protect patient privacy.

6.14 Analytics & Reporting APIs

6.14.1 Get Care Analytics

Endpoint: GET /analytics/care

Description: Retrieves care analytics for caregivers.

Access: Authenticated caregivers

Query Parameters:

- **patientId:** Filter by patient (optional)
- **startDate:** Analysis start date
- **endDate:** Analysis end date
- **metrics:** Comma-separated metrics to include

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "period": {
      "startDate": "2026-01-01T00:00:00Z",
      "endDate": "2026-01-31T23:59:59Z"
    },
  },
}
```

```
"taskMetrics": {
  "totalScheduled": 420,
  "totalCompleted": 388,
  "totalMissed": 15,
  "completionRate": 92.38,
  "averageCompletionDelay": "5 minutes"
},
"patientMetrics": [
  {
    "patientId": "usr_patient_456",
    "patientName": "Jane Smith",
    "tasksCompleted": 88,
    "tasksScheduled": 95,
    "completionRate": 92.63,
    "healthTrend": "stable"
  }
],
"categoryBreakdown": {
  "medication": {
    "scheduled": 180,
    "completed": 175,
    "completionRate": 97.22
  },
  "appointment": {
    "scheduled": 12,
    "completed": 11,
    "completionRate": 91.67
  }
}
}
```

6.14.2 Get QuickSight Dashboard URL

Endpoint: GET /api/v1/analytics/dashboard

Description: Generates an embedded Amazon QuickSight dashboard URL with temporary authentication for caregiver analytics visualization. The dashboard provides interactive charts and reports on patient health trends, task completion rates, and care quality metrics.

Access: Authenticated caregivers and admins

RBAC Enforcement:

- Required Role: Caregiver or Admin
- Permissions: can_view_analytics
- Backend Validation:
 - Generates dashboard with row-level security (RLS)
 - Caregivers see only assigned patients' data
 - Admins see organization-wide data

Query Parameters:

dashboardType: string (optional)

- Values: patient_overview, care_metrics, health_trends, compliance
- Default: patient_overview

timeRange: string (optional)

- Values: 7d, 30d, 90d, 1y
- Default: 30d

patientId: string (optional)

- Filter to specific patient (caregiver must be assigned)
- If omitted, shows all assigned patients

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "dashboardUrl": "https://us-east-1.quicksight.aws.amazon.com/sn/embed/share/accounts/123456789012/dashboards/abc-def-123?code=xyz...",
    "dashboardId": "careconnect-patient-overview",
    "expiresAt": "2026-01-31T11:30:00Z",
    "sessionDuration": 600,
    "availableFilters": [
      {
        "name": "PatientID",
        "type": "dropdown",
        "values": ["usr_patient_456", "usr_patient_789"]
      },
      {
        "name": "DateRange",
        "type": "date_range",
        "defaultValue": "30d"
      }
    ],
    "metadata": {
```

```
    "dashboardTitle": "Patient Care Overview",  
    "lastUpdated": "2026-01-31T10:00:00Z",  
    "dataFreshness": "15 minutes"  
  }  
}  
}
```

Dashboard Types:

patient_overview:

- Patient demographics and assignment summary
- Task completion rates by patient
- Recent health alerts and anomalies
- Upcoming appointments and medication schedules

care_metrics:

- Caregiver productivity metrics
- Visit frequency and duration
- Documentation completion rates
- Response time to patient requests

health_trends:

- Vital sign trends over time
- Medication adherence patterns
- Activity level progression
- Health risk scores

compliance:

- HIPAA access logs
- Documentation quality scores
- Required training completion
- Audit trail summaries

Security Features:

- Embedded dashboard uses anonymous embedding with IAM authentication
- Row-Level Security (RLS) enforces data filtering by caregiver assignment
- Session-based access with automatic expiration (10 minutes)
- No persistent QuickSight user accounts required
- Dashboard URL is single-use and time-limited

Error Responses:

QuickSight Not Configured:

```
json
{
  "status": "error",
  "error": {
    "code": "ANALYTICS_NOT_AVAILABLE",
    "message": "Analytics dashboard is currently unavailable. Please contact support.",
    "details": "QuickSight subscription not active"
  }
}
```

No Data Available:

```
json
{
  "status": "error",
  "error": {
    "code": "INSUFFICIENT_DATA",
    "message": "Not enough data available to generate analytics dashboard.",
    "minimumDataRequirements": {
      "patients": 1,
      "dataPoints": 10,
      "timeSpan": "7 days"
    }
  }
}
```

Performance Requirements:

- Embed URL generation: < 2 seconds
- Dashboard load time: < 5 seconds
- Data refresh: Every 15 minutes (automatic)
- Interactive query response: < 3 seconds

Testing Criteria:*AC-001: Dashboard URL generated successfully*

- Given: Caregiver has 3 assigned patients with 30 days of data
 - When: Caregiver requests patient_overview dashboard
 - Then:
 - Response status is 200 OK
 - Valid QuickSight embed URL returned
 - URL expires in 10 minutes
-

- Dashboard loads with caregiver's patient data only
- Interactive filters functional

AC-002: Row-level security enforced

- Given: Caregiver A assigned to Patient 1, Caregiver B assigned to Patient 2
- When: Caregiver A views dashboard
- Then:
 - Dashboard shows only Patient 1 data
 - Patient 2 data not visible
 - Filters show only Patient 1 as option
 - Cannot manually add Patient 2 filter

UI/UX:

- Embedded dashboard renders in iframe
- Loading indicator while generating URL
- Responsive layout for mobile and desktop
- Export to PDF button
- Print-friendly format option
- Refresh data button
- Help documentation link

AWS Architecture Notes:

- Lambda Function: `analytics-dashboard-url-generator` (Node.js 18, 512MB, 30s)
- API Gateway: GET `/api/v1/analytics/dashboard`
- Service: Amazon QuickSight (Enterprise Edition)
- Authentication: QuickSight anonymous embedding with RLS
- Data Source: Amazon Athena (queries S3 data lake)
- Data Lake: S3 bucket with Glue catalog
- ETL: AWS Glue jobs (nightly data aggregation)
- Database: RDS PostgreSQL (source operational data)
- Security: IAM role with `QuickSight:GenerateEmbedUrlForRegisteredUser` permission

6.14.3 Refresh Analytics Data

Endpoint: POST `/api/v1/analytics/refresh`

Description: Triggers immediate refresh of analytics data aggregations. Normally data refreshes automatically every 15 minutes, but this endpoint allows on-demand refresh when real-time data is needed.

Access: Authenticated caregivers and admins

Request Body:

```
json
{
  "datasetType": "patient_metrics",
  "incremental": true
}
```

Dataset Types:

- **patient_metrics:** Patient health data and vital signs
- **task_analytics:** Task completion and scheduling data
- **communication:** Messaging and interaction data
- **billing:** Subscription and payment analytics

Success Response (202 Accepted):

```
json
{
  "status": "success",
  "data": {
    "refreshJobId": "refresh_abc123",
    "datasetType": "patient_metrics",
    "status": "running",
    "estimatedCompletionTime": "2026-01-31T10:35:00Z",
    "lastRefreshCompleted": "2026-01-31T10:00:00Z"
  }
}
```

Performance Requirements:

- Incremental refresh: < 2 minutes
- Full refresh: < 10 minutes

AWS Architecture Notes:

- Lambda Function: **analytics-refresh-trigger** (Node.js 18, 256MB, 30s)
- ETL: AWS Glue job (Python)
- Data Source: RDS PostgreSQL, DynamoDB, Timestream
- Data Target: S3 Parquet files
- Query Engine: Amazon Athena

6.15 Social Networking APIs

6.15.1 Create Activity Post

Endpoint: POST /social/posts

Description: Creates activity feed post.

Access: Authenticated caregivers, patients, family members

Request Body:

```
json
{
  "patientId": "usr_patient_456",
  "content": "Had a great walk in the park this morning!",
  "visibility": "care_circle",
  "attachments": [
    {
      "type": "image",
      "url": "https://cdn.careconnect.com/media/media_234.jpg"
    }
  ]
}
```

Visibility Options: care_circle, private, public

Success Response (201):

```
json
{
  "status": "success",
  "data": {
    "postId": "post_678",
    "authorId": "usr_patient_456",
    "content": "Had a great walk in the park this morning!",
    "visibility": "care_circle",
    "createdAt": "2026-01-31T10:30:00Z",
    "likesCount": 0,
    "commentsCount": 0
  }
}
```

6.15.2 Get Activity Feed

Endpoint: GET /social/feed

Description: Retrieves activity feed posts.

Access: Authenticated users

Query Parameters:

- patientId: Filter by patient
 - page: Page number (default: 1)
-

- **limit:** Items per page (default: 20, max: 50)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "posts": [
      {
        "postId": "post_678",
        "author": {
          "userId": "usr_patient_456",
          "firstName": "Jane",
          "lastName": "Smith",
          "profilePictureUrl":
"https://cdn.careconnect.com/profiles/usr\_patient\_456.jpg"
        },
        "content": "Had a great walk in the park this morning!",
        "attachments": [
          {
            "type": "image",
            "url": "https://cdn.careconnect.com/media/media\_234.jpg",
            "thumbnailUrl":
"https://cdn.careconnect.com/media/media\_234\_thumb.jpg"
          }
        ],
        "createdAt": "2026-01-31T10:30:00Z",
        "likesCount": 3,
        "commentsCount": 1,
        "userHasLiked": false
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 5,
      "totalItems": 92,
      "itemsPerPage": 20
    }
  }
}
```

6.16 Admin APIs (Role-Based Access Control)

6.16.1 Get All Users (Admin Only)

Endpoint: GET /admin/users

Description: Retrieves paginated list of all system users.

Access: Admin role only

Query Parameters:

- **role:** Filter by role
- **status:** Filter by account status [active, inactive, suspended]
- **search:** Search by name or email
- **page:** Page number (default: 1)
- **limit:** Items per page (default: 50, max: 100)

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "users": [
      {
        "userId": "usr_1a2b3c4d5e",
        "email": "user@example.com",
        "firstName": "John",
        "lastName": "Doe",
        "role": "patient",
        "status": "active",
        "createdAt": "2026-01-15T08:00:00Z",
        "lastLoginAt": "2026-01-31T09:45:00Z"
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 20,
      "totalItems": 1000,
      "itemsPerPage": 50
    }
  }
}
```

6.16.2 Update User Role (Admin Only)

Endpoint: PATCH /admin/users/{userId}/role

Description: Updates user's role assignment.

Access: Admin role only

Request Body:

```
json
{
  "role": "caregiver",
  "reason": "User requested role change"
}
```

Success Response (200):

```
json
{
  "status": "success",
  "data": {
    "userId": "usr_1a2b3c4d5e",
    "previousRole": "patient",
    "newRole": "caregiver",
    "updatedAt": "2026-01-31T10:30:00Z",
    "updatedBy": "usr_admin_999"
  }
}
```

6.17 Webhook Configuration APIs

6.17.1 Register Webhook

Endpoint: POST /webhooks

Description: Registers webhook endpoint for event notifications.

Access: Authenticated caregivers (for patient events), Admin (for system events)

Request Body:

```
json
{
  "url": "https://external-system.com/webhooks/careconnect",
  "events": ["task.completed", "health.metric.recorded",
"alert.created"],
  "secret": "webhook_secret_key_123",
  "description": "Integration with external care management system"
}
```

Success Response (201):

```
json
```

```
{
  "status": "success",
  "data": {
    "webhookId": "webhook_123",
    "url": "https://external-system.com/webhooks/careconnect",
    "events": ["task.completed", "health.metric.recorded",
"alert.created"],
    "status": "active",
    "createdAt": "2026-01-31T10:30:00Z"
  }
}
```

Supported Events:

- task.created, task.completed, task.missed
- health.metric.recorded
- alert.created, alert.acknowledged
- subscription.created, subscription.cancelled
- message.sent, message.received

6.18 API Security & Compliance

6.18.1 Authentication Flow

User Login: Submit credentials to /auth/login
Receive JWT: Access token valid for 1 hour, refresh token valid for 30 days

Include Token: Add Authorization: Bearer {token} header to all requests

Token Refresh: When access token expires, use /auth/refresh with refresh token

Logout: Call /auth/logout to invalidate tokens

6.18.2 Rate Limiting

Rate limits are enforced per user/IP address:

User Type	Rate Limit
Unauthenticated	10 requests/minute
Authenticated	100 requests/minute
Premium Subscription	200 requests/minute

Rate Limit Headers:

http

X-RateLimit-Limit: 100

X-RateLimit-Remaining: 87

X-RateLimit-Reset: 1706702400

429 Response:

```
json
{
  "status": "error",
  "error": {
    "code": "RATE_LIMIT_EXCEEDED",
    "message": "Rate limit exceeded. Try again in 45 seconds.",
    "retryAfter": 45
  }
}
```

6.18.3 Data Encryption

- **In Transit:** All API traffic uses TLS 1.3
- **At Rest:** Sensitive data encrypted with AES-256
- **PHI/PII:** Protected Health Information and Personally Identifiable Information receive additional encryption layers

6.18.4 Audit Logging

All API requests are logged with:

- Request ID
- User ID
- Endpoint accessed
- Request timestamp
- Response status
- IP address
- User agent

Audit logs are retained for 7 years for HIPAA compliance.

6.18.5 HIPAA Compliance

All API endpoints handling Protected Health Information (PHI) comply with HIPAA requirements:

1. **Access Control:** Role-based access enforced on all endpoints
 2. **Audit Controls:** Comprehensive logging of PHI access
 3. **Integrity Controls:** Data validation and checksums
 4. **Transmission Security:** TLS 1.3 encryption
 5. **Authentication:** Strong password requirements and MFA support
-

6.19 API Versioning & Deprecation

6.19.1 Versioning Strategy

CareConnect uses URL path versioning (/api/v1/, /api/v2/). Version increments occur for:

- Breaking changes to request/response formats
- Removal of endpoints
- Changes to authentication mechanism

Non-breaking changes (new optional parameters, new endpoints) do not require version increments.

6.19.2 Deprecation Process

1. **Announcement:** 90 days before deprecation
2. **Warning Headers:** Deprecated endpoints return X-API-Deprecated: true header
3. **Migration Guide:** Provided in API documentation
4. **Sunset Date:** Clearly communicated in advance
5. **Version Support:** Previous version supported for minimum 6 months after new version release

6.20 Error Codes Reference

Error Code	HTTP Status	Description
AUTH_REQUIRED	401	Authentication required
INVALID_TOKEN	401	JWT token invalid or expired
INSUFFICIENT_PERMISSIONS	403	User lacks required role/permission
RESOURCE_NOT_FOUND	404	Requested resource does not exist
VALIDATION_ERROR	422	Request validation failed
DUPLICATE_RESOURCE	409	Resource already exists
RATE_LIMIT_EXCEEDED	429	Too many requests
PAYMENT_REQUIRED	402	Subscription required
SERVICE_UNAVAILABLE	503	Temporary service outage
INTERNAL_ERROR	500	Unexpected server error

6.21 API Testing Recommendations

6.21.1 Unit Testing

The new unit testing framework should include:

1. Endpoint Tests: Verify each endpoint returns expected status codes and response formats
2. RBAC Tests: Validate role-based permissions are enforced correctly
3. Validation Tests: Ensure request validation catches invalid inputs
4. Error Handling Tests: Verify error responses follow standard format

6.21.2 Integration Testing

1. End-to-End Flows: Test complete user workflows (registration → login → task creation → completion)
2. Third-Party Integration: Test Stripe payment processing, wearable device sync
3. Database Integrity: Verify data consistency across related endpoints
4. Performance Testing: Load testing for high-traffic endpoints

6.21.3 Security Testing

1. Authentication Bypass: Attempt to access protected endpoints without valid tokens
2. Authorization Bypass: Attempt to access resources outside assigned role permissions
3. Input Validation: Test SQL injection, XSS, command injection prevention
4. Rate Limit Testing: Verify rate limiting functions correctly

6.22 API Documentation & Developer Resources

6.22.1 Interactive API Documentation

CareConnect provides interactive API documentation via Swagger/OpenAPI at:

- Development: <https://dev-api.careconnect.com/docs>
- Production: <https://api.careconnect.com/docs>

6.22.2 Postman Collection

A complete Postman collection is available for API testing and includes:

- Pre-configured environments (dev, staging, production)
- Sample requests for all endpoints
- Authentication token management
- Example request bodies and responses

6.22.3 Code Examples

SDK libraries and code examples are available for:

- JavaScript/TypeScript (Node.js)
-

- Python
- Java
- Flutter/Dart

6.23 WebSocket APIs (Real-Time Communication)

CareConnect provides WebSocket APIs for real-time bidirectional communication between clients and servers. WebSocket connections enable instant delivery of notifications, live health data updates, task status changes, and real-time messaging without requiring constant polling.

6.23.1 WebSocket Connection

Endpoint: `wss://api.careconnect.com/ws`

Protocol: WebSocket (RFC 6455)

Authentication: JWT token required in connection request

Connection Methods:

Web Browser (JavaScript):

```
javascript
const accessToken = localStorage.getItem('accessToken');
const ws = new
WebSocket(`wss://api.careconnect.com/ws?token=${accessToken}`);

ws.onopen = function(event) {
  console.log('WebSocket connected');

  // Subscribe to channels
  ws.send(JSON.stringify({
    action: 'subscribe',
    channels: ['tasks', 'health', 'messages']
  }));
};

ws.onmessage = function(event) {
  const message = JSON.parse(event.data);
  console.log('Received:', message);

  // Handle different event types
  switch(message.channel) {
    case 'tasks':
      handleTaskUpdate(message.data);
      break;
```

```
        case 'health':
            handleHealthUpdate(message.data);
            break;
        case 'messages':
            handleNewMessage(message.data);
            break;
    }
};

ws.onerror = function(error) {
    console.error('WebSocket error:', error);
};

ws.onclose = function(event) {
    console.log('WebSocket closed:', event.code, event.reason);
    // Implement reconnection logic
    setTimeout(reconnect, 5000);
};
```

Mobile (Flutter):

```
dart
import 'package:web_socket_channel/web_socket_channel.dart';

final channel = WebSocketChannel.connect(
    Uri.parse('wss://api.careconnect.com/ws?token=$accessToken'),
);

// Subscribe to channels
channel.sink.add(jsonEncode({
    'action': 'subscribe',
    'channels': ['tasks', 'health', 'messages'],
})));

// Listen for messages
channel.stream.listen(
    (message) {
        final data = jsonDecode(message);
        print('Received: $data');
    },
    onError: (error) => print('Error: $error'),
    onDone: () => print('Connection closed'),
);
```

Connection Lifecycle:

1. **Connect:** Client initiates WebSocket connection with JWT token
2. **Authenticate:** Server validates JWT and establishes authenticated session
3. **Subscribe:** Client subscribes to desired channels
4. **Receive:** Server pushes events to client in real-time
5. **Heartbeat:** Periodic ping/pong to maintain connection
6. **Disconnect:** Clean disconnection or automatic reconnection on failure

Message Format:

All WebSocket messages use JSON format with consistent structure:

Client to Server (Actions):

```
json
{
  "action": "subscribe",
  "requestId": "req_12345",
  "params": {
    "channels": ["tasks", "health"],
    "patientId": "usr_patient_456"
  }
}
```

Server to Client (Events):

```
json
{
  "channel": "tasks",
  "event": "task_completed",
  "eventId": "evt_67890",
  "timestamp": "2026-01-31T10:30:00Z",
  "data": {
    "taskId": "task_890",
    "patientId": "usr_patient_456",
    "completedAt": "2026-01-31T10:29:58Z",
    "completedBy": "usr_patient_456"
  }
}
```

Heartbeat (Keep-Alive):

```
json
{
  "type": "ping",
  "timestamp": "2026-01-31T10:30:00Z"
}
```

```
}
```

Client Response:

```
json
{
  "type": "pong",
  "timestamp": "2026-01-31T10:30:00Z"
}
```

6.23.2 Supported Actions

subscribe: Subscribe to channels for real-time updates

Request:

```
json
{
  "action": "subscribe",
  "requestId": "req_001",
  "params": {
    "channels": ["tasks", "health", "messages", "alerts"],
    "patientId": "usr_patient_456"
  }
}
```

Response:

```
json
{
  "action": "subscribe",
  "requestId": "req_001",
  "status": "success",
  "subscribedChannels": ["tasks", "health", "messages", "alerts"],
  "timestamp": "2026-01-31T10:30:00Z"
}
```

unsubscribe: Unsubscribe from channels

Request:

```
json
{
  "action": "unsubscribe",
  "requestId": "req_002",
  "params": {
    "channels": ["health"]
  }
}
```

Response:

```
json
{
  "action": "unsubscribe",
  "requestId": "req_002",
  "status": "success",
  "remainingChannels": ["tasks", "messages", "alerts"],
  "timestamp": "2026-01-31T10:31:00Z"
}
```

update_preferences: Update real-time notification preferences

Request:

```
json
{
  "action": "update_preferences",
  "requestId": "req_003",
  "params": {
    "muteChannels": ["messages"],
    "priority": "high_only"
  }
}
```

6.23.3 Supported Channels and Events

tasks channel: Task-related events

Events:

- **task_created:** New task assigned
 - **task_updated:** Task details changed
 - **task_completed:** Task marked complete
 - **task_missed:** Task deadline passed without completion
 - **task_deleted:** Task removed
-

Example Event:

```
json
{
  "channel": "tasks",
  "event": "task_created",
  "eventId": "evt_task_001",
  "timestamp": "2026-01-31T10:30:00Z",
  "data": {
    "taskId": "task_901",
    "patientId": "usr_patient_456",
    "title": "Morning medication",
    "scheduledTime": "2026-02-01T08:00:00Z",
    "priority": "high",
    "createdBy": "usr_caregiver_123"
  }
}
```

health channel: Real-time health metric updates

Events:

- **metric_recorded**: New health data point
- **metric_anomaly**: Abnormal reading detected
- **device_synced**: Wearable device data synchronized
- **trend_alert**: Concerning trend identified

Example Event:

```
json
{
  "channel": "health",
  "event": "metric_recorded",
  "eventId": "evt_health_002",
  "timestamp": "2026-01-31T10:30:15Z",
  "data": {
    "metricId": "metric_456",
    "patientId": "usr_patient_456",
    "metricType": "blood_pressure",
    "value": {
      "systolic": 142,
      "diastolic": 88
    },
    "unit": "mmHg",
    "source": "manual",
    "status": "elevated",
  }
}
```

```
    "alert": true
  }
}
```

messages channel: Real-time messaging

Events:

- **message_sent**: New message received
- **message_read**: Message marked as read
- **typing_indicator**: User is typing

Example Event:

```
json
{
  "channel": "messages",
  "event": "message_sent",
  "eventId": "evt_msg_003",
  "timestamp": "2026-01-31T10:30:30Z",
  "data": {
    "messageId": "msg_789",
    "conversationId": "conv_123",
    "senderId": "usr_caregiver_123",
    "senderName": "Dr. Sarah Johnson",
    "messageText": "How are you feeling today?",
    "sentAt": "2026-01-31T10:30:30Z"
  }
}
```

alerts channel: Urgent notifications and system alerts

Events:

- **health_alert**: Critical health condition detected
- **emergency**: Emergency situation requiring immediate attention
- **system_alert**: Important system notification
- **appointment_reminder**: Upcoming appointment

Example Event:

```
json
{
  "channel": "alerts",
  "event": "health_alert",
```

```
"eventId": "evt_alert_004",
"timestamp": "2026-01-31T10:30:45Z",
"priority": "urgent",
"data": {
  "alertId": "alert_901",
  "patientId": "usr_patient_456",
  "type": "high_blood_pressure",
  "severity": "high",
  "message": "Blood pressure reading significantly elevated: 165/95
mmHg",
  "recommendedAction": "Contact patient immediately",
  "metricId": "metric_457"
}
```

notifications channel: General application notifications

Events:

- notification_received: New notification
- notification_dismissed: Notification cleared

video_call channel: Video call signaling

Events:

- call_incoming: Incoming call request
- call_accepted: Call accepted by recipient
- call_rejected: Call declined
- call_ended: Call terminated

6.23.4 Connection Management

Authentication:

- JWT token required in connection URL or initial message
- Token validated on connection establishment
- Connection terminated if token invalid or expired
- Token refresh requires reconnection

Connection Limits:

- Maximum 5 concurrent connections per user
-

- Oldest connection auto-disconnected when limit exceeded
- Connection limit prevents resource exhaustion

Heartbeat / Keep-Alive:

- Server sends ping every 30 seconds
- Client must respond with pong within 10 seconds
- Connection closed if 3 consecutive pings missed
- Prevents zombie connections

Reconnection Strategy:

- Exponential backoff: 1s, 2s, 4s, 8s, 16s, 30s (max)
- Automatic reconnection on connection loss
- Resubscribe to previous channels on reconnect
- Missed events retrieved via REST API

Connection States:

- **connecting**: Initial connection in progress
- **open**: Connected and authenticated
- **subscribed**: Subscribed to one or more channels
- **closing**: Graceful shutdown in progress
- **closed**: Connection terminated

6.23.5 Error Handling

Error Message Format:

```
json
{
  "type": "error",
  "errorCode": "AUTHENTICATION_FAILED",
  "message": "Invalid or expired authentication token",
  "requestId": "req_001",
  "timestamp": "2026-01-31T10:30:00Z"
}
```

Rate Limiting:

- Maximum 100 messages per minute per connection
 - Burst allowance: 20 messages in 10 seconds
 - Exceeding limit results in temporary throttling
 - Excessive violations cause connection termination
-

6.23.6 Performance and Scalability

Connection Capacity:

- Supports 100,000 concurrent connections per region
- Auto-scaling based on connection count
- Geographic distribution via CloudFront

Message Latency:

- Average: < 100ms from event to delivery
- p95: < 500ms
- p99: < 1000ms

Throughput:

- 10,000 messages per second per channel
- Fan-out to multiple subscribers in parallel

Data Transfer:

- Compression enabled (WebSocket permessage-deflate)
- Binary encoding option for large payloads
- Message size limit: 128KB

6.23.7 Security Considerations

Connection Security:

- WSS protocol (WebSocket over TLS 1.3)
- Certificate pinning on mobile apps
- No unencrypted WS connections allowed

Authentication:

- JWT validation on every connection
- Token expiration enforced
- Session binding to connection ID

Authorization:

- Channel subscriptions validated against user permissions
 - Caregivers can only subscribe to assigned patients
 - Patients can only subscribe to own data
 - Data filtering at source (no unauthorized data sent)
-

Message Security:

- All messages encrypted in transit
- Sensitive PHI redacted from messages
- Audit logging of all WebSocket activity
- Anomaly detection for suspicious patterns

Compliance:

- HIPAA-compliant WebSocket implementation
- All connections logged for audit
- Message delivery tracked
- Encryption meets regulatory requirements

6.23.8 Testing Criteria*AC-001: Successful WebSocket connection*

- Given: User has valid access token
- When: User initiates WebSocket connection
- Then:
 - Connection established successfully
 - Authentication validated
 - Connection ID assigned
 - Ready to subscribe to channels

AC-002: Real-time task update delivery

- Given: Caregiver subscribed to "tasks" channel
- When: Patient completes assigned task
- Then:
 - Task completion event sent via WebSocket
 - Caregiver receives update within 1 second
 - Dashboard updates automatically
 - No page refresh required

AC-003: Channel authorization enforced

- Given: Caregiver A assigned to Patient 1
 - When: Caregiver A attempts to subscribe to Patient 2 health channel
 - Then:
 - Subscription rejected
 - Error indicates insufficient permissions
 - No Patient 2 data transmitted
 - Connection remains active for authorized channels
-

AC-004: Automatic reconnection on disconnect

- Given: WebSocket connection established and subscribed
- When: Network interruption occurs
- Then:
 - Client detects disconnection
 - Automatic reconnection attempted
 - Exponential backoff applied
 - Previous subscriptions restored
 - Missed events retrieved via REST API

6.23.9 AWS Architecture Notes

Infrastructure:

- Service: Amazon API Gateway WebSocket APIs
- Compute: AWS Lambda (connection/disconnection handlers)
- Compute: AWS Lambda (message routing)
- Message Broker: Amazon EventBridge (event ingestion)
- Connections Table: DynamoDB (active connections)
- Authorization: Lambda Authorizer (JWT validation)
- Monitoring: CloudWatch Logs and Metrics
- Alarms: Connection count, error rates, latency

Lambda Functions:

- websocket-connect-handler (Node.js 18, 512MB, 30s)
- websocket-disconnect-handler (Node.js 18, 256MB, 10s)
- websocket-message-handler (Node.js 18, 512MB, 30s)
- websocket-event-publisher (Node.js 18, 1GB, 1min)

DynamoDB Table (websocket_connections):

Partition Key: `connectionId`

Sort Key: `userId`

Attributes:

- `subscriptions` (List)
- `connectedAt` (Timestamp)
- `lastActivityAt` (Timestamp)
- `deviceType` (String)
- `tTL` (Number, auto-deletion after 24h inactivity)

Cost Optimization:

- Pay per message and connection minute
 - No idle capacity costs
-

- Automatic scaling included
- Connection cleanup removes stale connections

7. Data Models

This section summarizes the CareConnect data model and database design. It is adapted from Section 7 (Data Models) of the SWEN 670 Fall 2025 Technical Design Document done by the previous cohort and has been revised to align with the current Spring 2026 scope, architecture, and technology decisions. Amazon RDS (PostgreSQL) is the primary relational store. DynamoDB supports sessions, user preferences, and notification queues. Timestream stores time-series wearable metrics. Redis (ElastiCache) provides caching for frequently accessed data. The model covers entities, relationships, normalization, indexing, and AWS-specific considerations. (TDD §6.1.2)

Note: Storage targets follow the current TDD polyglot persistence decisions. (TDD §6.1.2)

7.1 Scope

This section is organized into four areas:

- **Conceptual Model:** Identifies high-level entities and their relationships, providing a broad view without implementation details.
- **Logical Model:** Refines the conceptual design by normalizing it into tables with defined attributes, ensuring consistency and minimizing redundancy.
- **Physical Model:** Translates the design into PostgreSQL DDL, including schemas, types, constraints, and indexes for integrity and performance.
 - Note: Physical conventions align to PostgreSQL rather than prior MySQL assumptions. (TDD §6.1.2)
- **AWS Deployment:** Covers AWS database deployment using RDS (PostgreSQL) or Aurora PostgreSQL, including security, backups, and cloud integration for scalability and reliability.

7.2 Conceptual Data Model

7.2.1 Key Entities

User: core identity (patients, caregivers, family, admins)

- **Tenant:** organizational boundary for multi-tenant RBAC
 - **Role:** security roles (RBAC)
 - **Permission:** granular access rights bound to roles
 - **Profile:** PatientProfile, CaregiverProfile
 - **Subscription:** plan enrollment and billing
 - **Payment:** payment methods & transactions
 - **Task / Notification:** scheduling & reminders
 - **Health Data:** symptoms, symptom logs, meals, moods, medications
-

- **Appointments:** scheduled visits and follow-ups
- **Audit Log:** security/audit events
- **Transcriptions:** AI-generated clinical transcription metadata
-

Note: Appointments, medications, audit logs, and transcription records are retained to align with the TDD feature set and API examples. (TDD §6.1.2; §6.6.2; §6.9.3)

WearableDevice / WearableMetric

- **Communication:** messages, media files, telehealth sessions
- **Achievement:** gamification

7.2.2 High-Level Relationships

This section summarizes primary entity relationships at a high level. It describes how components interact and provides an overview across one-to-one and many-to-many associations.

- **A User can hold multiple Roles (many-to-many):** Users may be assigned more than one role to represent different access levels or permissions.
- **A User belongs to a Tenant (many-to-one):** Tenant scoping isolates roles, permissions, and data.
- **Roles and Permissions are tenant-scoped:** Role/permission associations are isolated per tenant to prevent cross-tenant access.
- **A PatientProfile is a one-to-one extension of User:** A user can have a single patient profile with patient-specific details.
- **CaregiverProfile links to multiple patients (many-to-many):** Caregivers and patients can be associated in a flexible many-to-many mapping.
- **A User may have multiple Subscriptions → one Plan:** A user can hold multiple subscriptions, each tied to a plan.
- **TaskTemplate → Task → Notification:** Templates define tasks, and task events trigger notifications.
- **PatientSymptom → SymptomLog:** Patient symptoms can be tracked through symptom logs over time.
- **PatientMealQuestion → MealEntry:** Assigned meal questions link to meal entries that record responses.
- **WearableDevice → WearableMetric:** Devices emit metrics used to monitor patient health.
- **PatientProfile → Appointment:** Appointments are scheduled for patients with providers and time slots.
- **PatientProfile → Medication:** Patient medication records support reminders and tracking.
- **User → AuditLog:** Security and compliance events link to the acting user when available.
- **Transcription → PatientProfile:** Transcription metadata is tied to patient records for clinical review.

7.3 Logical Data Model

Tables target 3NF. Sensitive fields (e.g., medical_notes) use TEXT or JSON; enumerations use PostgreSQL enums (or CHECK constraints where enums are not ideal).

Note: Apply tenant scoping consistently, either via tenant_id on domain tables or through joins to app_user/role tables. (SRS §3.3.4)

7.3.1 Core Tables

Table	Description
app_user	Identity: email, auth provider/subject or password hash, status
role	RBAC roles
user_role	join table
patient_profile	demographics, PHI notes
caregiver_profile	license, type
caregiver_patient	mapping caregiver ↔ patient

7.3.2 RBAC & Tenancy

Table	Description
tenant	tenant/organization boundary
permission	atomic access rights
role_permission	join table (role ↔ permission)

Note: Tenant, permission, and role-permission tables are retained to satisfy the multi-tenant RBAC requirement. (SRS §3.3.2–3.3.4)

7.3.3 Billing

Table	Description
plan	subscription plans
subscription	user enrollments
payment_method	tokenized payment methods
payment	payment attempts

7.3.4 Scheduling & Notifications

Table	Description
task_template	predefined reminders

task	scheduled tasks
notification	sends (push, SMS, email)

7.3.5 Care Management (Appointments & Medications)

Table	Description
appointment	scheduled visits and follow-ups
medication	medication master records
patient_medication	patient-specific medication regimen

Note: Care management tables are retained to align with TDD examples that include appointments and medications. (TDD §6.6.2)

7.3.6 Health Tracking

Table	Description
symptom	master list
patient_symptom	active symptoms per patient
symptom_log	scores & notes
meal_question	question bank
patient_meal_question	assigned questions
meal_entry	patient responses
mood_entry	mood diary

7.3.7 Wearables

Table	Description
wearable_device	linked device records
wearable_metric	time-series metrics (Timestream)

7.3.8 Communication & Media

Table	Description
message	in-app messaging
media_file	uploads, scans

telehealth_session	virtual visit records
webhook_subscription	outbound webhook registrations

Note: Webhook registrations are stored to support Webhook Configuration APIs. (TDD §6.17.1)

7.3.9 Audit & Transcriptions

Table	Description
audit_log	security and compliance events
transcription	AI transcription metadata and status

7.3.10 Gamification

Table	Description
achievement	badges & points
user_achievement	earned badges

7.4 Non-Relational Stores

Note: Non-relational stores reflect the polyglot persistence split defined in the TDD. (TDD §6.1.2)

7.4.1 DynamoDB Collections

- **sessions**: pk=session_id → user_id, expires_at, token metadata
- **user_preferences**: pk=user_id → preferences JSON, updated_at
- **notification_deliveries**: pk=recipient_id, sk=notification_id → channel, status, delivered_at, payload refs
- **websocket_connections**: pk=connection_id, sk=user_id → subscriptions, connected_at, last_activity_at, device_type, ttl

Note: WebSocket connection metadata is stored in DynamoDB for real-time channel management. (TDD §6.23.9)

7.4.2 Timestream

- **wearable_metrics**: dimensions (patient_id, device_id, metric_type), measures (value, unit), time (recorded_at)

7.4.3 Redis (ElastiCache)

- **cache**: ephemeral cache of frequently accessed data; no durable records

7.5 Data Dictionary

The Data Dictionary describes the database schema by listing each column in the relevant tables.

For each column, the name, data type, key status (PK/FK), and purpose are captured to keep storage and relationships clear for implementation.

Timestamp Convention: Use TIMESTAMPTZ for date/time fields in PostgreSQL unless a field is explicitly date-only.

7.5.1 Tenant

Entity Name: Tenant

Entity Description: Logical organization boundary used to scope RBAC data.

Main Attributes of Tenant:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique tenant identifier.	PK
name	VARCHAR(100)	Tenant display name.	NOT NULL
slug	VARCHAR(64)	Short unique key.	UQ
created_at	TIMESTAMPTZ	Creation timestamp (UTC).	—
updated_at	TIMESTAMPTZ	Last updated timestamp (UTC).	—

7.5.2 User

Entity Name: User

Entity Description: The user of the CareConnect application. Can be a caregiver, patient, or family member.

Main Attributes of User:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique user identifier.	PK
tenant_id	BIGINT	Tenant boundary for multi-tenant RBAC.	FK to tenant (id)
email	VARCHAR(254)	Login email	UQ (tenant_id, email)

		address.	
auth_provider	VARCHAR(32)	Authentication provider (e.g., LOCAL, COGNITO).	NOT NULL
auth_subject	VARCHAR(255)	Provider user id (e.g., Cognito sub). Nullable for LOCAL auth.	UQ (tenant_id, auth_provider, auth_subject)
password_hash	CHAR(60)	Hashed password for LOCAL auth.	NULL unless auth_provider=LOCAL
status	account_status (enum)	Account status.	ACTIVE/SUSPENDED
created_at	TIMESTAMPTZ	Record creation timestamp (UTC).	—
updated_at	TIMESTAMPTZ	Last updated timestamp (UTC).	—

7.5.3 Role

Entity Name: Role

Entity Description: The role of the user.

Main Attributes of Role:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique role identifier.	PK
tenant_id	BIGINT	Tenant boundary.	FK to tenant (id)
name	VARCHAR(50)	Role name (e.g. ADMIN, PATIENT).	UQ (tenant_id, name)

7.5.4 Permission

Entity Name: Permission

Entity Description: Atomic access right bound to a tenant.

Main Attributes of Permission:

Name	Type	Description	Constraints

id	BIGSERIAL	Unique permission identifier.	PK
tenant_id	BIGINT	Tenant boundary.	FK to tenant (id)
code	VARCHAR(100)	Permission code (e.g., TASK_READ).	UQ (tenant_id, code)
description	TEXT	Human-readable description.	—

7.5.5 Role Permission

Entity Name: Role Permission

Entity Description: Junction table between Role and Permission.

Main Attributes of Role Permission:

Name	Type	Description	Constraints
role_id	BIGINT	Role identifier.	PK, FK to role (id)
permission_id	BIGINT	Permission identifier.	PK, FK to permission (id)

7.5.6 User Role

Entity Name: User Role

Entity Description: The junction between User entity and Role entity.

Main Attributes of User Role:

Name	Type	Description	Constraints
user_id	BIGINT	Unique user identifier.	PK, FK to app_user (id)
role_id	BIGINT	Unique role identifier.	PK, FK to role (id)

7.5.7 Patient Profile

Entity Name: Patient Profile

Entity Description: Represents the patient user's profile information.

Main Attributes of Patient Profile:

Name	Type	Description	Constraints
user_id	BIGINT	Unique user	PK, FK to app_user

		identifier.	(id)
dob	DATE	Patient's date of birth.	—
sex	VARCHAR(10)	Sex of patient.	M/F/OTHER
medical_notes	TEXT	Free-text PHI notes.	—

7.5.8 Caregiver Profile

Entity Name: Caregiver Profile

Entity Description: Represents the caregiver user's profile information.

Main Attributes of Caregiver Profile:

Name	Type	Description	Constraints
user_id	BIGINT	Unique user identifier.	PK, FK to app_user (id)
type	caregiver_type (enum)	Type of caregiver	PROFESSIONAL/FAMILY
license_number	VARCHAR(100)	License number	—

7.5.9 Caregiver Patient

Entity Name: Caregiver Patient

Entity Description: Represents the link between a caregiver and a patient.

Main Attributes of Caregiver Patient:

Name	Type	Description	Constraints
caregiver_user_id	BIGINT	Caregiver's user id.	PK, FK to caregiver_profile(user_id)
patient_user_id	BIGINT	Patient's user id.	PK, FK to patient_profile(user_id)

role	VARCHAR(20)	Role.	PRIMARY/SECONDARY.
linked_at	TIMESTAMP Z	Timestamp p of link.	—

7.5.10 Plan

Entity Name: Plan

Entity Description: Represents the CareConnect plan type for billing purposes.

Main Attributes of Plan:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique plan identifier.	PK
code	VARCHAR(50)	Plan code (e.g. BASIC)	UQ
name	VARCHAR(100)	Human-readable name.	—
price_cents	INT	Price in cents.	—
billing_period	billing_period (enum)	Billing period of time.	MONTH.

7.5.11 Subscription

Entity Name: Subscription

Entity Description: Represents the user's subscription

Main Attributes of Subscription:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique subscription identifier.	PK
user_id	BIGINT	Subscrib	FK to app_user (id)

		er	
plan_id	BIGINT	Chosen plan	FK to plan (id)
status	subscription_status (enum)	Subscription status.	ACTIVE/SUSPENDED/GRACE/CANCELLED
started_at	TIMESTAMPTZ	Start time of subscription.	—
current_period_end	TIMESTAMPTZ	End time of subscription.	—

7.5.12 Payment Method

Entity Name: Payment Method

Entity Description: Represents the user's payment method (tokenized identifiers only; no raw card data stored).

Main Attributes of Payment Method:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique payment identifier.	PK
user_id	BIGINT	Owner.	FK to app_user (id)
provider	VARCHAR(20)	Payment provider.	STRIPE
stripe_payment_method_id	VARCHAR(255)	External token for Stripe (e.g., pm_*)	—
last4	CHAR(4)	Last four digits of payment method.	—
brand	VARCHAR(20)	Card brand.	—
exp_month	SMALLINT	Expiry month.	—
exp_year	SMALLINT	Expiry year.	—
created_at	TIMESTAMPTZ	Timestamp payment was added.	—

Note: Stripe is the payment provider; store tokenized identifiers only. (TDD §5.2.1)

7.5.13 Payment

Entity Name: Payment

Entity Description: Represents the payment transaction.

Main Attributes of Payment:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique payment identifier.	PK
subscription_id	BIGINT	Subscription identifier.	FK to subscription (id)
payment_method_id	BIGINT	Payment method identifier.	FK to payment_method (id)
amount_cents	INT	Amount charged to user.	—
status	payment_status (enum)	Payment status.	SUCCEEDED/FAILED
attempted_at	TIMESTAMPTZ	Payment attempt timestamp.	—

7.5.14 Task Template

Entity Name: Task Template

Entity Description: Represents the task template.

Main Attributes of Task Template:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique template identifier.	PK
name	VARCHAR(100)	Template name.	—
category	VARCHAR	Task category.	MEDICATION/MEAL/...
default_payload	JSON	Default JSON.	—

7.5.15 Task

Entity Name: Task

Entity Description: Represents the task.

Main Attributes of Task:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique task identifier.	PK
patient_id	BIGINT	Patient.	FK to patient_profile (user_id)
template_id	BIGINT	Template.	FK to task_template (id)
title	VARCHAR(150)	Custom title.	—
frequency	task_frequency (enum)	Frequency of task.	ONCE/DAILY/...
start_time	TIMESTAMPTZ	When to start the task.	—
next_run	TIMESTAMPTZ	When to next run the task.	—
created_by	BIGINT	Who created the task.	FK to app_user (id)

7.5.16 Notification

Entity Name: Notification

Entity Description: Represents the notification.

Main Attributes of Notification:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique notification identifier.	PK
user_id	BIGINT	User id of recipient.	FK to app_user (id)
task_id	BIGINT	Task identifier.	FK to task (id)

type	VARCHAR(50)	Type of task.	REMINDER.
channel	notification_channel (enum)	Type of channel notification	PUSH/SMS/EMAIL
sent_at	TIMESTAMPTZ	Notification send time.	—
delivered_at	TIMESTAMPTZ	Notification delivery time.	—

7.5.17 Appointment

Entity Name: Appointment

Entity Description: Represents scheduled patient visits and follow-ups.

Main Attributes of Appointment:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique appointment identifier.	PK
patient_id	BIGINT	Patient user id.	FK to patient_profile (user_id)
caregiver_id	BIGINT	Caregiver user id (if assigned).	FK to caregiver_profile (user_id)
title	VARCHAR(150)	Appointment title.	—
scheduled_time	TIMESTAMPTZ	Scheduled time (UTC).	—
provider	VARCHAR(150)	Provider name.	—
location	VARCHAR(255)	Location or clinic address.	—
status	VARCHAR(30)	Status of appointment.	SCHEDULED/COMPLETED/CANCELLED

7.5.18 Medication

Entity Name: Medication

Entity Description: Represents a medication record.

Main Attributes of Medication:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique medication identifier.	PK
name	VARCHAR(150)	Medication name.	—
description	TEXT	Medication description/instructions.	—

7.5.19 Patient Medication**Entity Name:** Patient Medication**Entity Description:** Links patient to medication regimen.

Main Attributes of Patient Medication:

Name	Type	Description	Constraints
patient_id	BIGINT	Patient user id.	PK, FK to patient_profile (user_id)
medication_id	BIGINT	Medication identifier.	PK, FK to medication (id)
dosage	VARCHAR(50)	Dosage (e.g., 10mg).	—
frequency	VARCHAR(50)	Frequency (e.g., daily).	—
start_date	DATE	Regimen start date.	—
end_date	DATE	Regimen end date.	—

7.5.20 Symptom**Entity Name:** Symptom**Entity Description:** Represents the symptom of the patient.

Main Attributes of Symptom:

Name	Type	Description	Constraints
id	BIGSERIAL	Unique symptom identifier.	PK
name	VARCHAR(100)	Symptom name.	UQ
critical_score	INT	Critical symptom score threshold.	—

7.5.21 Patient Symptom

Entity Name: Patient Symptom

Entity Description: Links patient to symptom.

Main Attributes of Patient Symptom:

Name	Type	Description	Constraints
patient_id	BIGINT	Patient identifier.	PK, FK to patient_profile (user_id)
symptom_id	BIGINT	Symptom name.	PK, FK to symptom (id)
active	BOOLEAN	If symptom is occurring or not.	TRUE/FALSE
started_at	DATE	Activation date.	

7.5.22 Symptom Log

Entity Name: Symptom Log

Entity Description: Represents the symptom log of the patient.

Main Attributes of Symptom Log:

Name	Type	Description	Constraints
id	BIGSERIAL	Log identifier.	PK
patient_id	BIGINT	Patient identifier.	PK, FK to patient_profile (user_id)
symptom_id	BIGINT	Symptom name.	PK, FK to symptom (id)
score	INT	Symptom score severity.	CHECK (score BETWEEN 0 AND 10)
notes	TEXT	Symptom notes.	Free-text.
logged_at	TIMESTAMPTZ	Time of logged symptoms.	

7.5.23 Meal Question

Entity Name: Meal Question

Entity Description: Represents a meal question.

Main Attributes of Meal Question:

Name	Type	Description	Constraints

id	BIGSERIAL	Question identifier.	PK
text	VARCHAR(255)	Question prompt.	—

7.5.24 Patient Meal Question

Entity Name: Patient Meal Question

Entity Description: Represents a patient's meal question.

Main Attributes of Patient Meal Question:

Name	Type	Description	Constraints
patient_id	BIGINT	Patient identifier.	PK, FK to patient_profile.
question_id	BIGINT	Meal question identifier.	PK, FK to meal_question
added_by	BIGINT	Which caregiver added the meal question.	FK to caregiver_profile
added_at	TIMESTAMPTZ	Question added time stamp.	—

7.5.25 Meal Entry

Entity Name: Meal Entry

Entity Description: Represents a patient's meal entry.

Main Attributes of Meal Entry:

Name	Type	Description	Constraints
id	BIGSERIAL	Meal entry identifier.	PK
patient_id	BIGINT	Patient identifier.	PK, FK to patient_profile.
question_id	BIGINT	Meal question	PK, FK to meal_question

		identifier.	
response_text	TEXT	Answer text.	—
response_voice_url	VARCHAR	Audio URL.	—
response_image_url	VARCHAR	Photo URL.	—
logged_at	TIMESTAMPTZ	Time answer was logged.	—

7.5.26 Mood Entry

Entity Name: Mood Entry

Entity Description: Represents a patient's mood entry.

Main Attributes of Mood Entry:

Name	Type	Description	Constraints
id	BIGSERIAL	Mood entry identifier.	PK.
patient_id	BIGINT	Patient identifier.	FK to patient_profile.
mood	mood_type (enum)	Mood of patient.	—
notes	TEXT	Notes text.	Free-text.
logged_at	TIMESTAMPTZ	Time answer was logged.	—

7.5.27 Wearable Device

Entity Name: Wearable Device

Entity Description: Represents a wearable device.

Main Attributes of Wearable Device:

Name	Type	Description	Constraints
id	BIGSERIAL	Device identifier.	PK
user_id	BIGINT	User identifier.	FK to app_user.
provider	wearable_provider (enum)	Device provider.	FITBIT/...
external_id	VARCHAR(128)	Vendor identifier.	—
linked_at	TIMESTAMPTZ	Link time.	—
revoked_at	TIMESTAMPTZ	Unlink time.	—

7.5.28 Wearable Metric

Entity Name: Wearable Metric

Entity Description: Time-series metrics from wearable devices (stored in Timestream; relational metadata optional).

Main Attributes of Wearable Metric:

Name	Type	Description	Constraints
id	BIGSERIAL	Metric identifier.	PK
device_id	BIGINT	User identifier.	FK to wearable_device.
metric	metric_type (enum)	Metric.	HEART_RATE/...
value	NUMERIC	Measured metric values.	—
recorded_at	TIMESTAMPTZ	Recorded time metrics.	—

7.5.29 Message

Entity Name: Message

Entity Description: Represents a message.

Main Attributes of Message:

Name	Type	Description	Constraints
id	BIGSERIAL	Message identifier.	PK
sender_id	BIGINT	Sender identifier.	FK to app_user.
receiver_id	BIGINT	Receiver identifier.	FK to app_user.
msg_type	message_type (enum)	Message text.	TEXT/AUDIO/...
content	TEXT	Body or URL.	—
sent_at	TIMESTAMPTZ	Send time.	—

7.5.30 Media File

Entity Name: Media File

Entity Description: Represents a media file.

Main Attributes of Media File:

Name	Type	Description	Constraints
id	BIGSERIAL	File identifier.	PK

uploader_id	BIGINT	Uploader identifier.	FK to app_user.
patient_id	BIGINT	Patient identifier.	FK to patient_profile.
file_type	VARCHAR(40)	File type.	MIME type
object_key	VARCHAR(512)	Object key.	S3 key
size_bytes	BIGINT	File size.	—
uploaded_at	TIMESTAMPTZ	Media upload time.	

7.5.31 Webhook Subscription

Entity Name: Webhook Subscription

Entity Description: Stores outbound webhook registrations for event notifications.

Main Attributes of Webhook Subscription:

Name	Type	Description	Constraints
id	BIGSERIAL	Webhook identifier.	PK
tenant_id	BIGINT	Tenant boundary.	FK to tenant.
owner_user_id	BIGINT	Owner user identifier.	FK to app_user.
url	VARCHAR(512)	Webhook endpoint URL.	—
events	JSONB	Subscribed event list.	—
secret	TEXT	Shared secret (store encrypted).	—
description	TEXT	Integration description.	—
status	VARCHAR(20)	Lifecycle status.	ACTIVE/INACTIVE
created_at	TIMESTAMPTZ	Creation time.	—
updated_at	TIMESTAMPTZ	Last updated time.	—

7.5.32 Audit Log

Entity Name: Audit Log

Entity Description: Records security and compliance events.

Main Attributes of Audit Log:

Name	Type	Description	Constraints

id	BIGSERIAL	Audit event identifier.	PK
tenant_id	BIGINT	Tenant boundary.	FK to tenant (id)
actor_user_id	BIGINT	Acting user (if applicable).	FK to app_user (id)
action	VARCHAR(100)	Action performed.	—
resource_type	VARCHAR(100)	Resource type.	—
resource_id	VARCHAR(100)	Resource identifier.	—
outcome	VARCHAR(30)	Allow/Deny/Fail outcome.	—
created_at	TIMESTAMPTZ	Event timestamp (UTC).	—
metadata	JSONB	Additional context.	—

7.5.33 Transcription

Entity Name: Transcription

Entity Description: Represents AI-generated transcription metadata.

Main Attributes of Transcription:

Name	Type	Description	Constraints
id	BIGSERIAL	Transcription identifier.	PK
patient_id	BIGINT	Patient user id.	FK to patient_profile (user_id)
created_by	BIGINT	Creator user id.	FK to app_user (id)
source_object_key	VARCHAR(512)	S3 object key for source audio.	—
status	VARCHAR(30)	Processing status.	QUEUED/PROCESSING/COMPLETED/FAILED
created_at	TIMESTAMPTZ	Creation timestamp (UTC).	—
completed_at	TIMESTAMPTZ	Completion timestamp (UTC).	—

entities	JSONB	Extracted medical entities.	—

7.5.34 Telehealth Session

Entity Name: Telehealth Session

Entity Description: Represents a telehealth session.

Main Attributes of Telehealth Session:

Name	Type	Description	Constraints
id	BIGSERIAL	Session identifier.	PK
creator_id	BIGINT	Creator identifier.	FK to caregiver_profile.
patient_id	BIGINT	Patient identifier.	FK to patient_profile.
zoom_meeting_id	VARCHAR(120)	External meeting identifier.	—
starts_at	TIMESTAMPTZ	Start time.	—
ends_at	TIMESTAMPTZ	End time.	—
status	session_status (enum)	Telehealth session status.	SCHEDULED/...

7.5.35 Achievement

Entity Name: Achievement

Entity Description: Represents an achievement.

Main Attributes of Achievement:

Name	Type	Description	Constraints
id	BIGSERIAL	Achievement identifier.	PK
code	VARCHAR(50)	Badge code.	UQ
title	VARCHAR(100)	Display name.	—
points	INT	Points awarded amount.	—

7.5.36 User Achievement

Entity Name: User Achievement

Entity Description: Represents a user achievement.

Main Attributes of User Achievement:

Name	Type	Description	Constraints
user_id	BIGINT	User identifier.	PK, FK to app_user.
achievement_id	BIGINT	Achievement identifier.	PK, FK to achievement.
earned_at	TIMESTAMPTZ	Time achievement was earned.	—

7.5.37 Location

Entity Name: Location

Entity Description: Represents a location.

Main Attributes of Location:

Name	Type	Description	Constraints
location_id	BIGINT	location identifier.	PK
latitude_position	DOUBLE PRECISION	Latitude position.	—
longitude_position	DOUBLE PRECISION	Longitude position.	—

7.6 Physical Data Model

7.6.1 Schemas & Convention

This section outlines the database schema design conventions and standards that will be followed to ensure consistency, type safety, and efficient data storage. The following conventions will be applied:

- **PostgreSQL (RDS)** is the primary relational store. Encoding is UTF8.
 - **Primary keys** use BIGSERIAL where auto-increment is needed; foreign keys BIGINT: Foreign keys will be defined as BIGINT to maintain consistency with primary key types.
 - **Timestamps** use TIMESTAMPTZ for UTC timestamps; defaults to NOW() at UTC: This helps avoid time zone issues and provides a standardized format for timestamp storage.
 - **JSON data stored in JSONB:** JSONB provides indexing and performance benefits in PostgreSQL.
 - **Enumerations defined as PostgreSQL enums (or CHECK constraints)** for type safety: This prevents invalid data entries by restricting values to allowed values.
-

7.6.2 Enum Types

```
CREATE TYPE account_status AS ENUM ('ACTIVE','SUSPENDED');
```

```
CREATE TYPE caregiver_type AS ENUM ('PROFESSIONAL','FAMILY');
```

```
CREATE TYPE subscription_status AS ENUM  
('ACTIVE','SUSPENDED','GRACE','CANCELLED');
```

```
CREATE TYPE payment_status AS ENUM ('SUCCEEDED','FAILED');
```

```
CREATE TYPE billing_period AS ENUM ('MONTH');
```

```
CREATE TYPE task_frequency AS ENUM ('ONCE','DAILY','WEEKLY');
```

```
CREATE TYPE notification_channel AS ENUM ('PUSH','SMS','EMAIL');
```

```
CREATE TYPE mood_type AS ENUM ('HAPPY','NEUTRAL','SAD','ANXIOUS','OTHER');
```

```
CREATE TYPE wearable_provider AS ENUM  
('FITBIT','APPLE_HEALTHKIT','GOOGLE_HEALTH_CONNECT','OTHER');
```

```
CREATE TYPE metric_type AS ENUM ('HEART_RATE','STEPS','SPO2');
```

```
CREATE TYPE message_type AS ENUM ('TEXT','VOICE','VIDEO');
```

```
CREATE TYPE session_status AS ENUM ('SCHEDULED','LIVE','ENDED','CANCELLED');
```

7.6.3 Table DDL

– Core tenancy & RBAC (PostgreSQL)

```
CREATE TABLE tenant ( id BIGSERIAL PRIMARY KEY, name VARCHAR(100) NOT  
NULL, slug VARCHAR(64) UNIQUE NOT NULL, created_at TIMESTAMPTZ NOT NULL  
DEFAULT NOW(), updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW() );
```

```
CREATE TABLE app_user ( id BIGSERIAL PRIMARY KEY, tenant_id BIGINT NOT NULL  
REFERENCES tenant(id), email VARCHAR(254) NOT NULL, auth_provider VARCHAR(32)  
NOT NULL, auth_subject VARCHAR(255), password_hash CHAR(60), status account_status  
NOT NULL DEFAULT 'ACTIVE', created_at TIMESTAMPTZ NOT NULL DEFAULT  
NOW(), updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(), UNIQUE (tenant_id,  
email), UNIQUE (tenant_id, auth_provider, auth_subject), CHECK (auth_provider IN  
( 'LOCAL', 'COGNITO' )), CHECK ((auth_provider = 'LOCAL' AND password_hash IS NOT  
NULL) OR (auth_provider <> 'LOCAL' AND auth_subject IS NOT NULL)) );
```

```
CREATE TABLE role ( id BIGSERIAL PRIMARY KEY, tenant_id BIGINT NOT NULL  
REFERENCES tenant(id), name VARCHAR(50) NOT NULL, UNIQUE (tenant_id, name) );
```

```
CREATE TABLE permission ( id BIGSERIAL PRIMARY KEY, tenant_id BIGINT NOT NULL REFERENCES tenant(id), code VARCHAR(100) NOT NULL, description TEXT, UNIQUE (tenant_id, code) );
```

```
CREATE TABLE role_permission ( role_id BIGINT NOT NULL REFERENCES role(id) ON DELETE CASCADE, permission_id BIGINT NOT NULL REFERENCES permission(id) ON DELETE CASCADE, PRIMARY KEY (role_id, permission_id) );
```

```
CREATE TABLE user_role ( user_id BIGINT NOT NULL REFERENCES app_user(id) ON DELETE CASCADE, role_id BIGINT NOT NULL REFERENCES role(id) ON DELETE CASCADE, PRIMARY KEY (user_id, role_id) );
```

– Profiles & linking

```
CREATE TABLE patient_profile ( user_id BIGINT PRIMARY KEY REFERENCES app_user(id) ON DELETE CASCADE, dob DATE, sex VARCHAR(10), medical_notes TEXT );
```

```
CREATE TABLE caregiver_profile ( user_id BIGINT PRIMARY KEY REFERENCES app_user(id) ON DELETE CASCADE, type caregiver_type NOT NULL, license_number VARCHAR(100) );
```

```
CREATE TABLE caregiver_patient ( caregiver_user_id BIGINT NOT NULL REFERENCES caregiver_profile(user_id) ON DELETE CASCADE, patient_user_id BIGINT NOT NULL REFERENCES patient_profile(user_id) ON DELETE CASCADE, role VARCHAR(20) NOT NULL DEFAULT 'PRIMARY', linked_at TIMESTAMPTZ NOT NULL DEFAULT NOW(), PRIMARY KEY (caregiver_user_id, patient_user_id), CHECK (role IN ('PRIMARY', 'SECONDARY')) );
```

– Appointments & medications

```
CREATE TABLE appointment ( id BIGSERIAL PRIMARY KEY, patient_id BIGINT NOT NULL REFERENCES patient_profile(user_id) ON DELETE CASCADE, caregiver_id BIGINT REFERENCES caregiver_profile(user_id) ON DELETE SET NULL, title VARCHAR(150), scheduled_time TIMESTAMPTZ, provider VARCHAR(150), location VARCHAR(255), status VARCHAR(30) DEFAULT 'SCHEDULED', CHECK (status IN ('SCHEDULED', 'COMPLETED', 'CANCELLED')) );
```

```
CREATE TABLE medication ( id BIGSERIAL PRIMARY KEY, name VARCHAR(150) NOT NULL, description TEXT );
```

```
CREATE TABLE patient_medication ( patient_id BIGINT NOT NULL REFERENCES patient_profile(user_id) ON DELETE CASCADE, medication_id BIGINT NOT NULL REFERENCES medication(id) ON DELETE CASCADE, dosage VARCHAR(50), frequency VARCHAR(50), start_date DATE, end_date DATE, PRIMARY KEY (patient_id, medication_id) );
```

– Billing

```
CREATE TABLE plan ( id BIGSERIAL PRIMARY KEY, code VARCHAR(50) NOT NULL
UNIQUE, name VARCHAR(100) NOT NULL, price_cents INT NOT NULL, billing_period
billing_period NOT NULL DEFAULT 'MONTH' );
```

```
CREATE TABLE subscription ( id BIGSERIAL PRIMARY KEY, user_id BIGINT NOT NULL
REFERENCES app_user(id) ON DELETE CASCADE, plan_id BIGINT NOT NULL
REFERENCES plan(id), status subscription_status NOT NULL DEFAULT 'ACTIVE',
started_at TIMESTAMPTZ NOT NULL DEFAULT NOW(), current_period_end
TIMESTAMPTZ );
```

```
CREATE TABLE payment_method ( id BIGSERIAL PRIMARY KEY, user_id BIGINT NOT
NULL REFERENCES app_user(id) ON DELETE CASCADE, provider VARCHAR(20) NOT
NULL DEFAULT 'STRIPE', stripe_payment_method_id VARCHAR(255), last4 CHAR(4),
brand VARCHAR(20), exp_month SMALLINT, exp_year SMALLINT, created_at
TIMESTAMPTZ NOT NULL DEFAULT NOW(), CHECK (provider IN ('STRIPE')) );
```

```
CREATE TABLE payment ( id BIGSERIAL PRIMARY KEY, subscription_id BIGINT NOT
NULL REFERENCES subscription(id) ON DELETE CASCADE, payment_method_id BIGINT
REFERENCES payment_method(id) ON DELETE SET NULL, amount_cents INT NOT
NULL, status payment_status NOT NULL, attempted_at TIMESTAMPTZ NOT NULL
DEFAULT NOW() );
```

– Audit & transcriptions

```
CREATE TABLE audit_log ( id BIGSERIAL PRIMARY KEY, tenant_id BIGINT NOT NULL
REFERENCES tenant(id), actor_user_id BIGINT REFERENCES app_user(id), action
VARCHAR(100) NOT NULL, resource_type VARCHAR(100), resource_id VARCHAR(100),
outcome VARCHAR(30), created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
metadata JSONB );
```

```
CREATE TABLE transcription ( id BIGSERIAL PRIMARY KEY, patient_id BIGINT NOT
NULL REFERENCES patient_profile(user_id) ON DELETE CASCADE, created_by BIGINT
REFERENCES app_user(id), source_object_key VARCHAR(512), status VARCHAR(30) NOT
NULL DEFAULT 'QUEUED', created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
completed_at TIMESTAMPTZ, entities JSONB, CHECK (status IN
('QUEUED', 'PROCESSING', 'COMPLETED', 'FAILED')) );
```

– Webhook subscriptions

```
CREATE TABLE webhook_subscription ( id BIGSERIAL PRIMARY KEY, tenant_id BIGINT
NOT NULL REFERENCES tenant(id), owner_user_id BIGINT NOT NULL REFERENCES
app_user(id), url VARCHAR(512) NOT NULL, events JSONB NOT NULL, secret TEXT,
description TEXT, status VARCHAR(20) NOT NULL DEFAULT 'ACTIVE', created_at
TIMESTAMPTZ NOT NULL DEFAULT NOW(), updated_at TIMESTAMPTZ NOT NULL
```

DEFAULT NOW(), CHECK (status IN ('ACTIVE','INACTIVE')));

– Additional domain tables follow the same conventions and are omitted here.

7.7 Physical Design Considerations

To ensure optimal performance and scalability for the database, the following physical design considerations will be implemented:

Indexes

- **FK columns auto-indexed:** Foreign key columns will be automatically indexed to improve query performance when joining tables based on these keys.
- **Multi-column indexes:** Multi-column indexes will be added to improve the performance of queries involving the combination of patient_id and logged_at in tables such as symptom_log and meal_entry. This will speed up data retrieval for time-based queries on these time-series tables.

Partitioning

- **Time-series tables partitioning:** Tables that store time-series data in RDS, such as symptom_log and meal_entry, will be range-partitioned by month. This approach helps with scaling as the data grows, improving performance for querying specific time ranges by limiting the amount of data to scan.

VACUUM & Autovacuum

- **Autovacuum tuning:** Autovacuum thresholds will be adjusted for high-write tables to ensure that table bloat is kept in check, especially in systems with frequent inserts or updates. This helps maintain database performance and space utilization.

Connection Pooling

- **Connection management:** To efficiently handle database connections and reduce overhead, connection pooling will be implemented using Amazon RDS Proxy or PgBouncer. These tools help manage database connections more effectively, allowing for better resource utilization and improved application performance.

7.8 AWS PostgreSQL Deployment

For the AWS deployment of PostgreSQL, the following configurations will be applied to ensure high availability, security, and efficient performance:

- **Engine:** The database will be deployed using Amazon RDS for PostgreSQL (or Aurora PostgreSQL if selected), providing robust and scalable solutions for handling relational workloads with enhanced performance and availability.
 - **High availability:** A Multi-AZ deployment will be used, with automated failover capabilities. This ensures that the database remains highly available in the event of hardware or infrastructure failure, minimizing downtime and ensuring continuous operation.
 - **Read scaling:** To handle analytics workloads and reduce the load on the primary database instance, read replicas will be deployed. This allows read-heavy queries, such as reports and analytics, to be offloaded from the primary instance, ensuring optimal
-

performance.

- **Encryption:** Both at-rest encryption using KMS (Key Management Service) and in-transit encryption via TLS 1.3 (or equivalent secure protocol) will be enabled to ensure that data is protected both while stored in the database and while being transferred over the network.
- **Backups:** Automated daily snapshots and point-in-time recovery (PITR) will be configured with a 35-day retention window for production to ensure durable recovery from operational failures and accidental data changes.
- **Monitoring:** To ensure the health and performance of the database, CloudWatch metrics will be used to monitor key performance indicators (KPIs). Additionally, Enhanced Monitoring will be enabled for more granular insights into query performance, allowing for proactive troubleshooting and performance tuning.

7.9 Security & Compliance

In terms of network security, the deployment will be configured within private subnets, with access restricted to authorized personnel via a bastion host or a VPN connection. For identity and access management (IAM), we will leverage IAM authentication for database credentials, provided it is supported by the database system in use. To ensure data protection, encryption will be enforced for data-at-rest through AWS KMS-managed encryption. Additionally, data-in-transit will be safeguarded by requiring TLS 1.3 encryption for all communications, ensuring secure data exchange across the network. Audit logging will be implemented using PostgreSQL audit tooling (e.g., pgaudit) and application audit_log records, with centralized collection in CloudWatch and immutable retention governed by AWS Lake Formation for compliance monitoring.

Furthermore, the principle of least privilege will be adhered to by ensuring that separate roles are defined for the application and read-only reporting purposes, minimizing the exposure of sensitive data to only those who need it.

7.10 Backup & Recovery

For backup and recovery, automated daily snapshots will be taken and retained for 35 days. Additionally, point-in-time recovery (PITR) will be configured with a 35-day retention window, enabling restoration of the database to a specific point in time. Manual snapshots will be created before significant schema changes to ensure recoverability if issues are introduced during updates. In the event of a disaster, a cross-region read replica promotion strategy will be employed within approved U.S. regions to minimize downtime while maintaining PHI data residency expectations.

8. UI/UX Design

8.1 User Roles & Management (Inherited from CareConnect TDD V3)

This section was inherited from section 8 of the technical design document from the SWEN 670

Summer/Fall 2025 cohort as Team C won't be making any changes to the UI/UX as part of their requirements. More detailed information about the UI/UX for CareConnect can be found in that TDD. Section 8.3 from the last technical design document specifically because it describes the user interface and experience in managing user rolls in CareConnect and the implications of the rolls. Team C has a requirement to refactor how roll-based access control, but any updates to the user interface or experience are out of scope for this development effort. The goal of this effort is to make backend processing more efficient and easier to maintain. If successful, users won't notice any difference in the UI/UX.

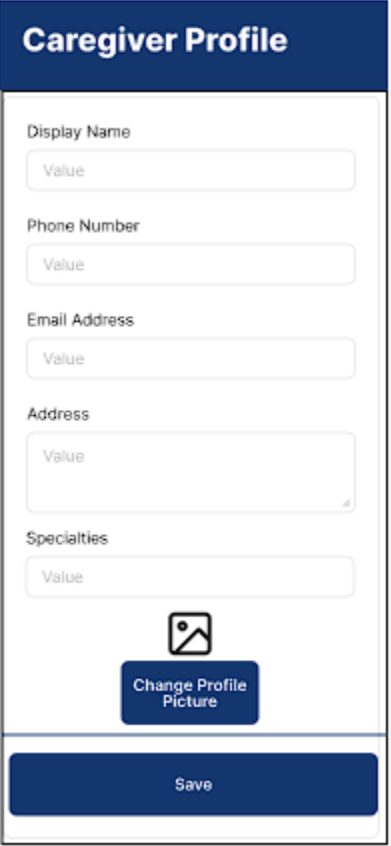
8.1.1 Caregiver Profile

8.1.1.1 Module Description

The Caregiver Profile screen allows the Caregiver to modify their display name, phone number, email address, and physical address, as well as view their unique QR Code and change their profile picture.

Roll Required: care giver

8.1.1.2 Images(s):

A mobile application form titled "Caregiver Profile" with a dark blue header. The form contains several input fields: "Display Name", "Phone Number", "Email Address", "Address", and "Specialties", each with a "Value" placeholder. Below these fields is a "Change Profile Picture" button with a camera icon. At the bottom is a large "Save" button.

Caregiver Profile

Display Name
Value

Phone Number
Value

Email Address
Value

Address
Value

Specialties
Value


Change Profile Picture

Save

8.1.1.3 Internal UI Functionality

ID	Description
IUF-CP-1	Enable input for display name, phone number, email address, physical address, specialties, and profile picture upload.
IUF-CP-2	Apply form validation of input fields before saving.
IUF-CP-3	Store locally before sending it to server in case of connection loss.

8.1.1.4 External UI Functionality

ID	Description
EUF-CP-1	New caregiver information is available for viewing on the Patient Dashboard screen.

8.1.2 Patient Profile

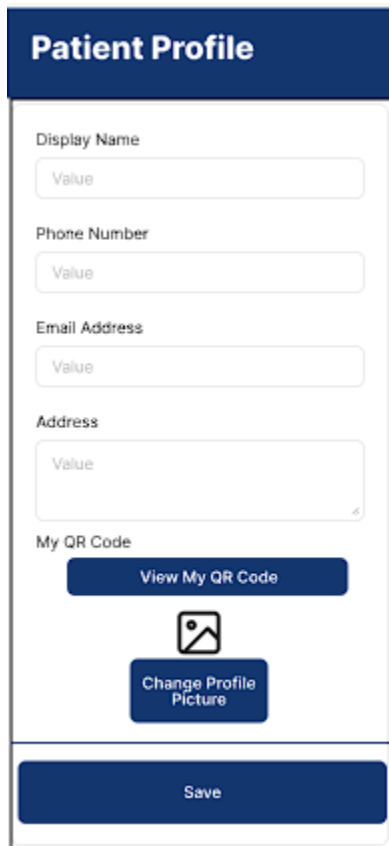
8.1.2.1 Module Description

The Patient Profile screen allows the Patient to modify their display name, phone number, email

address, and physical address, as well as view their unique QR Code and change their profile picture.

Roll Required: Patient

8.1.2.2 Image(s):

A screenshot of a mobile application form titled "Patient Profile". The form has a dark blue header with the title in white. Below the header, there are four text input fields, each with a label above it: "Display Name", "Phone Number", "Email Address", and "Address". Each input field contains the placeholder text "Value". Below the "Address" field is a section labeled "My QR Code" which contains a dark blue button with the text "View My QR Code". Below this button is a small square icon with a camera symbol, and below that is another dark blue button with the text "Change Profile Picture". At the bottom of the form is a large dark blue button with the text "Save".

8.1.2.3 Internal UI Functionality

ID	Description
IUF-PP-1	Enable input for display name, phone number, email address, physical address, and profile picture upload.
IUF-PP-2	Enable view access of unique QR Code.
IUF-PP-3	Apply form validation of input fields before saving.
IUF-PP-4	Store locally before sending it to server in case of connection loss.

8.1.2.4 External UI Functionality

ID	Description
EUFP-PP-1	New patient information is available for viewing on the Caregiver Dashboard screen.

8.1.3 Multi-Caregiver Support

8.1.3.1 Module Description

Multi-Caregiver Support allows more than one caregiver to be linked to a patient at once. Multiple caregivers, with varying specialties, can provide care for a patient and support their needs. Patients can have multiple caregivers and caregivers can be linked to multiple patients.

Roll Required: N/A

8.1.3.2 Image(s): N/A

8.1.3.3 Internal UI Functionality

ID	Description
IUF-MCS-1	Allow multiple caregivers to be linked to a patient.

8.1.3.4 External UI Functionality

ID	Description
EUFP-MCS-1	All linked caregivers display on the Patient Dashboard.
EUFP-MCS-2	All linked patients display on the Caregiver Dashboard.

8.1.4 Profile Linking

8.1.4.1 Module Description

Profile linking of Caregiver to Patient gives the Caregiver access to view, edit, and delete Patient records. Using the QR Code from the Patient's Profile screen, the Caregiver can link to the Patient through the Add New Patient form.

Roll Required: Patient to get QR code caregiver to use it

8.1.4.2 Image(s):





8.1.4.3 Internal UI Functionality

ID	Description
IUF-PL-1	Enable patient viewing of unique QR Code.
IUF-PL-2	Enable caregiver scanning of QR Code on Add New Patient screen.
IUF-PL-3	Autofill information on the Add New Patient form fields.
IUF-PL-4	Apply form validation of input fields before saving.
IUF-PL-5	Store locally before sending it to server in case of connection loss.

8.1.4.4 External UI Functionality

ID	Description
EUF-PL-1	Unique QR Code is generated for each patient user in My Profile.

EUF-PL-2	Submission of Add New Patient form submits link between caregiver and patient.

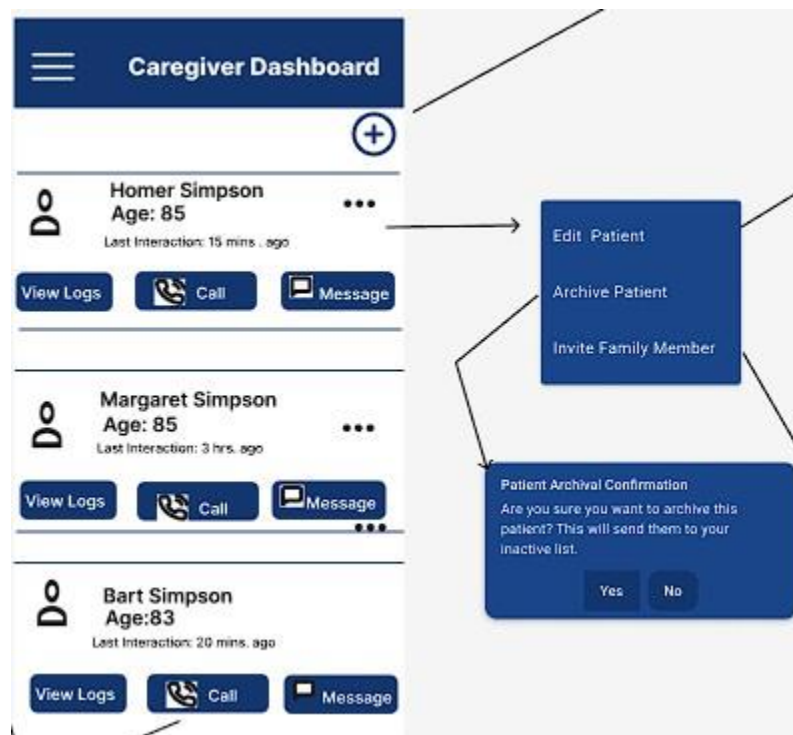
8.1.5 Access Control & Permissions

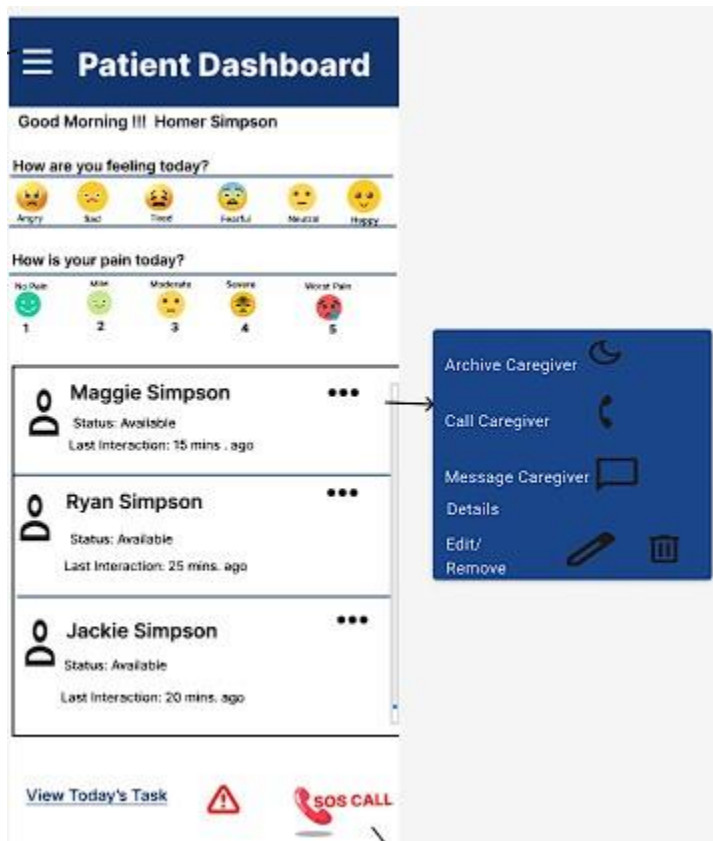
8.1.5.1 Module Description

User roles are managed through authentication in the login process. Caregivers are given all functional features to take care of, monitoring, and calls with their patients. Patients are given limited functional features aligned with displaying their symptoms, health, and wellness.

Roll Required: caregiver

8.1.5.2 Image(s):





8.1.5.3 Internal UI Functionality

ID	Description
IUF-ACP-1	Caregiver Dashboard hold all functional features for caregiver's needs.
IUF-ACP-2	Patient Dashboard hold all functional features for patient's needs.

8.1.5.4 External UI Functionality

ID	Description
EUf-ACP-1	Family member roles have “read only” access to patient records.

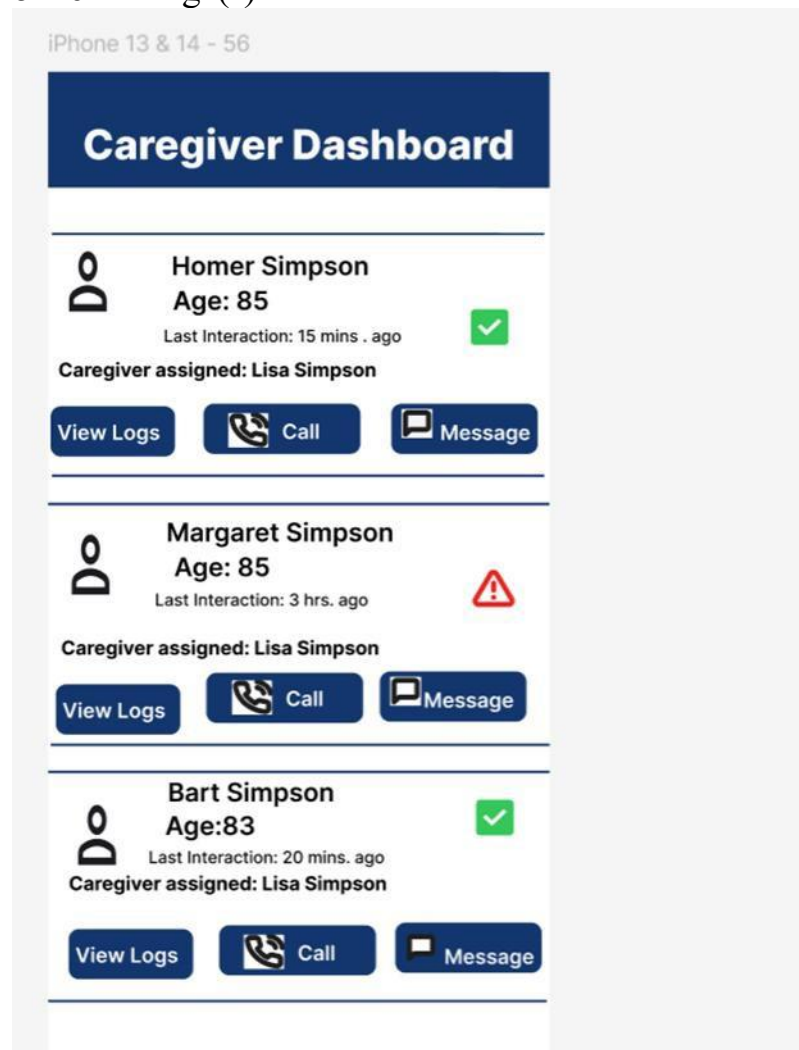
8.1.6 Family “Read-Only” Access

8.1.6.1 Module Description

Family “Read-Only” Access allows the approved family members to take care of the Patient.

They can view the Patient's record (symptoms, care schedules and caregiver notes). However, they don't have access to modify or delete any patients record.

8.1.6.2 Image(s):



8.1.6.3 Internal UI Functionality

ID	Description
IUF-WS-1	Invited Family Member will be given role-based access. Restricting to read only access.
IUF-WS-2	Fetch real-time patient data. (symptoms, schedule)
IUF-WS-3	Disabled functions such as "Edit", "Add", "Call"

8.1.6.4 External UI Functionality

ID	Description
EUF-WS-1	View patient profile with

	name, age, caregiver assigned.
EUF-WS-2	Access read-only version of shared document.

9. Security Considerations

9.1 Data Privacy & HIPAA/GDPR Compliance

- Encrypt PHI in transit (TLS 1.3) and at rest (AES-256 via KMS).
- Use HIPAA-eligible AWS services (e.g., S3, Cognito, RDS) and store PHI in US regions.
- Apply data minimization and GDPR rights (access/correction/deletion where applicable).
- Maintain automated backups with defined 35-day retention window.

9.2 Authentication & Authorization (RBAC)

- Enforce RBAC consistently across UI navigation and backend API access.
- Use least-privilege roles and tenant isolation for all protected resources.
- Require MFA for administrative access (hardware-backed MFA where supported).

9.3 Secure Data Transmission

- Enforce TLS 1.3 (or equivalent) for all client/server and service-to-service traffic.
- Do not expose PHI in insecure channels or push notifications.

9.4 API & Application Security

- Require authn/authz on every endpoint; validate inputs; apply rate limiting.
- Prevent IDOR, enforce session timeouts, and protect tokens/refresh flows.
- Require appropriate roll on each end point

9.5 Infrastructure / Hosting Security (Fargate)

- IAM least-privilege roles, secrets management, and hardened images.
- Network segmentation and ingress controls (security groups/WAF if applicable).

9.6 Monitoring, Logging, and Audit Trails

- Centralize logs in CloudWatch; generate audit events via pgaudit and application audit_log records; and enforce immutable audit-trail retention through AWS Lake Formation governance.
- Alert on abnormal access patterns, auth failures, and error spikes.

9.7 Mobile / Client Security

- Use secure storage for tokens/credentials; avoid PHI in notifications.
- Limit screen capture on sensitive views where supported.

9.8 AI / Third-Party Integrations

- AI outputs informational only; minimize PHI exposure and flag risky outputs.
- Secure third-party integrations (e.g., Stripe) and avoid storing payment data.

9.9 Security Testing

- Add RBAC/auth regression tests to the new testing norms.
- Run periodic vulnerability scans and penetration tests.

Note: Section 9 (Security Considerations) is adapted from the previous SWEN 670 Fall 2025 cohort's TDD for the CareConnect project and has been modified to reflect the current team's scope and design decisions.

10. Error Handling

10.1 Client / UI

- Provide clear, user-friendly error feedback for network, service, and authorization failures.
- For unauthorized access, display a clear message with a defined exit path (redirect or return to a safe screen).
- Support limited offline behavior by showing cached data where available and clearly indicating sync status.

(SRS: UI feedback, RBAC exit path, offline mode)

10.2 Server / API

- Enforce RBAC on every protected endpoint, denying access regardless of client navigation state.
- Return error responses in JSON consistent with the REST-based API interface described in the SRS.
- Use standard HTTP status codes appropriate to the REST interface.
- Ensure error messages avoid sensitive data or internal implementation details.

(SRS: RBAC enforcement, REST/JSON interfaces, HIPAA/PHI)

10.3 Monitoring, Logging, and Audit

- Centralize error and security logs using AWS CloudWatch.
 - Preserve immutable audit trails through Lake Formation governance of centralized audit data sourced from pgaudit and application audit_log records.
 - Log security events and system errors to support troubleshooting and compliance.
 - Ensure logs do not expose PHI or other sensitive data.
-

(SRS: monitoring, audit trails, HIPAA/PHI, operations logging)

10.4 Testing

- The redesigned unit testing paradigm must include coverage for error paths and RBAC denial scenarios within existing functionality.

(SRS/PMP: testing norms + reliability)

11. Performance & Scalability

11.1 Objectives

This section defines the performance and scalability goals for the Spring 2026 CareConnect effort. Since the scope is non-functional modernization (AWS hosting migration to Fargate, new unit testing paradigm, and improved RBAC enforcement) the goals focus on maintaining or improving system responsiveness and reliability during and after these changes.

Objectives

- Maintain responsive performance for normal usage and multiple concurrent users, with no “significant delays,” while enabling future growth without major redesign.
- Support scalable deployment on AWS using a managed hosting option Fargate to allow dynamic scaling based on demand.
- Ensure reliability and availability by using AWS infrastructure capabilities (load balancing, monitoring, automation) and reducing operational risk during migration.
- Preserve security/compliance behavior under load, especially around RBAC (least privilege) and auditability.

11.2 Key Metrics

To keep metrics simple and defensible, we align to the “Four Golden Signals” (latency, traffic, errors, saturation).

Metric	Target / Threshold	Why it matters
API latency (p50 / p95)	p50 ≤ 300ms; p95 ≤ 800ms	User-perceived responsiveness (golden

		signal: latency)
Traffic / throughput	Track RPS + peak hour baseline	Capacity planning + scaling validation (golden signal: traffic).
Error rate	< 1%; alert if elevated	Detect failures/regressions (golden signal: errors).
Saturation	CPU or Memory sustained > 70%	Early warning before user impact (golden signal: saturation).

11.3 Scalability Strategy

The Spring 2026 project requires migrating the backend to **AWS Fargate** as the managed hosting approach.

Team C shall;

- Use ECS Service Auto Scaling to increase/decrease the number of running tasks based on a target tracking policy (commonly CPU and/or memory).
- Target tracking policies manage CloudWatch alarms automatically and adjust desired task count to keep metrics near the target.

Load balancing and throttling

- For Fargate, place services behind a load balancer (commonly ALB) and/or an API front door if already in use by your architecture.
- Regardless of host, define rate limits at the API boundary to protect reliability during spikes and prevent abuse (especially important for auth/RBAC endpoints).

11.4 Monitoring and Capacity Management

Because the project plan emphasizes that AWS infrastructure “handles scaling, monitoring, backups, and deployment automation,” monitoring must be in place early—before and during migration.

Monitoring approach

- CloudWatch metrics + alarms for latency, error rate, and saturation thresholds.
- Use percentiles (p95/p99) for latency where possible to avoid averages hiding real user pain.

Capacity management

- Establish a baseline (current environment) and a post-migration baseline (Fargate).
- Use trend review to anticipate future scale needs (especially after hosting changes). This directly supports the SRS requirement to scale without major redesign.

11.5 Bottleneck Mitigation

When metrics breach thresholds, mitigation follows a consistent practice:

Common bottlenecks → mitigations

- **High CPU / Memory saturation:** scale out (tasks/instances), right-size CPU/memory, and profile hotspots. (Fargate scaling + best practices.)
- **Slow DB queries / connection pressure:** add indexes, reduce chatty queries, and consider connection pooling patterns appropriate to your DB host.
- **Auth/RBAC performance:** cache role/permission lookups appropriately, minimize repeated policy evaluation, and ensure endpoints fail fast on unauthorized access (security + performance).

11.6 Review Cadence

Performance and scalability must be reviewed regularly to ensure the system continues to meet expectations as infrastructure changes are introduced. Since the Spring 2026 scope includes migrating the application to AWS-managed hosting service Fargate, regular review helps confirm that performance remains stable and predictable throughout the project lifecycle.

Review Schedule

- **Weekly reviews:**

The team will review basic system health indicators (such as response time, error rates, and system resource usage) once per week. This ensures that performance problems are identified early and can be addressed before they impact users.

- **Pre-release reviews:**

Before each milestone demo or release, the team will verify that the system responds consistently and does not slow down under normal use. This check confirms that recent changes (such as hosting or security updates) did not negatively affect performance.

- **Post-issue reviews:**

If performance issues or outages occur, the team will conduct a short review to understand what happened, document the cause, and identify improvements to prevent similar issues in the future.

Continuous Improvement

Results from these reviews will be documented and used to:

- Adjust performance thresholds if needed
- Improve monitoring coverage
- Refine testing and deployment practices

12. Testing Strategy

Strategy	Description
Perform Testing at Different Levels	• Unit Testing – Verifies individual components and business logic, with emphasis on authorization rules and backend services. • Integration Testing – Confirms correct interaction between services, authentication, and role-based access control. • System Testing – Ensures the full system

	functions correctly after deployment to AWS-managed hosting. • Regression Testing – Validates that existing features continue to work after infrastructure and security changes.
Perform Different Types of Testing	• Functional Testing – Confirms core workflows (login, role-based access, dashboards) continue to operate correctly. • API Testing – Verifies REST endpoints for correctness, security enforcement, and error handling. • UI Testing – Confirms layouts and navigation remain stable after backend and deployment changes. • Security Testing – Focuses on role-based access control, authentication, and protection of sensitive data.
Use Appropriate Testing Tools	• Unit Testing – JUnit and Mockito for backend logic validation. • API Testing – Postman and Swagger/OpenAPI for endpoint verification. • Automation / CI – GitHub Actions to run automated tests during builds. • Issue Tracking – GitHub Issues for documenting and resolving defects.
Define Clear Entry and Exit Criteria	• All required unit and integration tests must pass before deployment to AWS environments. • No critical or high-severity defects may remain

	unresolved at release time. • Regression testing must confirm that existing functionality remains stable.

13. Deployment Strategy

The deployment strategy for the CareConnect project focuses on migrating the existing application to AWS-managed hosting while minimizing risk and disruption. Since the Spring 2026 scope does not introduce new features, deployment activities are centered on infrastructure modernization, reliability, and repeatability.

13.1 Deployment Preparation

Before deploying the application to AWS, the following preparation steps will be completed:

- **Environment configuration:**
AWS resources (such as compute services, networking, and permissions) will be configured according to project requirements and security best practices.
- **Application readiness:**
The application will be packaged and configured to run in a managed AWS Fargate environment, ensuring compatibility with the selected hosting option.
- **Security validation:**
Role-based access control (RBAC) rules and authentication settings will be verified to ensure users only have access to permitted functionality.
- **Testing verification:**
All required unit, integration, and regression tests must pass before deployment to ensure system stability. Integration and regression tests will also be executed post deployment.

These preparation steps reduce deployment risk and help ensure a smooth transition to AWS-managed infrastructure.

13.2 CI/CD Flows

Continuous Integration and Continuous Deployment (CI/CD) processes will be used to support consistent and repeatable deployments. Automation helps reduce manual errors and ensures deployments follow the same steps each time.

13.2.1 Backend

- Backend code changes are stored and managed in a version-controlled repository.
- Automated builds and tests are triggered when changes are committed.
- If all tests pass, the backend application is packaged and deployed to the AWS hosting environment.
- Deployment failures or test errors prevent release and require review before retrying.

This approach ensures backend changes are validated before reaching the deployed environment.

13.2.2 Frontend

- Frontend code is maintained separately and follows a similar automated build process.
- Builds verify that the application compiles correctly and integrates with backend services.
- Successful builds are deployed to the appropriate hosting environment.
- Any build or deployment issues must be resolved before release.

14. Known Issues & Limitations

Team C at this point has not identified any known issues or limitations.

Technical Design Document

CareConnect

University of Maryland Global Campus
SWEN 670 - Software Engineering Capstone
Professor Mir Mohammed Assadullah, DBA, PhD
February 7, 2026

Contributors:

Justinah Bashua (Technical Lead/Architect)

Corey Bayliss (Lead Tester)

Matthew Lester (Lead Software Developer)

James Stevens (Team Lead)

Brandon Sutan (Lead Business Analyst)

REVISION HISTORY

[illegible]

TABLE OF CONTENTS

1. INTRODUCTION.....	7
1.1 PURPOSE	7
1.2 INTENDED AUDIENCE	7
1.3 RELATED PROJECT DOCUMENTS	8
1.4 ACRONYMS, DEFINITIONS, AND ABBREVIATIONS.....	8
1.5 REFERENCES.....	9
2. SYSTEM OVERVIEW.....	10
2.1 SYSTEM SUMMARY	10
2.2 SYSTEM USERS AND STAKEHOLDERS	10
2.3 SYSTEM CAPABILITIES	10
2.4 SYSTEM ARCHITECTURE OVERVIEW.....	11
2.5 SYSTEM ARCHITECTURE DESIGN RATIONALE	11
2.6 IN-SCOPE AND OUT-OF-SCOPE FEATURES.....	11
3. ARCHITECTURE	13
3.1 SYSTEM ARCHITECTURE DIAGRAM.....	13
3.2 ARCHITECTURAL COMPONENTS	17
3.2.1 Static Analysis and Security Scanning Components	17
3.2.2 CI/CD Pipeline Components.....	19
3.2.2.5 Dependency Vulnerability Scanning (SCA) (Team D Enhancement)	21
3.2.3 Payment Processing Components	24
3.2.4 Notification and Messaging Components	26
4. TECHNOLOGY STACK	28
4.1 DEVELOPMENT AND BUILD TECHNOLOGIES	29
4.2 STATIC ANALYSIS AND SECURITY TOOLING	30
4.3 CI/CD TOOLING	32
4.4 CLOUD SERVICES.....	33
4.5 MOBILE PLATFORM DEPENDENCIES	35
5. DESIGN SPECIFICATIONS BY BUSINESS NEED.....	36

5.1 BUSINESS NEED 1 – AUTOMATED IDENTIFICATION OF BUGS, VULNERABILITIES, AND CODE QUALITY ISSUES.....	36
5.1.1 Design Overview	36
5.1.2 Tool Selection and Justification	37
5.1.3 Local Developer Workflow Design	39
5.1.4 CI Integration Design.....	41
5.1.5 Findings Classification and Prioritization.....	42
5.1.6 Documentation and Deferral Strategy.....	44
5.1.7 Constraints and Limitations	45
5.1.7.1 Performance and Resource Constraints	45
5.2 BUSINESS NEED 2 – CI/CD BUILD AND DEPLOYMENT CONTROLS.....	47
5.2.1 Design Overview	47
5.2.2 Pipeline Flow and Stages	47
5.2.4 Developer Feedback and Reporting.....	48
5.2.5 Promotion and Approval Logic.....	48
5.3 BUSINESS NEED 3 – NATIVE MOBILE PAYMENT MIGRATION	50
5.3.1 Design Overview	50
5.3.2 Stripe Decommissioning Strategy.....	51
5.3.3 Apple Native Payment Design	52
5.3.4 Google Native Payment Design.....	54
5.3.5 Subscription and State Management.....	55
5.3.6 Error Handling and Recovery	63
5.3.7 Compliance and Policy Considerations	63
5.4 BUSINESS NEED 4 – CENTRALIZED NOTIFICATIONS AND REMINDERS	65
5.4.1 Design Overview	65
5.7 Notification Triggers and Event Sources	65
5.4.3 Cloud Messaging Integration.....	66
5.4.4 Email Delivery Integration	71
5.4.5 Configuration and Extensibility	72
5.4.6 Monitoring and Observability.....	72
5.4.7 Documentation and Reproducibility	73

6. DATA DESIGN.....	74
6.1 DATA SCOPE AND OWNERSHIP	74
6.2 CONCEPTUAL DATA MODEL	74
6.3 LOGICAL DATA MODEL	78
6.4 DATA DICTIONARY	79
6.5 DATA STORAGE AND RETENTION CONSIDERATIONS	79
7. SECURITY CONSIDERATIONS.....	80
7.1 AUTHENTICATION AND AUTHORIZATION	80
7.2 DATA PRIVACY AND COMPLIANCE	80
7.3 SECURE DATA TRANSMISSION.....	80
7.4 LOGGING, AUDITING, AND MONITORING	81
8. PERFORMANCE AND SCALABILITY	82
8.1 PERFORMANCE OBJECTIVES.....	82
8.2 KEY PERFORMANCE METRICS.....	83
8.3 SCALABILITY STRATEGY	84
8.4 MONITORING AND CAPACITY MANAGEMENT	84
8.5 BOTTLENECK IDENTIFICATION AND MITIGATION.....	85
8.6 REVIEW AND OPTIMIZATION CADENCE	85
9. TESTING STRATEGY	86
9.1 SCOPE AND OBJECTIVES.....	86
9.2 REQUIREMENTS TRACEABILITY	86
9.3 TEST LEVELS AND TYPES	86
9.4 TEST ENVIRONMENTS AND DATA	87
9.5 AUTOMATION AND CI/CD	87
9.6 TEST TOOLING AND ORACLES	87
9.7 ENTRY AND EXIT CRITERIA	88
9.8 DEFECT MANAGEMENT AND REPORTING	88
9.9 ONGOING VALIDATION IN PRODUCTION	88
10. DEPLOYMENT STRATEGY	89
10.1 DEPLOYMENT PREPARATION	89

10.2 CI/CD DEPLOYMENT FLOWS.....	90
10.2.1 Backend Deployment Flow.....	90
10.2.2 Frontend Deployment Flow	91
10.3 RUNTIME DEPLOYMENT SAFETY AND ENFORCEMENT	92
10.4 RUNTIME USE CASE DEPLOYMENT SEQUENCES	92
11. KNOWN ISSUES AND LIMITATIONS.....	95
11.1 CURRENT GAPS AND DEFECTS	95
11.2 TECHNICAL CONSTRAINTS	95
11.3 DEFERRED WORK	96
11.4 OPERATIONAL LIMITATIONS	96
11.5 WORKAROUNDS AND MITIGATIONS	97
11.6 ASSUMPTIONS AND EXTERNAL DEPENDENCIES	97

1. INTRODUCTION

1.1 PURPOSE

This Technical Design Document (TDD) describes the architectural and engineering decisions supporting Team D's enhancements to the CareConnect platform. The purpose of this document is to provide a precise implementation-level description of how the system satisfies the assigned business needs while preserving continuity with prior cohorts.

This document captures the design rationale, component interactions, data structures, and operational controls introduced to support:

- Automated code quality and security enforcement,
- CI/CD deployment gating,
- Native mobile payment compliance,
- Centralized notification and reminder infrastructure.

The TDD serves as a durable engineering reference that enables reproducibility, maintainability, and safe extension of the system by future teams.

1.2 INTENDED AUDIENCE

This document is intended for:

- Current and future CareConnect developers,
- Instructors and academic reviewers,
- System architects,
- DevOps and operations maintainers,
- Future capstone cohorts that are responsible for system continuation.

Readers are assumed to have a working knowledge of software engineering principles, CI/CD pipelines, cloud infrastructure, and mobile application architecture.

1.3 RELATED PROJECT DOCUMENTS

This document complements the following artifacts:

- Software Requirements Specification (SRS)
- Business Needs Statement (BD1-BD4)
- Software Test Plan
- CI/CD pipeline documentation
- Repository README and setup guides
- Infrastructure-as-Code configuration
- Prior cohort technical documents

Together, these documents define requirements, implementation, validation, and operational governance.

1.4 ACRONYMS, DEFINITIONS, AND ABBREVIATIONS

Term	Definition
CI/CD	Continuous Integration / Continuous Deployment
SNS	Simple Notification Service (AWS)
SES	Simple Email Service (AWS)
IAP	In-App Purchase
DLQ	Dead Letter Queue
RBAC	Role-Based Access Control
JWT	JSON Web Token
SRS	Software Requirements Specification
TDD	Technical Design Document

1.5 REFERENCES

- IEEE 1016 – Software Design Description Standard
- Apple App Store Review Guidelines
- Google Play Payments Policy
- AWS Architecture Best Practices
- OWASP Secure Coding Guidelines
- SonarQube Documentation
- HIPAA Security Rule

2. SYSTEM OVERVIEW

2.1 SYSTEM SUMMARY

CareConnect is a modular, service-oriented healthcare coordination platform designed to support caregivers, care recipients, and administrators in managing reminders, communication, subscription services, and care workflows.

Team D's scope strengthens CareConnect's engineering foundation by implementing automated quality controls, modernizing the mobile payment architecture, and centralizing notification delivery. These enhancements improve reliability, compliance, maintainability, and long-term scalability while preserving compatibility with prior system components.

2.2 SYSTEM USERS AND STAKEHOLDERS

Role	Interest
Care Recipients	Reliable reminders, secure subscriptions
Caregivers	Timely alerts and workflow continuity
Administrators	Reporting, auditing, and system stability
Developers	Maintainable architecture
Future cohorts	Reproducible engineering environments

2.3 SYSTEM CAPABILITIES

Team D introduces and strengthens the following capabilities:

- Automated identification of bugs and vulnerabilities
- CI/CD enforcement preventing unsafe deployments
- Compliant native Apple and Google billing integration
- Centralized, auditable notification infrastructure
- Improved traceability and engineering reproducibility

These capabilities support both operational reliability and future extensibility.

2.4 SYSTEM ARCHITECTURE OVERVIEW

The CareConnect architecture separates responsibilities across:

- Client layer (Flutter applications),
- Backend services (Spring Boot),
- Relational data layer (PostgreSQL),
- Cloud messaging infrastructure (AWS SNS/SES),
- CI/CD enforcement pipelines,
- Platform native payment integrations.

This layered approach isolates concerns, enforces security boundaries, and enables independent component evolution.

2.5 SYSTEM ARCHITECTURE DESIGN RATIONALE

Architectural decision priority:

- Maintainability across academic cohorts,
- Automated enforcement over manual processes,
- Security first design,
- Platform compliance,
- Reproducibility and traceability,
- Cloud native scalability.

Every design decision aims to reduce long-term technical debt while enabling the safe extension of the system.

2.6 IN-SCOPE AND OUT-OF-SCOPE FEATURES

In Scope

- Static analysis enforcement
- CI/CD deployment gating
- Native payment migration
- Centralized notifications
- Auditability and traceability

Out of Scope

- Redesign of core healthcare workflows
- Wearable integration changes
- AI assistant redesign
- UI redesigns beyond payment/notification impact
- Unrelated feature additions.

3. ARCHITECTURE

3.1 SYSTEM ARCHITECTURE DIAGRAM

The CareConnect system follows a modular, service-oriented architecture designed to support secure healthcare workflows, enforce automated quality controls, and enable platform-compliant payments and notifications. The architecture separates concerns across presentation, application, data, infrastructure, and external services to improve maintainability, security enforcement, and future extensibility.

The architecture presented in this section represents a continuation of the CareConnect system established by prior cohorts. Core platform components—including client applications, backend services, data persistence, and cloud infrastructure—are retained and maintained. This baseline architecture is extended by formalizing automated quality enforcement, integrating platform-compliant native billing systems, and implementing a centralized notification and reminder infrastructure.

The System Architecture Diagram (Figure 3-1) illustrates the major system components, their interactions, and the integration points required to satisfy the project's business needs, including static analysis enforcement, CI/CD quality gates, native payment processing, and centralized notification delivery.

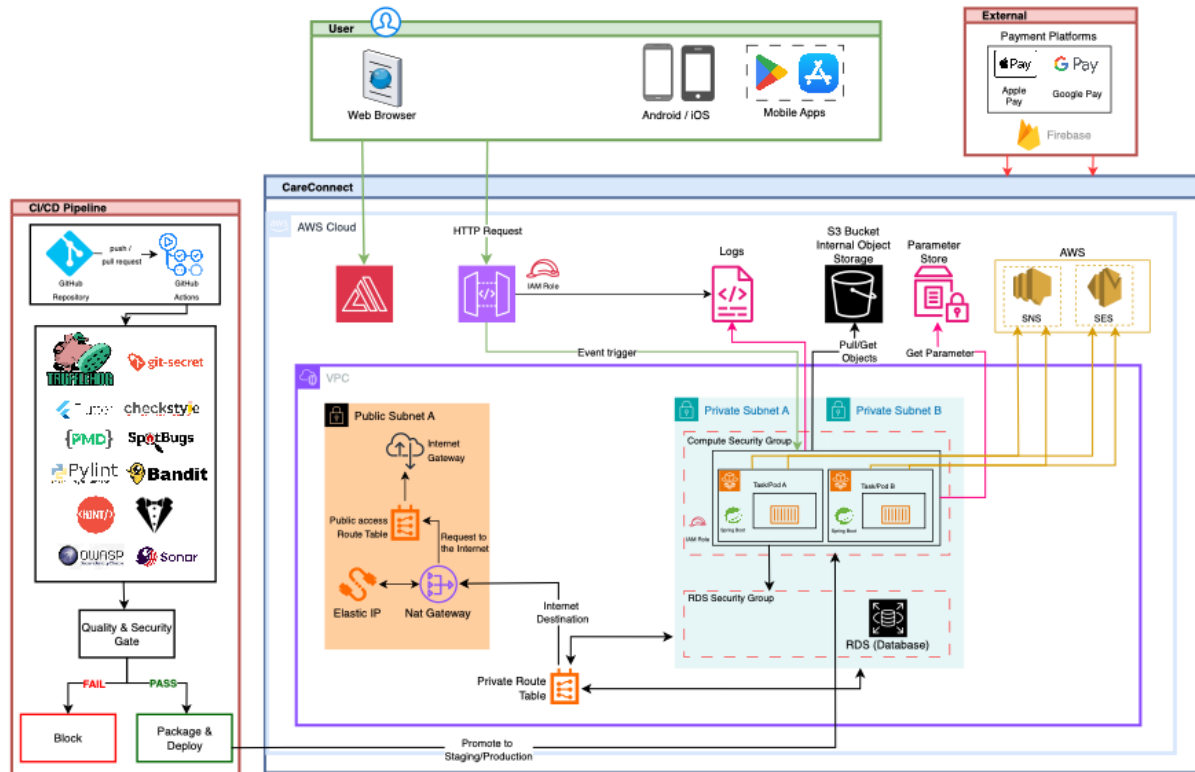


Figure 3-1: CareConnect System Architecture

Figure 3-1 depicts the flow of interactions from client applications through backend services, external payment providers, and AWS-managed notification services. The diagram also highlights the CI/CD pipeline as a parallel enforcement path, ensuring that all code changes pass automated quality and security checks before deployment.

External integrations present in prior cohort implementations (e.g., wearable data sources, AI services, and third-party communication platforms) remain logically supported within the architecture and are treated as external service integrations. These integrations are preserved to ensure continuity while allowing future cohorts to extend functionality without impacting core system components.

3.1.1 Architectural Layers

3.1.1.1 Client Layer

The client layer consists of mobile and web user interfaces built using Flutter. This layer handles user interactions for caregivers, care recipients, and administrators, including authentication, subscription management, and reminder acknowledgment. Client applications communicate exclusively with backend services through secured REST APIs.

3.1.1.2 Application Layer

The application layer is implemented using a Java Spring Boot backend. It contains core business logic for care coordination, subscription management, reminder scheduling, and notification escalation. The backend exposes RESTful endpoints consumed by the client layer and enforces role-based access control and validation.

3.1.1.3 Data Layer

Data is stored in a PostgreSQL relational database. This layer maintains user profiles, subscription status, billing metadata, reminder configurations, notification history, and audit-relevant records. The data layer is isolated from direct client access and is accessed only through the application layer.

3.1.1.4 Payment Integration Layer

To meet mobile platform compliance requirements, the architecture integrates native payment systems:

- Apple In-App Purchase for iOS clients
- Google Play Billing for Android and web clients

Payment processing is abstracted behind backend services to decouple client logic from billing providers. This abstraction reduces compliance risk and allows future payment changes without disrupting core application functionality.

3.1.1.5 Notification & Messaging Layer

A centralized notification and messaging architecture is implemented using AWS-managed services to ensure reliable, scalable, and auditable message delivery across multiple channels.

- AWS Simple Notification Service (SNS) is used as the primary orchestration mechanism for notification events. SNS topics publish reminder and alert messages and fan out events to multiple subscribers.
- AWS Simple Email Service (SES) is used for transactional and reminder email delivery.
- Push notifications are supported either through SNS mobile platform endpoints or through container-based push workers that process SNS events and deliver platform-specific payloads to APNs and FCM.
- SMS delivery is supported directly through SNS subscriptions where required.

This layered approach enables configurable reminder escalation rules, delivery retries, and delivery history tracking. By centralizing messaging through SNS and SES, the system avoids hardcoded notification logic, improves consistency across delivery channels, and allows future cohorts to extend or modify notification behavior without impacting core application functionality.

3.1.1.6 DevOps & Quality Enforcement Layer

The DevOps layer integrates automated quality and security controls into the development and delivery pipeline. This includes:

- Local static analysis tools (SpotBugs, PMD, Checkstyle)
- CI-based static analysis, dependency scanning, container scanning, and dynamic testing
- GitHub Actions pipelines enforcing quality gates

Builds, merges, and deployments are automatically blocked when unit tests fail or when high or critical vulnerabilities are detected. This layer ensures enforcement occurs consistently and early, reducing the risk of insecure or unstable releases.

This architecture emphasizes separation of concerns, automated enforcement, and reproducibility, supporting both current project requirements and long-term maintainability for future cohorts.

3.2 ARCHITECTURAL COMPONENTS

3.2.1 Static Analysis and Security Scanning Components

The system incorporates a set of automated static analysis components that are responsible for identifying software defects, security vulnerabilities, and code quality issues throughout the development and build processes. These components operate continuously across the software lifecycle, providing early detection of risks, enforcing coding standards, and preventing defective or insecure code from being integrated into the main codebase.

The static analysis framework consists of multiple complementary tools that analyze source code and compiled artifacts without executing the application. These tools collectively support defect detection, security assurance, and maintainability improvement.

3.2.1.1 SonarQube

SonarQube serves as the centralized platform for continuous code quality monitoring and reporting. It aggregates analysis results from multiple static analysis tools and presents them through a unified dashboard. SonarQube is responsible for:

- Identifying software bugs and reliability issues
- Detecting security vulnerabilities and high-risk coding patterns
- Assessing maintainability through code smells and technical debt metrics
- Enforcing organizational coding and security standard
- Applying automated quality gates within the CI/CD pipeline

During automated builds, SonarQube evaluates the codebase against predefined quality criteria. If the code violates critical thresholds, the build is automatically failed, preventing defective code from being merged into protected branches.

3.2.1.2 PMD

PMD performs rule-based static analysis of source code to detect structural and logical weaknesses. The tool focuses on maintainability and defect prevention by identifying:

- Duplicate or redundant code segments
- Unused variables, methods, and classes
- Inefficient or risky programming constructs
- Potential logical errors and bad practices

PMD supports the reduction of technical debt and improves long-term maintainability of the software.

3.2.1.3 SpotBugs

SpotBugs conducts deep analysis of compiled bytecode to identify defects that may not be apparent at the source code level. The tool is particularly effective at detecting:

- Null pointer dereferences
- Race conditions and concurrency risks
- Memory management issues
- Potential runtime failures
- Dangerous or error-prone coding patterns

By analyzing bytecode rather than source code alone, SpotBugs provides an additional layer of reliability assurance.

3.2.1.4 Checkstyle

Checkstyle enforces standardized coding conventions across the development team. It verifies compliance with predefined formatting, naming, and structural rules, including:

- Class and method naming standards
- Indentation and spacing consistency
- File organization and structure
- Readability and style best practices

The use of Checkstyle ensures uniformity across the codebase, improving collaboration, review efficiency, and long-term maintainability.

3.2.1.5 Dependency Vulnerability Scanning (Software Composition Analysis)

To address security risks introduced through third-party libraries and frameworks, the system incorporates Software Composition Analysis (SCA) tooling. Dependency vulnerability scanning identifies known security issues in externally sourced components that are not detectable through source-code analysis alone.

OWASP Dependency-Check is used to analyze project dependencies against publicly maintained vulnerability databases. The tool identifies known Common Vulnerabilities and Exposures (CVEs), assigns severity levels, and generates reports used during quality gate evaluation.

Dependency vulnerability scanning complements static application security testing by ensuring that external libraries meet defined security standards before software artifacts are promoted through the CI/CD pipeline.

3.2.2 CI/CD Pipeline Components

The system employs an automated Continuous Integration and Continuous Deployment (CI/CD) pipeline to support reliable, repeatable, and secure software delivery. The pipeline is responsible for build orchestration, automated quality and security analysis, enforcement of gating mechanisms, and controlled artifact generation and storage. Team D's enhancements strengthen this pipeline by embedding static analysis and dependency vulnerability scanning directly into the build process and preventing the promotion of non-compliant software artifacts. Each numbered subsection below corresponds directly to one or more stages depicted in Figure 3-2.

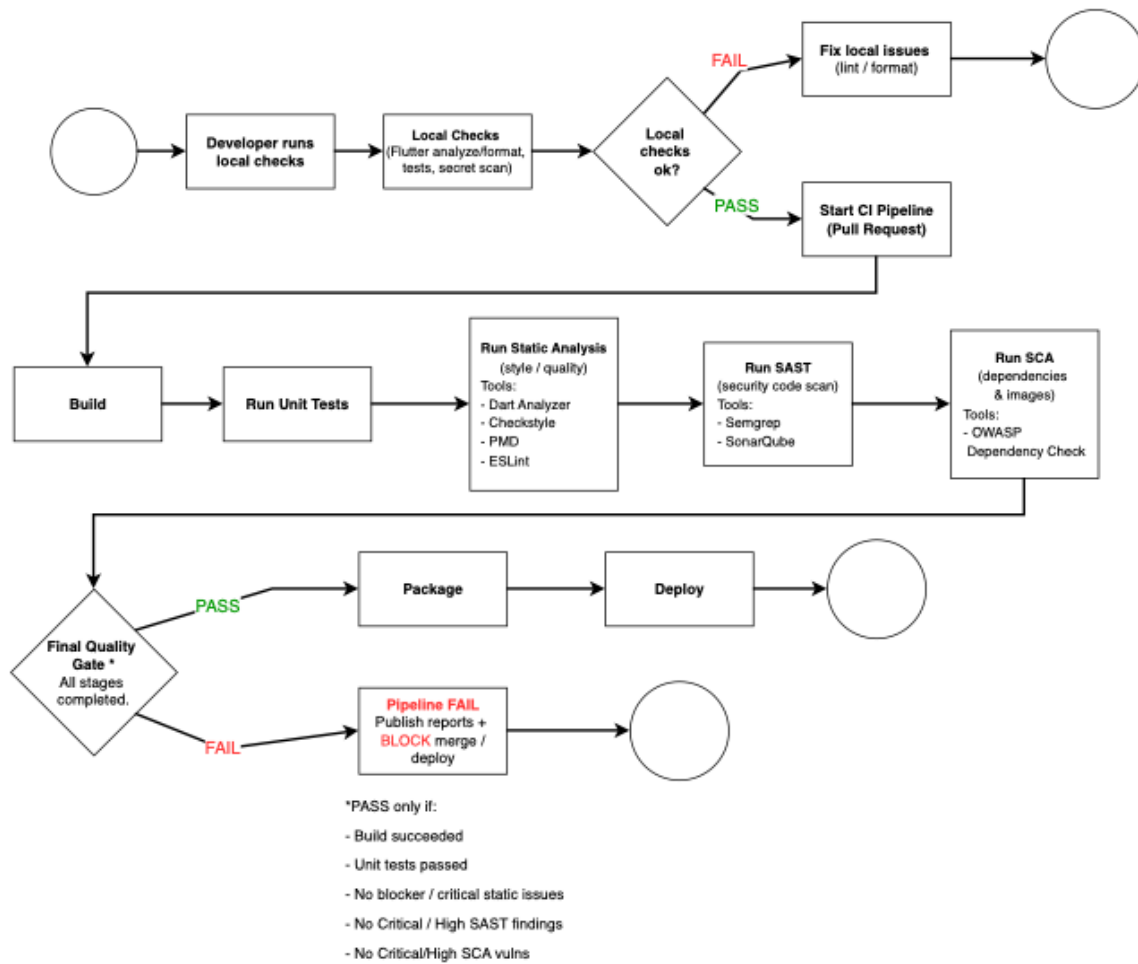


Figure 3.2 - CI/CD Pipeline Flow with Quality and Security Gates

This diagram illustrates the CI/CD pipeline flow from local developer checks through automated build, testing, static analysis (SAST), dependency vulnerability scanning (SCA), and final quality gate evaluation. Only artifacts that pass all required quality and security gates are allowed to proceed to merge or deployment, ensuring consistent and controlled software delivery.

3.2.2.1 Source Control Integration and Triggers

- Triggered by pull requests, pushes to feature branches, or merge attempts into protected branches (e.g., main, develop)
- Retrieves the latest committed source code from GitHub
- Initializes a clean build environment to prevent dependency contamination

3.2.2.2 Dependency Resolution and Build Environment Setup

- Installs required runtime dependencies, build tools, and libraries
- Configures build environments
- Applies caching mechanisms where appropriate to improve efficiency

3.2.2.3 Compilation and Unit Testing

- Compiles the application using the project's primary build system
- Executes unit tests to validate basic functional correctness
- Compilation and test execution results are recorded for pipeline evaluation

3.2.2.4 Static Analysis Integration (Team D Enhancement)

- After successful compilation, static analysis tools are executed automatically to identify code-level defects and security risks:
 - **Checkstyle** – verifies coding standards and formatting compliance.
 - **PMD** – analyzes source code for structural defects and maintainability issues.
 - **SpotBugs** – inspects compiled bytecode for potential runtime defects.
 - **SonarQube** – aggregates results and performs centralized quality evaluation.
- Results from these tools are collected and transmitted to SonarQube for consolidated reporting and quality gate evaluation.

3.2.2.5 Dependency Vulnerability Scanning (SCA) (Team D Enhancement)

- To identify known vulnerabilities in third-party libraries, the pipeline performs Software Composition Analysis (SCA):
 - **OWASP Dependency-Check** scans project dependencies against public vulnerability databases
 - Generates detailed vulnerability reports identifying CVEs and severity levels
 - Results are retained as pipeline artifacts and included in overall quality evaluation
- Dependency scanning complements SAST by addressing risks originating from external components rather than application source code.

3.2.2.6 Quality Gate Evaluation

- SonarQube applies predefined quality gates incorporating results from static analysis and dependency scanning, including:
 - Zero critical security vulnerabilities
 - No blocker-level bugs
 - Threshold limits on code smells and technical debt
 - Minimum test coverage requirements (if applicable)
- If the quality gate fails, the pipeline is marked as failed and blocks merges and deployments into protected branches.

3.2.2.7 Artifact Packaging

- If all required checks execute successfully, the system produces deployable artifacts such as:
 - JAR/WAR files (for Java-based components)
 - Compiled Flutter binaries
 - Bundled frontend artifacts
 - Container images (if applicable)
- Artifacts are labeled with version identifiers or commit hashes for traceability.

3.2.2.8 Artifact Storage and Repository Management

- Approved artifacts are stored in a centralized repository (e.g., GitHub Packages)
- Ensures controlled access, version history, and reproducibility

3.2.2.9 Build Orchestration

Build orchestration is managed using GitHub Actions:

- Defines execution order of pipeline stages
- Coordinates tool execution (build → tests → SAST → SCA → quality gate)
- Records execution logs for auditing and debugging

3.2.2.10 Gating Mechanisms

- The pipeline enforces multiple promotion gates:
 - **Branch Protection Gate** – requires passing CI checks, quality gates, and code review approval
 - **Automated Quality Gate (SonarQube)** – blocks promotion of vulnerable or non-compliant code
 - **Test Failure Gate** – all tests execute to completion; failures block promotion
 - **Static Analysis and SCA Failure Gate** – all analysis tools execute; violations block promotion
- Execution continues where technically possible to ensure complete reporting, while promotion decisions are enforced post-execution.

3.2.2.11 Artifact Traceability and Audit Support

- Each build produces traceable artifacts:
 - Tagged with commit identifiers
 - Timestamped
 - Linked to specific pipeline runs
- Artifacts can be:
 - Retrieved for deployment
 - Rolled back if issues are discovered
 - Audited for compliance and security
- Artifacts support deployment, rollback, and compliance auditing.

3.2.2.12 Design Rationale

These enhancements:

- Embed SAST and SCA directly into CI/CD
- Prevent insecure or defective code from advancing
- Provide complete, reproducible quality reporting
- Align the delivery pipeline with modern DevSecOps principles

3.2.3 Payment Processing Components

The payment processing components of CareConnect are designed to support secure, compliant, and platform-aligned subscription transactions for mobile users. Because CareConnect delivers digital services within mobile applications distributed through the Apple App Store and Google Play Store, payment functionality must comply with platform-specific policies that require the use of native payment systems. As a result, CareConnect transitions away from third-party processors such as Stripe for mobile subscriptions and instead adopts Apple In-App Purchases and Google Play Billing.

Payment responsibilities are intentionally split between the client and backend to balance compliance and maintainability. Mobile clients are responsible for initiating purchases, handling user authentication with the app store, and completing transactions using native SDKs. Backend services do not process payments directly but instead validate transaction receipts, persist in subscription metadata, and enforce access control based on subscription state. This separation ensures compliance with app store policies while preserving centralized control over entitlements, analytics, and auditing.

The Billing Service acts as the authoritative system of record for subscription status, storing platform-neutral subscription identifiers, renewal dates, cancellation status, and references to Apple or Google transaction tokens. This approach allows CareConnect to maintain consistent behavior across platforms while remaining resilient to platform-specific implementation changes. These components must incorporate robust error handling, multi-stage validation, and clear user feedback for scenarios like subscription renewal failures, offline verification, and receipt refresh cycles.

Key Responsibilities

Client

- Initiate purchase flows
- Present native billing UI
- Capture receipt/token securely
- Handle transaction lifecycle events (e.g., defer, cancel, refund)

Backend (Billing Service)

- Validate tokens/receipts with platform APIs
- Persist subscription state in a normalized database schema
- Generate entitlement decisions (active, grace period, past due)

Database

- Store consolidated subscription metadata
- Maintain audit history for compliance and reporting

Examples of Transaction Flows & Edge Cases

- Successful purchase: User taps subscribe → native payment dialog → receipt is generated → backend verifies → status records as “ACTIVE.”
- App offline at purchase: App caches purchase token → retries send when online → backend screws consistent state.
- Subscription cancellation: Platform notifies backend via webhook → backend updates state → UI reflects cancellation.

Below is a Payment processing sequence diagram

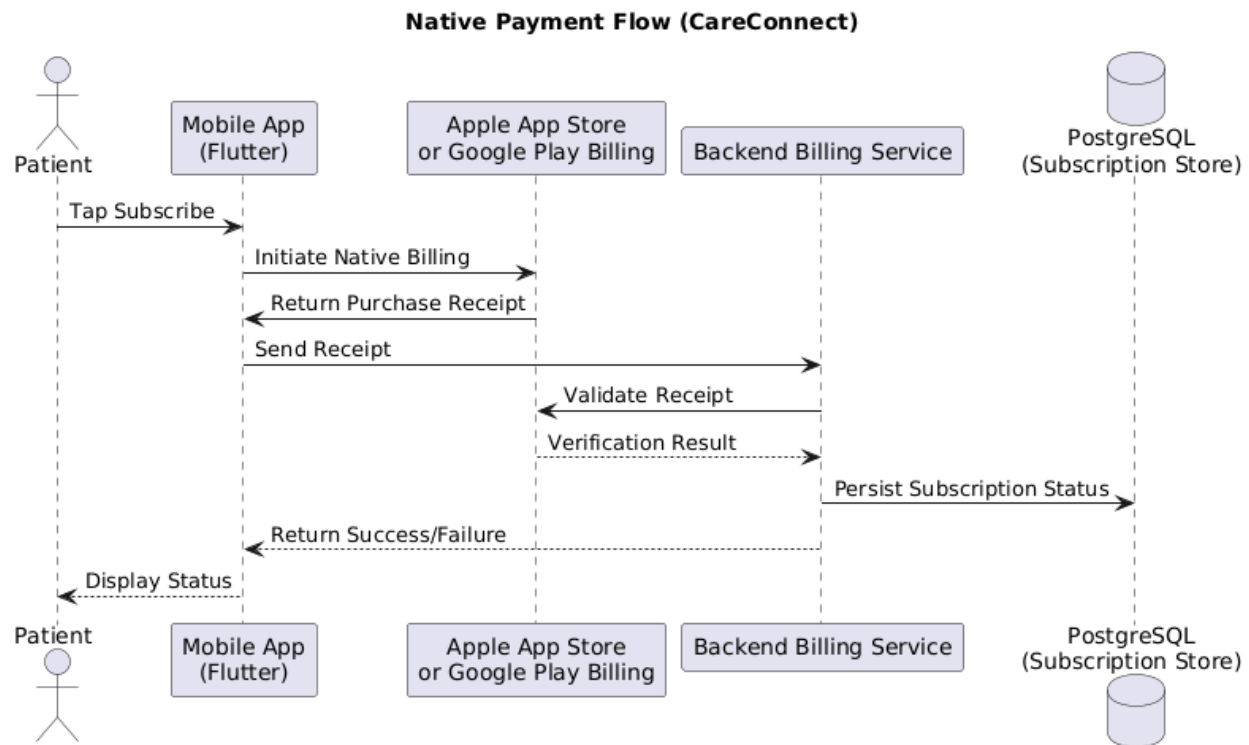


Figure 3.3 - Native Payment Processing and Subscription Validation Flow

This figure shows the native subscription payment flow for CareConnect, including purchase initiation from the Flutter mobile app, receipt validation by backend billing services, and persistence of subscription status in PostgreSQL. The flow ensures platform-compliant billing and centralized subscription management.

3.2.4 Notification and Messaging Components

The CareConnect system uses a centralized notification and messaging architecture to deliver reminders, alerts, and system messages across multiple channels in a reliable and scalable manner. This architecture is built on AWS-managed services to reduce operational complexity while supporting time-based reminders and escalation workflows.

AWS Simple Notification Service (SNS) serves as the primary event distribution mechanism. Backend services publish notification events—such as reminder due, escalation triggered, or system alerts—to dedicated SNS topics. SNS fans out these events to the appropriate delivery mechanisms based on message type and configured policies.

Supported notification channels include:

- Mobile push notifications (via platform-specific services)
- SMS delivery
- Email notifications

Channel selection and escalation behavior are driven by backend logic and user preferences rather than hardcoded client rules. For example, routine reminders may be delivered via push notifications, while critical alerts may additionally trigger SMS or email delivery.

Notification delivery is decoupled from core business logic. Backend services are responsible only for emitting notification events; channel-specific delivery is handled by SNS subscriptions and supporting workers. This design allows new notification channels or escalation rules to be added without modifying existing publishing services.

Delivery outcomes are recorded to support auditing, troubleshooting, and compliance reporting. Each notification attempt is logged with status information indicating whether it was sent, delivered, or acknowledged. Centralized logging and monitoring provide visibility into notification health and delivery performance.

Security and operational controls follow least-privilege and environment isolation principles. Services are granted access only to the notification topics they require, and sensitive data is minimized within message payloads. Topics and delivery components are separated by environment to support safe development, testing, and production operation.

This centralized messaging approach improves reliability, simplifies future enhancements, and provides a scalable foundation for reminders and notifications across the CareConnect platform.

4. TECHNOLOGY STACK

Technology stack decisions are fundamental to the non-functional success of CareConnect. They influence maintainability, cost, scalability, and ease of future development. The stack was chosen to balance cross-platform expediency, strong ecosystem support, and cloud-native reliability.

Layer	Technology Used	Purpose
Frontend	Flutter (Dart)	Cross-platform mobile UI development
Backend	Java, Spring Boot	REST APIs and business logic
Database	PostgreSQL (AWS RDS)	Persistent relational data storage
CI/CD	GitHub Actions	Automated build, test, and deployment
Static Analysis	SonarQube, SpotBugs, PMD, Semgrep	Code quality and vulnerability detection
Dependency Scanning	OWASP Dependency-Check	Third-party vulnerability identification
Cloud Platform	AWS	Hosting, networking, and managed services
Notifications	AWS SNS, AWS SES	Push notifications and email delivery
Payments (iOS)	Apple StoreKit 2	Native iOS subscription handling
Payments (Android)	Google Play Billing	Native Android subscription handling
API Gateway	AWS	Entry point for mobile app to backend (manages routing)
Compute	Amazon ECS (Fargate)	Containerized Spring Boot services running as replicated ECS tasks in private subnets.
Infrastructure Provisioning	Terraform	Infrastructure as Code (Iac) to deploy AWS services reproducibly.

Layer	Technology Used	Purpose
Authentication	Spring Security(Custom JWT Authentication), or Amazon Cognito	User login and session management
Security Compliance	HIPAA AND GDPR compliant	Ensures every component is secure and compliant.
File & Media Storage	Amazon S3	Stores images, videos, PDFs

4.1 DEVELOPMENT AND BUILD TECHNOLOGIES

The development and build environment underpins code quality, version traceability, and reproducibility. Using industry-standard tools ensures that developers across platforms can follow uniform workflows.

Flutter & Dart:

- Used to build cross-platform UI components with a single codebase.
- Features built-in tooling for formatting and static analysis (e.g., dart analyze).

Spring Boot & Java:

- Provides a robust server-side framework with dependency injection, REST API support, and modular architecture.
- Build automation via Maven ensures dependency consistency and version control.

Examples:

- Flutter's reactive widget tree simplifies complex UI for reminders and billing screens.
- Spring Boot's Actuator aids in runtime metrics and health checks.

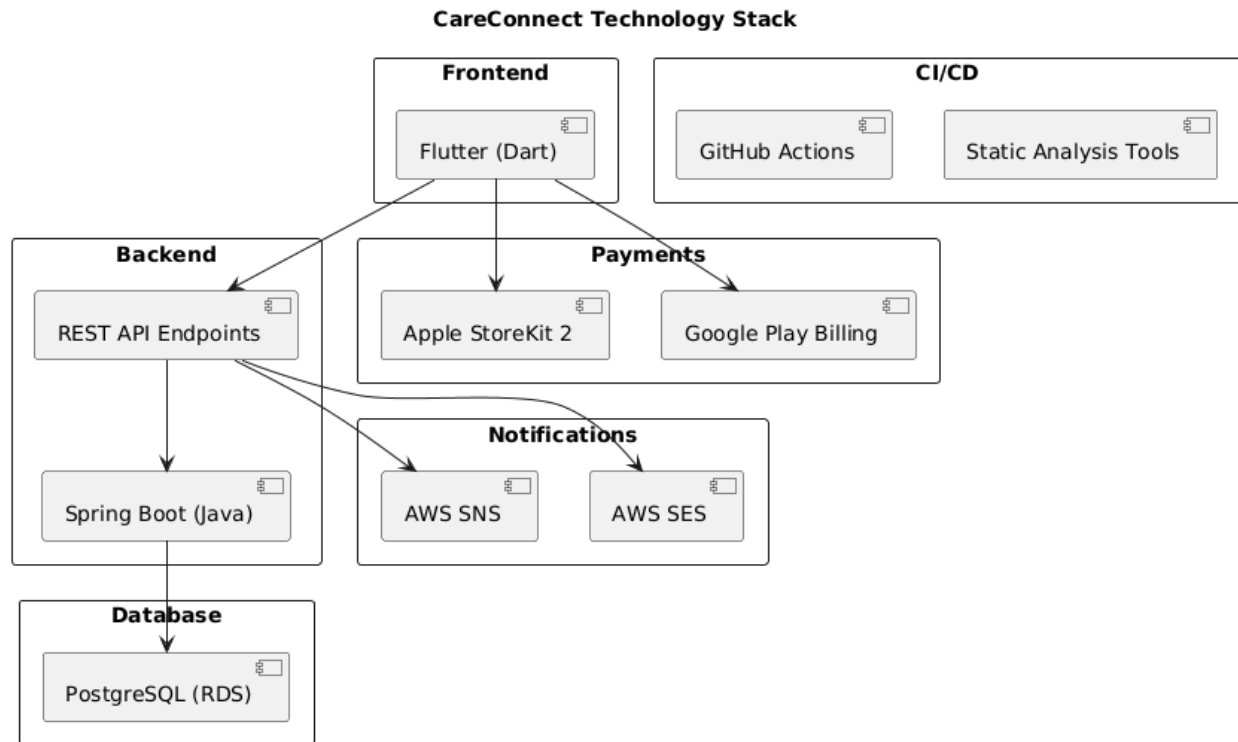


Figure 4.1 - CareConnect Technology Stack Overview

Figure 4.1 provides a high-level view of the CareConnect technology stack, showing the core frontend, backend, database, notification, payment, and CI/CD technologies and how they integrate to support application functionality and automated quality enforcement.

4.2 STATIC ANALYSIS AND SECURITY TOOLING

Static analysis and security tooling are integrated into both local development workflows and the continuous integration pipeline to detect defects early and prevent insecure code from advancing toward production.

During local development, tools such as Flutter Analyzer and Dart formatting enforce coding standards and surface potential issues before code is committed. For backend services, SpotBugs, PMD, and Checkstyle analyze source code and compiled artifacts to identify defect patterns, maintainability concerns, and style violations.

Security-focused static analysis is performed using SonarQube and Semgrep, which detect common vulnerability classes such as injection flaws, insecure cryptographic usage, and improper error handling. In addition, OWASP Dependency-Check is used to identify known vulnerabilities in third-party libraries and transitive dependencies.

All critical and high-severity findings must be resolved or formally documented with justification before code is eligible for deployment, ensuring consistent enforcement of security and quality standards.

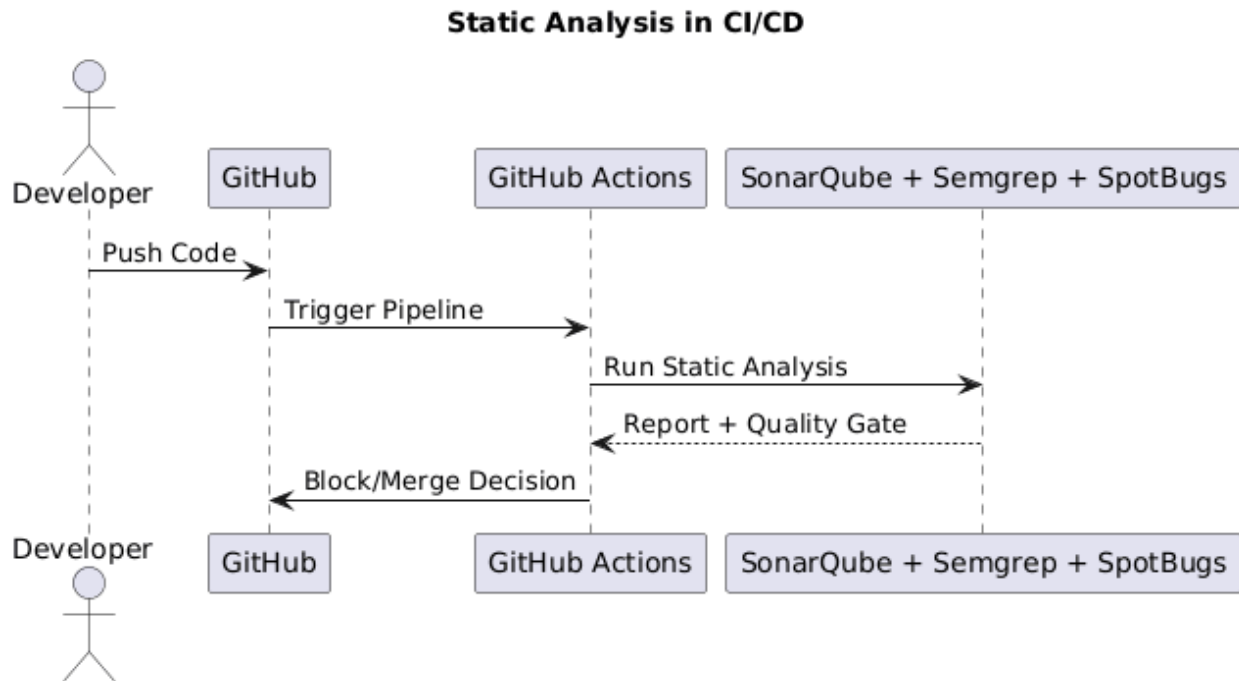


Figure 4.2 - Static Analysis Enforcement in CI/CD

This figure illustrates how static analysis tools are integrated into the CI/CD pipeline. Code changes pushed to GitHub trigger automated analysis through GitHub Actions, where tools such as SonarQube, Semgrep, and SpotBugs evaluate code quality and security. Analysis results are assessed against defined quality gates to determine whether code may be merged.

4.3 CI/CD TOOLING

CI/CD tooling in CareConnect automates build, test, and validation workflows to reduce human error and ensure consistent, repeatable software delivery. The CI/CD implementation emphasizes enforcement of quality and security controls prior to code promotion.

In CareConnect:

- **GitHub Actions** orchestrates automated builds, tests, and analysis workflows.
- **Quality gates** prevent code with failed tests or high-severity vulnerabilities from being merged into protected branches.
- **Deployment blocking** ensures that non-compliant builds cannot advance toward production environments.

Deployment triggers include:

- Merges to the development branch initiating integration or staging workflows
- Merges to the main branch initiating production-candidate pipelines

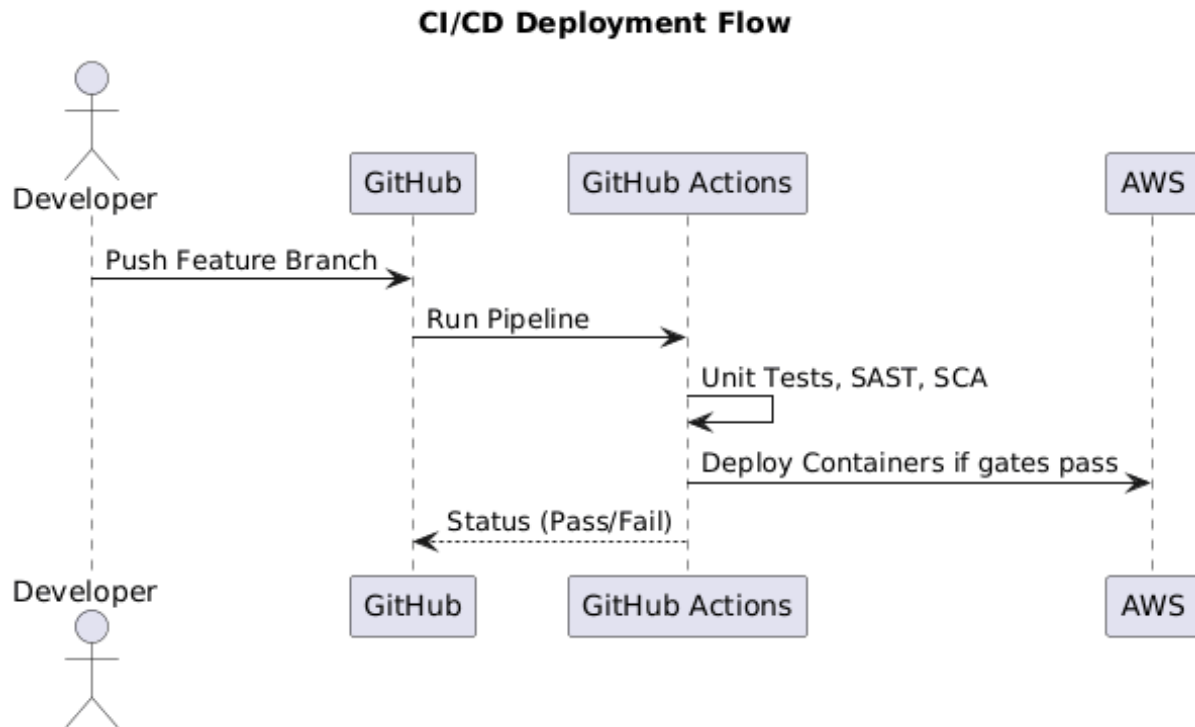


Figure 4.3 - CI/CD Deployment Flow with Quality Gates

This figure depicts the end-to-end CI/CD deployment flow. Feature branch changes trigger automated pipelines that execute tests, security scans (SAST and SCA), and quality checks. Only builds that pass all required gates are eligible for deployment to the AWS environment, ensuring controlled and secure releases.

4.4 CLOUD SERVICES

Amazon Web Services (AWS) provides the cloud infrastructure for CareConnect. Backend services are deployed as containerized applications running on Amazon ECS with AWS Fargate, eliminating the need for server management. Amazon API Gateway serves as the secure entry point for client requests, integrating with Amazon Cognito for authentication and authorization. Amazon RDS with PostgreSQL is used as the primary relational datastore, while Amazon S3 supports object storage for documents and static assets. AWS SNS and SES provide centralized notification and email delivery capabilities. Observability is achieved using Amazon CloudWatch for logging, metrics, and alerting.

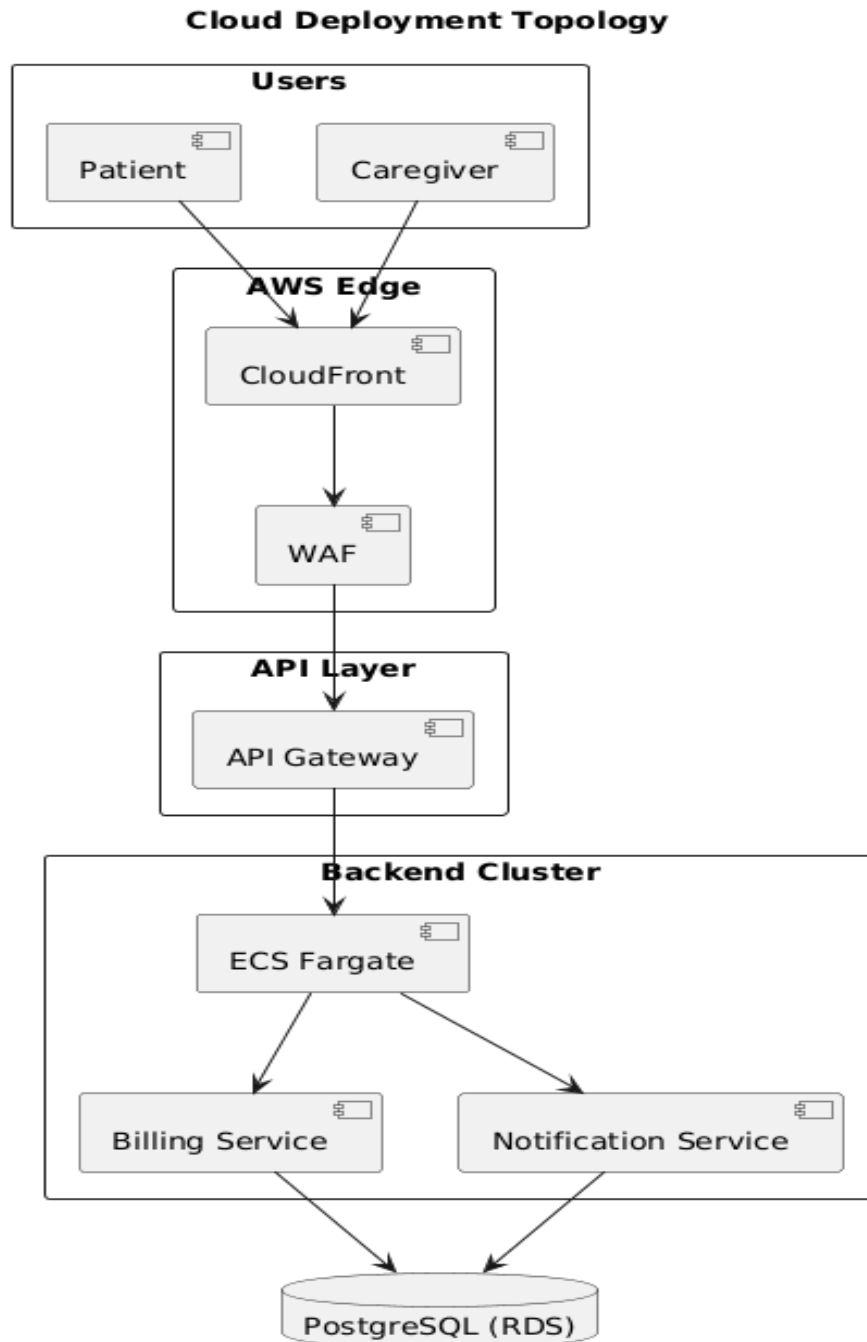


Figure 4.4 - Cloud Deployment Topology

This figure illustrates the CareConnect cloud deployment topology on AWS, showing how user traffic flows from edge services through the API layer to backend application services.

CloudFront and AWS WAF provide edge-level performance optimization and security, API Gateway manages controlled access to backend services, and ECS Fargate hosts the billing and notification services, which persist data in a managed PostgreSQL (RDS) database.

4.5 MOBILE PLATFORM DEPENDENCIES

CareConnect relies on platform-native dependencies to deliver compliant and optimized mobile experiences. On iOS, Apple StoreKit 2 is used to manage in-app subscriptions, transaction lifecycle events, and receipt generation. On Android, Google Play Billing Library is used for equivalent functionality. Both platforms require application-level configuration, entitlement validation, and secure handling of platform credentials. Push notification delivery depends on Firebase Cloud Messaging (FCM) for Android and Apple Push Notification Service (APNs) for iOS, with AWS SNS acting as the centralized broker.

5. DESIGN SPECIFICATIONS BY BUSINESS NEED

5.1 BUSINESS NEED 1 – AUTOMATED IDENTIFICATION OF BUGS, VULNERABILITIES, AND CODE QUALITY ISSUES

The project requires an automated mechanism to continuously identify software defects, security vulnerabilities, and code quality issues throughout the development and build lifecycle. Reliance on manual code reviews and post-deployment testing alone is insufficient to ensure consistent software quality, security, and maintainability at scale. As a result, there is a need for an integrated, tool-based approach that detects risks early, enforces standards consistently, and prevents defective or insecure code from being merged into protected branches or released to production.

5.1.1 Design Overview

The overall design approach is based on a shift-left quality and security strategy, in which automated analysis is embedded as early as possible in the software development lifecycle rather than being treated as a late-stage validation activity. The intent is to make code quality, security assurance, and defect detection a continuous and integral part of everyday development rather than a separate or optional process.

This approach emphasizes automation, consistency, and prevention by integrating static analysis tools directly into both the developer environment and the CI/CD pipeline. The design minimizes manual intervention while ensuring that objective, tool-based assessments govern code acceptance into protected branches.

5.1.2 Tool Selection and Justification

Rather than relying on one tool, we have decided to implement multiple tools into the development pipeline. Having multiple tools enables us the capability of having multiple layers of protection against software defects and software vulnerabilities.

Key selection criteria included:

- Breadth of defect coverage
- Integration with CI/CD pipelines
- Compatibility with the project's technology stack
- Performance and scalability
- Level of configurability
- Industry adoption and support

No single tool satisfied all these criteria; therefore, a multi-tool ecosystem was intentionally adopted.

5.1.2.1 SonarQube

SonarQube was chosen because it:

- Provides comprehensive analysis covering bugs, security vulnerabilities, code smells, duplication, and maintainability.
- Offers a centralized dashboard that enables leadership, developers, and reviewers to track quality trends over time.
- Integrates seamlessly with GitHub Actions and other CI/CD tools, enabling automated enforcement.
- Supports quality gates that can block merges or releases when thresholds are violated.
- Supports multiple languages and frameworks, making it suitable for multi-platform projects.

5.1.2.2 PMD

PMD was chosen because it:

- Effectively identifies dead code, unused variables, and overly complex methods.
- Allows highly customizable rule sets tailored to project needs.
- Is lightweight and fast enough to run reliably in CI builds.
- Has strong support for Java and integrates easily with Maven and Gradle.

5.1.2.3 Checkstyle

Checkstyle was chosen because it:

- Ensures uniform formatting, naming, and structural standards.
- Provides immediate feedback in IDEs and during builds.
- Has minimal performance impact, making it suitable for continuous integration.
- Helps onboard new developers by automatically enforcing team standards.

5.1.2.4 Spotbugs

SpotBugs was chosen because it:

- Detects critical issues such as null pointer dereferences, concurrency problems, and incorrect equals/hashCode implementations.
- Analyzes compiled bytecode, which increases accuracy in some cases compared to source-only tools.
- Has a relatively low false-negative rate for serious defects.
- Performs efficiently in CI pipelines.
- Supports security-focused plugins such as FindSecBugs.

5.1.2.5 OWASP Dependency-Check

OWASP Dependency-Check was chosen because it:

- Identifies known vulnerabilities in third-party libraries and transitive dependencies.
- Matches dependencies against publicly maintained vulnerability databases (CVEs).
- Provides severity ratings that support risk-based remediation decisions.
- Integrates with CI/CD pipelines to prevent vulnerable dependencies from being promoted.
- Complements static code analysis by addressing risks outside application source code.

Dependency vulnerability scanning ensures that security risks introduced through external components are identified and managed consistently alongside application-level defects.

Each tool excels in different areas and implementing them together covers coding style consistency, maintainability, design quality, runtime defect risks, security, metrics, and governance.

5.1.3 Local Developer Workflow Design

Developers are expected to execute static analysis tools locally prior to committing code or opening a pull request. This practice ensures that defects, security risks, and coding standard violations are identified and resolved early in the development lifecycle, reducing pipeline failures and rework. Local execution of analysis tools supports a shift-left quality strategy by enabling immediate feedback within the developer's workspace rather than relying solely on CI/CD enforcement.

Developers execute automated analysis using a combination of IDE-integrated plugins and command-line tools, depending on their development environment and workflow.

5.1.3.1 IDE-Based Analysis (Real-Time Feedback)

Developers primarily interact with analysis tools through their integrated development environment (IDE), such as IntelliJ IDEA, or Visual Studio Code. The following capabilities are available:

- **Checkstyle Integration**

- Developers receive real-time warnings for formatting, naming, and structural violations.
- Violations are highlighted directly in the code editor.
- Developers can configure the IDE to block commits or display errors for severe violations.

- **PMD Integration**

- PMD rules are applied to source files as they are edited.
- Issues such as unused variables, duplicated code, or overly complex methods are flagged in the IDE.

Developers can click on findings to view rule explanations and remediation suggestions.

-

- **SpotBugs Integration**

- SpotBugs plugins analyze compiled code within the IDE.
Potential runtime defects (e.g., null pointer risks, concurrency issues) are surfaced with contextual warnings.
- Findings include descriptions of the risk and recommended fixes.

This real-time feedback reduces the likelihood that problematic code reaches version control. Before pushing code to the repository, developers are expected to run the project's build and analysis tools locally via the command line.

If any tool reports critical violations, developers are expected to remediate issues before proceeding.

Developers follow a structured review process when analysis results are generated:

1. Prioritize Critical Issues First

- Security vulnerabilities
- Blocker or critical bugs
- High-risk runtime defects

2. Resolve Functional Risks Next

- Null pointer risks
- Concurrency problems
- Logic errors

3. Address Maintainability and Style Issues

- Code duplication
- Unused variables
- Formatting violations

4. Document Exceptions When Necessary

- If a finding is a known false positive, developers may annotate code or request rule exceptions through team governance processes.

5.1.4 CI Integration Design

Describe how automated analysis is triggered and executed within CI pipelines.

Automated static analysis is integrated as a mandatory component of the continuous integration (CI) pipeline to ensure that all code changes are systematically evaluated for defects, security vulnerabilities, and coding standard violations before being merged into protected branches. The analysis is triggered automatically by repository events and executed in a controlled, repeatable build environment. This approach eliminates reliance on manual checks and enforces consistent quality standards across the development lifecycle.

5.1.5 Findings Classification and Prioritization

Findings generated by automated static analysis tools are categorized according to severity and priority to enable consistent triage, risk management, and remediation. Severity reflects the technical impact, while priority reflects the urgency based on business, operational, and release considerations.

5.1.5.1 Blocker (Highest Severity)

Issues likely to cause system failure, data loss, or security breaches.

Examples include:

- Critical security vulnerabilities
- Guaranteed runtime crashes (e.g., certain null pointer dereferences)
- Serious concurrency defects
- Logic flaws that break core functionality

Handling:

- Automatically fails the CI pipeline
- Blocks pull request merges
- Requires immediate remediation

5.1.5.2 Critical

High-risk defects that may cause severe operational or security issues.

Examples include:

- Potential memory leaks
- Race conditions
- High-risk security weaknesses
- Major architectural violations

Handling:

- Fails the SonarQube quality gate
- Must be resolved before merging to protected branches

5.1.5.3 Major

Significant maintainability or reliability risks.

Examples include:

- Complex methods that are hard to maintain
- Significant code duplication
- Risky programming patterns
- Poor error handling

Handling:

- Typically must be addressed before release
- May be allowed temporarily in feature branches with justification

5.1.5.4 Minor

Low-risk quality or readability concerns.

Examples include:

- Style inconsistencies
- Minor naming violations
- Small maintainability improvements

Handling:

- Do not block builds by default
- Addressed through normal hygiene

5.1.5.5 Informational (Lowest Severity)

Informational findings provide suggestions or best-practice recommendations rather than warnings of defects.

Examples include:

- Readability suggestions
- Documentation recommendations
- Code clarity improvements

Handling:

- Optional remediation
- Used for continuous improvement rather than enforcement

5.1.6 Documentation and Deferral Strategy

Not all static analysis findings can or should be resolved immediately. Some findings may be false positives, acceptable technical debt, legacy issues, or require significant architectural changes. To ensure transparency, accountability, and traceability, unresolved findings are formally documented with clear rationale and, where applicable, remediation guidance. This process prevents issues from being ignored while allowing controlled, risk-based decision-making.

All unresolved findings remain visible in SonarQube and associated tracking systems to ensure they are revisited over time rather than permanently dismissed.

A finding may remain unresolved under controlled circumstances, including:

- **Confirmed false positives** (tool misinterpretation of code)
- **Legacy code constraints** that cannot be easily refactored
- **High remediation cost relative to risk**
- **Planned future refactoring or migration**
- **Dependency or framework limitations**
- **Temporary exceptions for active development**

In all cases, a documented justification is required.

5.1.7 Constraints and Limitations

Document known limitations, tradeoffs, or tool constraints.

While the selected static analysis toolchain significantly improves software quality, security, and maintainability, it is not a complete or perfect solution. This section documents known limitations, operational tradeoffs, and inherent tool constraints to ensure realistic expectations, informed decision-making, and proactive risk management. These constraints are acknowledged and mitigated through process controls, governance practices, and complementary testing strategies.

5.1.7.1 Performance and Resource Constraints

- **SonarQube Performance Overhead**

- **Limitation:**

- SonarQube is resource-intensive, particularly for large codebases or multi-module projects. High CPU and memory usage can slow down analysis and increase infrastructure costs.

- **Tradeoff:**

- Benefit: Centralized governance, trend tracking, and automated gating.
 - Cost: Additional compute resources and longer analysis times in CI.

- **Mitigation Strategy:**

- Enable incremental analysis where possible.
 - Optimize rule sets to reduce unnecessary checks.
 - Cache dependencies in CI to reduce runtime.

- **CI Pipeline Execution Time**

- **Limitation:**

- Running Checkstyle, PMD, SpotBugs, and SonarQube sequentially increases build duration.

- **Tradeoff:**

- Benefit: Higher confidence in code quality and security.
 - Cost: Slower feedback cycles for developers.

- **Mitigation Strategy:**

- Parallelize analysis steps where feasible.
 - Use fast fail behavior for critical failures.
 - Encourage local analysis before opening pull requests.

5.1.7.2 False Positives Across Tools

- **Limitation:**

- All tools may occasionally flag issues that are not real defects or security risks.

- SonarQube: Some rules are overly conservative without tuning.
 - PMD: Generic rules may misinterpret framework-specific patterns.
 - SpotBugs: Can misjudge complex modern Java constructs.
 - Checkstyle: May flag stylistic issues that have no functional impact.

- **Tradeoff:**

- Benefit: Lower risk of missing real defects.
 - Cost: Developer effort spent reviewing non-issues.

- **Mitigation Strategy:**

- Maintain curated rule sets.
 - Allow documented suppressions for validated false positives.
 - Periodically review and refine rules.

5.2 BUSINESS NEED 2 – CI/CD BUILD AND DEPLOYMENT CONTROLS

5.2.1 Design Overview

CI/CD build and deployment controls ensure that software is built, validated, and promoted in a consistent, secure, and repeatable manner while minimizing risk to production systems. These controls automate compilation, testing, static analysis, dependency vulnerability scanning, and quality validation so that defects, vulnerabilities, and non-compliant changes are detected before integration or release.

The scope of these controls spans the entire delivery lifecycle—from code commit through artifact generation, quality gating, versioning, and controlled deployment. By embedding technical and policy-based checkpoints directly into the pipeline, CI/CD controls enforce standards, prevent unsafe or unauthorized changes, and provide traceability, auditability, and reliability across software delivery.

5.2.2 Pipeline Flow and Stages

CI/CD build and deployment controls ensure that software is built, validated, and promoted in a consistent, secure, and repeatable manner while minimizing risk to production systems. These controls automate compilation, testing, static analysis, dependency vulnerability scanning, and quality validation so that defects, vulnerabilities, and non-compliant changes are detected before integration or release.

The scope of these controls spans the entire delivery lifecycle—from code commit through artifact generation, quality gating, versioning, and controlled deployment. By embedding technical and policy-based checkpoints directly into the pipeline, CI/CD controls enforce standards, prevent unsafe or unauthorized changes, and provide traceability, auditability, and reliability across software delivery.

5.2.3 Gate Conditions and Failure Criteria

Builds, merges, and deployments are automatically blocked when predefined technical, quality, or security conditions are not met. This prevents unstable or vulnerable code from advancing. Builds are blocked if compilation fails, required tests fail, dependencies cannot be resolved, or critical pipeline steps encounter errors. Merges into protected branches are blocked when

required status checks, peer reviews, or quality gates fail, or when critical findings from tools such as SonarQube, SpotBugs, PMD, Checkstyle, or OWASP Dependency-Check remain unresolved.

Deployments are blocked if artifacts do not match approved builds, vulnerability thresholds are exceeded, environment checks fail, or required approvals are missing. These controls ensure that only verified, secure, and compliant artifacts are eligible for production release.

5.2.4 Developer Feedback and Reporting

CI/CD results and failures are communicated to developers through integrated feedback mechanisms that support rapid remediation. Pipeline status is displayed directly on commits and pull requests using pass/fail indicators, execution logs, and test reports.

Findings from static analysis and security tools—including SonarQube dashboards and dependency vulnerability reports—are presented through centralized reporting interfaces and pull request annotations. Critical issues are clearly highlighted and linked to remediation guidance. Where configured, alerts for blocked builds or merges are surfaced through team communication channels to ensure visibility and timely resolution.

5.2.5 Promotion and Approval Logic

Promotion of builds through pipeline stages is governed by predefined technical, quality, and security criteria. Advancement requires successful compilation, passing automated tests, and completion of all required analysis stages with quality gates satisfied, including the absence of blocking vulnerabilities and unresolved dependency risks.

Artifacts must be correctly versioned, validated, and stored in the artifact repository before promotion. Advancement to higher environments additionally requires alignment with the approved commit, successful environment verification, and completion of required human approvals or change controls. This ensures that only stable, secure, and compliant builds progress toward deployment.

5.2.6 Traceability and Auditability

All pipeline activities and decisions are automatically logged to provide end-to-end traceability and auditability. Build logs, test results, static analysis findings, dependency scan reports, and quality gate outcomes are retained within the CI/CD platform and associated artifact repositories. Key decisions—such as blocked builds, approved promotions, or deployment actions—are recorded with associated commit identifiers, timestamps, and responsible actors. These records are preserved according to defined retention policies to support compliance verification, incident investigation, and continuous improvement.

5.3 BUSINESS NEED 3 – NATIVE MOBILE PAYMENT MIGRATION

5.3.1 Design Overview

The Native Mobile Payment Migration addresses compliance, reliability, and platform alignment concerns arising from the continued use of third-party payment processors for mobile digital goods. Apple App Store and Google Play Store policies require that digital subscriptions and in-app content purchases be processed using platform-provided billing systems. Failure to comply introduces significant risk, including application rejection, forced delisting, and revenue disruption.

To mitigate these risks, CareConnect adopts a hybrid billing architecture that separates payment execution from subscription state management. Native billing SDKs (Apple StoreKit 2 and Google Play Billing) are used exclusively for initiating, confirming, and managing payments on mobile devices, while the backend Billing Service maintains a platform-agnostic representation of subscription state.

This design ensures:

- Full compliance with platform billing policies
- Consistent backend authorization logic
- Minimal coupling between business logic and vendor-specific APIs
- Future extensibility for additional platforms or pricing models

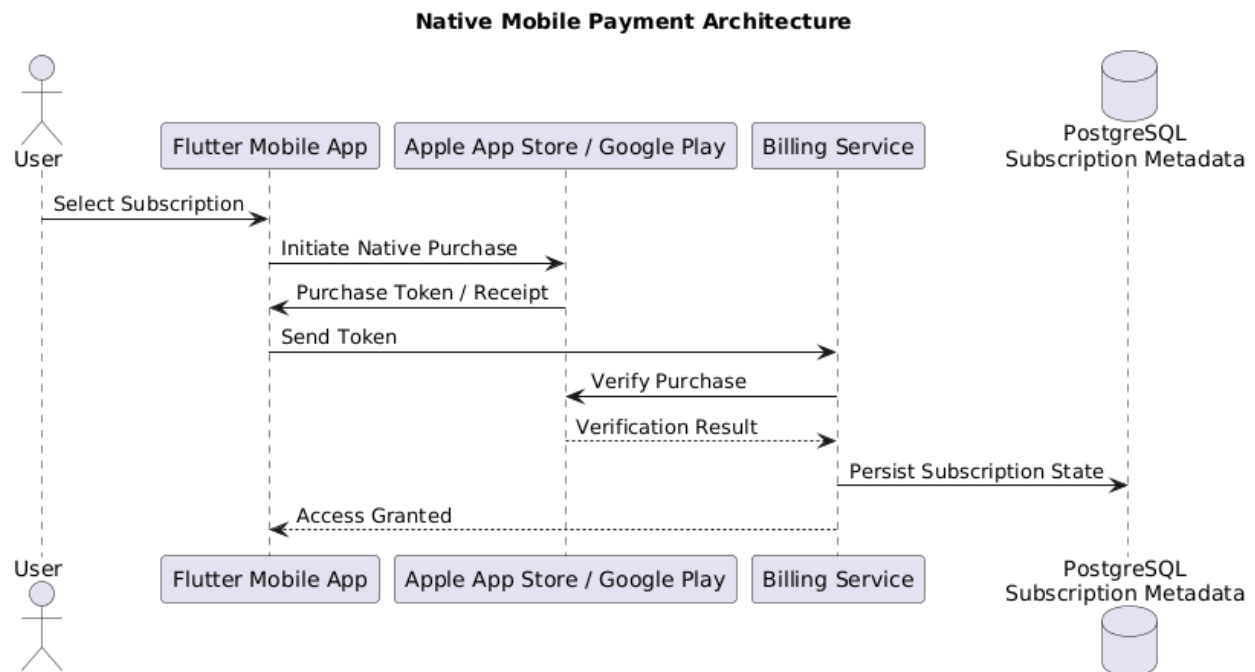


Figure 5.1 - Native Mobile Payment Architecture

This diagram illustrates the native mobile subscription payment flow for CareConnect, showing how the Flutter mobile application initiates purchases through Apple App Store or Google Play billing services, verifies purchase tokens with the backend billing service, and persists subscription state in PostgreSQL before granting user access.

5.3.2 Stripe Decommissioning Strategy

Stripe is currently used as the payment processor; however, it is not permitted for mobile digital subscriptions under Apple and Google policies. To avoid service disruption while maintaining backward compatibility, CareConnect follows a phased decommissioning strategy.

Phase 1: Isolation

- Stripe logic is encapsulated within the Billing Service.
- New mobile clients are routed exclusively through native billing paths.
- Stripe remains active only for legacy or non-mobile use cases.

Phase 2: Dual State Management

- Stripe subscription identifiers are mapped to internal subscription records.
- Platform-neutral subscription states (ACTIVE, GRACE, EXPIRED) are introduced.
- Access control logic references internal state, not Stripe directly.

Phase 3: Deactivation

- Stripe APIs are disabled for mobile subscription flows.
- Historical Stripe data is retained for auditing and reporting.
- Stripe keys are revoked and removed from Secrets Manager.

This approach minimizes risk while ensuring compliance and continuity.

5.3.3 Apple Native Payment Design

Apple payments are implemented using StoreKit 2, which provides modern APIs for handling subscriptions, renewals, refunds, and grace periods.

Key Characteristics

- Purchases occur entirely within Apple-controlled UI.
- Receipts are cryptographically signed by Apple.
- Subscription lifecycle events are asynchronously communicated.

Implementation Flow

- Flutter app invokes StoreKit via platform channel.
- User completes purchase through native dialog.
- Receipt is generated and cached locally.
- Receipt is sent to backend Billing Service.
- Backend validates receipt with Apple servers.
- **Subscription state is updated in PostgreSQL.**

Edge Case Handling

- Offline purchases queued for later verification
- Receipt refresh on device reinstall
- Grace period enforcement during billing issues

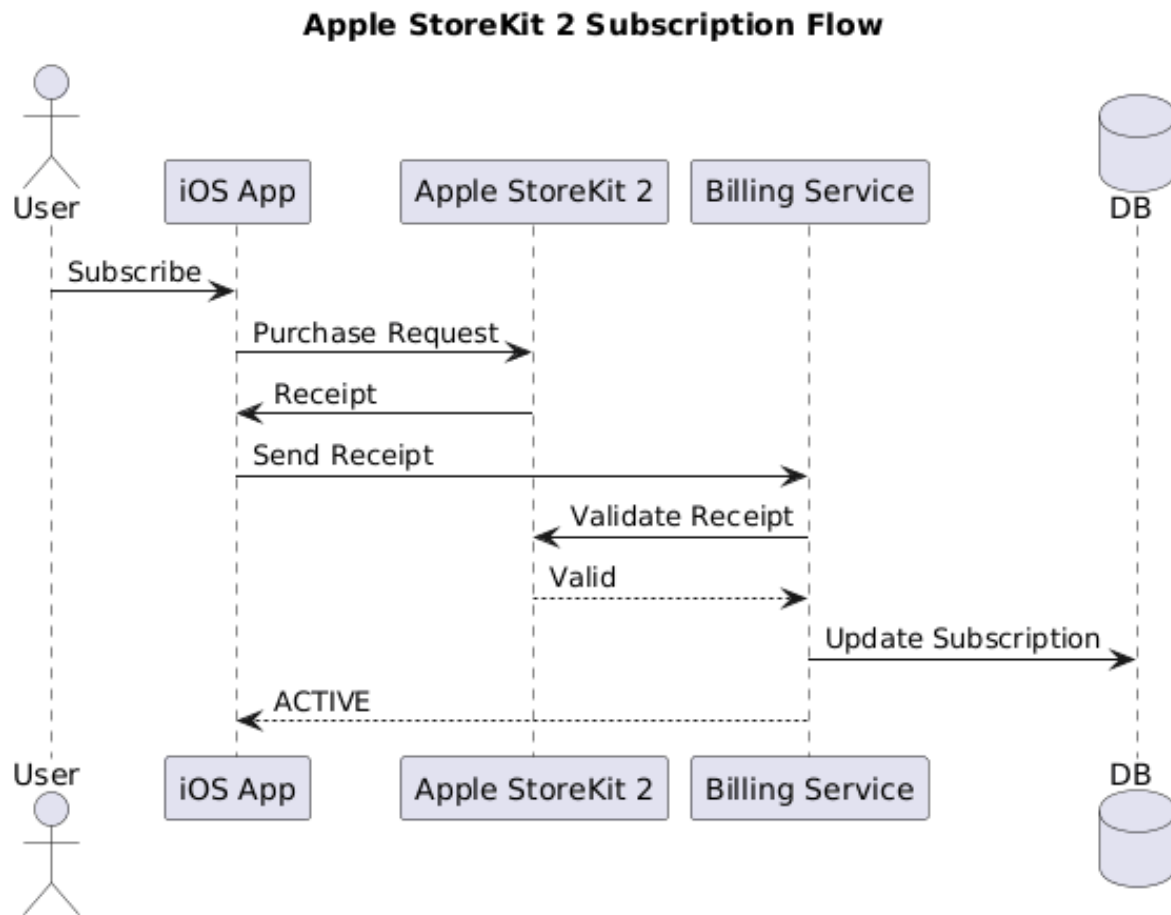


Figure 5.2 - Apple StoreKit 2 Subscription Flow

This diagram shows the iOS subscription purchase flow using Apple StoreKit 2, where the iOS app initiates a subscription request, receives a purchase receipt, submits it to the backend billing service for validation, and updates the subscription state in the database before activating access for the user.

5.3.4 Google Native Payment Design

On Android, the Google Play Billing Library manages subscriptions and lifecycle events.

Purchase tokens are securely transmitted to the backend, where the Billing Service verifies them using Google Play Developer APIs before persisting subscription metadata. This design mirrors Apple's model while providing Android-specific features such as retry policies and purchase acknowledgment.

Key Characteristics

- Purchase tokens uniquely identify subscriptions.
- Acknowledgment is mandatory to prevent auto-refunds.
- Backend validation ensures integrity and replay protection.

Implementation Flow

- Flutter app invokes Google Play Billing client.
- User confirms subscription via Play UI.
- Purchase token returned to app.
- Token sent to backend for validation.
- Subscription acknowledged and persisted.

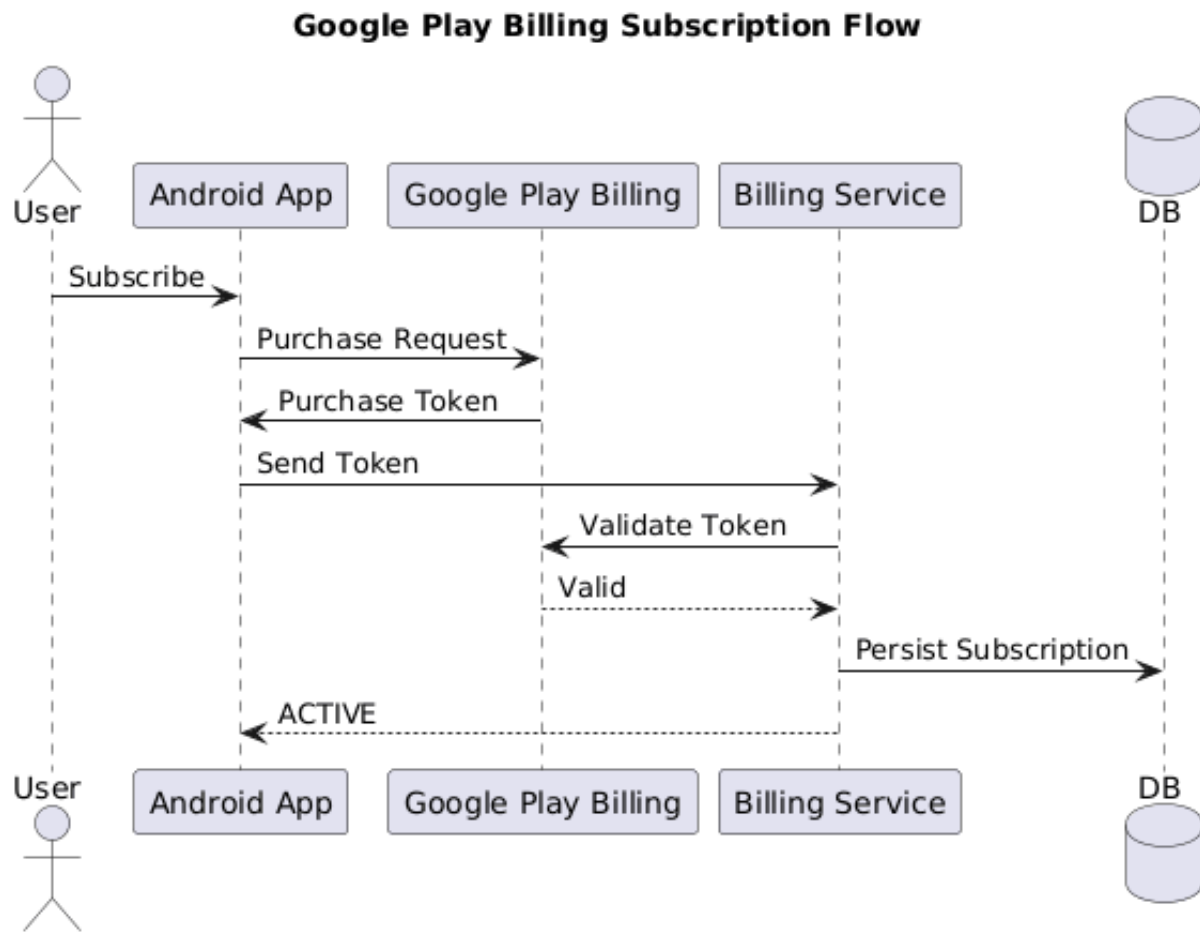


Figure 5.3 - Google Play Billing Subscription Flow

This diagram illustrates the Android subscription flow using Google Play Billing, where the Android app initiates a purchase request, receives a purchase token, submits it to the backend billing service for validation, and persists the subscription state in the database before granting active access to the user.

5.3.5 Subscription and State Management

Subscription and state management within CareConnect is centralized in the backend and implemented using a platform-neutral data model stored in PostgreSQL. This design ensures that subscription authorization and access control logic remain consistent across both iOS and Android clients, regardless of the native payment system used to complete a transaction. Each subscription record captures essential metadata including the originating platform, external transaction or purchase reference, current subscription status, renewal and expiration dates, and

cancellation indicators. By abstracting these attributes away from platform-specific details, CareConnect maintains a single authoritative view of a user's entitlement status that can be reliably referenced by all backend services.

The system defines a standardized set of canonical subscription states that represent the full lifecycle of a user's subscription, including ACTIVE, GRACE_PERIOD, SUSPENDED, CANCELED, and EXPIRED states. These states are designed to align with common behaviors across both Apple and Google billing ecosystems, allowing CareConnect to handle scenarios such as payment retries, temporary billing failures, and voluntary cancellations in a predictable and auditable manner. Centralizing these states enables backend services to make consistent authorization decisions while also simplifying reporting, analytics, and support workflows. Responsibility for enforcing subscription access resides exclusively with the backend, ensuring that business rules cannot be bypassed or inconsistently applied across client platforms. Mobile applications are responsible only for displaying subscription status to the user and initiating platform-native purchase flows, but they do not directly control entitlement enforcement. Platform-generated events such as renewals, cancellations, or payment failures are processed by the backend Billing Service, which validates these events and updates subscription state accordingly. This abstraction decouples authorization logic from payment vendors, allowing CareConnect to evolve billing providers or introduce additional platforms in the future without requiring changes to core access control mechanisms.

5.3.5.1 Subscription Lifecycle State Model

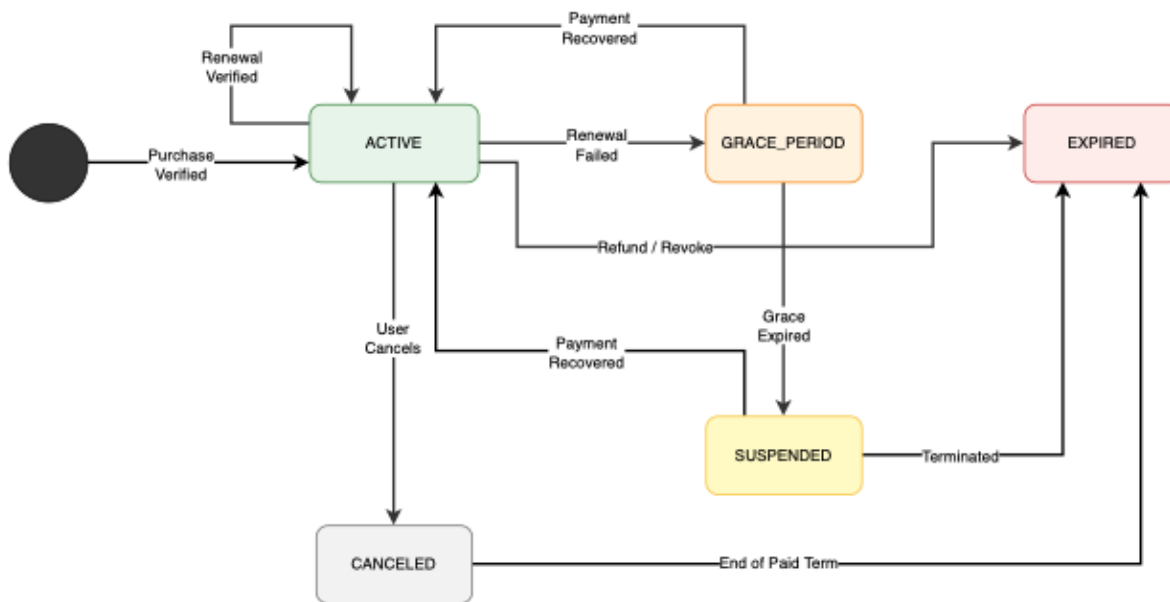


Figure 5.4 – Subscription Lifecycle State Diagram

CareConnect manages subscriptions using a platform-neutral lifecycle model to ensure consistent entitlement enforcement across iOS and Android platforms. The transitions illustrated in this diagram are triggered by validated purchase events and renewal webhooks described in Figures 5.5 (Payment Purchase and Validation Sequence) and 5.6 (Renewal/Webhook Processing Sequence). The Billing Service acts as the authoritative source of truth for subscription state, validating store events, persisting state transitions, and returning entitlement decisions to client applications.

A successful purchase verification transitions the subscription from its initial state to **ACTIVE**. Subsequent renewal verification events return the subscription to **ACTIVE**, preserving uninterrupted access. If a renewal attempt fails but the platform indicates a grace window, the subscription transitions to **GRACE_PERIOD**, allowing continued entitlement while payment issues are resolved.

If payment is recovered during the grace window, the subscription transitions back to **ACTIVE**. If the grace period expires without recovery, the subscription transitions to **SUSPENDED**. A suspended subscription may be restored to **ACTIVE** if payment is successfully recovered within

platform-defined limits. If suspension persists or the platform issues a termination event, the subscription transitions from **SUSPENDED** to **EXPIRED**.

User-initiated cancellations transition the subscription from **ACTIVE** to **CANCELED**, where entitlement remains valid until the end of the paid billing term. Upon reaching the natural expiration of that term, the subscription transitions from **CANCELED** to **EXPIRED**. Refunds, revocations, or charge reversals may immediately transition a subscription from **ACTIVE** to **EXPIRED**, reflecting authoritative store state.

Each state transition results in:

- An update to the `Subscription.status` field (see Figure 6.1 - Logical Data Model),
- The insertion of a corresponding `PaymentTransaction` record to preserve immutable transaction history,
- And the creation of an `AuditRecord` entry to maintain compliance-relevant traceability.

By explicitly modeling subscription states and transitions, the system ensures deterministic entitlement decisions, consistent cross-platform behavior, and complete traceability across purchase, renewal, cancellation, recovery, suspension, and expiration workflows.

5.3.5.2 Payment Purchase and Validation Sequence

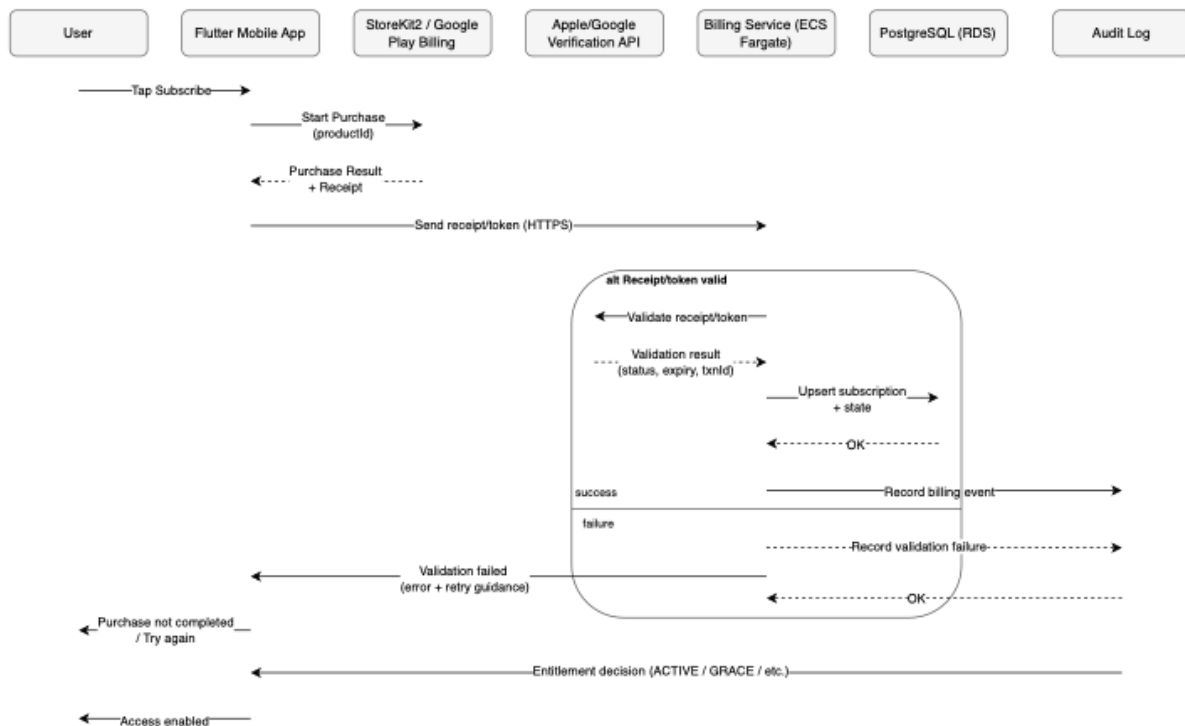


Figure 5.5 – Native Purchase and Receipt Validation Sequence Diagram

This sequence diagram illustrates the end-to-end subscription purchase flow using platform-native billing systems with backend receipt validation and centralized entitlement enforcement. The user initiates a subscription purchase from the Flutter mobile application. The mobile client invokes the platform-native billing SDK (Apple StoreKit 2 on iOS or Google Play Billing on Android), and the user completes the purchase through the app store’s native UI. The platform returns a signed receipt (iOS) or purchase token (Android) to the client. The client securely transmits the receipt or purchase token to the backend Billing Service over authenticated HTTPS. The Billing Service calls the Apple/Google Verification API to validate the receipt or token and confirm authenticity, subscription status, expiration details, and transaction integrity.

If validation succeeds, the Billing Service upserts the Subscription record and updated lifecycle state in PostgreSQL and records the billing event in the Audit Log for traceability. The Billing Service then returns an entitlement decision (e.g., ACTIVE, GRACE_PERIOD) to the client application.

If validation fails, the Billing Service records the validation failure in the Audit Log and returns an error response with retry guidance. In this case, no subscription state change is applied.

Upon receiving a successful entitlement decision, the client updates the user interface and enables subscription-gated functionality.

This design ensures platform policy compliance by enforcing server-side validation before any entitlement is granted. The backend remains the single source of truth for subscription state, preventing client-side entitlement bypass and ensuring consistent behavior across platforms.

5.3.5.3 – Renewal / Webhook Processing Sequence

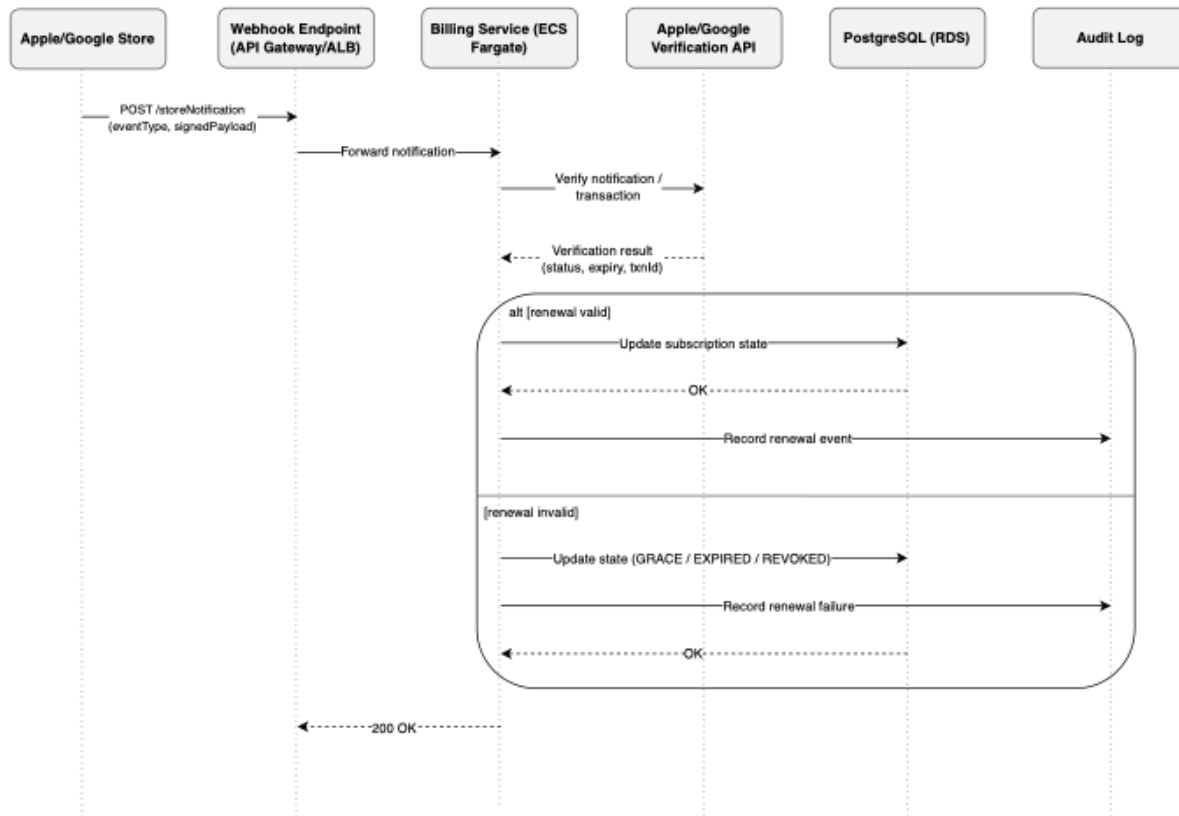


Figure 5.6 – Renewal / Webhook Processing Sequence Diagram

This sequence diagram illustrates the asynchronous renewal and billing status update workflow triggered by Apple or Google store server notifications. Unlike the client-initiated purchase flow shown in Figure 5.5, this process is event-driven and originates from the platform store, ensuring subscription state remains synchronized even when the user is not actively using the application. When a renewal, cancellation, expiration, refund, or revocation event occurs, the Apple/Google Store sends a signed POST /storeNotification payload to the system’s Webhook Endpoint (API Gateway / ALB). The endpoint forwards the request to the Billing Service running on ECS Fargate.

The Billing Service verifies the authenticity and integrity of the webhook payload by validating the notification or associated transaction with the Apple/Google Verification API before applying

any state change. This validation confirms the subscription status, expiration metadata, and transaction identifiers before any state change is applied.

If the renewal or billing event is valid:

- A new `PaymentTransaction` record is inserted to preserve immutable event history.
- The corresponding `Subscription` record is updated to reflect the authoritative lifecycle state (e.g., `ACTIVE`, `GRACE_PERIOD`, `EXPIRED`, `REVOKED`).
- An `AuditRecord` entry is generated to capture the state transition and correlation identifiers.

If the renewal is invalid or indicates a failed payment:

- The `Subscription.status` field is updated accordingly (`GRACE_PERIOD`, `SUSPENDED`, or `EXPIRED`).
- A `PaymentTransaction` record is still inserted to maintain traceability.
- An audit entry records the failure condition for compliance and reconciliation purposes.

All updates correspond directly to the entities shown in Figure 6.1 (Logical Data Model). Once processing is complete, the Billing Service returns an HTTP 200 OK response through the Webhook Endpoint, preventing store-side retries.

This asynchronous webhook mechanism ensures that subscription state remains accurate and platform-authoritative without requiring client interaction. By validating all renewal events server-side and persisting each transition, the system maintains deterministic entitlement enforcement, full billing traceability, and resilience against missed or delayed client-side updates.

5.3.6 Error Handling and Recovery

The payment architecture incorporates comprehensive error handling and recovery mechanisms to address the inherent variability of mobile networks, third-party platform dependencies, and asynchronous transaction processing. Transient failures, including temporary network disruptions, delayed receipt validation, or short-lived store service outages, are handled through automated retry mechanisms designed to resolve issues without user intervention whenever possible. These retries are executed in a controlled manner to prevent duplicate processing while maintaining system responsiveness and data integrity.

In cases where payment validation fails definitively, such as invalid receipts, revoked transactions, or expired subscriptions, the system responds by revoking access and recording the event in an immutable audit log. This ensures both compliance and traceability while protecting the system from unauthorized access. Client-side error handling is intentionally designed to provide clear, user-friendly feedback that guides users toward resolution, such as retrying a purchase or updating payment information, without exposing internal system details or security-sensitive information.

Recovery strategies extend beyond automated processes to include administrative tooling that supports manual reconciliation when required. Grace periods may be applied during subscription renewal disputes or delayed platform confirmations, allowing continued access while backend verification is completed. Administrative interfaces enable authorized staff to review transaction histories, reconcile discrepancies, and resolve user-reported issues efficiently. Together, these mechanisms ensure that CareConnect remains resilient in the face of payment-related failures while maintaining a high-quality user experience and strong operational integrity.

5.3.7 Compliance and Policy Considerations

CareConnect's native payment architecture is explicitly designed to comply with the policies and regulatory requirements enforced by both Apple and Google for digital goods and subscription-based services. All in-app purchases are processed exclusively through Apple In-App Purchases and Google Play Billing, ensuring adherence to platform rules governing payment flow, pricing disclosure, cancellation handling, and refund policies. By delegating payment execution and sensitive financial data handling to the platform providers, CareConnect eliminates the need to

store or process payment credentials directly, significantly reducing security risk and compliance burden.

This approach ensures alignment with Apple App Store Review Guidelines, including Section 3.1.1, and with Google Play's Payments Policy, which mandate the use of native billing systems for digital services. Because CareConnect never handles raw cardholder data, the scope of PCI DSS compliance is substantially reduced, simplifying security management while maintaining strong protections for end users. Platform providers are responsible for encryption, fraud detection, and payment authorization, allowing CareConnect to focus on application-level security and healthcare data protection.

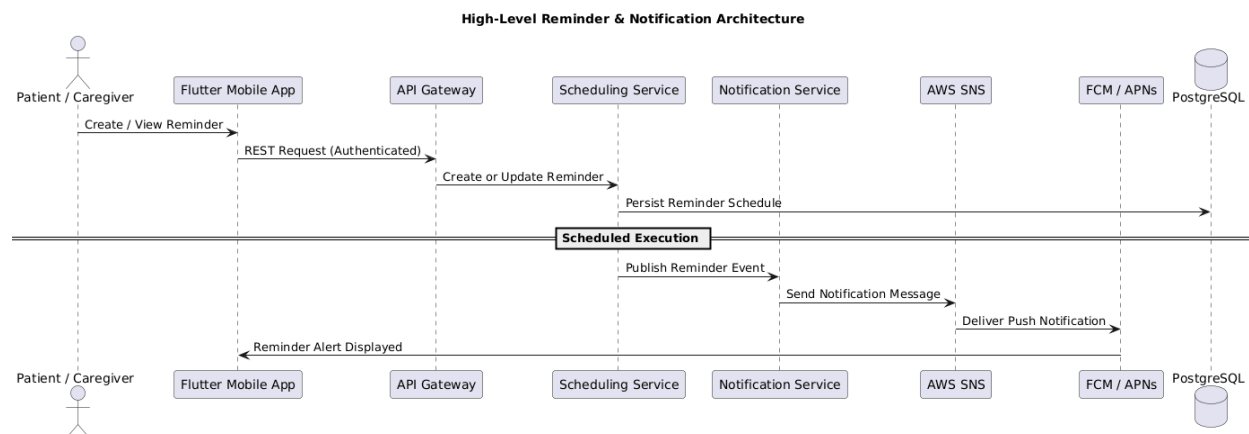
To support auditing, compliance verification, and dispute resolution, the system maintains detailed audit logs that record all subscription state changes, validation timestamps, and references to platform-generated transaction identifiers. These logs provide a transparent and traceable history of billing-related events, supporting internal reviews, regulatory audits, and future system maintenance. By combining strict policy adherence with comprehensive auditing, CareConnect ensures that its payment system remains compliant, secure, and sustainable as the platform evolves.

5.4 BUSINESS NEED 4 – CENTRALIZED NOTIFICATIONS AND REMINDERS

5.4.1 Design Overview

Describe the centralized notification architecture and its objectives.

This design centralizes all user notifications and time-based reminders behind a single, cloud-managed messaging backbone using AWS Simple Notification Service (SNS). Application services publish domain events (for example, `ReminderDue`, `EscalationTriggered`, `SystemAlert`) to well-defined SNS topics. Channel-specific workers or managed integrations subscribe to those topics and deliver messages via push (APNs/FCM), SMS, and email, applying user preferences, quiet hours, and localization. The objectives are: consistent behavior across channels, reduced coupling between business logic and delivery mechanisms, scalable fan-out, auditable delivery lifecycles, and straightforward extensibility for new channels or rules without rewriting core services.



5.7 Notification Triggers and Event Sources

Explain where notification events originate within the system.

Notification events originate from several bounded contexts. The Scheduling service emits events when a reminder reaches its due time or enters grace/overdue states. The Care Plan service emits adherence and follow-up reminders based on plan logic. The Communications service emits chat mentions or appointment confirmations. The Monitoring/Alerts service emits threshold and exception alerts. Each source publishes a compact JSON payload with message

attributes (urgency, channel hints, locale, tenant, user and caregiver identifiers, correlation IDs) to the appropriate SNS topic. Acknowledgment or completion events from clients are published back to an “ack” endpoint, updating state and canceling further retries or escalations.

5.4.3 Cloud Messaging Integration

AWS Simple Notification Service (SNS) serves as the centralized publish-subscribe backbone for all notification events within CareConnect. Backend services running on Amazon ECS (Fargate) publish domain events—such as reminder due, escalation triggered, or system alert—to dedicated SNS topics using least-privilege IAM roles.

SNS distributes events to subscribed delivery components without embedding channel-specific logic in the publishing services. Subscriptions include HTTPS endpoints backed by containerized services (ECS tasks), Amazon SQS queues for buffered processing and retry management, and direct SMS delivery where appropriate.

All notification transformation, routing logic, and templating are handled by containerized notification workers running on Amazon ECS (Fargate). No AWS Lambda functions are used in this architecture. This ensures architectural consistency, operational predictability, and alignment with the system’s containerized deployment model.

SNS message attributes are used to implement filter policies so that subscribers receive only relevant events (for example, channel = sms and urgency = critical). For mobile push notifications, a dedicated push worker translates normalized domain events into platform-specific APNs or FCM payloads, applying deep links, time-to-live (TTL), and collapse identifiers as required.

Dead-letter queues (Amazon SQS) are attached to SNS subscriptions to capture delivery failures and enable controlled replay and operational recovery. All publishing, processing, and delivery orchestration remain within containerized backend services running on Amazon ECS (Fargate), preserving separation of concerns and eliminating serverless dependencies.

5.4.3.1 Notification Service Component Architecture

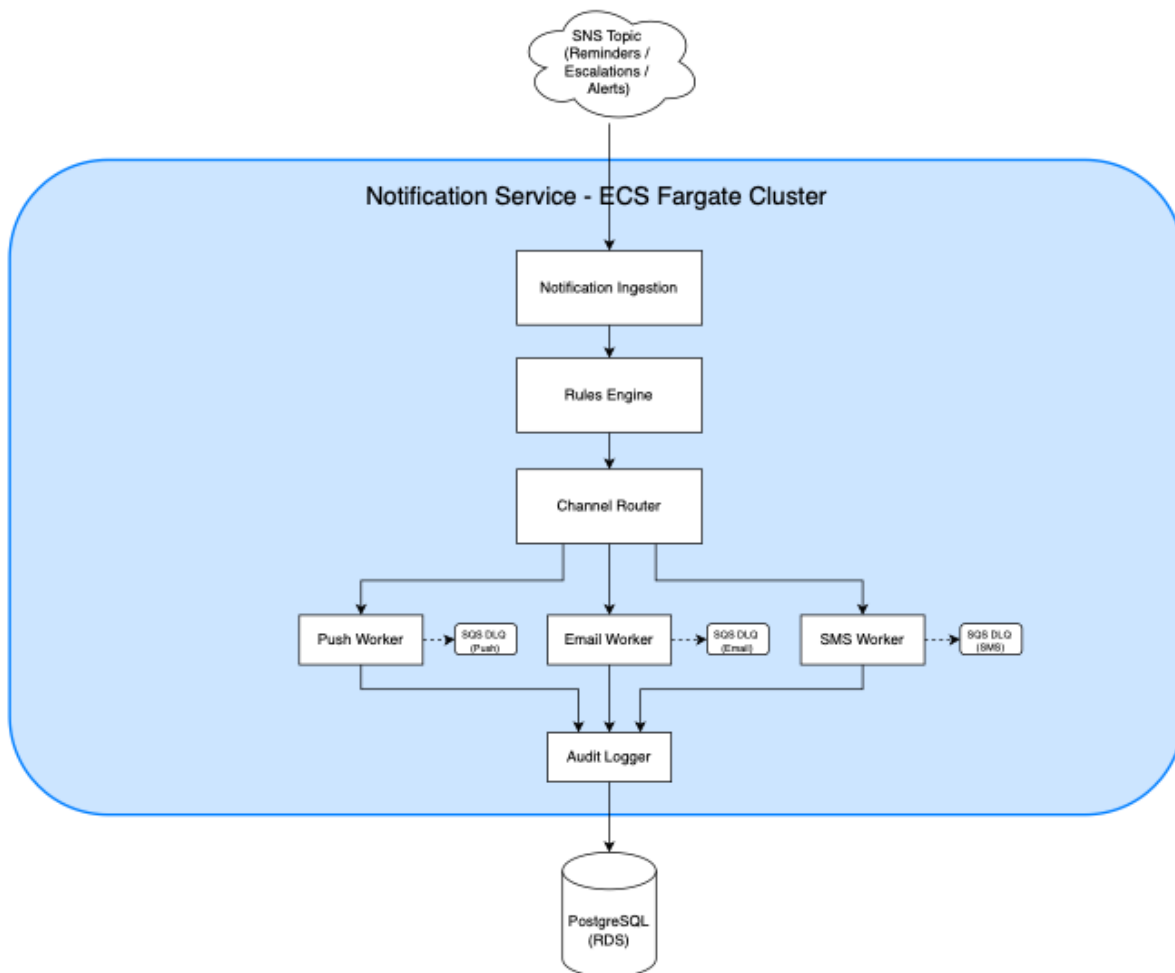


Figure 5.8 – Notification Service Component Architecture

The Notification Service is implemented as a collection of containerized components deployed within an Amazon ECS (Fargate) cluster. The architecture separates event ingestion, decision logic, channel processing, and auditing responsibilities to ensure modularity, scalability, and fault isolation. This component structure supports the runtime behaviors illustrated in Figures 5.9 (Notification Lifecycle State Model) and 5.10 (Reminder Delivery and Escalation Sequence). Incoming domain events (e.g., `ReminderDue`, `EscalationTriggered`, `SystemAlert`) are published to an SNS topic by upstream services. The Notification Ingestion component subscribes to the SNS topic and performs schema validation, correlation identifier assignment,

and payload normalization. Upon successful ingestion, a corresponding `NotificationEvent` record is inserted into PostgreSQL (see Figure 6.1 – Logical Data Model), establishing the initial lifecycle state.

The Rules Engine evaluates user preferences, urgency classification, quiet-hour constraints, retry policies, and escalation thresholds. Based on this evaluation, the Channel Router dispatches the event to one or more channel-specific workers (Push, Email, SMS). Each Channel Worker performs provider-specific formatting and transmits the message to external delivery systems (APNs, FCM, SES, or SMS gateway).

For each channel attempt:

- A `NotificationDeliveryAttempt` record is inserted to capture the send attempt.
- Provider confirmations update the `NotificationDeliveryAttempt.status` field (SENT, DELIVERED).
- Lifecycle transitions update the associated `NotificationEvent.current_state`.

Delivery outcomes and lifecycle state transitions are persisted to PostgreSQL through the Audit Logger component, ensuring traceability and compliance across acknowledgment, retry, and escalation workflows.

Each channel worker is associated with a dedicated SQS dead-letter queue (DLQ). If retry thresholds are exceeded or delivery fails irrecoverably, the failed message is moved to the DLQ. The DLQ reference is recorded in the corresponding `NotificationDeliveryAttempt` entry to support replay and operational investigation.

This modular architecture ensures:

- Asynchronous decoupling between publishers and delivery processors
- Independent horizontal scaling of channel workers
- Deterministic lifecycle state transitions
- Reliable auditing and delivery traceability
- Resilience through DLQ-based failure isolation

By separating ingestion, routing, channel execution, and persistence responsibilities within the ECS Fargate cluster, the Notification Service maintains clear flow of events and data from SNS publication through final state resolution in the database.

5.4.3.2 Notification Event Lifecycle State Model

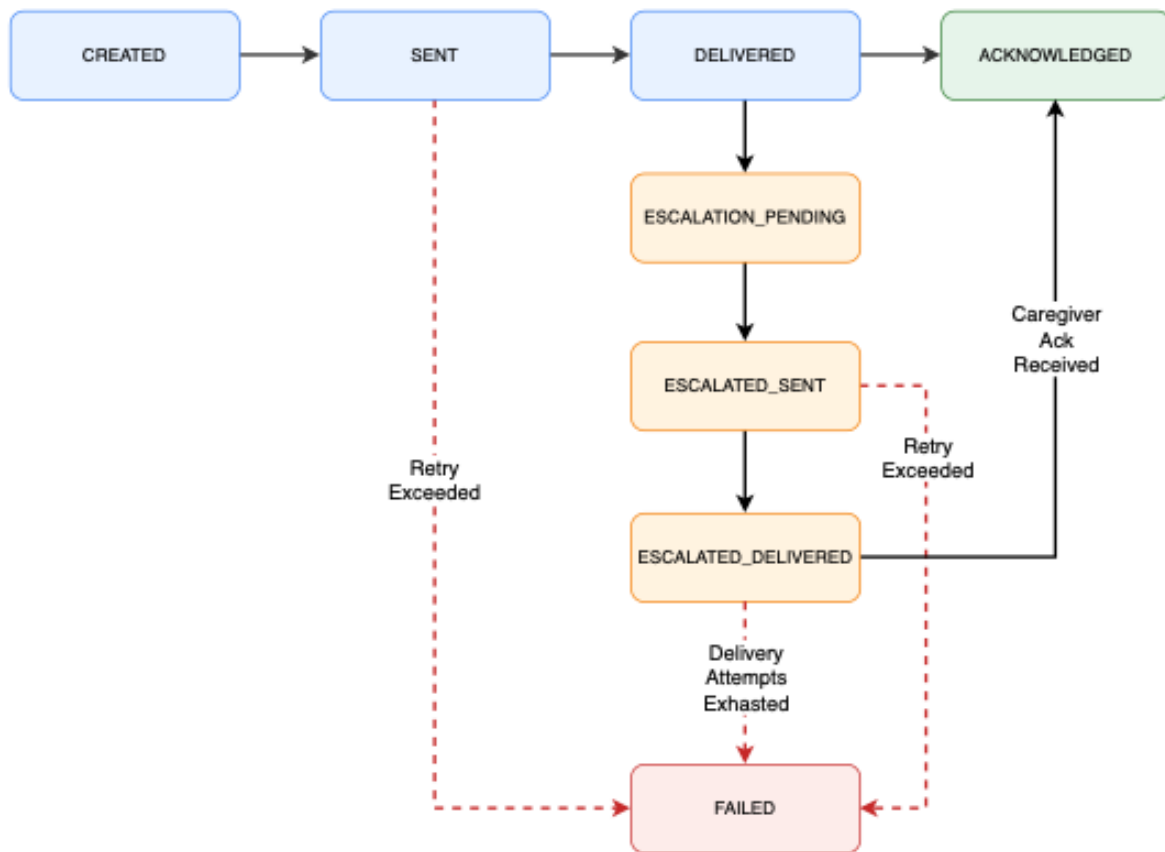


Figure 5.9 – Notification Event Lifecycle State Diagram

Each notification event progresses through a defined lifecycle to ensure reliability, traceability, and controlled escalation. The state model governs how reminders and alerts transition from creation to final resolution.

When a domain event is generated, a `NotificationEvent` record is created with state = `CREATED` and the event is published to SNS. Upon successful dispatch to a channel worker, the `NotificationEvent.current_state` transitions to `SENT`. If the delivery provider confirms transmission, the state advances to `DELIVERED`.

If the user acknowledges the notification within the configured acknowledgment window, the state transitions to `ACKNOWLEDGED`, terminating further escalation logic. If no

acknowledgment is received within the defined interval, the notification transitions to ESCALATION_PENDING, triggering escalation policies through the Rules Engine. Escalated notifications transition through ESCALATED_SENT and ESCALATED_DELIVERED, and transition to FAILED if retry thresholds are exceeded or delivery attempts are exhausted. All transitions are persisted in the database with correlation identifiers to support auditing and replay.

This explicit state model ensures deterministic behavior, prevents duplicate escalation, and supports reliable operational monitoring.

5.4.3.3 Reminder Delivery and Escalation Sequence

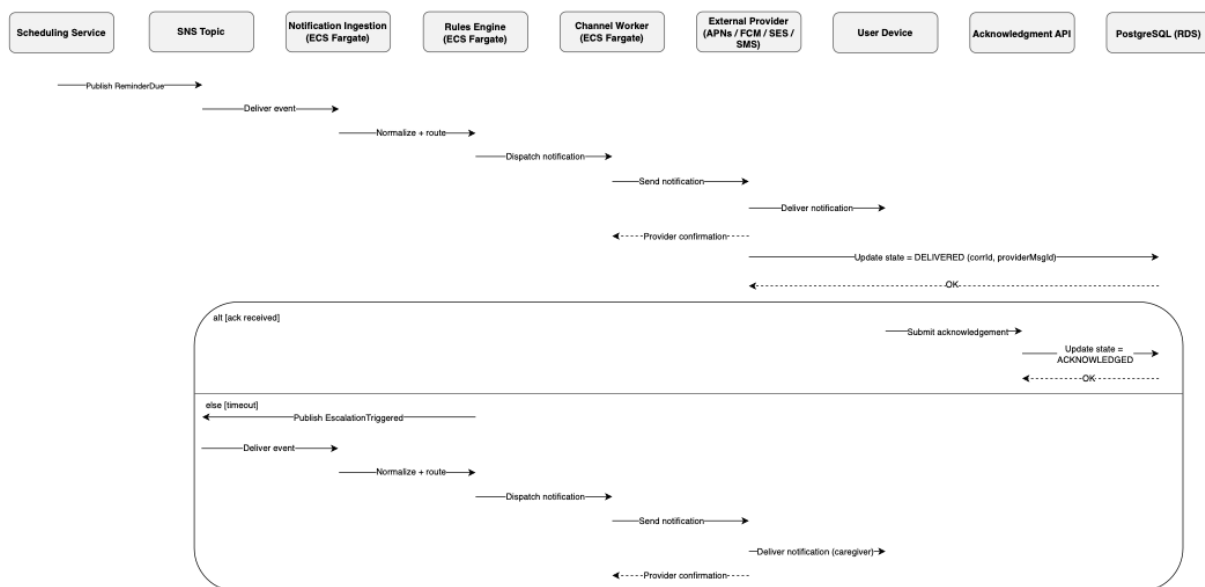


Figure 5.10 – Reminder Delivery and Escalation Sequence Diagram

This sequence diagram illustrates the end-to-end flow of a scheduled reminder event from initial trigger through delivery and potential escalation.

1. The Scheduling Service detects that a reminder has reached its trigger time and generates a domain event.
2. The event is published to an SNS topic.

3. The Notification Ingestion component (running on ECS Fargate) consumes the event, creates a `NotificationEvent` record, and forwards the normalized payload to the Rules Engine.
4. The Rules Engine evaluates user preferences, urgency classification, quiet-hour policies, and escalation thresholds.
5. The Channel Router directs the message to the appropriate worker (Push, Email, or SMS).
6. The channel worker transmits the notification to the external provider (APNs, FCM, SES, or SMS gateway).
7. Provider confirmation updates the `NotificationDeliveryAttempt` and `NotificationEvent` state in PostgreSQL through the Audit Logger component.
8. If the user acknowledges the reminder within the configured interval, the acknowledgment API updates the notification state to `ACKNOWLEDGED` and terminates the workflow.
9. If no acknowledgment is received within the defined window, the Rules Engine publishes an `EscalationTriggered` event to SNS, initiating a new delivery sequence to the designated caregiver or fallback contact.

All service interactions occur within containerized backend services running on Amazon ECS (Fargate). SNS provides asynchronous decoupling between publishing and delivery components, ensuring resilience and independent scalability. Dead-letter queues capture failed deliveries to support replay and operational recovery.

All state transitions in this sequence correspond directly to the lifecycle states defined in Figure 5.9 and the entities defined in Figure 6.1.

5.4.4 Email Delivery Integration

Describe integration with cloud-based email delivery services.

Email is handled via Amazon Simple Email Service (SES). The email worker subscribes to SNS topics (typically reminders and escalations) and renders templates (localized, branded) with SES Template API or precompiled handlebars. SES configuration includes verified identities/domains, DKIM, SPF, DMARC, dedicated or pooled IPs as needed, and sending quotas/guardrails. Delivery, bounce, and complaint notifications from SES are routed to an SNS

topic and consumed by a feedback processor that updates notification status, suppresses problematic addresses, and feeds analytics. For transactional reliability, the worker records SES message IDs and correlates them with the originating event and user.

5.4.5 Configuration and Extensibility

Explain how notification rules and channels can be configured or extended.

Notification behavior is driven by data, not code. A Preferences store (for example, DynamoDB or relational) holds per-user and per-tenant channel preferences, quiet hours, fallback contacts, and localization settings. A Rules registry defines which event types map to which channels and escalation ladders, expressed as JSON/YAML and deployed via configuration pipelines. SNS subscription filter policies implement coarse routing (by tenant, channel, urgency), while workers apply fine-grained rules (for example, suppress during quiet hours unless critical). To add a new channel, provision a new subscription and worker, extend the mapping in the Rules registry, and supply new templates; publishers remain unchanged. Feature flags allow gradual rollout per tenant or cohort.

5.4.6 Monitoring and Observability

Describe logging, metrics, and failure visibility for notifications.

End-to-end visibility includes application and infrastructure layers. Each notification attempt is persisted with `notification_id`, `user_id`, `event_type`, `topic`, `channel`, `provider_message_id`, `created_at`, `delivered_at`, `acknowledged_at`, and `status`. CloudWatch metrics and alarms cover SNS publish failures, throttling, subscription delivery failures, DLQ depth, worker service errors and latency, and SES bounces/complaints. Structured logs include correlation IDs tying domain events to downstream provider IDs and client acknowledgments. Optional traces (AWS X-Ray or OpenTelemetry) show the path from publisher through SNS to workers and providers. Dashboards present “sent → delivered → acknowledged” funnels, latency percentiles, error budgets, and per-tenant health.

5.4.7 Documentation and Reproducibility

Describe how setup, testing, and validation are documented for future cohorts.

All infrastructure is defined as code (for example, Terraform or AWS CDK) including SNS topics, policies, subscriptions, DLQs, SES identities, and monitoring. The repository includes example event payloads, message attribute conventions, and a contract document for publishers and subscribers. Runbooks cover environment bootstrapping, SES domain verification, template deployment, rotating credentials, and handling DLQ replays. Test assets include Postman or HTTPie scripts, a local event replayer, and synthetic tests that publish canary events and assert end-to-end delivery across channels. Versioned templates and rules are documented with change history so future cohorts can reproduce setup, run smoke tests, and validate delivery and acknowledgment flows consistently across environments.

6. DATA DESIGN

6.1 DATA SCOPE AND OWNERSHIP

Team D's changes introduce and modify data structures related to:

- Subscription state tracking
- Notification delivery history
- Static analysis reporting artifacts
- CI/CD traceability records
- Audit-relevant operation events

Ownership of this data resides in backend services. Client applications never directly manipulate authoritative records.

6.2 CONCEPTUAL DATA MODEL

Core entities include:

- User
- Subscription
- Payment Transaction
- Notification Event
- Notification Delivery Attempt
- Audit Record
- Quality Gate Result

Relationships:

- A user may have one active subscription
- A subscription generates multiple payment events
- A notification event may produce multiple delivery attempts
- Audit records reference system events.

6.2.1 Logical Data Model (ERD)

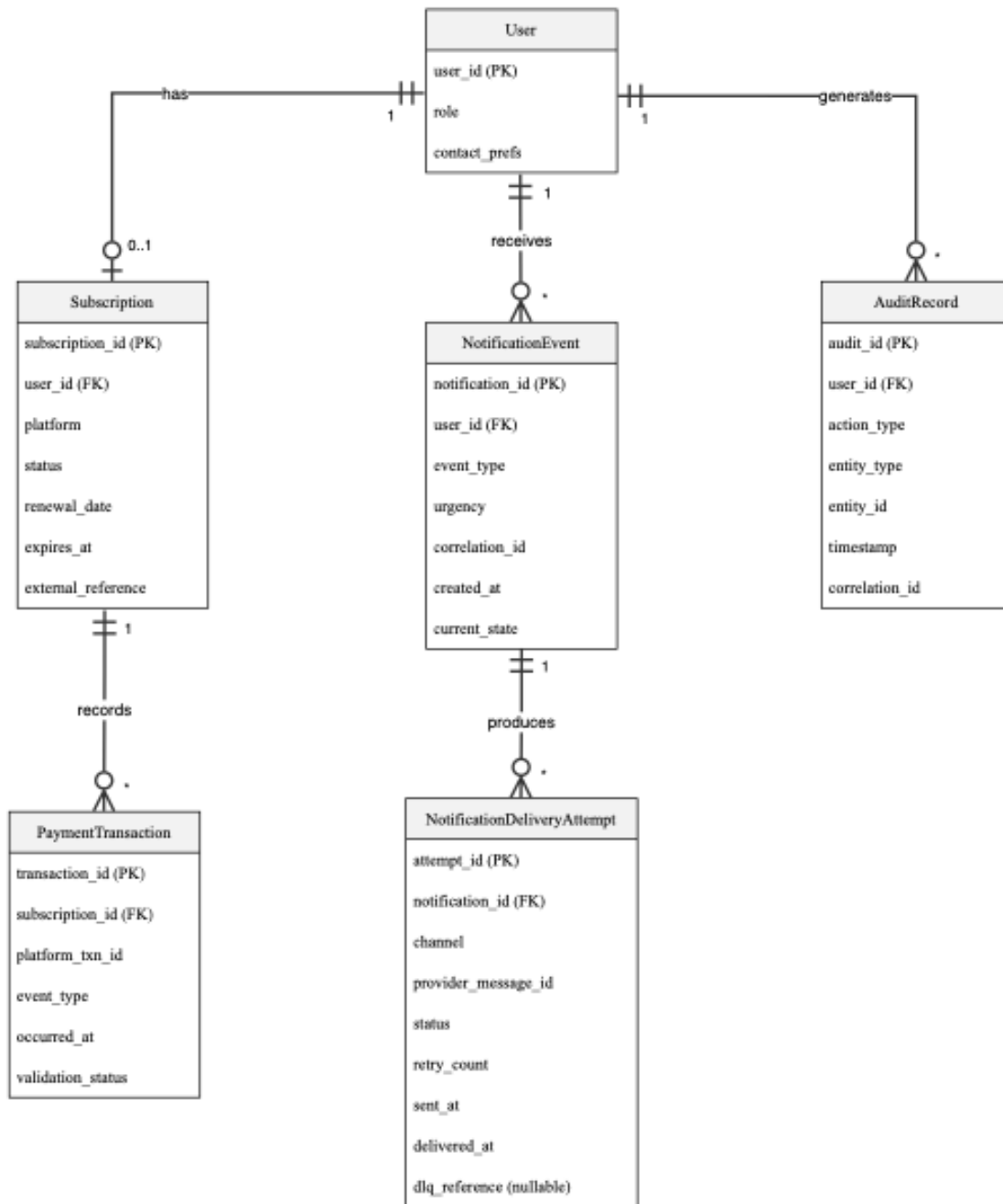


Figure 6.1 – CareConnect Logical Data Model (ERD) for Subscriptions and Notifications

This entity-relationship diagram (ERD) defines the logical data model supporting Team D's subscription lifecycle management and centralized notification architecture. The model serves as the authoritative persistence layer for the workflows illustrated in Figures 5.5 (Payment Purchase

and Validation Sequence), 5.6 (Renewal/Webhook Processing Sequence), and 5.9 - 5.10 (Notification Lifecycle and Delivery Sequences).

The User entity represents the primary system actor. A user may have zero or one active Subscription, which stores the normalized platform source (Apple/Google), external transaction reference, renewal and expiration timestamps, and the canonical lifecycle state (ACTIVE, GRACE_PERIOD, CANCELED, SUSPENDED, EXPIRED). Subscription state transitions are driven exclusively by validated purchase events and renewal webhooks processed by the Billing Service.

Each billing-related event generates an immutable `PaymentTransaction` record linked to its parent subscription. This design separates current entitlement state from historical billing events. The Subscription table reflects the current authoritative state, while `PaymentTransaction` preserves the transaction history that produced that state, enabling reconciliation and audit support.

Notifications are modeled using a two-layer structure to ensure reliability and traceability. A `NotificationEvent` represents the logical intent of a reminder, escalation, or system alert. This record is created when a scheduling or escalation event is published. Each `NotificationEvent` may produce one or more `NotificationDeliveryAttempt` records, representing channel-specific send attempts (push, email, SMS). These attempts capture provider message identifiers, delivery status, retry counts, timestamps, and optional DLQ references to support replay and operational recovery.

Lifecycle transitions such as SENT, DELIVERED, ESCALATION_PENDING, ESCALATED_SENT, ESCALATED_DELIVERED, ACKNOWLEDGED, and FAILED are persisted in the `NotificationEvent.current_state` field. Provider confirmations update the corresponding `NotificationDeliveryAttempt.status`, ensuring consistency between event-level and attempt-level tracking.

The **AuditRecord** entity provides cross-cutting traceability across both billing and notification domains. Audit records link to the originating user and reference the affected entity (subscription, transaction, or notification) through polymorphic identifiers. This structure enables end-to-end trace reconstruction across purchase verification, renewal processing, notification delivery, acknowledgment, escalation, and administrative workflows.

Flow of Events and Data Across the Model

The logical data model reflects the runtime behavior of the system:

1. Purchase Verified

- Insert `PaymentTransaction`
- Upsert `Subscription` (`state = ACTIVE`)
- Insert `AuditRecord`

2. Renewal Webhook Received

- Insert `PaymentTransaction`
- Update `Subscription` (`ACTIVE / GRACE_PERIOD / SUSPENDED / EXPIRED`)
- Insert `AuditRecord`

3. Reminder Due Published

- Insert `NotificationEvent` (`state = CREATED`)
- Insert one or more `NotificationDeliveryAttempt` records

4. Provider Confirmation

- Update `NotificationDeliveryAttempt.status`
- Update `NotificationEvent.current_state` (`SENT / DELIVERED`)

5. Acknowledgment Received

- Update `NotificationEvent.current_state = ACKNOWLEDGED`
- Insert `AuditRecord`

6. Escalation Triggered

- Insert new `NotificationEvent`
- Insert associated `NotificationDeliveryAttempt` records

This layered structure ensures:

- Clear separation between current state and historical events
- Deterministic entitlement enforcement
- Measurable notification delivery performance
- Reproducible operational troubleshooting
- Long-term auditability and compliance

By aligning persistence entities directly with lifecycle states and sequence flows, the ERD provides a consistent foundation for billing validation, notification processing, escalation logic, and auditing across the system.

6.3 LOGICAL DATA MODEL

Subscription records contain:

- Platform source
- Transaction identifier
- Status (ACTIVE, GRACE, EXPIRED)
- Renewal Timestamps
- Cancellation flags

Notification records contain:

- Event type
- Channel
- Delivery status
- Timestamps
- Acknowledgment status
- Correlation ID

Audit records include immutable timestamps and actor metadata.

6.4 DATA DICTIONARY

Field	Type	Description
subscription_id	UUID	Unique subscription record
Platform	ENUM	Apple / Google
Status	ENUM	ACTIVE / GRACE / EXPIRED
notification_id	UUID	Unique message event
Channel	ENUM	Push / email / sms
delivered_at	Timestamp	Delivery confirmation time
Acknowledged	Boolean	User response state

6.5 DATA STORAGE AND RETENTION CONSIDERATIONS

- Subscription records retained for audit duration
- Notification metadata retained for operational analytics
- Audit logs are immutable
- Sensitive healthcare data excluded from messaging payloads
- Encryption enforced at rest and in transit

Retention policies prioritize compliance, transparency, and minimal exposure to sensitive data.

7. SECURITY CONSIDERATIONS

7.1 AUTHENTICATION AND AUTHORIZATION

Authentication is handled using JWT-based identity management integrated with backend services. Role-based access control enforces separation of duties among care recipients, caregivers, and administrative privileges.

Payment and notification actions require authenticated service access and signed tokens.

7.2 DATA PRIVACY AND COMPLIANCE

CareConnect is designed with healthcare adjacent privacy safeguards:

- No PHI transmitted through SNS/SES payloads
- Minimal metadata exposure
- Backend-only entitlement decision
- Strict role isolation
- Audit logging for compliance traceability

Payment data is handled exclusively by the Apple/Google platform, reducing PCI scope.

7.3 SECURE DATA TRANSMISSION

All communication uses TLS encryption. API endpoints enforce HTTPS-only access. Cloud messaging is encrypted in transit and authenticated using IAM-scoped credentials.

Secrets are stored in AWS Secrets Manager and never committed to source control.

7.4 LOGGING, AUDITING, AND MONITORING

Security-relevant events are logged with:

- Immutable timestamps
- Actor identity
- Action metadata
- Correlation IDs

Monitoring detects:

- Abnormal login behavior
- Delivery failures
- CI/CD enforcement bypass attempts
- Suspicious billing anomalies

Logs support compliance audit and incident response.

8. PERFORMANCE AND SCALABILITY

This section defines the performance objectives, measurement criteria, scalability approach, and ongoing optimization practices for the CareConnect system. It enhances the established performance and scalability framework from prior releases by incorporating automated pipelines, notification workflows, and payment processing into the system's non-functional performance model.

8.1 PERFORMANCE OBJECTIVES

CareConnect is designed to support time-sensitive caregiving workflows while maintaining predictable system behavior under expected and peak load conditions. Performance objectives extend beyond runtime behavior to include automated delivery and operational workflows.

The system is designed to:

- **Maintain low-latency API responses** for core application operations, including authentication, data retrieval, reminder acknowledgment, and subscription validation.
- **Ensure reliable execution of CI/CD pipelines**, enabling timely feedback on build quality, security posture, and deployment readiness.
- **Deliver notifications and reminders within clinically relevant time windows**, even during periods of elevated event volume.
- **Process payment and subscription workflows predictably**, without introducing blocking behavior in core user interactions.
- **Enable early identification of performance degradation** through monitoring, logging, and pipeline feedback before users are impacted.

8.2 KEY PERFORMANCE METRICS

Performance and reliability are evaluated using observable, repeatable metrics aligned with system workflows and operational controls.

Key metrics include:

Metric	Target
API response time	< 300 ms (median), < 500 ms (95th percentile)
Pipeline execution success rate	100% completion for required stages
Notification processing success	No missed or dropped events
Resource utilization	CPU < 70%, Memory < 75%
Error rate	< 1% failed requests or transactions

These metrics provide benchmarks for performance validation, capacity planning, and operational review across application runtime, automation pipelines, and external integrations.

8.3 SCALABILITY STRATEGY

The CareConnect architecture supports controlled scalability across users, events, and delivery workflows through modular design and separation of concerns.

Scalability is achieved through:

- **Horizontal scaling of backend services**, enabled by stateless request handling and controlled service boundaries.
- **Asynchronous notification processing**, decoupling reminder generation from message delivery to prevent load amplification.
- **Scalable CI/CD workflows**, designed to accommodate growth in codebase size, test coverage, and analysis depth without degrading delivery cadence.
- **Delegation of payment processing** to platform-managed billing services, reducing internal scalability and performance risk.

This strategy allows the system to grow incrementally while preserving predictable performance characteristics.

8.4 MONITORING AND CAPACITY MANAGEMENT

Operational visibility is maintained through integrated monitoring, logging, and automation feedback mechanisms.

Monitoring and capacity management practices include:

- **Pipeline monitoring**, providing immediate visibility into build, test, and analysis outcomes.
- **Centralized application logging**, capturing execution events, errors, and workflow outcomes for diagnosis and trend analysis.
- **Cloud service observability**, using managed service metrics to observe availability, throughput, and message processing behavior.
- **Trend-based capacity review**, using historical data to anticipate growth and adjust capacity before performance thresholds are exceeded.

These practices support proactive performance management rather than reactive remediation.

8.5 BOTTLENECK IDENTIFICATION AND MITIGATION

Potential performance bottlenecks are identified through monitoring data, pipeline feedback, and operational review.

Key bottlenecks and mitigation strategies include:

Bottleneck	Mitigation Strategy
High CPU or memory utilization	Scale service instances, optimize code paths
Pipeline execution delays	Fail fast, parallelize analysis stages where appropriate
Notification volume spikes	Use asynchronous messaging and independent scaling
Slow database queries	Apply indexing, query optimization, or read isolation
Payment processing latency	Minimize synchronous dependencies and rely on platform-managed billing

8.6 REVIEW AND OPTIMIZATION CADENCE

Performance and scalability are reviewed continuously as part of system delivery and operational governance.

Review activities include:

- **Pre-merge and pre-deployment reviews** of pipeline execution results
- **Regular analysis of notification and payment workflow behavior**
- **Post-incident reviews**, documenting root causes and corrective actions
- **Periodic updates to performance thresholds and mitigation strategies** based on observed system behavior

This cadence ensures that performance and scalability considerations remain aligned with system growth and operational demands.

9. TESTING STRATEGY

9.1 SCOPE AND OBJECTIVES

This strategy validates Team D's design end to end, confirming that functional and nonfunctional requirements are met and that outcomes trace to Business Need 4 (Centralized Notifications and Reminders) and adjacent needs (security, reliability, usability). It aligns with the program Software Test Plan (STP) by defining test levels, environments, data management, automation, and reporting. Success is measured by requirement coverage, defect escape rate, delivery SLAs, and user acceptance.

9.2 REQUIREMENTS TRACEABILITY

A bidirectional traceability matrix maps each requirement (Must/Should/Could) to one or more test cases, automated checks, and acceptance criteria. Each test case references: requirement ID, business need, component(s) under test, expected behavior, data preconditions, and pass/fail thresholds. Daily builds publish coverage (requirements and code), with gaps flagged as exit blockers. Any change request updates the matrix and associated tests before merge.

9.3 TEST LEVELS AND TYPES

- **Unit tests:** Validate domain logic, rules evaluation, and message attribute shaping (e.g., urgency, channel filters). Target $\geq 80\%$ critical-path coverage.
- **Contract and component tests:** Verify API schemas, SNS/SES/SQS integrations and notification worker services, idempotency, and malformed input handling using mock providers or local emulators where feasible.
- **Integration tests:** Exercise end-to-end notification flows from event emitters through SNS to channel workers (push/SMS/email) using sandbox accounts and ephemeral topics/identities.
- **System and end-to-end (E2E) tests:** Validate user journeys (create care task → reminder sent → acknowledge → no escalation) and negative paths (quiet hours, opt-outs, DLQ replay).

- **Nonfunctional tests:** Performance (publish throughput, worker latency, p95/p99), resilience (DLQ, retries, partial outages), security (authZ, least-privilege IAM, transport encryption), accessibility (WCAG for notification settings UI), and compliance (logging without PHI exposure).

9.4 TEST ENVIRONMENTS AND DATA

Dedicated dev, test, and staging environments mirror production topology for SNS topics, SES identities, DLQs, and metrics. Test data uses synthetic users and seeded scenarios (routine reminder, critical escalation, bounce/complaint cases). Secrets and keys are rotated and scoped per environment. Observability is enabled in every environment to collect comparable metrics and traces.

9.5 AUTOMATION AND CI/CD

A continuous test pipeline runs on every pull request: static analysis, unit, contract, and component tests, followed by ephemeral integration tests that spin up temporary resources (topics, queues, identities) and tear them down on completion. Nightly scheduled suites run heavier E2E and nonfunctional tests, including canary flows that publish sample events and assert channel delivery. Results, coverage, and traces are pushed to dashboards and used to gate releases.

9.6 TEST TOOLING AND ORACLES

- **Contract testing:** OpenAPI/JSON Schema with negative and backward-compatibility checks.
- **Messaging:** Publishers and subscribers include test harnesses to validate message attributes, filter policies, deduplication, and idempotency.
- **Email:** Template snapshot tests (localized), SES sandbox for deliverability, bounce/complaint simulation.
- **Mobile push:** Payload validation against APNs/FCM, device-token lifecycle tests, collapse keys, TTL behavior.
- **Oracles:** Business rules specify observable outcomes (e.g., “no escalation if acknowledged within grace window”), encoded as assertions in E2E tests.

9.7 ENTRY AND EXIT CRITERIA

- **Entry:** Requirements baselined in the traceability matrix; environments available; test data ready; observability configured.
- **Exit:** All Must/Should requirements covered with passing tests; no open Sev-1/Sev-2 defects; nonfunctional SLOs met (throughput, latency, error budgets); regression and security suites green; stakeholder sign-off from Product and QA leads.

9.8 DEFECT MANAGEMENT AND REPORTING

Defects are triaged by severity and mapped to affected requirements. Root-cause analysis is required for Sev-1/Sev-2 with remediation tasks added to the backlog. Dashboards report trend lines for pass rates, flakiness, MTTR, and defect escape. Weekly quality reviews ensure alignment with the STP and visibility for stakeholders.

9.9 ONGOING VALIDATION IN PRODUCTION

Synthetic monitors continually publish probe events to validate end-to-end delivery, with alerts on drift (latency, failure rate, DLQ depth). Sampling correlates “sent → delivered → acknowledged” to verify real-user outcomes. Releases use feature flags and progressive rollouts with automated rollback on SLO breach, ensuring continuous alignment to business needs.

10. DEPLOYMENT STRATEGY

The CareConnect deployment strategy is designed to ensure that only secure, validated, and policy-compliant software artifacts are promoted into shared and production-like environments. Given the healthcare-adjacent nature of the platform and the requirement to enforce strict quality and security gates, deployment automation is tightly integrated with continuous integration workflows, static analysis tooling, and vulnerability scanning processes. This strategy prioritizes reliability, traceability, and rollback safety while supporting rapid iteration during the milestone-driven development lifecycle.

Deployment workflows are fully automated and event-driven, minimizing manual intervention and reducing the risk of configuration drift between environments. All infrastructure and application deployments are performed using reproducible pipelines that enforce quality thresholds before any runtime changes are applied.

10.1 DEPLOYMENT PREPARATION

Prior to deployment, all code changes must satisfy a defined set of pre-deployment requirements that collectively ensure functional correctness, security posture, and operational readiness. These checks are executed automatically as part of the CI pipeline and must complete successfully before deployment steps are initiated.

Deployment preparation includes the execution of unit and integration tests for backend services and mobile clients, static code analysis to detect bugs and insecure coding patterns, and dependency vulnerability scans to identify known third-party risks. Static analysis tooling such as SpotBugs, PMD, Checkstyle, and SonarQube is configured to block promotion when critical or high-severity findings are detected, directly supporting Business Needs BD1 and BD2. For mobile and frontend components, Flutter and Dart analysis tools are executed locally and in CI to ensure formatting consistency, correctness, and maintainability.

Infrastructure readiness is ensured through infrastructure-as-code validation. Cloud resources, including ECS services, networking configurations, IAM roles, and managed AWS services, are defined declaratively and validated prior to deployment to guarantee repeatability across

environments. Secrets, credentials, and environment variables are provisioned using AWS Secrets Manager and injected at runtime through IAM-scoped permissions, ensuring that sensitive data is never embedded in source code or build artifacts.

10.2 CI/CD DEPLOYMENT FLOWS

CareConnect uses automated CI/CD pipelines to orchestrate build, test, analysis, and deployment activities across backend, frontend, and mobile platforms. Pipelines are triggered by repository events such as code pushes or pull request merges and are designed to enforce quality gates before any deployment occurs.

10.2.1 Backend Deployment Flow

Backend deployments are triggered by changes to backend service modules within the CareConnect repository. GitHub Actions orchestrates the pipeline, executing a series of stages that include code checkout, compilation, automated testing, static analysis, and vulnerability scanning. Only when all quality gates pass is a deployable artifact produced.

Container images are built from validated source code and pushed to a secure container registry. Deployment to AWS ECS Fargate is performed using controlled rollout strategies, allowing new versions of services to be gradually introduced while health checks monitor system stability. If health checks fail or runtime errors exceed defined thresholds, automated rollback procedures restore the previous stable version, minimizing service disruption.

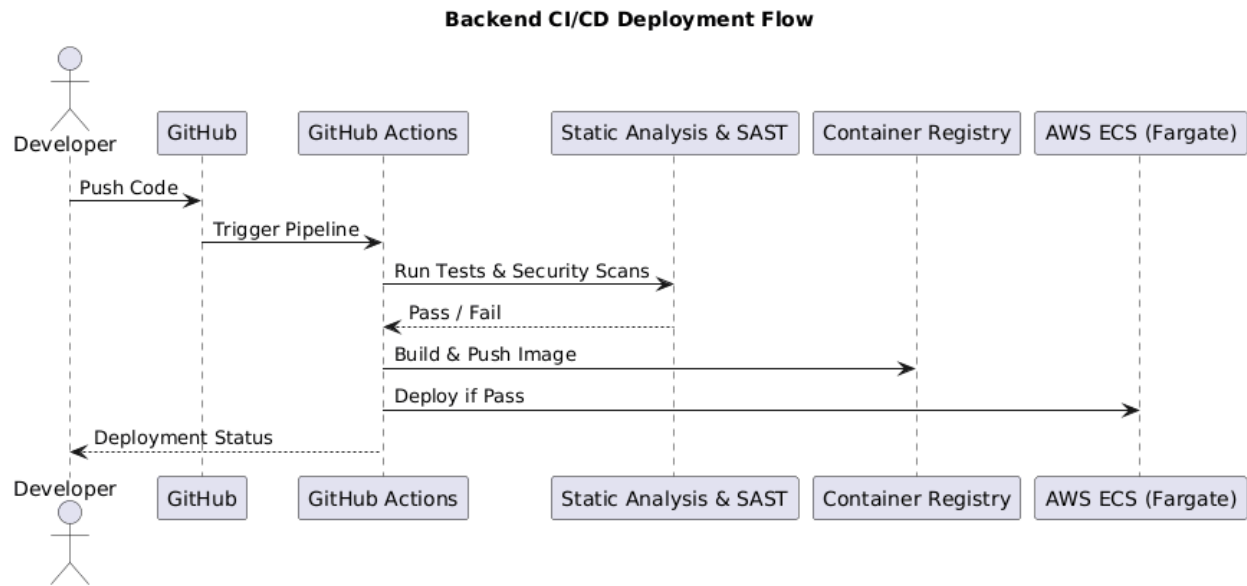


Figure 10.1 - Backend CI/CD Deployment Flow

This diagram shows the backend CI/CD deployment flow, where code changes pushed to GitHub trigger automated pipelines in GitHub Actions to run tests and security scans, build and publish container images, and deploy approved artifacts to AWS ECS (Fargate) when all quality and security gates pass.

10.2.2 Frontend Deployment Flow

Frontend deployment workflows differ slightly depending on the target platform but follow the same principle of automated validation and controlled release. For the web frontend, changes to the frontend codebase trigger a GitHub Actions workflow that builds the Flutter web application and deploys it to managed hosting services. Infrastructure provisioning and updates are handled through infrastructure-as-code to ensure hosting environments remain consistent.

Mobile application deployments for iOS and Android are handled through a mobile-focused CI pipeline that builds platform-specific artifacts, executes automated tests, and prepares release bundles for distribution through the Apple App Store and Google Play Store. Native platform review processes are respected, and no runtime code is distributed outside approved store channels.

10.3 RUNTIME DEPLOYMENT SAFETY AND ENFORCEMENT

Deployment safety is enforced through multiple layers of protection embedded directly into the CI/CD pipeline. Quality gates ensure that builds cannot progress when unit tests fail, static analysis reports critical findings, or dependency scans identify unacceptable vulnerabilities. These controls prevent unstable or insecure code from reaching runtime environments, directly addressing Business Need BD2. Monitoring and observability tools continuously track system health following deployment. Structured logs, metrics, and alerts enable rapid detection of anomalies, while deployment automation ensures that corrective action, such as rollback or service restart, can be executed quickly without manual intervention. This approach ensures that CareConnect remains resilient even as new features and architectural changes are introduced.

10.4 RUNTIME USE CASE DEPLOYMENT SEQUENCES

Deployment workflows also support runtime operational use cases that depend on reliable and consistent system behavior.

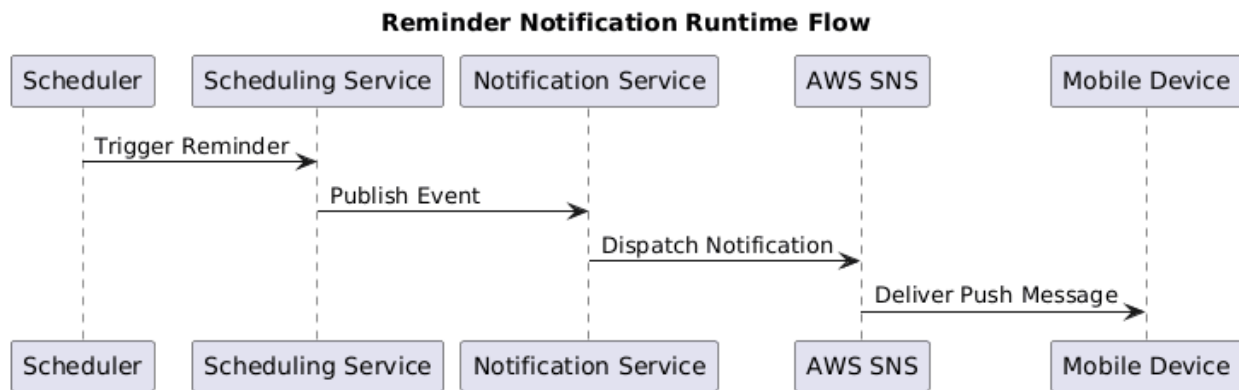


Figure 10.2 - Reminder Notification Runtime Flow

This figure illustrates the runtime sequence for delivering reminder notifications within CareConnect. A scheduled event triggers the Scheduling Service, which publishes a reminder event to the Notification Service. The Notification Service dispatches the message through AWS SNS, which delivers the push notification to the user's mobile device.

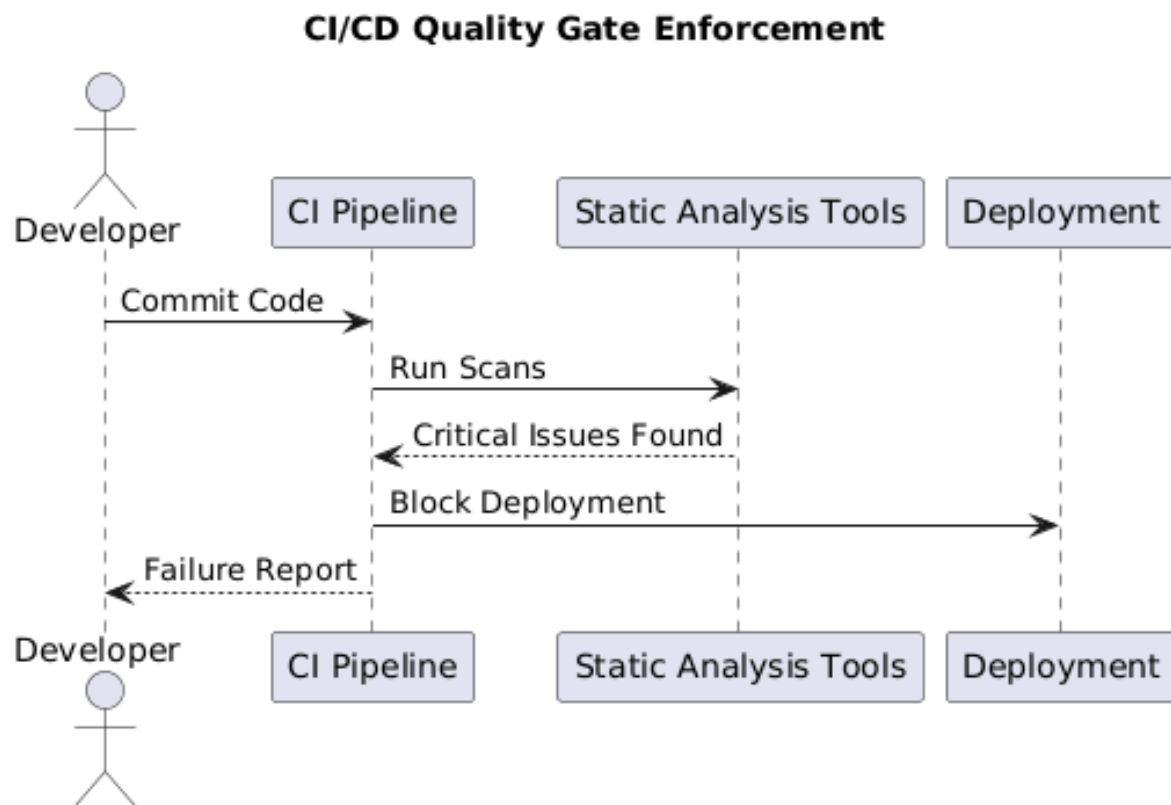


Figure 10.3 - CI/CD Quality Gate Enforcement

This figure illustrates how automated quality gates are enforced within the CI/CD pipeline. After a developer commits code, the pipeline executes static analysis scans. If critical issues are detected, the quality gate blocks deployment and returns a failure report to the developer, preventing non-compliant code from advancing.

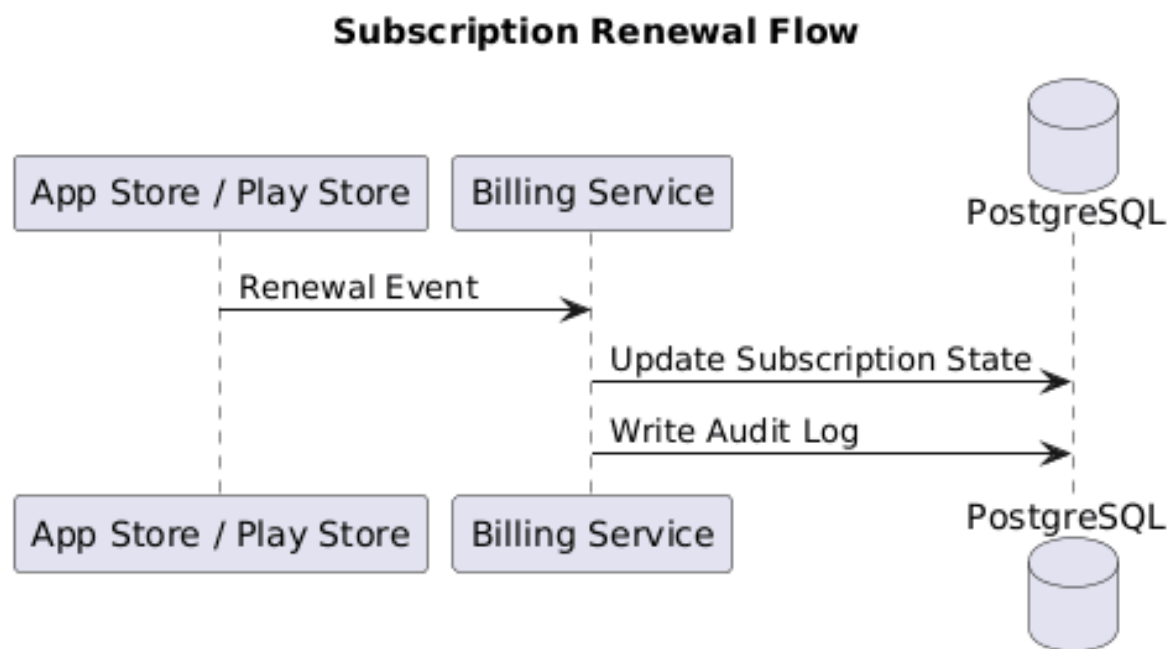


Figure 10.4 - Subscription Renewal Flow

This figure shows how subscription renewals are processed using native mobile platform billing systems. Renewal events from the Apple App Store or Google Play are received by the backend billing service, which updates the subscription state and records an audit entry in the PostgreSQL database to maintain consistency and traceability.

This deployment strategy ensures that CareConnect maintains a high standard of quality, security, and operational stability while supporting rapid iteration and compliance with healthcare-adjacent requirements. By integrating CI/CD enforcement, static analysis, native mobile deployment workflows, and automated rollback mechanisms, the system is well-positioned to scale across future milestones and cohorts without compromising safety or maintainability.

11. KNOWN ISSUES AND LIMITATIONS

Document known issues, technical constraints, deferred work, and limitations relevant to Team D's scope.

11.1 CURRENT GAPS AND DEFECTS

- Occasional duplicate notifications can occur during worker retries if upstream publishers reuse correlation IDs inconsistently; de-duplication is partial and scoped to a short time window.
- Quiet-hours suppression has edge cases across time zones and daylight saving transitions; rare off-by-one-hour deliveries have been observed.
- Device token lifecycle hygiene is incomplete; invalid APNs/FCM tokens are removed asynchronously, which can inflate “sent” counts vs. “delivered.”
- Escalation timing relies on scheduler clocks; minor clock skew between services can delay or prematurely trigger escalations.
- SES sandbox in non-production constrains external address testing, limiting realism of end-to-end email tests.
- SMS delivery receipts vary by carrier; delivered vs. accepted is sometimes indistinguishable, reducing accuracy of channel-level analytics.

11.2 TECHNICAL CONSTRAINTS

- Push notifications (APNs/FCM) and SMS are best-effort networks without strict delivery guarantees or SLAs; device OS background restrictions may defer or drop non-critical pushes.
- AWS SNS, SES, and SMS have regional availability, throughput quotas, and rate limits that may throttle spikes without pre-warming or limit increases.
- SMS coverage and short code/alpha-sender support vary by country; compliance requirements (opt-in, STOP keywords) restrict messaging content and frequency.
- For compliance, payloads must avoid PHI; this limits message personalization and requires additional app-side fetching via deep links.
- Email deliverability depends on domain reputation, DKIM/SPF/DMARC alignment, and IP warming; changes to templates or sending patterns can temporarily impact inbox placement.

- Multi-tenant isolation is logical (per-topic filters and IAM), not physical; noisy tenants can affect shared throughput unless per-tenant throttles are configured.

11.3 DEFERRED WORK

- In-app inbox and read-state synchronization for all channels (single source of truth for “seen” and “acknowledged”).
- Full multilingual template coverage and locale fallbacks for email/SMS/push, including right-to-left layouts.
- Self-service tenant rules editor with safe validation and preview; currently requires config changes via code or ops.
- Advanced analytics dashboards (funnel: sent → delivered → acknowledged, cohort retention, per-tenant error budgets).
- Per-tenant rate limiting and fair-share throttling across channels to isolate traffic spikes. Additional channels (voice call, WhatsApp/RCS) pending legal, compliance, and provider selection.
- Automated DLQ replay tooling with guardrails; replays are currently manual and operator-driven.

11.4 OPERATIONAL LIMITATIONS

- Observability is uneven across providers; unified correlation from domain event to provider receipt is not guaranteed for all carriers and locales.
- Incident playbooks exist but are light on rollback procedures for template regressions; restoring prior versions may require manual intervention.
- Cross-region disaster recovery for messaging is not yet in place; topics and SES identities are single-region per environment.
- Synthetic canary coverage focuses on push and email; SMS canaries are limited due to cost and carrier anti-spam heuristics.
- Redaction of sensitive fields in logs is enforced at workers, but legacy publishers may emit verbose payloads if misconfigured.

11.5 WORKAROUNDS AND MITIGATIONS

- Idempotency keys and short-lived de-duplication caches reduce, but do not eliminate, duplicate sends during retries.
- Feature flags and progressive rollouts mitigate blast radius for new templates and rule changes.
- Backoff and jitter strategies on publish/subscribe operations smooth out quota bursts; alarms are configured for throttling and DLQ depth.
- Deep links carry opaque identifiers only; the app fetches details securely to mitigate PHI exposure over email/SMS.
- Time synchronization (NTP) and scheduler drift alarms reduce escalation timing errors; grace windows include buffers for skew.

11.6 ASSUMPTIONS AND EXTERNAL DEPENDENCIES

- Upstream services emit well-formed events with stable schemas and provide timely acknowledgments for user actions.
- Identity and access management is available to validate user/channel preferences at send time.
- Mobile apps support deep links and register for push reliably; users have granted notification permissions.
- Verified SES domains and SMS sender identities remain in good standing; reputation management and complaint handling are maintained.
- Accurate tenant configuration (time zones, quiet hours, escalation contacts) is curated by tenant admins; data quality is not programmatically enforced yet.