**Programmer’s Guide Document**

CareConnect

University of Maryland Global Campus

SWEN 670 - Software Engineering Capstone

Dr. Mir Assadullah

August 3rd, 2025

Contributors: Diane Angeles, Torie Bias, Juan Gaucin Calderon, Luke Curran, Ashenafi Grbreegziabhere, Alyssa Marielle Harding, Astha Malla-Paudel, Maria Ramirez, Edwenson Raphael, Dat Truong, Alex Vecchioni, Christian Yawn

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 07/26/2025 | 1.0 | Initial Release | CareConnect Team |
| 08/03/2025 | 2.0 | Adding new challenge faced | CareConnect Team |
|  |  |  |  |

**Sign-off Sheet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Role** | **Name** | **Signature** | **Date** |
| Project Manager | Alireza Minagar |  |  |
| Team Lead | Alyssa Harding |  |  |
| Lead Tester | Juan Gaucin |  |  |
| Client | Dr. Mir Assadullah |  |  |

Table of Contents

[1. Introduction 5](#_Toc205126516)

[1.1 Purpose 5](#_Toc205126517)

[1.2 Intended Audience 5](#_Toc205126518)

[1.3 Document Organization 5](#_Toc205126519)

[1.4 Project Documents 5](#_Toc205126520)

[1.5 Definitions, Abbreviations, and Acronyms 6](#_Toc205126521)

[1.6 References 6](#_Toc205126522)

[2. Technical Summary 7](#_Toc205126523)

[2.1 Business Case 7](#_Toc205126524)

[2.2 Architecture 7](#_Toc205126525)

[2.3 Typical Use Scenarios 8](#_Toc205126526)

[3. Rationales 8](#_Toc205126527)

[3.1 Development Environment 8](#_Toc205126528)

[3.1.1 Source Control 9](#_Toc205126529)

[3.2 Frameworks & Languages 9](#_Toc205126530)

[4. Project Structure 11](#_Toc205126531)

[4.1 Overall Project Structure 11](#_Toc205126532)

[4.4 Terraform Infrastructure Structure 15](#_Toc205126533)

[5. Code Structure 16](#_Toc205126534)

[5.1 Overall CodeStructure 16](#_Toc205126535)

[5.2 Front-end Code Structure 16](#_Toc205126536)

[5.3 Backend Code Structure 16](#_Toc205126537)

[5.4 Coding Conventions 16](#_Toc205126538)

[6. Known Bugs & Workarounds 17](#_Toc205126539)

[6.1 UI Inconsistencies on Older Android Devices 17](#_Toc205126540)

[6.2 Push Notifications Delayed on iOS 17](#_Toc205126541)

[6.3 Wearable Integration 18](#_Toc205126542)

[6.3.1 Fitbit Integration on Web and iOS 18](#_Toc205126543)

[6.3.2 Google Health and Apple Health Kit 18](#_Toc205126544)

[6.4 Home Monitoring 19](#_Toc205126545)

[6.5 Smart Devices 19](#_Toc205126546)

[6.6 Speech to Text Package 19](#_Toc205126547)

[6.7 Automated Amplify Deployments 20](#_Toc205126548)

[6.8 Implementation of Long-Lived connections 20](#_Toc205126549)

# 1. Introduction

## 1.1 Purpose

This Programmer Guide has been created to enable a thorough knowledge of developing, maintaining, and troubleshooting the CareConnect system in the minds of the developers. This guide will discuss all the technical factors, such as development setup, coding standards, system integrations, and tools, in the development cycle. This document will help developers understand the code structure and organization and ensure that the developer works according to the overall system architecture.

## 1.2 Intended Audience

This document addresses developers, engineers, and system administrators working on the project CareConnect. It also acts as a great form of reference for new developers who find themselves entering the project, as well as helping these new developers get acquainted within the tools and processes implemented by the project in developing CareConnect. This guide is essential for anyone who must write, revise, or maintain the source code.

## 1.3 Document Organization

This document is divided into multiple sections, each focusing on a key area of the CareConnect system:

* **Technical Summary**: An overview of the CareConnect architecture and its components.
* **Rationales**: Explains the reasons behind key technical decisions made during the development.
* **Project Structure**: Describes the organization of the CareConnect codebase.
* **Code Structure**: Details the structure of the source code, including coding conventions and best practices.
* **Known Bugs**: Lists known issues in the system that are yet to be resolved.

## 1.4 Project Documents

This guide is one of several key documents in the CareConnect project. These documents are integral to understanding the overall system and ensuring that the development aligns with the project’s goals. The table below highlights the documents within the entire document package of CareConnect.

**Table 1**

*CareConnect Documentation Table*

|  |  |  |
| --- | --- | --- |
| **Document** | **Version** | **Date** |
| Project Plan | 1.0 | 05/31/2025 |
| Software Requirements Specification | 1.0 | 05/31/2025 |
| Technical Design Document | 1.0 | 06/14/2025 |
| Software Test Plan | 1.0 | 06/14/2025 |
| Programmer’s Guide | 1.0 | 07/26/2025 |
| Deployment & Operations Guide | 1.0 | 07/26/2025 |
| User Guide | 1.0 | TBD |
| Test Report | 1.0 | TBD |

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

## 1.5 Definitions, Abbreviations, and Acronyms

* **API**: Application Programming Interface, used to connect different software components.
* **AWS**: Amazon Web Services.
* **Flutter**: An open-source UI toolkit used for building cross-platform applications.
* **SDK**: Software Development Kit, a collection of tools for developing software applications.
* **CI/CD**: Continuous Integration/Continuous Deployment, a set of practices for automating software development and deployment.
* **EHR**: Electronic Health Rrecords.
* **EULA**: End User Licensing Agreement.
* **RDS**: Relational Database Service, used to store relational data in AWS.

## 1.6 References

*AWS Documentation Overview*. (n.d.-b). Amazon Web Services, Inc. <https://aws.amazon.com/documentation/>

*Flutter documentation*. (n.d.-b). <https://flutter.dev/docs>

*Postman: The world’s leading API platform | Sign up for free*. (n.d.). Postman API Platform. <https://www.postman.com/>

# 2. Technical Summary

CareConnect is a cross-platform application developed with Flutter as a front-end on the mobile side, and Spring Boot microservices as the back end, running on AWS. This system is intended to enable the care of patients through a task management tool, a messaging system, and health tracking functions available to caregivers. Its backend is on AWS RDS to store user information and Amazon S3 to store media. The front-end to the back-end communication (the two-way traffic) is done through REST APIs, and the notifications are delivered using Firebase Cloud Messaging (FCM). The system can be scaled and expanded by adding newer functions, such as incorporating wearable devices.

## 2.1 Business Case

The CareConnect system was designed to provide a solution to one of the most serious gaps in the healthcare industry, the coordination of patient care among caregivers, patients, and healthcare providers. Modern systems tend to be isolated and sharing patient health information with the team as well as assigning and monitoring tasks and their outcomes in real-time can be problematic. CareConnect is an all-in-one, scalable platform that simplifies communication tasks management, and health tracking. As a cross-platform mobile-enabled access with a cloud-native backend infrastructure, CareConnect is not only a valuable tool to existential healthcare institutions, but it also enables an increased level of efficiency in operations, caregiver communication, and eventually, patient outcomes. Due to the increased opportunities through scalability to small clinics and large hospitals, as well as the capability to integrate with EHR and wearable health devices in the future, this system has the potential to grow with the trend towards digital health solutions worldwide.

## 2.2 Architecture

CareConnect is designed using a microservices architecture that ensures modularity, scalability, and maintainability.

* **Front-end:** Built using Flutter for cross-platform mobile application delivery (iOS and Android), providing a consistent user experience.
* **Backend:** Composed of Spring Boot microservices deployed on AWS, each responsible for a specific domain: user management, task scheduling, notifications, etc.
* **Data Layer:** Relational data is managed through Amazon RDS, while multimedia content is stored on Amazon S3.
* **Communication:** RESTful APIs enable secure and efficient communication between the front-end and back-end services.
* **Notifications:** Firebase Cloud Messaging (FCM) is used to deliver real-time notifications across devices.
* **Infrastructure:** AWS Lambda functions are integrated for serverless processing tasks, and Terraform scripts are used for infrastructure-as-code (IaC) management.

This architecture allows independent scaling, continuous deployment, and efficient error isolation, supporting both current operations and future enhancements.

## 2.3 Typical Use Scenarios

CareConnect addresses multiple real-world healthcare workflows, including:

* **Caregiver-Patient Task Management:** Caregivers are also able to delegate duties like medication, wound dressing, or daily assessment to patients or other caregivers that can be tracked in real-time and monitor completion and adherence.
* **Secure Messaging:** Enables secure, HIPAA-compliant communication between healthcare providers and caregivers for coordination of care plans, urgent alerts, or follow-ups.
* **Health Data Monitoring:** Patients will be able to monitor their vital signs or wellness indicators, which will be gathered and displayed to caregivers to evaluate their progress or take action where needed.
* **Notifications & Alerts:** Users are reminded and alerted in a timely manner (e.g., medication times, appointment reminders). This minimizes errors and enhances adherence to care plans.
* **Administrative Oversight:** Healthcare administrators can monitor system usage, task performance, and caregiver efficiency through dashboards and reports.

These scenarios illustrate the practical, patient-centered benefits that CareConnect offers to healthcare systems.

# 3. Rationales

## 3.1 Development Environment

The development environment for CareConnect is designed to support seamless collaboration, efficient development workflows, and reliable builds across multiple platforms. It includes all necessary tools, services, and configurations to enable developers to contribute to both front-end and back-end components of the system.

### 3.1.1 Source Control

* **Git:** Git is a version control system used to track changes in code and collaborate with other developers. It allows multiple team members to work on the CareConnect project at the same time, manage updates, and safely experiment without affecting the main codebase. Git also keeps a history of all changes, making it easy to review, undo mistakes, or go back to previous versions of the project.
* **GitHub:** GitHub was chosen for CareConnect because it integrates seamlessly with Git, allowing the team to manage code changes, track issues, and review each other’s work in one place. GitHub also supports branching, pull requests, and continuous integration tools, which help maintain a clean codebase and streamline the development process across the entire team.
* **MS Teams-** Microsoft Teams was chosen for CareConnect because it provides a centralized platform for communication and collaboration. It allows team members to chat, hold video meetings, share files, and manage tasks all in one place.

3.1.2 Integrated Development Environments

* **Visual Studio Code**: Visual Studio Code was chosen for CareConnect as a development environment because it is a lightweight, fast, and highly customizable code editor. It supports a wide range of programming languages.
* **Android Studio**: Android Studio was chosen for CareConnect primarily for developing and testing the mobile app on Android devices. It is the official IDE for Android development, offering built-in support for Flutter and Dart, as well as powerful tools like an emulator, layout editor, and debugger.
* **IntelliJ**: IntelliJ IDEA was chosen for CareConnect because it is a powerful, full-featured IDE that works especially well for Java and Spring Boot development. IntelliJ helps backend developers work more efficiently and with few errors, making it ideal for building and maintaining the complex service that powers the CareConnect application.

### 3.2 Frameworks & Languages

* **Flutter & Dart**: Flutter was chosen as the front-end framework for CareConnect because of its cross-platform capabilities, allowing a single codebase to run on both iOS and Android devices. This supports faster development and consistent user experiences. Paired with Flutter, Dart was selected as the programming language since it is optimized for UI development and is the native language of Flutter. Dart’s features like hot reload, strong typing, and the ability to compile to both native code and JavaScript make it ideal for building fast, flexible, and interactive healthcare interfaces such as real-time notifications and patient training.
* **Java & Spring Boot**: Java was selected as the backend language for CareConnect due to its reliability, security, and wide adoption in building large, stable systems. It integrates seamlessly with Spring Boot, the chosen backend framework, which simplifies the development of secure, scalable, and production-ready APIs. Together, Java and Spring Boot enable the CareConnect team to develop and maintain well-structured backend services efficiently, supported by a rich ecosystem of tools and strong community support.

3.3 Infrastructure

* **Operating Systems**: For desktop development and testing, CareConnect supports major desktop operating systems, including Windows, macOS, and Linux. Developers can build and run the application on Windows 10 or later, macOS 10.15 (Catalina) or later, and most modern Linux distributions such as Ubuntu or Fedora. These operating systems provide the necessary tools and runtime environments to support Flutter's desktop framework, enabling programmers to test UI components, manage backend integrations, and simulate real-world usage scenarios during development. As a result, CareConnect is fully compatible with both iOS and Android, allowing it to function seamlessly across a wide range of smartphones and tablets.
* **Amazon Web Service**: The scalability and security of cloud infrastructure required AWS to be selected. By storing relational data using Amazon RDS, we can take advantage of managed databases, automated backups, and scale. AWS Lambda allows serverless execution, such as data processing, that enhances efficiency in processing and notifications. For more details, please view the “CareConnect Technical Design Document”.
* **Fit Bit & Apple**: Fitbit and Apple (HealthKit/Apple Watch) were considered for integration with CareConnect to enhance health data tracking and provide real-time insights into a patient’s physical activity, sleep, and vital signs. These platforms allow the app to collect important health metrics such as heart rate, steps, and sleep patterns, which can be used to monitor patient well-being and trigger personalized recommendations or alerts. Integrating with Fitbit and Apple devices helps CareConnect support a more comprehensive, connected, and proactive approach to patient care.

3.4 UX Design

* **Figma**: Figma was chosen for CareConnect as the primary UI/UX design tool because it allows for collaborative, cloud-based interface design and prototyping. Designers and developers can work together in real time, making it easier to create and iterate on the app’s layout, visuals, and user flows. Figma also allows for seamless exporting of design assets (such as icons and images) directly into Flutter, speeding up the development process and ensuring consistency between the design and the final product. For these reasons, Figma was chosen for CareConnect.

3.5 Database

* **MySQL**: MySQL was chosen for CareConnect because it is reliable, widely supported and easy to integrate with Java and Spring Boot. It handles structured data efficiently, which is important for storing patient information, appointments, and logs in an organized way. Additionally, MySQL is open-source and well-documented, making it a cost-effective and developer-friendly choice for managing healthcare data.

3.6 Licensing

* **End User License Agreement**: EULA was chosen for CareConnect to protect the rights of the developers and to clearly define the responsibilities of users when using the app. Since CareConnect deals with sensitive healthcare data, it's important to establish rules for proper use, prevent unauthorized actions (like tampering or redistribution), and limit legal liability. The EULA also ensures users understand how their data will be handled, helping the app stay compliant with privacy regulations such as HIPAA. Overall, it provides a legal foundation that supports security, accountability, and trust between the development team and the end users.

# 4. Project Structure

## 4.1 Overall Project Structure

The CareConnect project is organized into a modular structure, allowing developers to work on separate components without affecting other areas of the system.

* **/backend**: Contains the Spring Boot microservices for backend logic.
  + **/user-service**: Handles user authentication, profile management, and access control.
  + **/task-service**: Manages tasks assigned to caregivers and patients, including task scheduling and completion tracking.
  + **/notification-service**: Handles the sending and tracking of notifications.
* **/front-end**: Contains the Flutter application, which is responsible for the UI/UX.
  + **/lib**: Holds the Dart code for the app.
  + **/assets**: Stores images and other assets used by the app.
* **/infrastructure**: Contains Terraform scripts for provisioning infrastructure on AWS.

4.2 Front-End Structure

The front-end of CareConnect is developed using the Flutter framework, which allows a single codebase to be deployed across Android, iOS, and web platforms. The project follows a modular and scalable architecture with a clear folder hierarchy to separate concerns and improve maintainability.

4.2.1 Folders

* android/, Android-specific files for building and running the app
* ios/, iOS-specific files for Apple device deployment
* linux/, Platform support for building on Linux (optional)
* macos/, Platform support for building on macOS (optional)
* windows/, Platform support for building on Windows (optional)
* web/, Files for compiling and deploying the Flutter web version
* assets/images/, Contains image files such as icons and logos used in the UI
* lib/assets/, May include helpers or static content related to asset management
* lib/config/, App-wide configuration (e.g., constants, themes, app settings)
* lib/core/services/, API logic, authentication, and backend service connections
* lib/features/, Organized by screen or module (e.g., login, SOS, dashboard)
* lib/frontend/, Contains shared UI elements or layouts used throughout the app
* lib/providers/, Handles state management using Provider or Riverpod
* lib/services/, Utility services like notifications, formatting, or local storage
* test/, Contains unit tests, widget tests, and integration tests
* test\_results/, Stores test outputs and result logs

4.2.2 Files

|  |  |
| --- | --- |
| * pubspec.yaml – Defines project dependencies, fonts, and asset paths. * .gitignore – Specifies files and folders that Git should ignore. * .metadata – Contains IDE and Flutter configuration metadata. * startup.sh – Script to launch the Flutter application. * quick-start.sh – One-command setup script for new developers. * generate-mocks.sh – Generates mock data for testing purposes. * migrate\_structure.sh – Helps restructure file/folder layout. * migrate\_imports.dart – Updates Dart import paths during refactoring. * add-profile-settings.sh – Automates user profile config setup. * security-check.sh – Runs security scans or validations. * run-all-tests.sh – Runs the complete front-end test suite. * simple-test-report.sh – Generates a basic summary of test results. * test-coverage.sh – Produces a test coverage report. * test\_all\_endpoints.dart – Tests all backend API endpoints from the Flutter front end. * test\_forgot\_endpoint.dart – Tests the 'forgot password' backend endpoint. * test-api-endpoints.sh – Validates multiple backend APIs via shell script. * test-jwt.sh – Tests authentication using JWT tokens. * test-jwt-manual.sh – Manual JWT authentication test script. * speech\_text.txt – Stores static content for speech-based or accessibility features. |  |

4.3 Back-End Structure

The backend of CareConnect is built using Java with the Spring Boot framework. The architecture follows a modular microservices approach, where each functional domain is encapsulated in its own service to ensure scalability, separation of concerns, and ease of maintenance.

Each microservice generally follows the same internal pattern:

* Controller Layer: Exposes REST APIs, handles incoming HTTP requests, and maps them to the correct services.
* Service Layer: Contains the business logic and orchestrates workflows.
* Repository Layer: Interfaces with the database, executing queries and returning data to the service layer.
* Configuration Layer: Sets up beans, external API access, database connections, and environment properties.
* DTOs & Models: DTOs are used to carry data between layers, while models are used for ORM (Object Relational Mapping).

Each microservice also includes its own unit tests and configuration files for local testing and deployment.

4.3.1 Folders

The backend folder in the CareConnect project contains the core Java-based Spring Boot services. Below are the key subfolders and their purposes:

* **billing/**  
   Contains logic and services related to subscription billing and payment processing.
* **core/**  
   Includes shared components, configuration files, and common utilities used across all backend modules.
* **user\_management/**  
   Handles authentication, authorization, user profiles, and account settings.
* **.mvn/**  
   Maven wrapper-related files for consistent builds across environments.
* **src/**  
   Standard Maven directory containing source code and resources.
* **github/actions/ci/**  
   Contains GitHub Actions configuration for CI/CD pipelines.
* **\_readme/**  
   Includes screenshots and visuals supporting documentation like setup guides and variable configurations.

4.3.2 Files

* pom.xml – Maven configuration file for dependency management and project build.
* Dockerfile – Contains the instructions to build the backend Docker image.
* .gitignore – Specifies intentionally untracked files to ignore in Git.
* mnvw and mnvw.cmd – Maven wrapper scripts for Linux/macOS and Windows.
* security-check.sh – Shell script for validating security configurations.
* README.md – Describes backend project details and instructions.
* API\_DOCUMENTATION.md – Markdown file outlining backend API endpoints.
* DATABASE\_CONFIGURATION.md – Explains database setup and integration.
* EMAIL\_TESTING.md – Documentation on email-related features and test configurations.
* env-entry.sh – Script for setting environment variables locally.

## 4.4 Terraform Infrastructure Structure

The infrastructure for CareConnect is defined using Terraform, an infrastructure-as-code tool that allows us to manage AWS resources consistently across environments.

Modules Explanation:

compute/: Includes Lambda function definitions, IAM roles, and permissions. Each module defines a serverless function that supports backend processing like notifications or background tasks.

db/: Manages provisioning of RDS databases. This includes subnet group creation, security group assignment, multi-AZ configuration, and secret handling via AWS Secrets Manager.

general/: Contains root Terraform configurations that reference modules. These configurations are reusable and define networking (VPC, subnets), monitoring, and IAM roles shared across services.

modules/: Shared modules used across environments (e.g., reusable IAM policies or tags).

s3\_tfstate/: Holds the backend configuration for storing Terraform’s state file in Amazon S3 with locking handled by DynamoDB. This ensures safe team collaboration and state consistency.

Initialization Process:

To deploy infrastructure:

cd infrastructure/general

terraform init

terraform plan

terraform apply

This will provision all necessary AWS resources, including VPCs, EC2, RDS, S3, Lambda, and SSM parameters.

4.4.1 Folders

* compute/Contains Terraform scripts for provisioning AWS Lambda compute resources.
* db/Manages Terraform modules and variables for the database layer.
* modules/ssm/ – Submodule for AWS SSM (System Manager) configurations.
* general/Contains reusable modules and core infrastructure definitions.
* modules/Shared reusable modules (included under general).
* s3\_tfstate/Stores Terraform backend state configurations (S3 bucket setup).

4.4.2 Files

* main.tf – Main configuration file for defining resources.
* outputs.tf – Defines outputs after resources are created.
* variables.tf – Lists input variables used in main configurations.
* README.md – Instructions for initializing and applying Terraform in each directory.
* .gitignore – Specifies files/folders to exclude from Git versioning.
* structure.yaml – (If present) may define module hierarchy or configurations.

# 5. Code Structure

## 5.1 Overall CodeStructure

The CareConnect project follows a modular, layered architecture, allowing the development team to work independently on different parts of the system while maintaining clear separation of concerns. The codebase is divided into three main components:

## 5.2 Front-end Code Structure

The front-end code follows the Model-View-Controller (MVC) pattern:

* **Model**: Contains data and logic for the application (e.g., user models, task models).
* **View**: The user interface - implemented using Flutter widgets.
* **Controller**: Logic that communicates with the backend to fetch data and manage the app’s state.

## 5.3 Backend Code Structure

The backend is organized into microservices that handle specific domains:

* **Controllers**: Handle HTTP requests.
* **Services**: Contain business logic.
* **Repositories**: Interface with the database for data persistence.
* **DTOs (Data Transfer Objects)**: Represent data structures that are passed between the controller and service layers.

## 5.4 Coding Conventions

* Use camelCase for variable names and function names.
* Classes and methods should be named in a descriptive, business-related manner.
* Every class and method must have a header comment explaining its functionality.
* Complex blocks of code should be well-commented.

# 6. Known Bugs & Workarounds

## 6.1 UI Inconsistencies on Older Android Devices

The application layout can also not display correctly on some Android devices with older versions (of less than 9.0) because of incompatibility with older SDK versions of Android. This has been reported, and a fix will be released next.

## 6.2 Push Notifications Delayed on iOS

The iOS is sometimes associated with delays in push notifications, especially when it experiences high load. That can be bound to network congestion or a push notification service provided by Apple. Additional tests are underway to determine the root cause.

**Front-end Development Workspace Setup**

To set up the front-end development environment for CareConnect, follow these steps:

1. **Install Flutter Plugin in IDE**
   1. Ensure you have **VS Code** or **Android Studio** installed.
   2. Install the **Flutter** plugin/extension in your IDE for enhanced Dart and Flutter support.
2. **Download and Install Flutter SDK**
   * Download the latest version of the **Flutter SDK** from the official Flutter website.
   * Add Flutter to your **PATH environment variable**: export PATH="$PATH:`pwd`/flutter/bin"

For Windows, modify the Environment Variables in the Control Panel and add the path to your Flutter SDK’s bin directory.

1. **Install Android Studio**
   * Download and install **Android Studio** from [here](https://developer.android.com/studio).
   * During installation, make sure to install the **Android SDK** and **Android Emulator** for testing.
2. **Enable Developer Mode (Windows)**
3. Open **Settings > Update & Security > For Developers** and turn on **Developer Mode**.

## 6.3 Wearable Integration

Wearable integration is a key feature of CareConnect that enhances patient monitoring through continuous data collection from devices such as smartwatches and fitness trackers. This integration provides visibility into physical activity, heart rate, sleep patterns, and other vital health metrics, allowing caregivers to respond proactively to changes in patient status.

The current scope of integration includes support for:

* **Fitbit** for Android (partial).
* **Google Health Connect** for Android.
* **Apple HealthKit** for iOS.

The integration relies on OAuth 2.0 authentication and secure data exchange with the wearable APIs. At present, support is limited due to platform-specific constraints and third-party API restrictions.

### 6.3.1 Fitbit Integration on Web and iOS

Currently Fitbit integration is not supported on the web and is untested in iOS. Fitbit requires the authentication callback to be registered on the Fitbit developer side regardless of the redirect URI being part of the request sent to the Fitbit API. Fitbit performs a validation check against the callback URI registered on their side.

Registering two callback URIs on the Fitbit developer does not allow Android to handle the callback gracefully if there is a web URI registered as well. While iOS remains untested.

If now or in the future the callback needs to change, the new URI needs to be registered directly here: <https://www.fitbit.com/dev.>

### 6.3.2 Google Health and Apple Health Kit

Each of these options are only available in their respective environments. Google Health only works on Android, and Apple Health only works on iOS. The option is not available on the web or any other platform.

Please note that for older Android devices who still utilize Google Fit, our implementation will not support a connection due to the Google Fit API being deprecated by Google. Users must have Google Health installed on their devices.

## 6.4 Home Monitoring

Nest camera integration is not currently supported. The UI components have been implemented but there is no backend functionality. Integration of this capability is hindered due to the following technical blocks:

* Google Smart Devices offers a loc cost “$5” tier that only supports 25 devices.
  + The free tier cannot be utilized in any deployment.
* Utilization of the enterprise/commercial tier can incur a cost and requires certification from Google to utilize their Smart Devices API. Certification is not immediately and requires various checks from Google. This certification ensures that the app integrates well with its ecosystem and allows the application to brand the “Works with Google Home” badge.
* Other ecosystems like Ring or Arlo do not have open APIs.

## 6.5 Smart Devices

Smart device integration is not currently supported. The UI components have been implemented but there is no backend functionality. Integration of this capability is hindered due to the following technical blocks which are similar in nature to the home monitoring blocks:

* Integrating Alexa compatible smart devices requires a certification from Amazon which may incur a cost.
* The certification process takes time and it's not immediate. This ensures Amazon reviews and vets the application before it is allowed to use its API. This ensures the app that integrates with their smart devices can brand the “Works with Alexa” branding.

For future development, CareConnect must support the Matter open standard which was developed for interoperability across devices and applications regardless of the device manufacturer. If the device supports the Matter standard, then it can be integrated into applications as long as the standard is implemented within the application.

## 6.6 Speech to Text Package

The use of speech-to-text functionality is limited to web platforms and is not implemented on mobile platforms. This is due to a dependency compatibility issue between the speech\_to\_text package and the Kotlin compiler. This build error is causing the mobile build to fail when trying to compile.

The current workaround regarding this issue is to comment out the speech\_to\_text package dependency in the pubspec.yaml file and not referencing or using any speech\_to\_text functionality in the mobile platform. In addition, any references to the speech\_to\_text package in corresponding dart files will need to comment out the package as well:

* app\_router.dart
* healthcare\_notes.dart
* voice\_command\_ai.dart

## 6.7 Automated Amplify Deployments

Currently, students cannot create continuous deployments from the course GitHub to a created AWS Amplify instance. This is because there is a permissions issue when setting up the automated deployments with new commit changes from a branch on the course GitHub and the created AWS Amplify instance. In order to connect an AWS Amplify instance to a GitHub account, the user is required to have admin permissions on the GitHub account to allow for the connection between the two resources to be made. Since students are only given user permissions to access the course GitHub, this creates a limitation of permissions allowed by users to connect to 3rd party applications. Thus, students are unable to created automated deployments to AWS Amplify.

One workaround to this that was identified was to collaborate with a member of the course staff that has admin permission on their GitHub account to create an AWS Amplify resource manually on their AWS account and establish the connection between the two resources. This way, Amplify can listen for any new commits to the selected GitHub branch and automatically deploy the changes to Amplify.

## 6.8 Implementation of Long-Lived connections

During the course, we had planned to implement several features using a WebSocket-based approach. However, due to changes in our infrastructure design, we were unable to support WebSocket effectively. The infrastructure we delivered was optimized for stateless, short-lived executions, which are not compatible with the persistent connection model required by WebSocket. Given the limited time available after the infrastructure shift, we explored alternatives and were able to deliver a working solution before the end of the course. This solution did not rely on WebSocket and stayed within the constraints of our serverless architecture. While WebFlux was considered as an alternative, it introduced its own set of challenges. Specifically, running a Spring Boot application with WebFlux in a Lambda environment led to conflicts between multiple dispatcher types. These conflicts made it difficult to reliably start and run the application, especially given the isolated nature of Lambda invocations.

It’s important to note that AWS Lambda is designed for quick, stateless invocations. Each invocation runs in isolation, without shared state or persistent connections. This makes it unsuitable for use cases that require long-lived sessions or real-time bidirectional communication.

We are leaving the exploration of WebSocket and WebFlux as a future consideration for upcoming teams. However, we want to emphasize that our infrastructure was intentionally designed with scale-to-zero as a priority. For this reason, we avoided using resources like ECS or Fargate, which could support persistent connections but would compromise the serverless nature of our solution.

We would recommend looking into API Gateway Websocket API to integrate with Lambda. The option of having micro-services could also help here.