



LIFESAVERS

COMPSCI 399 Capstone Course Project Report

Team Members:

HaoChen Li - Backend Lead

Danarta Sanyata - Backend

Lucy Baldwin - Frontend Lead

Rajat Kumar - Frontend

Mshari Fahad Alajaji - Frontend

Jiachen Xu - Backend

Executive Summary

Caregivers often grapple with demanding, round-the-clock shifts and an intense workload, taking care of another person's life. Furthermore, the changing social, economic, and political landscape has resulted in a shortage of caregivers. This responsibility places immense pressure on their performance and requires unyielding diligence. The chronic understaffing in elder care facilities exacerbates the issue, resulting in escalated stress and burden for the existing caregivers.

Enter LifeSavers, our groundbreaking application that aims to lighten the load for these diligent caregivers. We propose an innovative solution wherein patients are equipped with a Fitbit smartwatch, which doubles as a continuous health data transmitter. This wearable device dutifully relays health data to caregivers daily, offering them an in-depth understanding of the patient's health. We replace the monotonous and tiresome manual checks with a modern, automated system that ensures superior care, enhanced with real-time notifications.

Our innovative solution rides on the cutting-edge AWS serverless architecture, exploiting the robustness of Python and Javascript for back-end processes. The mobile application has been adeptly crafted using the Flutter framework, while Javascript powers the Fitbit application.

With LifeSavers, we're instigating a paradigm shift in the caregiving sector. We invite you to join us on this transformative journey to render caregiving more efficient and less strenuous, ultimately improving the quality of life for patients and their selfless caregivers.

Table of Contents

[Executive Summary](#)

[Table of Contents](#)

[Introduction](#)

[Background](#)

[Project Specification](#)

[Project Design](#)

[Project Implementation](#)

[Specific Technical Features](#)

[Results & Evaluation](#)

[Future Work](#)

[Conclusion](#)

[References](#)

[Appendices](#)

[Declaration of Authorship](#)

Introduction

In light of the increasing population of elders in New Zealand diagnosed with dementia, there is growing concern among them and their families regarding the quality of care they will receive. Compounded by the issue of staff shortages in aged care facilities, there is a risk of compromised care quality.

Aims and Objectives:

LifeSavers is a project designed to create a wearable app that targets issues that come with staff shortages in elderly dementia care. Our aim is to relieve stress on the caregiver shortage by making patient monitoring more efficient. The app aims to gather real time health data from people with dementia through a wearable device. It also aims to send notifications to their caregivers when that health data indicates the patient is in need of attention. Through this alert system our app aims to relieve the caregivers workload, and enhance the freedom of those with dementia.

Target audience:

Our primary target audience comprises caregivers for elderly individuals, particularly those diagnosed with dementia. We specifically are aiming to help those in elderly care facilities, with caregivers that need to look after more than one person. Additionally, we aim to help those with more advanced levels of dementia, and those that particularly struggle with wandering and getting lost, those that move and pace when anxious or agitated.

Scope:

The scope of the LifeSavers project includes the following:

- Acquiring real-time data from a wearable device.
- Creating a consistent manner to store patient health metrics for future data analysis
- Developing a companion mobile app capable of monitoring multiple patients, featuring:
 - User authentication functionality.
 - The ability to add and remove patients from the caregivers' interface.
 - Alert notifications to caregivers based on real-time data indicating the immediate care needs of patients.

The overarching purpose of our project is to initialise a foundation for patient monitoring. Advanced functionalities like long term data analytics, a patient dashboard for caregivers and more advanced health metric notifications are deemed outside the scope of our project.

Methodology/Approach:

To realise our project objectives, we had to accomplish the following:

- Design the system architecture using the selected frameworks and services (Flutter, AWS, Fitbit).
- Research aged care and dementia.
- Perform UX/UI design research for mobile applications.
- Design the database to successfully retrieve data from Fitbit and transmit it to the app.

In terms of the technical strategy for the project, our team elected to adopt a procedural programming methodology rather than a functional or object-oriented approach. On the non-technical side of our methods, we primarily used the Kanban model as our agile development strategy, attributed to our team members' adaptability in task completion. We will delve deeper into these aspects in the implementation segment of our report.

Outcomes :

The primary outcomes of our project include the development of an app that enables caregivers to monitor better and assist dementia patients through alerts. This should alleviate stress on the caregivers and those under their care.

Background

Preliminary background

Over recent years, wearable technology has become increasingly integrated into our lives. Our primary focus will be on devices developed for people with dementia. Dementia affects millions worldwide, and the number of cases will increase in the coming years due to ageing populations. With that, the demand for a device to help make their lives easier is high.

Alzheimers New Zealand has stated that due to staff shortages, they estimate that 30000 people with dementia are missing out on vital care (RNZ, 2022). Our project aims to assist in solving the shortage of care and support services by making it easier for caregivers to manage multiple patients.

Dementia can affect people in multiple ways, including sleep issues; wandering; agitation and anxiety; memory loss and confusion, and hallucinations (Alzheimer's Association, 2019). We decided to target wandering, sleep issues and agitation and anxiety. We were mindful of what would cause resource strain on caregivers, so we wanted to target wandering and sleep. Additionally, we wanted to relieve stress for those with dementia. We found a study that tracked step count in those with dementia as an indicator of agitation (Moyle, 2018). We also saw that increased heart rates indicate stress/anxiety (Stress, 2021).

Analysis of existing solutions







Our research found a few solutions targeted at elder care, like the AWG-A19 Elderly GPS watch (Smartwatches NZ, 2023). We liked the Geo-fence capabilities and the capability to track multiple watches simultaneously and wanted to include these in our approach. However, a key feature we wanted to have was to track health data. The voice chat and SOS message are nice features; however, we wanted to make the Fitbit application as user-friendly to the elderly as possible, and in New Zealand, the Saint John Medical alarms already exist. We found a few similar solutions; however, accessing this technology in New Zealand takes a lot of work. We believed that extending the capabilities of the cheaper, widely available Fitbit would be more accessible for carers in New Zealand.

The Value of this Project

Regarding the value we are bringing to the table, what differentiates this app is that it acts as a proactive advisor on top of the data-collecting aspect that most apps have today.

Another value that this app brings is that it helps not only dementia patients but also caregivers. Caregivers are typically subject to exhaustion due to the physical, mental and emotional stress they must endure while caring for their patients. Their ability to track patients' data, along with their locations, will significantly reduce their stress.

Methods and Tools used for the project

Tool	Description
 Flutter	Flutter is the framework used for the development of the mobile.
 AWS	AWS is where we host our serverless backend
 Javascript	Javascript is used in the visual design of the Fitbit application, and in our lambda functions on AWS
 Python	Python is used in the lambda functions on AWS
 Fitbit	Fitbit is an external API queried for data collection purposes
 Mapbox	Mapbox is an external API queried for patient location tracking functionalities

Project Specification

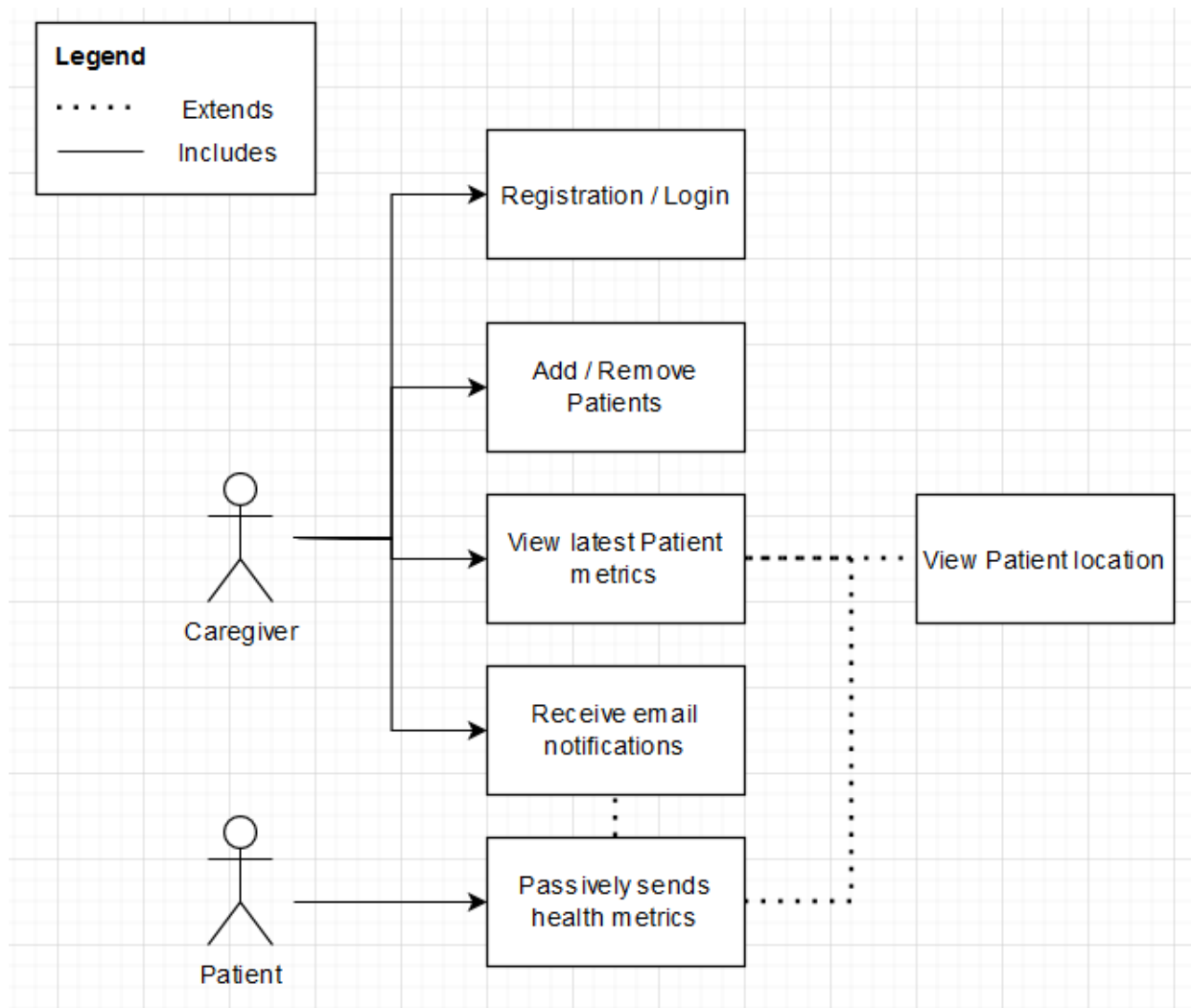
This project is primarily centred around the needs of caregivers. The chief user requirements are as follows:

1. As a caregiver, I want to receive notifications and alerts when a patient's health metrics exhibit anomalies so that I can promptly attend to their needs.
2. As a caregiver, I want to manage multiple patients simultaneously so that I can enhance my capacity to deliver quality care to patients.
3. As a patient, I want a care model that harmoniously integrates with my daily life so that I can continue to be facilitated in my wellness.

Based on the user requirements, we derived a comprehensive list that guided our application development process:

Stakeholder	Functional	Non-functional
Caregiver	<ul style="list-style-type: none">• Capable of associating themselves with specific patients• Enabled to receive health alert notifications for assigned patients• Granted visibility to health data of multiple patients• Provided with login capabilities• Allowed to remove patients from their list of care• Provisioned with sign-up functionality	<ul style="list-style-type: none">• Required to verify their email address• Enforced to use strong passwords
Patient	<ul style="list-style-type: none">• Able to retrieve their unique patient code• Capable of passively transmitting health data from their Fitbit device	

We employed the UML Use-Case diagram to establish a robust framework for the development process.



The project addresses all the user requirements that were identified initially. Most features are accessible to all users, regardless of their profiles. The project's feature set includes the following:

For Caregivers:

Addition/Removal of Patients from Care List (2):

Caregivers can add or remove patients from their care list using the application. The adding process requires the patient's unique ID to be displayed on their Fitbit device. By doing so, caregivers can manage and monitor the health metrics of multiple patients in real time, enhancing their ability to respond promptly to any health issues. Removing patients from their list can be done quickly when the caregiver's services are no longer needed, ensuring privacy and data integrity.

Receiving Email Notifications from Patients (1, 2):

The application is designed to alert caregivers by email when it detects irregularities in a patient's health data. This feature was designed after extensive consultations with clients who expressed the need for a system that provides a comprehensive and easily accessible history of notifications. This means that caregivers can refer to these email alerts anytime, offering a seamless way to track patient health over time.

Monitor Patient's Connectivity Status (1)

This feature enables caregivers to view the connectivity status of each patient instantly. This could indicate device issues or suggest that a patient has forgotten to wear their device. This immediate insight into a patient's status helps caregivers troubleshoot potential problems and ensures consistent health tracking.

Review Latest Health Metrics of the Patient (1)

With this feature, caregivers can remotely check in on the health status of their patients. It allows access to up-to-the-minute health data, such as heart rate, activity levels, sleep patterns, and more. This not only saves time but also avoids disturbing the patient unnecessarily.

Location Tracking of the Patient (1, 2)

Using Mapbox in our application allows caregivers to know the location of the patients they care for. This feature is handy in scenarios where a patient isn't at their regular place and requires immediate attention or care. The location data can help caregivers or emergency services to reach the patient promptly.

For Patients:

Passive Transmission of Health Metrics (3)

Patients do not have to take any active steps to send their health data. Their Fitbit device will passively collect and send data to our backend, making it available on our mobile application. This setup ensures that the patient can go about their day without disruptions while their health is continuously monitored.

Responsive design that lets the patient know the device is working (3)

Our application's Fitbit device design displays the current time, the unique patient ID, and a count of data transmission instances. This approach reassures the patient about the device's proper functioning. Additionally, this introduces a layer of security, as only caregivers who have met the patient in person and have seen their unique patient ID on Fitbit can subscribe to their health data. This protects patient data from unauthorised access while keeping the patient informed about the device's working status.

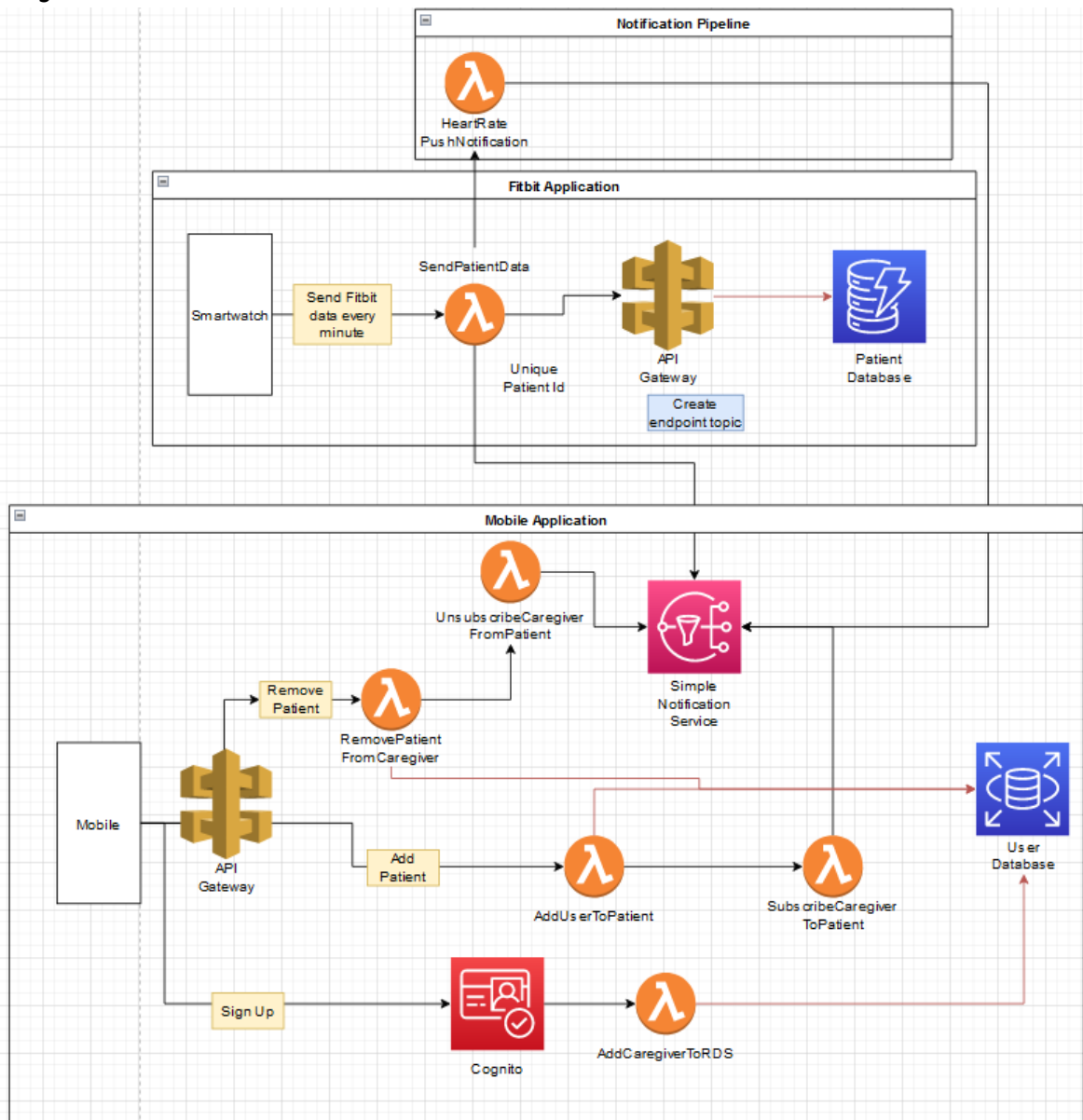
Project Design

In our initial software project phase, our prime objective in architectural selection was to secure reliability, scalability, and maintainability. The client's original technology recommendations include Flutter for the mobile application, .NET C# for the backend, and AWS for deployment.

Further analysis prompted us to adopt a deployment-centric approach, choosing our deployment strategy before initiating the coding phase. We used a serverless application through lambda functions instead of a containerised EC2 instance on AWS. The decision was primarily driven by the fact that our application wasn't anticipated to perform intensive processing tasks but mainly relay data to our database. Additionally, leveraging a serverless backend allowed us to use the team's expertise in Python and JavaScript, thus, circumventing the need for C# altogether. Consequently, the technology-related risks were narrowed down to AWS and Flutter for our team.

Flutter emerged as our chosen framework owing to its cross-platform compatibility and the potential for rapid, iterative front-end development. Its impressive feature of supporting hot-reloading for mobile front-end design further facilitated a swift development process. Even though our final application was designed to support Android devices exclusively, Flutter was deemed the right fit. Its strength lies in its ability to lay a robust foundation for future IOS development with minimal requirements for significant alterations in the existing codebase.

AWS System architecture:



This report outlines the overall system architecture, which integrates with Amazon Web Services (AWS). Our architecture comprises three key components: the alert mechanism, the Fitbit interface, and the mobile app module.

The alert mechanism is the primary module that flags irregularities in patient health data. At present, the system is set up to alert heart rate anomalies, but we envision expanding this feature to include additional health notifications.

The Fitbit interface plays a crucial role by consistently transmitting data to our backend DynamoDB. This architecture section is expected to remain relatively stable as data modifications are dispatched as JSON documents complete with the patient's full dataset. Consequently, any requirement for further data would entail alterations on the client side rather than the backend.

Like a web application, the mobile app module comprises most of our system's logic. It encompasses all Create, Read, Update, and Delete (CRUD) operations and leverages Cognito for user authorisation and SNS for email-based push notifications.

Our architecture contains four primary data flows. The first occurs in response to an irregularity in patient health data and follows these steps:

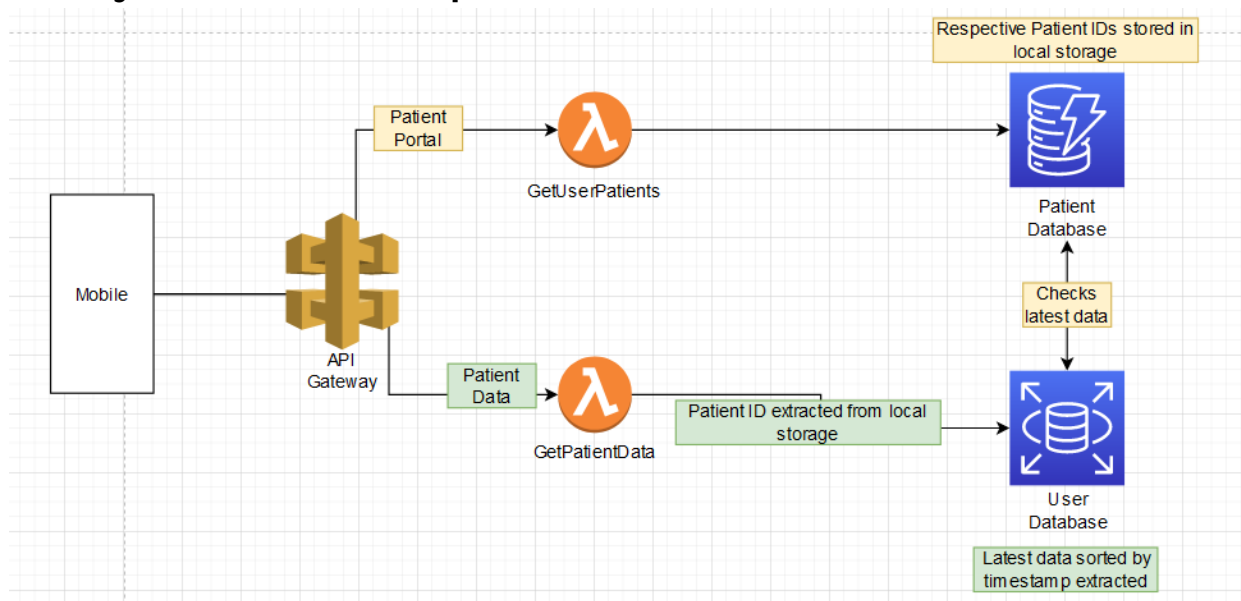
1. Health data is dispatched from the Fitbit interface.
2. Lambda functions within the alert mechanism scan for any health irregularities.
3. Relevant caregivers receive an email alert.

The second data flow involves everyday interactions between the caregiver and the mobile app module, including:

1. Caregiver logging into the mobile app using Cognito for authentication.
2. Lambda functions manage the addition and removal of patients into the RDS.

The third data flow is more complex and will be elaborated on in the subsequent section of this report.

AWS System architecture: patient data retrieval



We've focused on user interface (UI) components for the AWS diagram to enhance comprehension instead of delving into the underlying functions.

The third data flow proceeds as follows:

1. The user engages with the patient portal.
2. This action triggers the execution of the 'GetUserPatients' Lambda function.
3. All patient identifiers (IDs) linked with the user are extracted and locally stored.
4. Each patient ID is subsequently cross-checked against our DynamoDB database to determine if the patient's latest entry is within a minute of the current time. This check assigns each patient a "connected" or "disconnected" status on the portal screen.

In terms of the final data flow, it involves these steps:

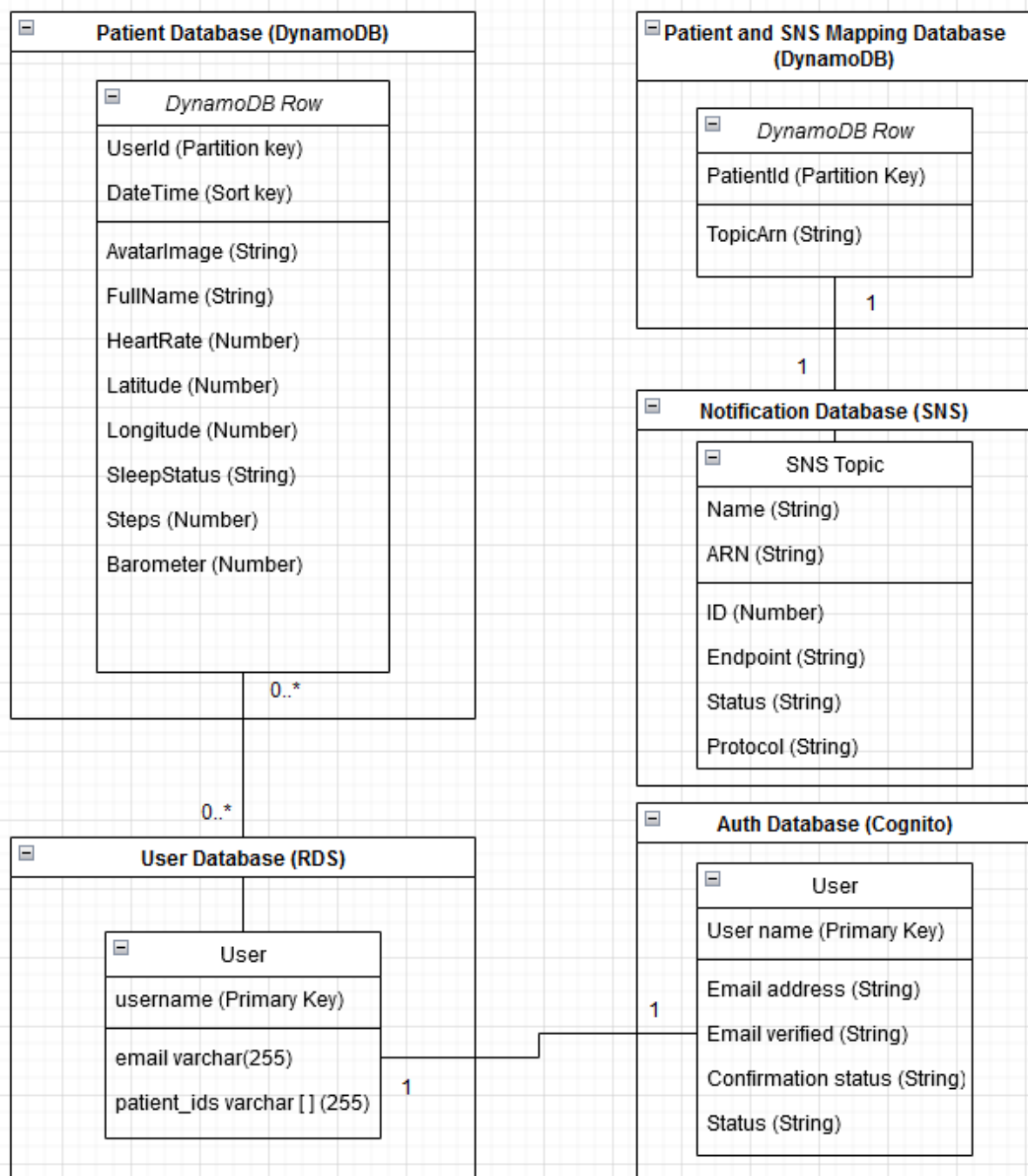
1. The user navigates to the individual patient data view from their patient portal screen.
2. The respective patient ID is retrieved from local storage.
3. The most recent data corresponding to this patient ID is then fetched from our DynamoDB database.

Programming Approach

Our team selected a procedural programming approach over functional or object-oriented methodologies for the development of our software. This decision stemmed from our team's limited experience with the functional programming paradigm. Moreover, considering our application was primarily comprised of CRUD (Create, Read, Update, Delete) operations interfacing with the database, we didn't view the project as sufficiently complex to necessitate an object-oriented strategy.

The choice of procedural programming aligned with our team's skill set and facilitated a swifter development process than the other alternatives. As a result of this choice, we designed our models as simple entities with immutable states equipped with straightforward getters and setters. The business logic associated with these models was managed by dedicated repository or service classes, separating the data representation from business operations.

Database design:



Per the client's specifications, the project involved utilising a NoSQL database via AWS DynamoDB and a SQL database via AWS RDS.

DynamoDB was chosen for its capacity to house various variables from the Fitbit API. The lack of a predefined schema in DynamoDB provided flexibility for accommodating diverse Fitbit variables, given the uncertainty about which variables effectively signified patient health. Therefore, a DynamoDB table was deemed appropriate for storing all obtainable variables from the Fitbit API, facilitating future analyses to enhance patient notifications' accuracy. Moreover, since patient data was to be collected from Fitbit every minute, DynamoDB was favoured for its superior performance in read/write operations compared to an SQL database like RDS.

On the other hand, RDS was employed to store more regular variables, such as authenticated users and their corresponding patients, through a one-to-many association. Furthermore, Postgresql was chosen as the database engine for its robustness, extensibility, standards compliance, and support for complex data types and advanced SQL queries. The decision to opt for Postgresql was also driven by the relationship-focused nature between users and patients and the ACID (Atomicity, Consistency, Isolation, Durability) properties it provides. These properties are crucial for ensuring data integrity and consistency in a database environment, making Postgresql an excellent choice for relational data management in our project.

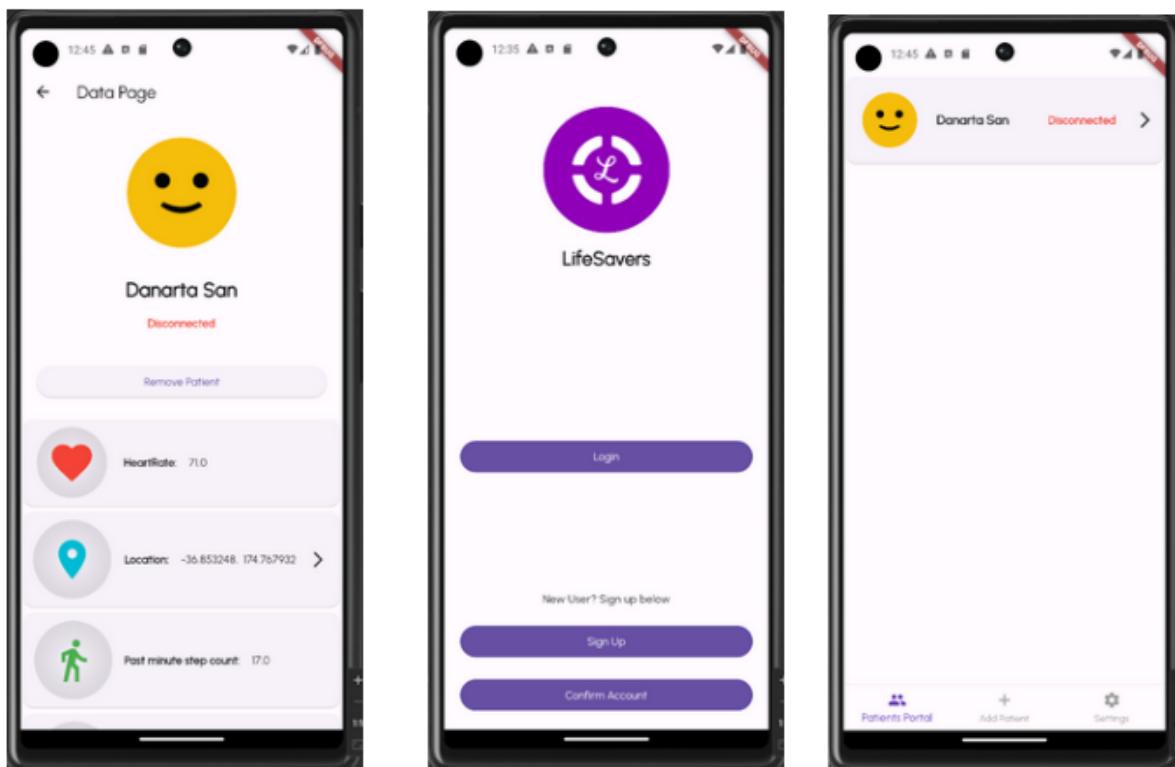
Due to the project's reliance on serverless functions and AWS services, our ERD diagram is quite simple. We have included it nonetheless.

User Experience/User Interface

The predominant colour in our application is purple, chosen primarily for its symbolic representation of independence and elegance, resonating with our client base. Furthermore, purple is soothing to the eye, making it a more accessible colour choice for the middle-aged segment of our target audience. It also offers a superior contrast in light and dark themes, enhancing overall usability.

We opted for a Material Design theme for its simplicity in implementation and seamless compatibility with Android devices. This design philosophy espouses a minimalist and intuitive user interface that is unobtrusive, ensuring an easy and smooth user experience for caregivers.

Below, you'll find the final designs of a few critical pages from LifeSavers, including the Home, Patient Portal, and Patient Data pages:

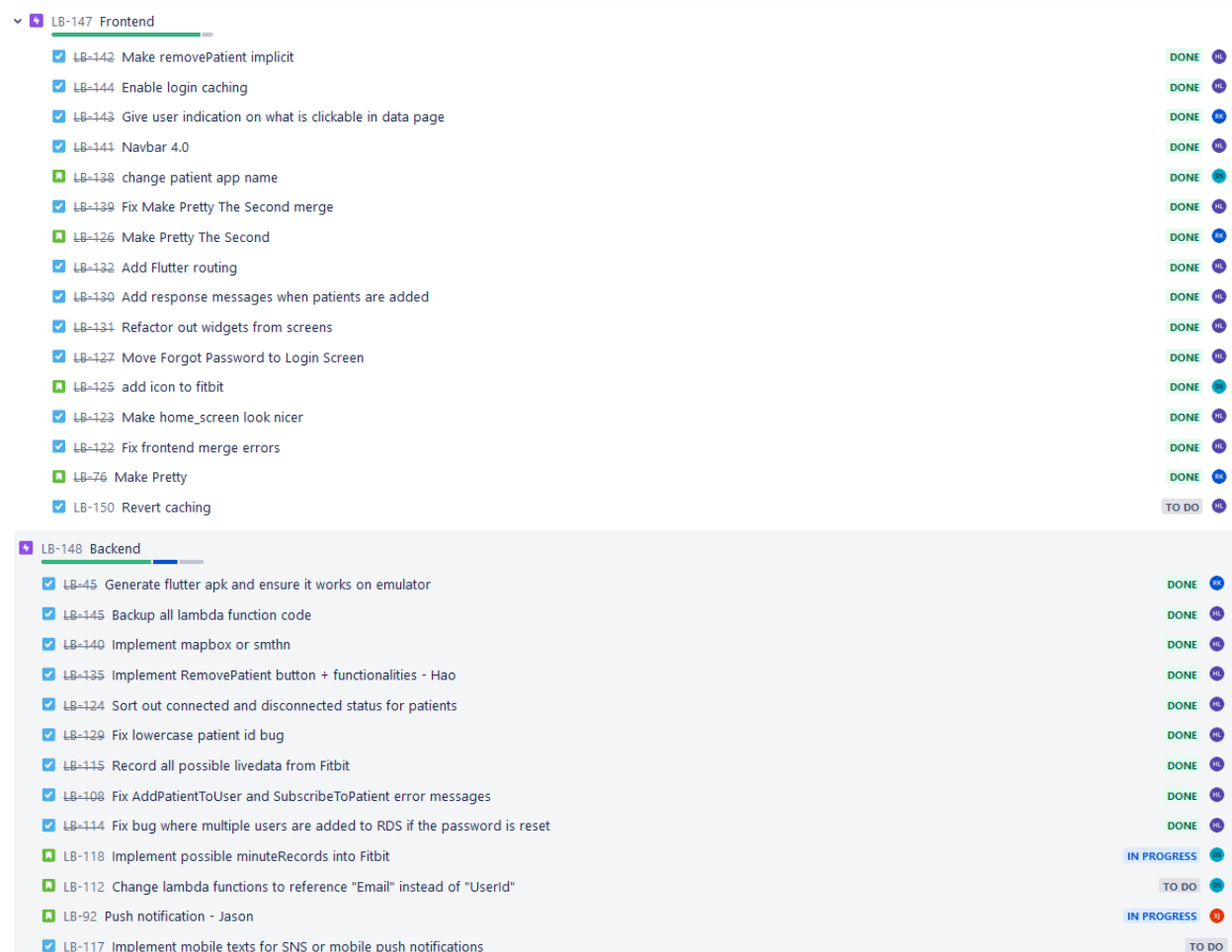


Project Implementation

While our team had initially outlined a Gantt Chart in the project proposal with the plan to employ a Scrum-based methodology, we pivoted to the Kanban approach due to ambiguous requirement specifications. Consequently, our workflow became more adaptive, accommodating new feature requests from the client and changes to existing features as needed.

Following every client meeting, we added tickets to our to-do list, allowing team members to choose tickets that suited their capacities and roles, such as front-end or back-end development. In cases where issues arose outside their designated roles or when members discovered unexpected bugs in previously implemented features, additional tickets were created.

Enclosed is a snapshot of the features we implemented during the project, as captured from the Jira board.



LB-147 Frontend

- ☒ LB-142 Make removePatient implicit **DONE**
- ☒ LB-144 Enable login caching **DONE**
- ☒ LB-143 Give user indication on what is clickable in data page **DONE**
- ☒ LB-141 Navbar 4.0 **DONE**
- ☒ LB-138 change patient app name **DONE**
- ☒ LB-139 Fix Make Pretty The Second merge **DONE**
- ☒ LB-126 Make Pretty The Second **DONE**
- ☒ LB-132 Add Flutter routing **DONE**
- ☒ LB-130 Add response messages when patients are added **DONE**
- ☒ LB-131 Refactor out widgets from screens **DONE**
- ☒ LB-127 Move Forgot Password to Login Screen **DONE**
- ☒ LB-125 add icon to fitbit **DONE**
- ☒ LB-123 Make home_screen look nicer **DONE**
- ☒ LB-122 Fix frontend merge errors **DONE**
- ☒ LB-76 Make Pretty **DONE**
- ☒ LB-150 Revert caching **TO DO**

LB-148 Backend

- ☒ LB-45 Generate flutter apk and ensure it works on emulator **DONE**
- ☒ LB-145 Backup all lambda function code **DONE**
- ☒ LB-140 Implement mapbox or smthn **DONE**
- ☒ LB-135 Implement RemovePatient button + functionalities - Hao **DONE**
- ☒ LB-124 Sort out connected and disconnected status for patients **DONE**
- ☒ LB-129 Fix lowercase patient id bug **DONE**
- ☒ LB-115 Record all possible livedata from Fitbit **DONE**
- ☒ LB-108 Fix AddPatientToUser and SubscribeToPatient error messages **DONE**
- ☒ LB-114 Fix bug where multiple users are added to RDS if the password is reset **DONE**
- ☒ LB-118 Implement possible minuteRecords into Fitbit **IN PROGRESS**
- ☒ LB-112 Change lambda functions to reference "Email" instead of "UserId" **TO DO**
- ☒ LB-92 Push notification - Jason **IN PROGRESS**
- ☒ LB-117 Implement mobile texts for SNS or mobile push notifications **TO DO**

Database Implementation

Our software solution was built on the foundation of two separate databases, both implemented through DynamoDB. The central NoSQL database was structured to contain patient-specific information, where "UserId" was utilised as the partition key, and "DateTime" served as the sort key. Upon the project's conclusion, each patient record comprised various parameters such as UserId, DateTime, AvatarImage, FullName, HeartRate, Latitude, Longitude, SleepStatus, Steps, and Barometer. Our secondary NoSQL database functioned to link SNS topic ARNs to the PatientId, thus creating a straightforward key-value mapping database that retrieves the associated SNS endpoint when a patient's data is queried.

In addition, we integrated a SQL database, namely PostgreSQL, using RDS (Relational Database Service) offered by AWS. This RDS database mainly accommodates User (caregiver) objects defined by attributes such as username, email, and raw patient ids. These patient ids were then used to draw the most recent entry linked to a particular patient id from our DynamoDB document database.

We encountered a few challenges during this process. It was recognised that DynamoDB and RDS have different service models within AWS: DynamoDB is a database as a service, while RDS is a database service as a service. We made our RDS endpoints public due to time limitations and initial hurdles in establishing VPC configurations. This inevitably presents potential security risks to our application due to the publicly exposed endpoints. Mitigation strategies for this security concern are actively being explored and discussed further in this report's future work section.

Additionally, we found it rather challenging to modify the nomenclature for our database and its corresponding fields; this will be expanded more in our report's "Results & Evaluation" section.

Version Control Strategy

We utilised GitHub as our primary system for version control. Our development process was orchestrated around the main branch, representing the project's "production" version. The workflow process is initiated with ticket creation on Jira, followed by creating a corresponding branch, offering visibility into ongoing tasks within the team.

Every branch was linked to a ticket, and each ticket, in turn, corresponded to either a feature development, a bug fix, or a user interface modification. Team members were responsible for their respective branches but had the flexibility to seek assistance from others when needed. This approach enhanced ticket completion and assignment workflow, minimising conflicts and duplications.

Each pull request (PR) would proceed through a GitHub Action pipeline and require approval from at least one other team member before merging. This practice ensured that our main branch remained operational, as the merge process required a successful build pipeline and a team member's review.

Project Folder Organisation

The project is systematically divided into three directories named "back-end", "Fitbit", and "front-end."

The "back-end" directory is essentially a repository for the backup versions of the code, employed directly in our AWS lambda functions. It does not enhance the overall features or capabilities of the project.

The "Fitbit" directory houses the Fitbit application code, which is primarily written in Javascript. The organisational structure of this application mirrors the one advised in the Fitbit API, with the 'app' directory accommodating the smartwatch application and the 'companion' hosting the Fitbit mobile phone application.

Lastly, the "front-end" directory accommodates the mobile application code developed in Flutter. Given the uncomplicated nature of our domain and anticipated database interactions, we have built our functionality on a screen-by-screen basis. For instance, the "Patient Screen" has its dedicated "patient_screen" directory, consisting of the main screen and the widgets employed on that particular screen. The routing has been executed via a custom routing logic, ensuring the functionality of the navigation bar. In addition, the 'data' directory contains the repository layer for interfacing with our back-end database, and the 'auth' directory encompasses the User Service and other files assisting with authentication and authorisation. In contrast, the 'domain' directory is home to our class models.

Specific Technical Features

Patient Model Implementation

Every patient record within our system corresponded directly to a single entry in our DynamoDB patient database. This design decision was based on the understanding that our patient records functioned as specific display objects. The front-end mobile application was built to present these Patient models to users without executing complex calculations or state transformations.

Response Model Implementation

The response model primarily comprises a boolean variable, "isSuccess", and a string variable, "message". The core objective of implementing this response model is its interaction with the status returns of our application, mainly through our lambda functions. This enables a more personalised approach to error messaging, notably when our snack bar visual component displays them.

User Model Implementation

Every user record in our system is accurately mirrored as a row in our Cognito service. Given the intricate nature of the application that necessitates data being visible solely to logged-in users and only presenting data specific to the patients to whom the user is linked, we implemented a distinct "user_service.dart" class. This class manages mobile authorisation, account verification, and logout operations. This strategy aligns with our chosen procedural programming approach, and the "user_service.dart" class is chiefly responsible for managing the user model's business logic.

We encountered an unexpected challenge when integrating Cognito with our Relational Database Service (RDS). As an Amazon Web Services (AWS) solution, Cognito delivers user authorisation and authentication, yet we aimed to link patient IDs to the user object, a task that could only be performed indirectly with Cognito. To navigate this issue, we duplicated a Cognito role into our RDS system, which allowed us to initialise the user's patient IDs as an empty list and proceed from there. Nonetheless, an optimal solution would be to avoid data duplication and conduct operations directly from Cognito.

Email Notification System

We emphasised email notifications over mobile push notifications since emails provided a more formal and traceable record of patient health data. Our email notification system was implemented using the AWS Simple Notification Service (SNS). When patient health data was first transmitted to our AWS back-end, an SNS producer endpoint was generated. Consequently, when caregivers added a patient, they subscribed to the same topic, effectively receiving notifications as subscribers. This system allowed multiple caregivers to receive messages from a single subscription if needed.

To bolster security, we required caregivers to confirm their subscription to the patient's updates before they could start receiving email notifications. This measure ensured that caregivers had greater control over their inboxes and safeguarded them from potential malicious activities.

Implementation of CRUD Operations

From the project's structure, our principal decision was to utilise lambda functions to execute create, read, update, and delete (CRUD) operations for patients in our RDS system. For instance, in a "GetUsersPatients" request, the request would originate from the client and pass through the repository layer, which would then query the AWS API Gateway endpoint. The corresponding lambda function would execute and return the desired result. This process allowed us to maintain a clear separation between the front-end and back-end and leverage the advantages of a REST-based architecture.

One issue with this approach is that our back-end might need to be more evident for future maintenance and documentation as they are housed in AWS, not the repository. To reduce repetition, we expanded more on this area in our report's "Results & Evaluation" section.

Synchronisation between Mobile Applications and Fitbit Smartwatch

The communication between the mobile application and the Fitbit smartwatch is a publisher-subscriber model. Interested subscribers (caregivers using the mobile application) subscribe to publishers (patients wearing the smartwatch) and receive updates accordingly.

A deliberate design choice was to eschew a search function and compel caregivers to input the ID associated with the smartwatch of the patient they are looking after. This decision was rooted in the commitment to protect user privacy - allowing users to search and subscribe to a random ID could potentially violate this privacy. Mandating the input of the patient's ID provides an inherent layer of security as it implies that the caregiver must have a real-life connection with the patient to obtain the ID, thereby enhancing the application's security.

However, an unintended consequence of this design choice is that we need a mechanism to alter the patient's ID. Consequently, if a caregiver had access to the patient's ID in the past, we have no security measures to prevent them from subscribing to the patient's updates again. This issue will be explored more extensively in our future work section.

Results & Evaluation

Evaluation of project goals:

Our project initiative revolved around improving the understanding and application of data obtained from wearable devices, specifically within the domain of elder care. Our partner for this endeavour was Fitbit, a well-established health and physical activity monitoring entity.

Utilising Fitbit's comprehensive APIs allowed us to gather essential health metrics in real time. Our choice of critical indicators - heart rate, step count, sleep statistics, and location - was based on their importance to elderly care:

1. **Heart Rate:** By integrating a live heart rate monitor into our system, we anticipated it would supply valuable insights into possible cardiac complications or other health issues, such as stress.
2. **Steps:** Tracking daily step counts was perceived as a means to provide a detailed understanding of physical activity levels. This not only aids us in ensuring our users uphold a healthy lifestyle but also allows the detection of significant changes that may hint at developing health problems.
3. **Sleep Data:** With the significance of sleep for overall health in mind, we integrated sleep data tracking. Keeping a close eye on sleep patterns alerts us to deviations that indicate potential health issues critical in elder care.
4. **Location:** For the sake of user safety, we incorporated location tracking. This functionality allows us to ensure that users remain within a predefined safe area, sending notifications to caregivers if deviations occur.

Using Fitbit APIs, our system can capture this data and relay it to caregivers via our application. This methodology closely aligns with our project's main goal and sets a solid foundation for the future incorporation of AI/Machine Learning techniques for increasingly personalised and predictive healthcare solutions.

Knowing that caregivers often oversee multiple individuals, we configured our application to enable the simultaneous monitoring of multiple patients. This feature dramatically improves caregiving efficiency by reducing caregivers' time on individual patient check-ups.

By aggregating this data into a user-friendly platform, we've drastically eased the burden on caregivers. They can now promptly access critical health and safety information about their charges, facilitating swift responses when necessary, along with alerts triggered by heart rate abnormalities.

We have achieved our initial goal of alleviating caregiver workload by offering a streamlined, efficient, and anticipatory method of monitoring health metrics, enabling them to devote more attention to personalised care and less to constant manual health tracking. Additionally, we've established the groundwork for a monitoring system that can readily accommodate further enhancements and sophisticated features in the future.

System testing process:

Due to the time limitations on our project and the initial learning curve with Flutter, which resulted in a lack of proper dependency injection in our classes and made them more difficult, or even impossible, to test, our GitHub Actions pipeline did not include unit, integration, or end-to-end tests. Furthermore, considering the project's tight schedule and uncertainty about its future continuation, we prioritised feature development more than automated testing.

The chief function of GitHub Actions in our project was to aid the application's build process. We had two pipelines running: one for the mobile application front-end and the other for the Fitbit wearable application. Successful execution of these pipelines indicated that the code was compiled without any errors and that the necessary secret keys were present and correctly integrated into our application.

Each pull request required further approval from at least one other team member. This measure ensured that a minimum of two team members understood the source code, and the same number had thoroughly reviewed the feature branch and manually validated the code before its approval.

Therefore, our approach to testing was primarily proactive in mitigating risks. In the grand scheme, the functionality of our "main" production branch was preserved through our GitHub actions pipeline and our pull request approval process.

Positive Aspects of our System:*Constant System Deployment*

The use of lambda functions guarantees the constant deployment of our system. Every modification directly affects production, eliminating the need to spin up separate EC2 instances.

Maintainability of Mobile Application

We employ linters and a logical folder structure in the front-end that streamlines development. Each folder represents a screen with its associated widgets and user interface. Moreover, there's an abstraction concerning how the mobile application retrieves data. The back-end and front-end are sufficiently separated such that switching the front-end framework from Flutter to another language would be straightforward, necessitating only the querying of the REST endpoints.

Negative Aspects of our System:*Challenges with Automated Testing*

Our system fundamentally acts as a binder for various services. The AWS back-end lacks testing since it's hosted on AWS. The task of automated testing for our Fitbit application presents challenges due to inadequate documentation from Fitbit and our heavy reliance on external services, making testing difficult.

Insufficient Back-end Maintainability

We contemplated setting up an automated AWS build and test process employing Serverless Application Model (SAM), CodeBuild, and CodeDeploy. However, considering our restricted access to IAM and the significant resources required, we opted against it. As a compensatory action, we have backed up all our lambda functions in a folder labelled "back-end" on our GitHub repository.

For instance, our patient database continues to carry the prototype name "TestingDatabaseFitbit", with its partition key remaining as "UserId" instead of the intended "PatientId". Such naming inconsistencies would indicate potential areas for enhancement in our database design and management, mainly if this project were to be extended.

Additionally, as all our back-end code is hosted on AWS, we lack an automated testing framework to verify the accuracy of the queries.

Future Work

Existing issues and potential areas of improvement:

During our project development, we came across several features we couldn't incorporate within the existing system. This section outlines such features that pose potential improvements for future iterations.

Our first unaddressed feature concerns the Integration with RDS. While working with the AWS-backed solution Cognito, which manages user authentication and authorization, we encountered a stumbling block when attempting to associate patient IDs with the user object. To circumvent this, we duplicated a Cognito role into our RDS system, permitting us to initialise the patient IDs as an empty list. However, an ideal solution would enable direct operations from Cognito without data duplication.

Next is the Implementation of CRUD Operations. Initially, we planned to employ lambda functions to handle CRUD operations for patients within our RDS system. The downside to this method is the lack of transparency in our back end for future maintenance and documentation since these operations are hosted on AWS, not within our repository. A solution we contemplated involved using Serverless Application Model (SAM), CodeBuild, and CodeDeploy for an automated AWS build and test process. Nonetheless, the significant resource requirements and limited IAM access led us to forgo this plan. As a compensatory measure, we've stored backups of all our lambda functions in a "back-end" folder within our GitHub repository.

The third feature we couldn't address was the Synchronisation between Mobile Applications and the Fitbit Smartwatch. The current communication design between the mobile application and the Fitbit smartwatch follows a publisher-subscriber model. However, this design may require a mechanism to alter the patient's ID in the future. If a caregiver previously had access to the patient's ID, there aren't any established security measures to prevent them from subscribing to the patient's updates again. We intend to delve deeper into this issue in the forthcoming future work section.

The final feature we could not incorporate pertains to the functionality of push notifications. Our efforts to set up Firebase and make the push notifications operational faced numerous technical challenges, which unfortunately could not be resolved in this iteration.

Future enhancements - Front-end:

Patient ID Change Feature

We plan to introduce a feature in the application that allows modifying patient IDs. The current situation is such that the patient ID is hard-coded and device-tied, potentially presenting future security concerns.

Geo-Alerts

We intend to augment user safety by creating a feature that allows caregivers to establish a specified geographic range for the user. This location-tracking system will notify the caregiver whenever the user moves outside the defined area. This feature is especially advantageous for caregivers managing users with dementia, where wandering is common.

Improvements in Usability

The upcoming enhancements are focused on maximising the user-friendliness of our application, which include:

1. **Handling Empty State:** Our goal is to create specialised screens for caregivers with no active subscriptions. This will offer an engaging user interface and provide clear guidance for new users of our application.
2. **Subscription Flow Refinement:** In our quest to make the subscription process more intuitive, we plan to enhance the user interface of our subscription portal. Patients will only appear on the subscription portal once they have confirmed the subscription, providing a seamless experience.

Communication Channels Extension

We plan to embed different communication mediums into our application. These could include methods like SMS or Firebase push notifications.

iOS Compatibility Checks

We intend to review the iOS application and implement any necessary alterations to guarantee compatibility and make our application more accessible.

Future enhancements - back-end:

Customised Health Insights and Analysis

We aim to provide personalised health insights by harnessing machine learning (ML). We plan to incorporate AWS Sagemaker into our workflow to analyse the health data stored in our DynamoDB tables. This will allow us to identify the unique health patterns of individual users, enabling our application to be suitable for large-scale industrial usage.

Notification System Enhancements

Our application can detect irregular heart rate irregularities and email the associated caregiver. However, we intend to develop and diversify this feature in several ways:

1. **Inclusion of Diverse Health Metrics:** We plan to integrate a broader range of health metrics into our anomaly detection system, facilitating more holistic health monitoring of users. Examples include notifications relating to abnormal number of steps taken at the last minute or whether the patient is awake when they shouldn't be.
2. **Integration of Machine Learning:** By adding a machine learning workflow, our system will be capable of sending out more accurate and timely alerts, thereby improving the reliability of our notifications.

Conclusion

Project Aims

The LiveSavers project was conceived to develop a wearable application to address the challenges posed by staff shortages in elderly dementia care. Our primary objective is to mitigate the stress caused by this caregiver shortage by enhancing the efficiency of patient monitoring. Our application is designed to collect real-time health data from dementia patients via a wearable device and send alerts to their caregivers when the health data signifies a need for attention. This alert system is aimed at easing the caregivers' workload and improving the autonomy of dementia patients.

Key Findings

We made several key findings that significantly influenced our approach throughout our project. These include adopting a Kanban approach, choosing Fitbit as our wearable device, employing lambda functions, developing an alert system, and incorporating CRUD architecture in the mobile app.

Initially, we used the SCRUM framework methodology for scheduling, but we transitioned to the Kanban methodology due to escalating workloads from other courses and external commitments. This shift was made as Kanban's agile and flexible nature enabled a more fluid team workflow.

In our search for suitable wearable devices to capture patient health metrics, we considered brands like Apple, Samsung, Garmin, and Huawei. Consequently, we opted for Fitbit, whose open and developer-friendly APIs allowed us to extract the needed health metrics.

In our initial phases of AWS usage, we had to choose between an EC2 approach or Lambda functions. We favoured Lambda functions, as our application was designed primarily for data relaying rather than intensive processing tasks. Additionally, concerns about unexpected costs from maintaining EC2 continuously active made Lambda functions a more budget-friendly choice.

While developing the alert system, we encountered several challenges leading to crucial findings. Initial attempts at creating Push Notifications ran into issues with Firebase and the necessary keys. Similar challenges arose with SMS notifications, leading us to adopt email alerts, a solution our clients prefer.

Major Outcomes and their Significance

We successfully developed and deployed a fully operational app on both Fitbit and a mobile platform, although yet to be suitable for real-world use. We implemented two-factor authentication on the mobile app, a crucial step considering the sensitivity of health data. The app displays patients' health data remotely, significantly easing the task of caregivers in aged care, enabling them to check on their charges' well-being remotely. We also added a map feature showing a patient's precise location, a critical tool for caregivers if a patient is lost or needs immediate assistance, allowing them to reach the patient quicker. The alert functionality was successfully incorporated, with future iterations set to refine further and enhance this system. This feature reassures caregivers of timely awareness of patients requiring urgent care. Given these accomplishments, our LifeSavers project has made considerable strides in developing wearable technology to alleviate some of the stress associated with understaffed elderly care and dealing with dementia.

References

Careerforce.org. (n.d.). A shortage in care and support services is looming as the New Zealand population ages. Retrieved from <https://www.careerforce.org.nz>

RNZ. (n.d.). Staff shortages mean 30,000 dementia patients missing vital care advocate. Retrieved from <https://www.rnz.co.nz/news/national/476773/staff-shortages-mean-30-000-dementia-patients-missing-vital-care-advocate>

Alzheimer's Association. (2019). Stages and Behaviors. Alzheimer's Disease and Dementia. <https://www.alz.org/help-support/caregiving/stages-behaviors>

Moyle, W., Jones, C., Murfield, J., Thalib, L., Beattie, E., Shum, D., O'Dwyer, S., Mervin, M. C., & Draper, B. (2018). Effect of a robotic seal on the motor activity and sleep patterns of older people with dementia, as measured by wearable technology: A cluster-randomised controlled trial. *Maturitas*, 110, 10–17. <https://doi.org/10.1016/j.maturitas.2018.01.007>

Staff shortages mean 30,000 dementia patients missing vital care - advocate. (2022, October 16). RNZ <https://www.rnz.co.nz/news/national/476773/staff-shortages-mean-30-000-dementia-patients-missing-vital-care-advocate>

Stress. (2022). Bhf.org.uk. <https://www.bhf.org.uk/information-support/risk-factors/stress#:~:text=When%20you%27re%20stressed%20your>

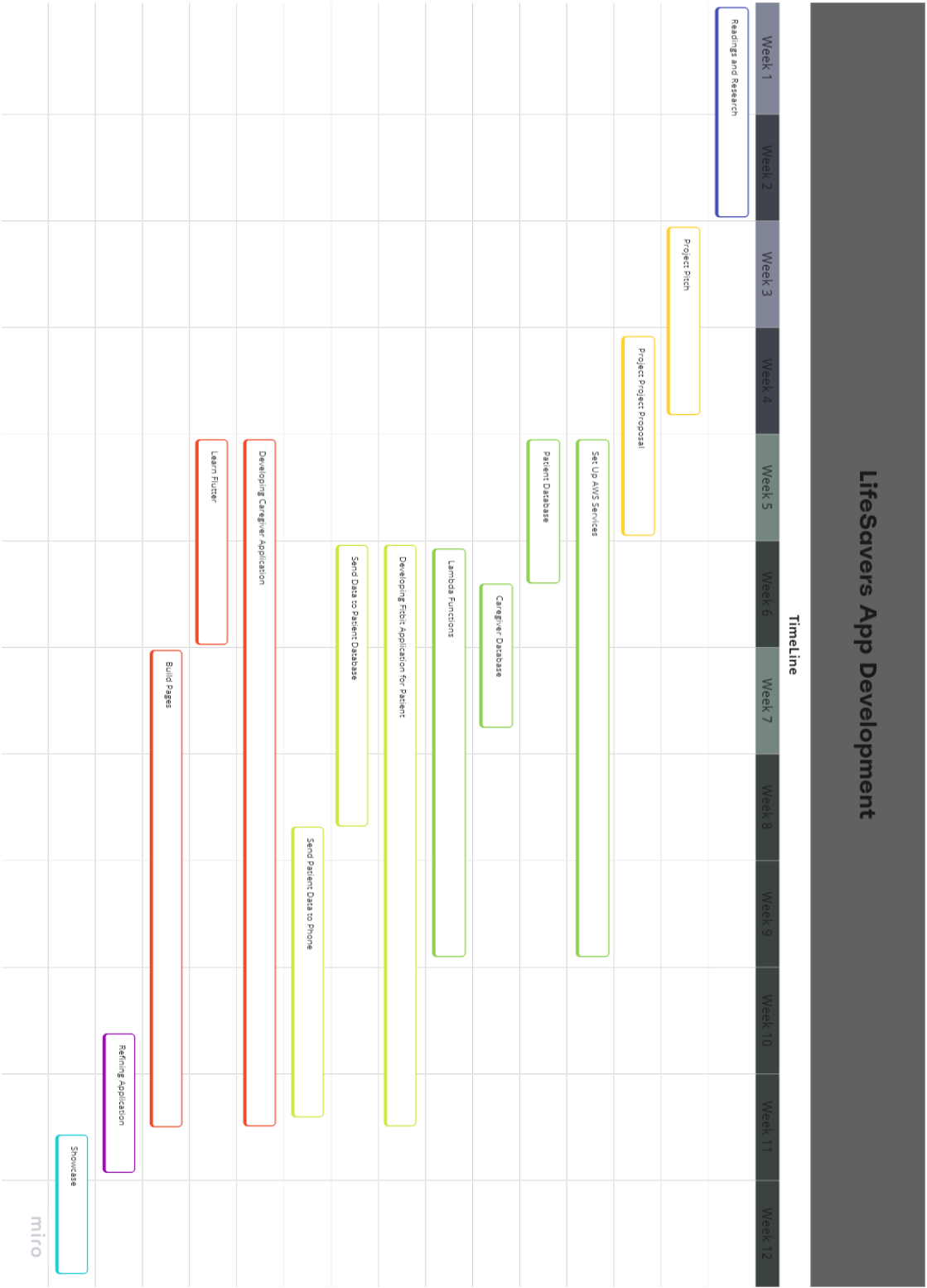
British Heart Foundation. (n.d.). Stress. Retrieved from <https://www.bhf.org.uk/information-support/risk-factors/stress#:~:text=When%20you're%20stressed%20your,should%20go%20back%20to%20normal.>

Alzheimer's Association. (2019). Stages and Behaviors. Alzheimer's Disease and Dementia. <https://www.alz.org/help-support/caregiving/stages-behaviors>

Smart Watches New Zealand.(2023). Elderly GPS Watch AWG-A19 <https://awesomemegadgets.nz/smartwatchesnz/product/awg-a19-elderly-gps-watch/>

Appendices

A. Gantt Chart



B. Link to final build

[Github Repository Final Build](#)

C. Link to video

[Video](#)

Declaration of Authorship

Section	Author(s)	Signature(s)
Title Page	Danarta Sanyata	D.S
Executive Summary	Danarta Sanyata HaoChen Li	D.S H.L
Table of Content	Danarta Sanyata	D.S
Introduction	Rajat Kumar Lucy Baldwin	R.K L.B
Background	Mshari Alajaji Lucy Baldwin	M.A L.B
Specification	HaoChen Li Lucy Baldwin	H.L L.B
Design	HaoChen Li	H.L
Implementation	HaoChen Li	H.L
Result and Evaluation	Rajat Kumar Mshari Alajaji HaoChen Li	R.K M.A H.L
Future Work	Xu Jiachen HaoChen Li Danarta Sanyata	J.X H.L D.S
Conclusion	Lucy Baldwin Danarta Sanyata	L.B D.S
Appendices	Danarta Sanyata	D.S