



# HEART FAILURE READMISSION APPLICATION



Edward Storey, Sai Anoop Avunuri, Jaehoon Song, Jacob Rogers, Michael Fu  
GEORGIA TECH DETAILED DESIGN DOCUMENT, CS-3311/3312-JIE-3342

## Table of Contents

Table of Figures.....	3
Terminology.....	4
Introduction.....	6
Document Summary .....	6
System Architecture.....	7
Architecture Diagram (Static).....	7
Use Case Diagram.....	9
System Sequence Diagram (Dynamic).....	12
Data Storage Design.....	14
ER Diagram .....	14
File Usage .....	14
Data Exchange .....	15
Security .....	15
Component Detailed Design .....	17
Component Diagram (Static) .....	17
Interaction Overview Diagram (Dynamic).....	19
UI Design .....	21
Mobile (Flutter Application).....	21
Web (Flask Application) .....	24
Appendix A: REST API Documentation.....	27
Response Codes .....	27
Request-Response Param and Descriptions (Mobile) .....	28
POST /submit_survey .....	29
POST /login.....	31
GET /get_weight.....	33
Request-Response Param and Descriptions (Web) .....	34
GET/staff_logout.....	35
GET/ .....	36
POST/staff_login .....	37
GET/staff_dashboard .....	38
POST/staff_create_patient .....	39
GET/staff_add_patient .....	40

POST/staff_remove_patient.....	41
POST/edit_patient.....	42
POST/send_notifications.....	43
GET/detail .....	44
Appendix B: Team Collaboration Roles .....	45

## Table of Figures

Figure 1: Architecture Diagram by Layers .....	7
Figure 2: Architecture Diagram by Tiers .....	8
Figure 3: Use Case Diagram.....	10
Figure 4: System Sequence Diagram .....	13
Figure 5: ER Diagram .....	14
Figure 6: Component Diagram .....	17
Figure 7: Interaction Overview Diagram (Moblie).....	19
Figure 8: Interaction Overview Diagram (Web) .....	20
Figure 9: Mobile UI Theme .....	21
Figure 10: Mobile UI Login.....	22
Figure 11: Mobile UI Questionnaire .....	22
Figure 12: Mobile UI Dashboard .....	23
Figure 13: Web UI Login .....	24
Figure 14: Web UI Dashboard .....	24
Figure 15: Web UI Risk Categories .....	25
Figure 16: Web UI Statistical Analysis .....	26

# Terminology

This dictionary contains terms and their definitions, covering both technical aspects such as the components used in our application and domain-specific terms related to our specific challenges and focus area (healthcare management). It's a crucial resource for understanding the components and concepts used throughout this document, providing clarity on both the inner workings of our application and the unique problems we address.

<b>3-Tier and 3-Layer Architecture</b>	An architectural model dividing functionalities into three layers
<b>Composite Primary Key (CPK)</b>	A primary key consisting of more than one attribute, often used in junction tables to ensure uniqueness.
<b>Entity Relationship Diagram (ERD)</b>	A visual representation of entities and their relationships in a database, providing insights into the structure and organization of data.
<b>Flask, Flutter, Derby</b>	Technologies used in the project - Flask (Python web framework), Flutter (Dart-based mobile application framework), and Derby (Java database).
<b>HTTP Protocol and SSL/TLS Protocol</b>	Protocols governing communication between clients and servers, with SSL/TLS providing secure data transmission.
<b>HTTPS</b>	A secure version of HTTP (Hypertext Transfer Protocol) using encryption for secure communication.
<b>Junction Table</b>	A table resolving many-to-many relationships by containing foreign keys from related entities.
<b>JWT Secret Key</b>	A secret key used in JSON Web Tokens (JWT) for secure data storage and protection.
<b>Microservices Architecture</b>	A design approach where applications are built as a collection of small, independent services, promoting agility, scalability, and resilience.
<b>Model-View-Controller (MVC)</b>	A design pattern dividing an application into three interconnected components - Model (backend logic and data), View (presentation layer), and Controller (mediates between Model and View).
<b>Relational Database Management System (RDBMS)</b>	A database management system that uses a relational model for storing and retrieving data.
<b>Socket Connection and TCP/IP Protocols</b>	A communication method between the Flask server and Derby database, utilizing standard TCP/IP protocols over a socket connection.

<b>SQL Injection</b>	A security vulnerability where malicious SQL code is inserted into input fields, often mitigated by using parameterized queries.
<b>Third Normal Form (3NF)</b>	A database normalization technique eliminating transitive dependencies, ensuring efficient organization and minimal redundancy.

# Introduction

In response to the growing need for effective and efficient healthcare management, our project focuses on developing a comprehensive system aimed at preventing readmissions related to heart failure. This initiative is driven by a commitment to enhance patient care, streamline medical processes, and provide valuable tools for medical staff. The project adopts the Model-View-Controller (MVC) architecture pattern, dividing functionalities into frontend presentation (View), backend logic and data management (Model), and intermediary processing (Controller).

The primary goal of our system is to empower patients and medical professionals through a platform-independent solution that seamlessly operates across various devices and platforms. This is achieved through the utilization of well-defined components, including a mobile application built with Flutter, a database powered by Apache Derby, and a versatile server facilitated by Flask.

## Document Summary

The Detailed Design document goes into the details of how the system is set up in the System Architecture section showing a picture and a moving view of how everything fits together. This includes clear diagrams that explain how the View, Model, and Controller components connect.

Moreover, the document provides insights into the Data Storage Design, outlining the structure of the database and offering an Entity Relationship Diagram (ERD) for a comprehensive understanding. It underscores the significance of file usage and data exchange, ensuring organizational clarity and compliance with industry standards. Security considerations are central to our design, addressing potential vulnerabilities and outlining measures taken to protect patient information.

Furthermore, the Component Detailed Design section goes into more detail about how everything is set up, explaining the roles and interactions of each component in the 3-Tier and 3-Layer architecture models as well as wraps up with an Interaction Overview Diagram, which gives a big-picture view of how users and the system interact in real-time. This diagram shows how information flows and how different parts work together to make sure users have a smooth experience.

In summary, this Detailed Design document serves as a comprehensive guide to the architecture, functionality, and interactions within our healthcare management system. It aims to provide a clear understanding of the project's structure and design principles, laying the foundation for the successful implementation and deployment of a platform that significantly contributes to improved patient care and healthcare processes.

# System Architecture

This project incorporates a system aimed at preventing readmissions related to heart failure, employing the Model-View-Controller (MVC) architecture pattern. The followings are the roles within this framework:

1. **View (Frontend):** This component functions as the presentation layer, interacting with the user interface (UI). Its primary responsibilities include displaying information to the user and capturing user input.
2. **Model (Backend):** The backend, known as the Model, encompasses both the application's data and its business logic.
3. **Controller (Backend):** Serving as an intermediary between the Model and the View. It receives user input from the View, processes it, potentially interacting with the Model, and updates the View accordingly.

This architecture is alternatively referred to as the 3-Tier and 3-Layer Architecture (or Client-Server-Database Architecture), dividing functionalities into UI, Business Logic, and Data layers. in certain contexts.

## Architecture Diagram (Static)

Here is a comprehensive view for 3 layers.

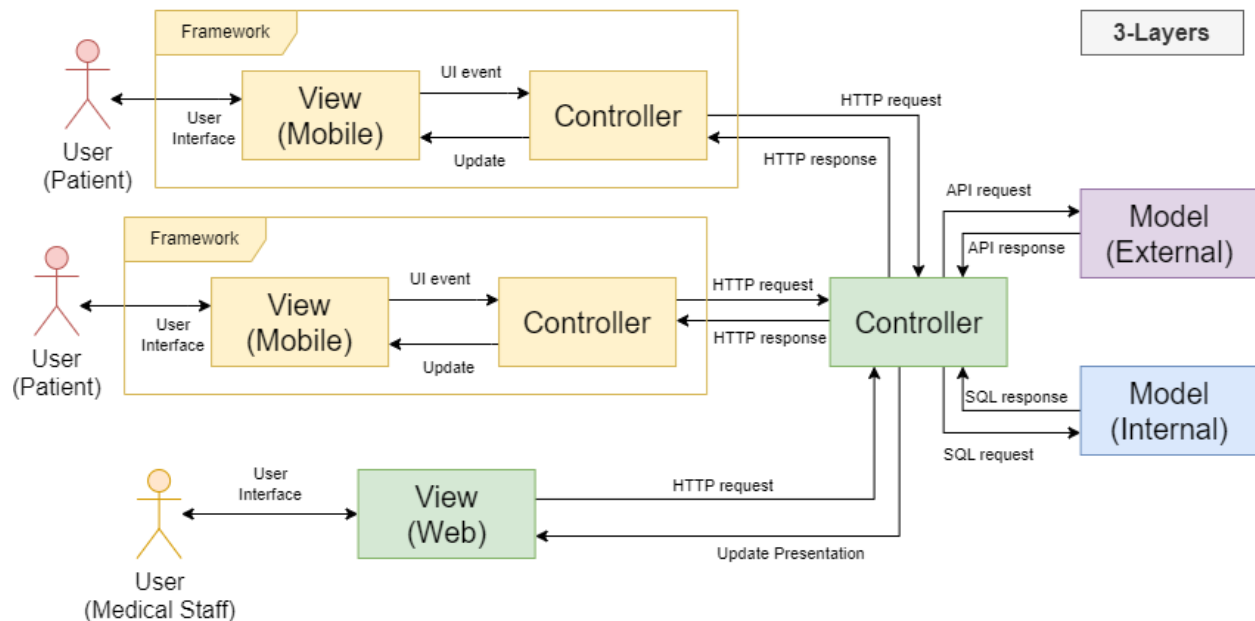


Figure 1: Architecture Diagram by Layers



Also, here is a comprehensive view for 3 tiers (physical separations of each).

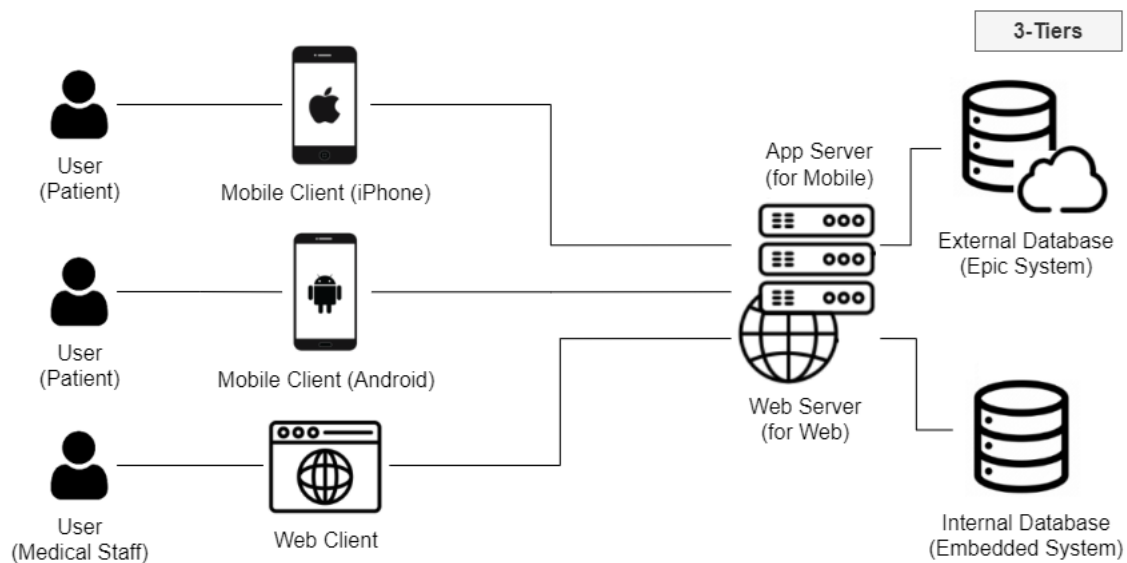


Figure 2: Architecture Diagram by Tiers

By the Model-View-Controller (MVC) architecture pattern, the system is essentially platform-independent, operating seamlessly across various platforms. This is achieved through the following components:

- **Frontend Application (Client: Flutter by Dart):** The client side is compatible with both iOS and Android operating systems, as well as their respective hardware platforms. It is a mobile application.
- **Database (Database: Apache Derby by Java):** The database component manages the storage and retrieval of data. It is a command line application.
- **Application Server / Web Server (Server: Flask by Python):** The server side handles the business logic and serves as an intermediary between the frontend and the database. At the same time, it presents data to the user through HTML templates. It is also a command line application.

**Note:** This system can also be treated as microservices architecture since applications are built as a collection of small, independent services (frontend, server, and database). Each service focuses on a specific function, allowing for agility, scalability, and resilience.

Specifically, in the 3-Tier and 3-Layer architecture models, our system manages the UI (frontend: client), the application server (backend: server), and the microservice managing data (backend: database). The user interface on the front end serves as the starting point for most of the control flow. Additionally, notifications are sent to users and permissions are also obtained from users. Furthermore, the client application is utilized to store tokens locally and to communicate with the server application. we employ built-in URL and send POST requests with headers containing JWT tokens.

Patients use this interface to input their daily conditions into the application, including information such as weight, blood pressure, and responses to questionnaires. These entries are stored in the

application's database. The structure of the database will be further investigated in the following "Data Storage Design" section. Additionally, the server validates this input by cross-referencing it with information from the EPIC system, which is retrieved when initializing the system, ensuring accurate matching of patient details.

## Use Case Diagram

A visual representation of user interactions with a system to achieve specific goals is provided by the use case diagram below. In this diagram, users, also known as actors, are depicted engaging with the system to accomplish various use cases. Each use case has its own dynamic architecture design, associated with SD.

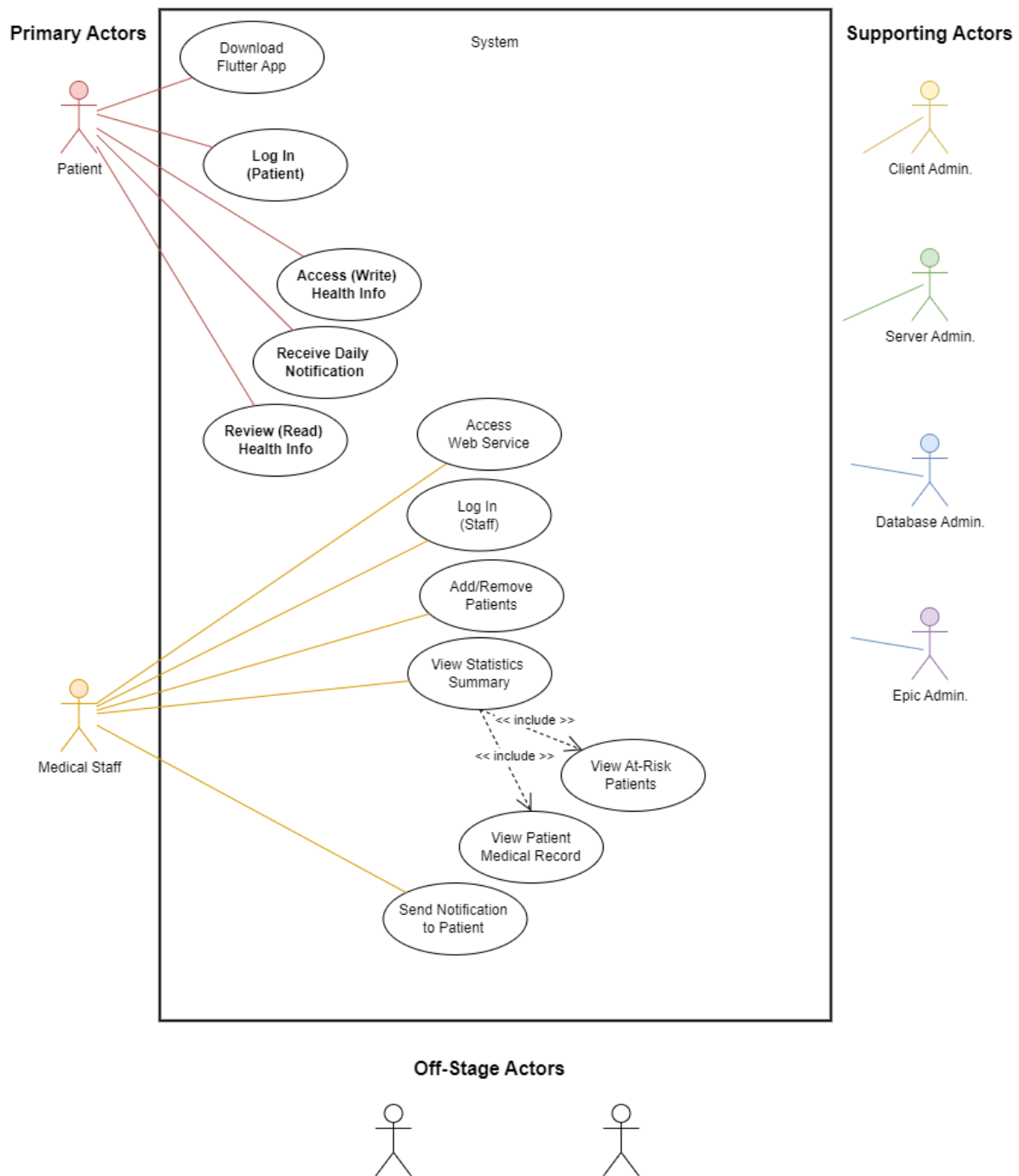


Figure 3: Use Case Diagram

**Patient Experience:**

1. As a patient, I want to log in conveniently using my Date of Birth, Full Name, and Medical Record Number (MRN) retrieved from the system to access my medical records and information.
2. As a patient, I want to easily access and enter information about my condition to make informed decisions about my health.
3. As a patient, I want to receive daily notifications to ensure I don't forget to record my weight.
4. As a patient, I want to review my previous weight/blood pressure responses on a visual timeline.

**Dashboard for Medical Staff:**

5. As a medical staff member, I want to log in to the website to view patient data.
6. As a medical staff member, I want to add new patients and remove old/deceased patients from my list to keep it up to date. [important: medical staff will be able to enter epic data manually]
7. As a medical staff member, I want to look at a patient's specific medical record to assess their risk.

**Patient Info on Dashboard:**

8. As a medical staff member, I want to see special highlighting for at-risk patients based on statistical models, allowing me to check their record and verify the need for extra support.
9. As a medical staff member, I want to view summary statistics for my patients to identify and contact those at risk.
10. As a medical staff member, I want to send a notification to the patient via the website to remind them to take their weight or schedule an appointment.

**Hosting & Deploying:**

11. As a user, I want to download the Flutter app from the App Store/Google Play Store.
12. As a medical staff member, I want to access our website hosted on the public Internet.
13. As a user, I want my data to be hosted on a secure cloud platform. (important: physically and logically secure local database in one of the rooms in a hospital will be the safest)

**Security Quality Debugging:**

14. As a patient, I want assurance that my data and information are secure and encrypted where necessary.
15. As a patient, I want to navigate the application without any unexpected bugs or glitches, regardless of the supported platform or device I am using.

**Note:** the color assigned to each component in static diagrams corresponds to its representation in the dynamic diagrams.

## System Sequence Diagram (Dynamic)

The Sequence Diagrams presented as dynamic architecture designs illustrate the sequence of events and interactions between system components for their respective use cases. These diagrams capture the exchange of information as users engage with the system, providing insights into the system's functionality. The following primary use cases will be included in the system sequence diagram.

- **Use Case 1: Log In (Patient)**
  - In this scenario, patients conveniently log in using their Date of Birth, Full Name, and Medical Record Number (MRN) for secure access to medical records. This ensures privacy and streamlines retrieval of health details.
- **Use Case 2: Access and Enter Health Info**
  - In this scenario, patients easily access crucial health information to make informed decisions. The user-friendly interface facilitates health information prompt with navigation of treatment plans and updates.
- **Use Case 3: Receive Daily Notification**
  - In this scenario, patients receive timely reminders to record vital health metrics, promoting adherence to medical recommendations and enhancing healthcare monitoring.
- **Use Case 4: Review Health Info**
  - In this scenario, patients track progress through a visual timeline of previous weight and blood pressure responses, fostering collaboration with healthcare professionals.

The diagram is provided on the page below.

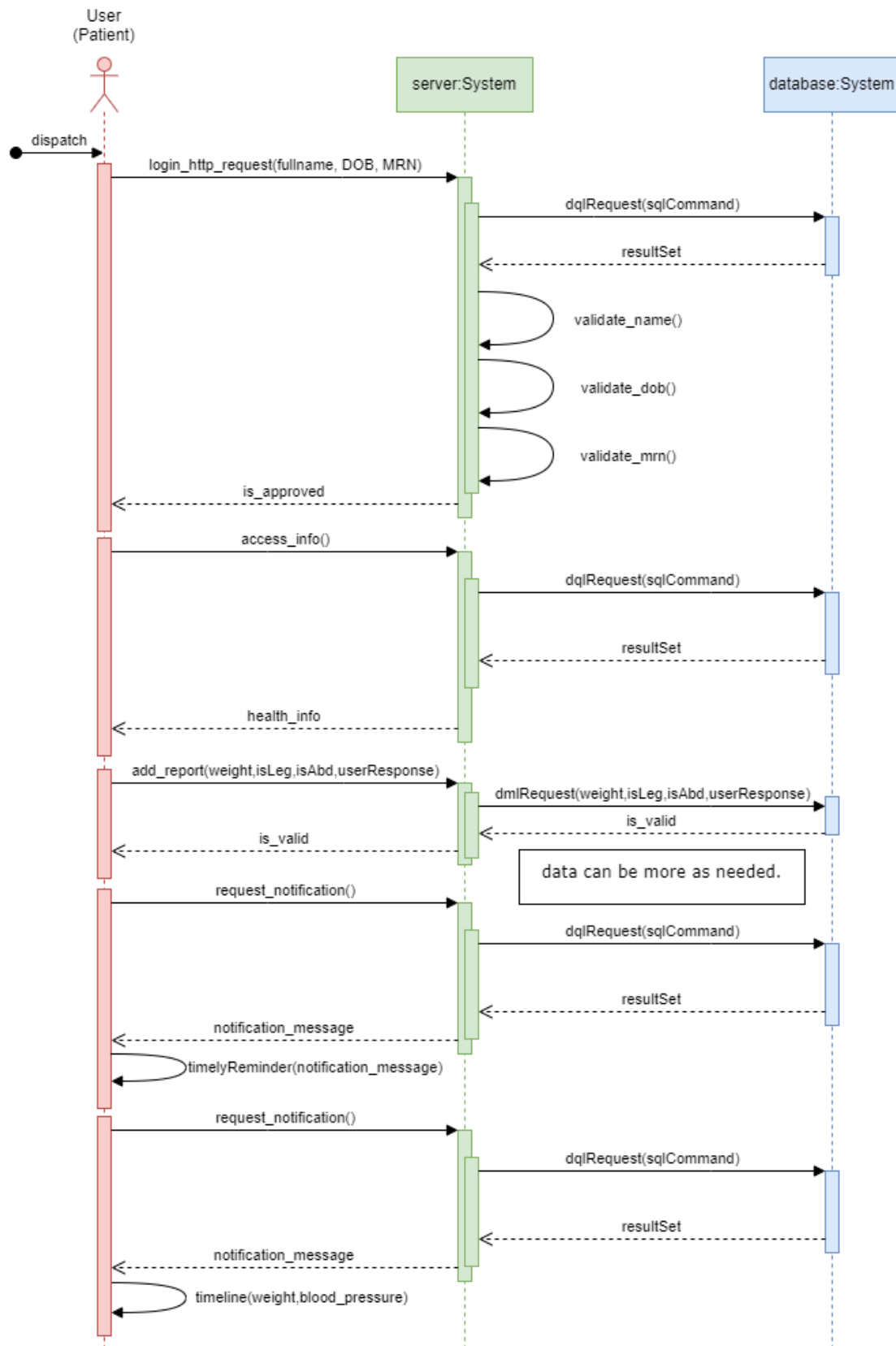


Figure 4: System Sequence Diagram

# Data Storage Design

Here are the details of the database's structure referenced in the section, "Architecture Diagram (Static)". The current project utilizes a customized Relational Database Management System (RDBMS). The database bridges the gap between SQL-based RDBMS and NoSQL paradigms, offering users the familiarity of SQL alongside the flexibility of JSON and the convenience of HTTP methods for data manipulation. The following Entity Relationship Diagram (ERD) provides insight into the system's architecture, deploying the entities (data) to be stored in the database.

## ER Diagram

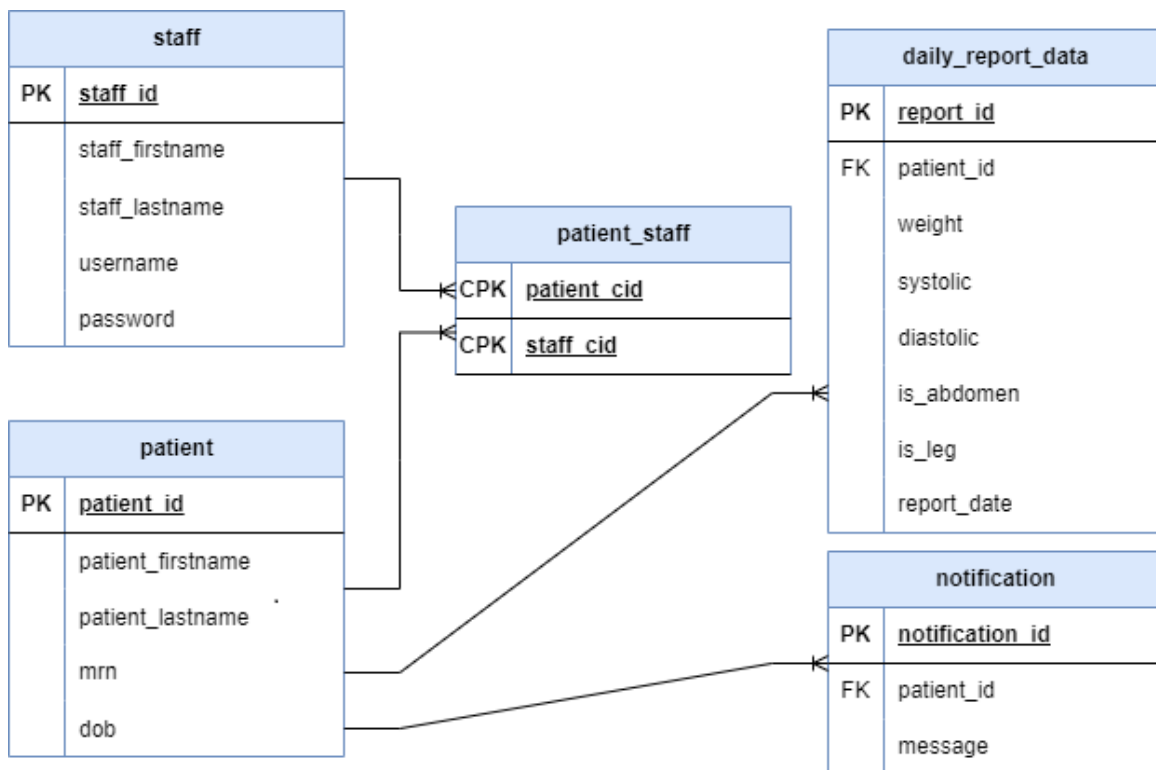


Figure 5: ER Diagram

**Note:** Third Normal Form (3NF) of RDBMS plays a crucial role in optimizing the database structure. The many-to-many relationship is resolved through a *junction table* (*StaffPatientPair*) which contains a composite primary key (CPK) comprising foreign keys from the related entities. To adhere to 3NF principles, the design ensures that non-key attributes in the junction table depend solely on the composite primary key, eliminating transitive dependencies and enhancing data organization by avoiding redundancy and maintaining database normalization.

## File Usage

In the context of file usage, it is crucial to maintain organizational clarity. On the Flask server side, the application is designed for web and API services, utilizing SQLAlchemy for database interaction with SQLite. **Template** files are used for rendering web pages and **static** files are used for styling as follows:

- templates/dashboard.html
- templates/detail.html
- templates/edit\_patient.html
- templates/index.html
- statics/styles.css
- etc....

Additionally, environment variables are stored in a **.env** file, including **JWT\_SECRET\_KEY** and **APP\_SECRET\_KEY**. Lastly, **SSL certificate** files (*cert.pem* and *key.pem*) are referenced for potential SSL activation. On the derby database side, **TSV** (Tab Separated Value) files function as repositories for table data in a relational database, facilitating both data backup and handling by medical staff. Additionally, **SQL** (Structured Query Language) files stand prepared for immediate use whenever client applications seek query data from the database.

## Data Exchange

In data exchange, it's crucial to carefully document the format in use. When a client, like our mobile application developed with Flutter, interacts with a website or a web service, the communication takes place through the **HTTP protocol** along with **SSL/TLS protocol**. The messages exchanged between the client and server strictly adhere to the rules defined by the chosen version of the HTTP protocol.

Additionally, the communication between the Flask server and the Derby database involves a **socket connection** utilizing standard **TCP/IP protocols**. Socket connections offer low-level control, allowing developers to fine-tune communication and optimize performance for specific needs. Moreover, they efficiently handle binary data transfer, unlike HTTP, which primarily deals with text-based data. This makes them ideal for applications needing to exchange raw data. Furthermore, sockets provide platform-independent abstraction for network communication, facilitating development across diverse operating systems and hardware platforms.

## Security

In the context of Flask server-side security, SQL injection vulnerabilities may arise due to raw SQL queries, and it might be expected to be replaced with Derby database service by employing parameterized queries. However, data processing using a **JWT secret key** ensures safe storage and protection with **SSL/TLS encryption** ensuring secure communication over a computer network.

Utilizing a socket connection with the TCP/IP protocol for the Derby database on localhost as a separate service is generally treated secure for internal services within the same machine since the database is configured to only accept connections from localhost and is not accessible over the internet, it reduces some security concerns. Even if the database is only accessible locally, there can still be risks within the system. Malicious insiders or compromised applications on the same server could potentially show up. For this aspect, the responsibility falls on the infrastructure team of Grady such as a firewall to permit authorized connections, particularly when handling sensitive data.



With the low-level control of networking, the database is protected against SQL injection since the communication protocol for this application has been intentionally simplified to only require a single command. This setup is designed specifically to prevent SQL injection attacks.

Lastly, Infrastructure security on AWS is managed by AWS itself, taking responsibility for securing data within the AWS environment, including encryption (both in transit and at rest). Enterprises may have specific compliance requirements (e.g., HIPAA, PCI DSS) that dictate the use of certain encryption algorithms and modes. **SSL/TLS encryption** on the server side can be optional since symmetric encryption algorithm (AES or Advanced Encryption Standard) has been applied to the database. AES in CBC (Cipher Block Chaining) mode can be a suitable choice for providing security in enterprise applications, but it's essential to consider factors such as authentication, integrity, key management, compliance requirements, and performance characteristics to ensure it meets the specific security needs of the enterprise.

# Component Detailed Design

The provided Component Diagram serves as a static illustration portraying a system along with its subsystems, noting the relationships and dependencies among components at a given point in time. Each component holds a distinct purpose within the overall system and engages solely with other crucial elements. The diagram and its description create conceptual integrity, with the Architecture Diagram (Static) section providing more details.

Interaction Overview Diagrams serve as a high-level abstraction for modeling interactions, providing an overview of the control flow within interactions. Given the system's significant reliance on frameworks and UI components, the choice was made to utilize Interaction Overview Diagrams instead of sequence diagrams. This decision is motivated by the small scalability of internal data available to describe within the context of a sequence diagram.

## Component Diagram (Static)

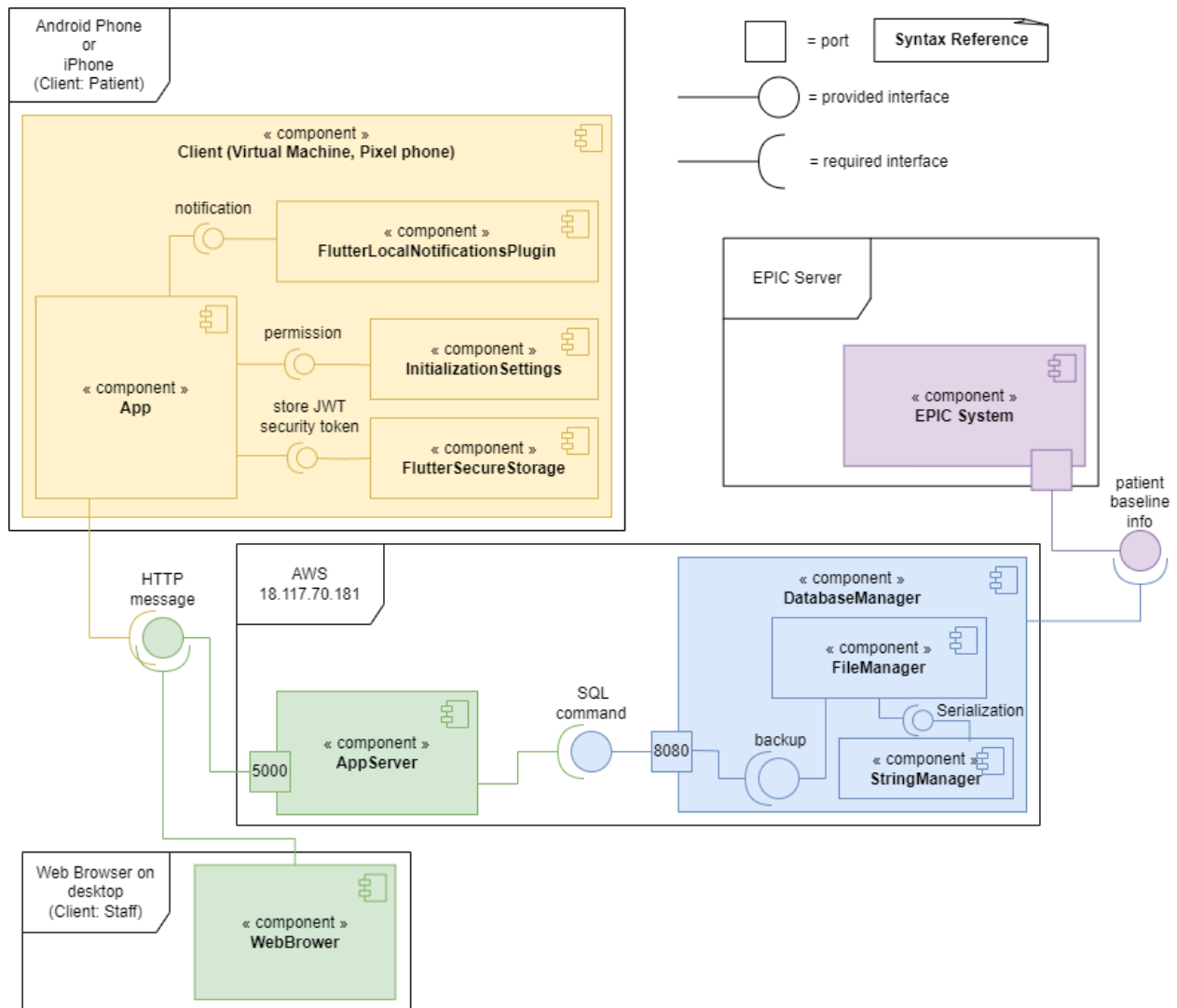


Figure 6: Component Diagram

In the 3-Tier and 3-Layer architecture models, our system manages the UI (frontend: client), the application server (backend: server), and the database (backend: database). The user interface on the front end serves as the starting point for most of the control flow.

On the client side (mobile), notifications are sent to users using *FlutterLocalNotificationsPlugin*, and Permissions are obtained from users through *DarwinInitializationSettings* and *AndroidInitializationSettings*. Furthermore, *FlutterSecureStorage* is utilized to store tokens locally and to communicate with the server application. We employ explicit URLs to send POST requests with headers containing JWT tokens. In addition, the server application operates in conjunction with the Flask web framework to provide web app services to medical staff using the web client.

For the server application infrastructure, Amazon Web Services (or AWS), providing an enterprise interface for administrators and developers to easily host secure web applications for the public, has been chosen to host the application service with the following IP address given.

18.117.70.181

**Note:** An official domain name for the service will be set by the enterprise by Grady later on.

Patients use these interfaces to input their daily conditions into the application, including information such as weight, blood pressure, and responses to questionnaires. These entries are then stored in the application's database. Once the data is sent to the database through the server, *DatabaseManager* will call *FileManager* for backup associated with the file IO of the local machine's operating system and *StringManager* to either serialize or deserialize interacted data into various string forms such as JSON, TSV, and SQL since adherence to the relational model's principles through the use of SQL is maintained, while JSON is incorporated for data retrieval in the development of a custom RDBMS. This strategic combination facilitates the integration with web services, accepting JSON's widespread adoption for data exchange over HTTP. Moreover, the system accommodates standard HTTP methods such as GET, POST, PUT, and DELETE, aligning with the CRUD (Create, Read, Update, Delete) operations typically associated with traditional databases.

## Interaction Overview Diagram (Dynamic)

The dynamic component design is evident through the detailed diagram presented below. This diagram illustrates the runtime interactions between the user and the system, showcasing how each individual component interacts.

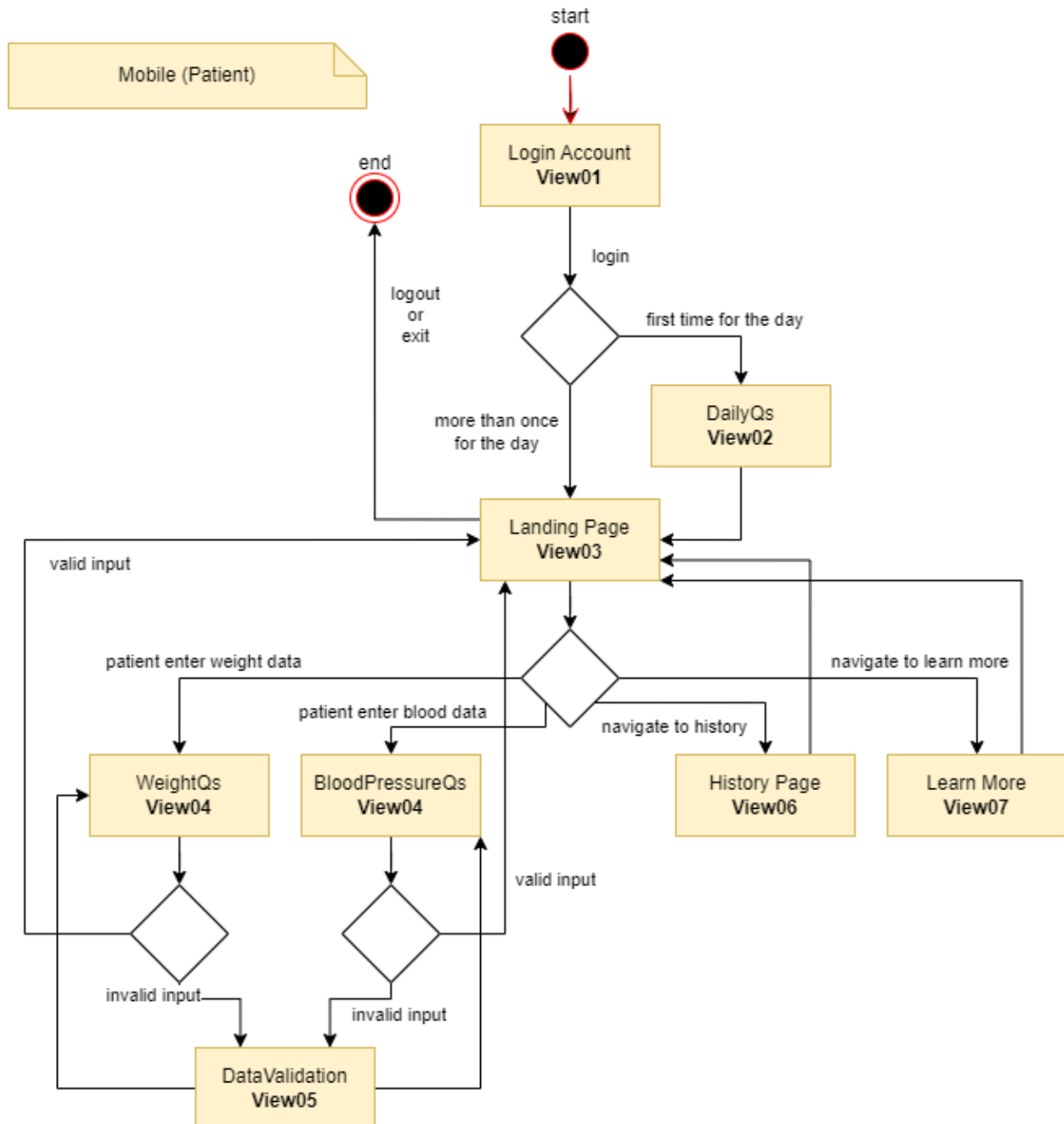


Figure 7: Interaction Overview Diagram (Mobile)

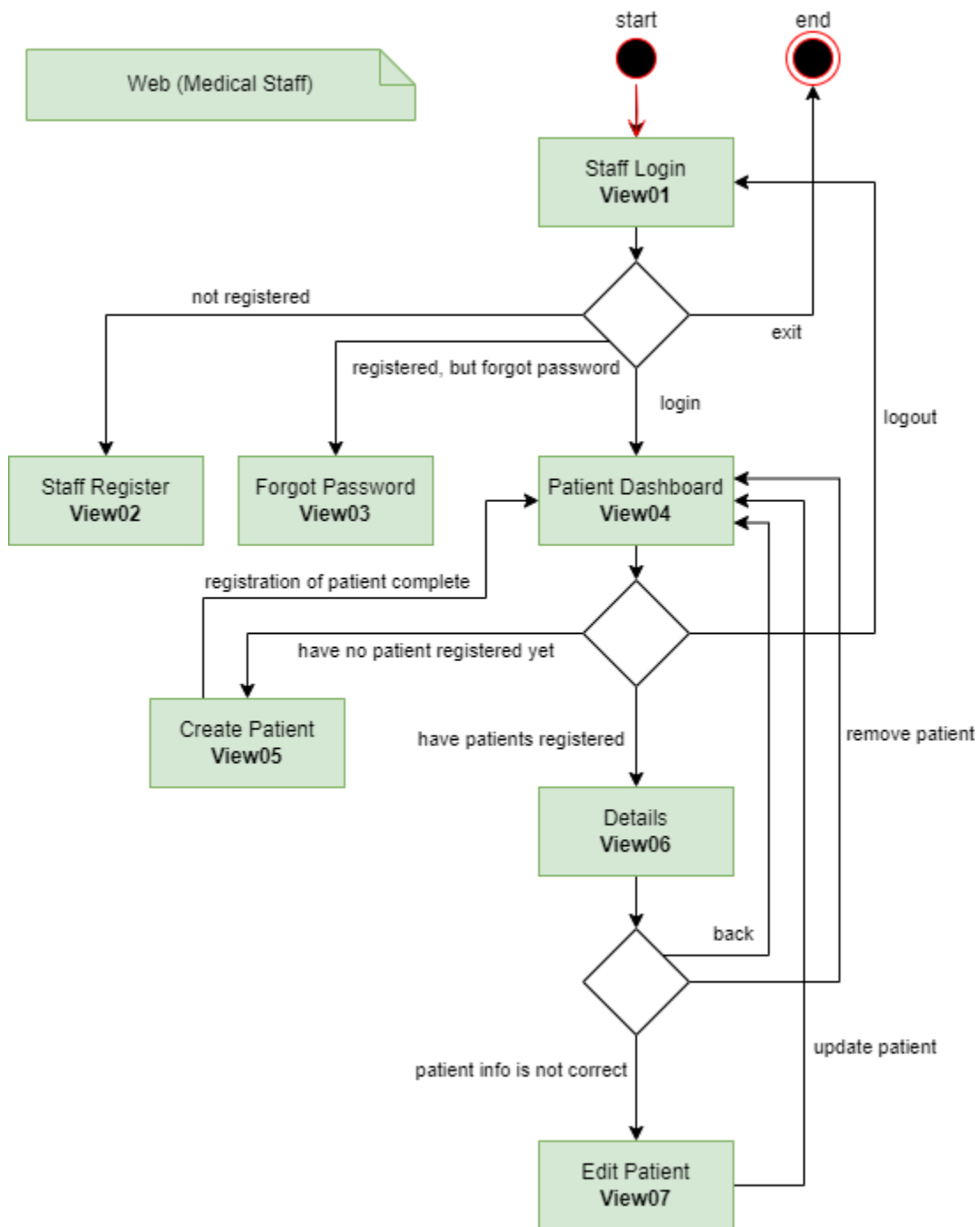


Figure 8: Interaction Overview Diagram (Web)

# UI Design

This section outlines the user interface design for the healthcare management system. It provides details on primary screens (UI) elements being developed for both mobile and web applications as well as the UI design decisions behind them.

## Mobile (Flutter Application)

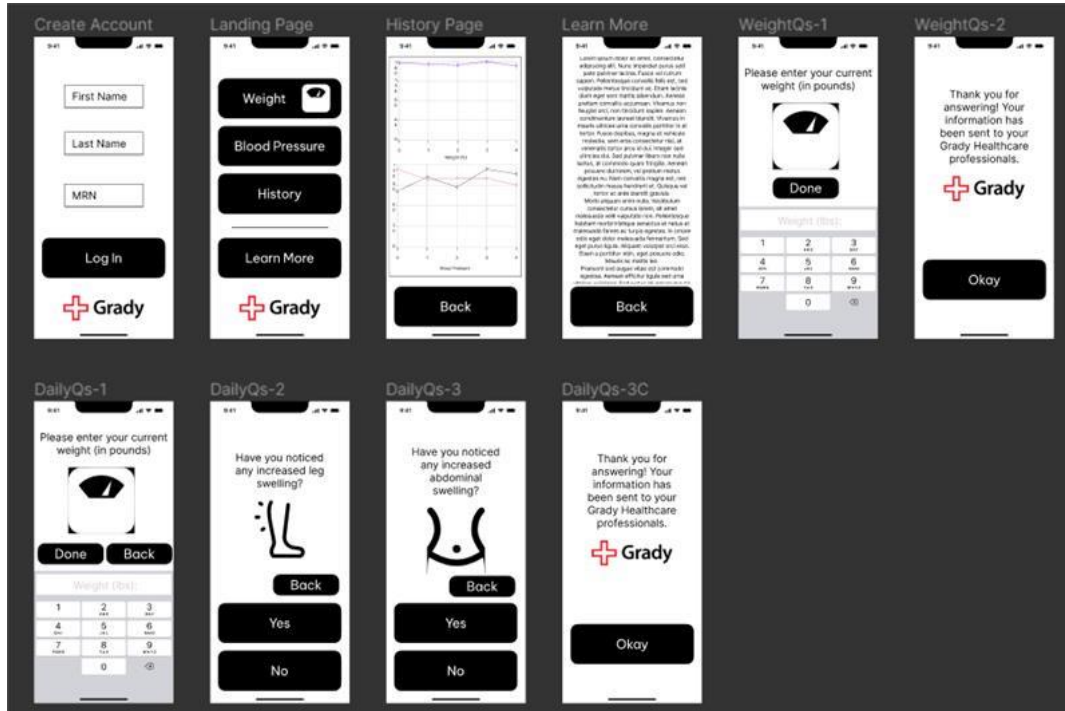


Figure 9: Mobile UI Theme

The interface design is consistent across all pages of the application. The main theme of the application (high contrast, black / white / gray) is maintained through the entire application with no jarring exceptions. So, the interface design maintains consistency throughout all pages of the application, adhering to the **Consistency and standards** principle. This ensures that users can easily navigate through different sections of the app without confusion.



Figure 10: Mobile UI Login

When the user enters their mobile application, they will be required to enter their first name, last name, and MRN to verify their patient data so they can login. As soon as they get logged in, they will be reporting their daily conditions as below.

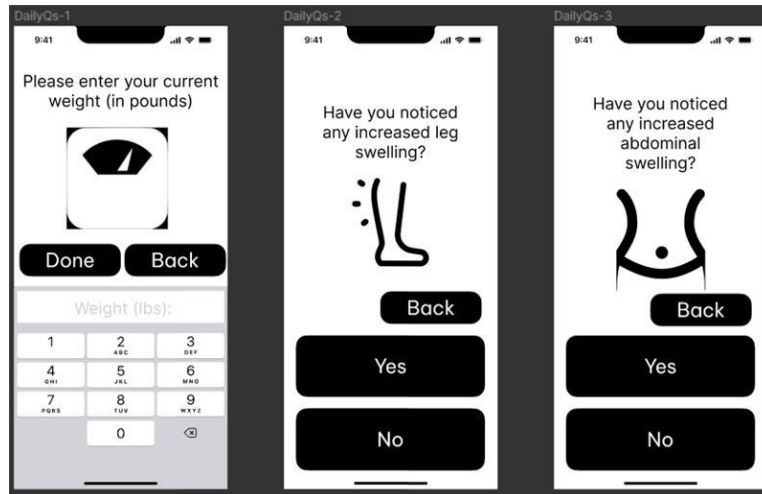
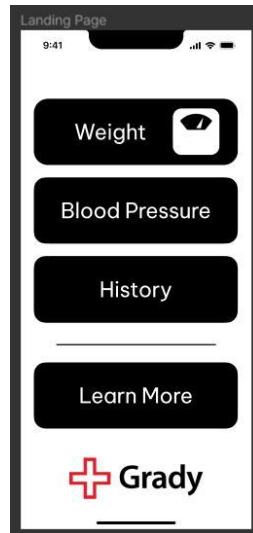


Figure 11: Mobile UI Questionnaire

In the main component of the application (the questionnaire), the future information that the patient will be asked is hidden behind several buttons, ensuring a linear procedure. This adheres to proper UI design since including all the questions on a single scrollable page would be potentially overwhelming, especially to our user demographic of low technologic literacy. By implementing progressive disclosure, the design prioritizes simplicity and ease of use, in accordance with the principle of progressive disclosure to prevent overwhelming users with too much information at once. Implementing progressive disclosure in the questionnaire section follows the **Flexibility and**

**efficiency of use** principle by simplifying the interface for users with low technological literacy, preventing overwhelming them with too much information at once.

Once they complete their daily report, they will proceed to the main board.

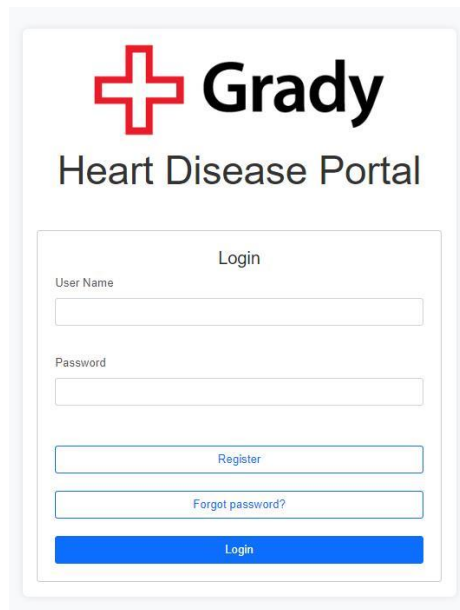


*Figure 12: Mobile UI Dashboard*

The design of this application features extremely high contrast. While this is important for all applications, it is especially important for our application due to the demographic of our users: the majority of the users will be older (55+) and underprivileged. This allows for users with lower levels of eyesight to still interact with the application, highlighting the significance of contrast in ensuring accessibility and usability for all users. This high contrast aligns with the **Match between system and the real-world** principle by considering the demographic's potential vision disabilities.



## Web (Flask Application)



The login form for the Grady Heart Disease Portal. It features a red cross logo and the text 'Grady Heart Disease Portal'. The form includes fields for 'User Name' and 'Password', a 'Register' button, a 'Forgot password?' link, and a 'Login' button.

Grady  
Heart Disease Portal

Login

User Name

Password

Register

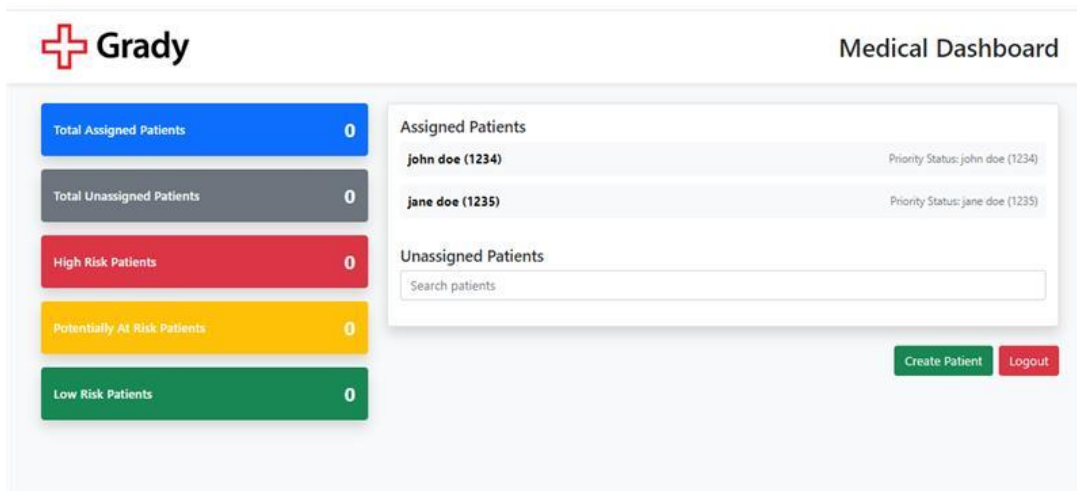
Forgot password?

Login

Figure 13: Web UI Login

The login page for the medical staff website is intuitive and familiar: it features the traditional components of a login screen in the order most found. This allows medical staff to use the software as quickly and easily as possible without having to search for any features. This intuitive and familiar layout of the login page follows **Flexibility and efficiency of use** principles, enabling medical staff to quickly access the software without needing to search for features.

After logging into the website, the medical staff will proceed to the medical dashboard.



The medical dashboard for the Grady Heart Disease Portal. It features a red cross logo and the text 'Grady Medical Dashboard'. The dashboard includes a sidebar with patient counts and a main area with assigned and unassigned patients.

Grady Medical Dashboard

Total Assigned Patients 0

Total Unassigned Patients 0

High Risk Patients 0

Potentially At Risk Patients 0

Low Risk Patients 0

Assigned Patients

john doe (1234) Priority Status: john doe (1234)

jane doe (1235) Priority Status: jane doe (1235)

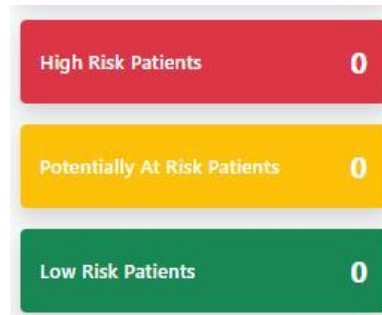
Unassigned Patients

Search patients

Create Patient Logout

Figure 14: Web UI Dashboard

In the main dashboard of the medical staff portal, staff can assign patients to themselves. They will return to this page to determine the risk levels of said patients. Using the column of counters on the left of the dashboard, the UI provides useful, dynamic feedback in an eye-catching, clear manner. This is especially important since these counters represent risk levels of patients. This feedback allows at-a-glance judgement of total patient status without forcing the doctors to pick through individual detailed views. In the main dashboard of the medical staff portal, staff can assign patients to themselves, aligning with **User control and freedom**.



*Figure 15: Web UI Risk Categories*

In the medical portal, patients are classified into three different risk categories based on their predicted levels of medical risk. These categories are then counted using these counting widgets. Red was selected for the high-risk patients, yellow for potentially at-risk patients, and green for low-risk patients. This stoplight-style approach is important since eyes should be drawn to the most important category: the high-risk patients. Making this counter a cooler color could be confusing and misleading. This coherent use of color ensures that the patients are categorized in an understandable, clear manner. This color-coded risk categories ensure **Match between system and the real world** by utilizing a stoplight-style approach familiar to users.

The staff also can view the statistical data result of each patient as below.

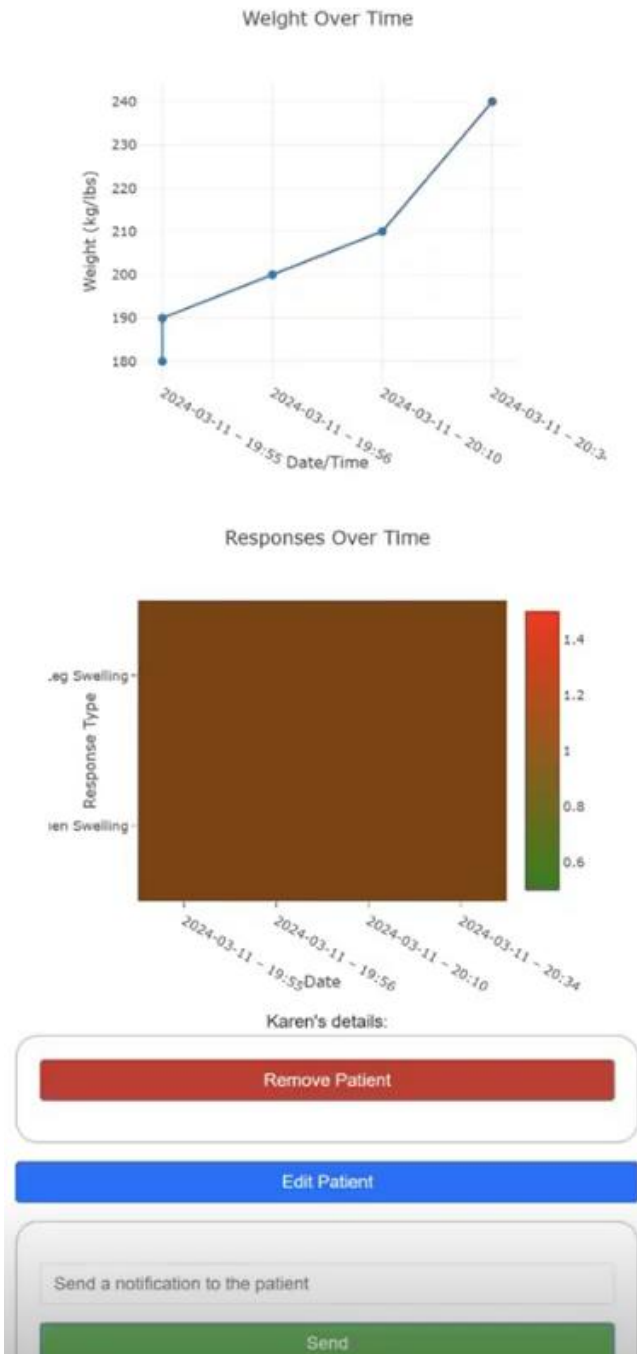


Figure 16: Web UI Statistical Analysis

# Appendix A: REST API Documentation

This REST API Documentation for healthcare management system outlines two critical aspects essential for effective API utilization: "Response Codes" and "Request-Response Parameters and Descriptions." These sections provide a detailed breakdown of HTTP status codes and the corresponding meanings, ensuring clarity and ease of troubleshooting during development and integration phases.

## Response Codes

These response codes are typically used in HTTP responses to indicate the status of the request made by the client.

Response Code	Meaning
200	OK - The request has succeeded
201	Created - The request has been fulfilled and resulted in a new resource being created
400	Bad Request - The server cannot or will not process the request due to an apparent client error
401	Unauthorized - Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided
404	Not Found - The requested resource could not be found but may be available in the future
500	Internal Server Error - A generic error message, given when an unexpected condition was encountered, and no more specific message is suitable

## Request-Response Param and Descriptions (Mobile)

The following is a list of REST API documents for the mobile client (patient) application, Flutter Routes.

## POST /submit\_survey

receives survey information from patient and stores it in database

**Resource URL:** [http://18.117.70.181/submit\\_survey](http://18.117.70.181/submit_survey)

### Request Information

- Request format: Json

### Resource Information

- Response Format: Json
- Requires Authentication: Yes (JWT Token issued to patient when logging in)
- Rate Limited: No

**Example Requests:** [http://18.117.70.181/submit\\_survey](http://18.117.70.181/submit_survey)

The header of the request must have

```
{  
  'Content-Type': 'application/json; charset=UTF-8',  
  'Authorization': 'Bearer $token'  
}
```

where \$token represents the JWT token issued to user when first logging in.

Json: {'weight': 140, 'isLeg':false, 'isAbdomen':false,'date':'2024-04-03 12:34'}

Parameter	Description
weight	A integer describing patient weight
isLeg	A boolean describing if the patient has seen leg swelling
isAbdomen	A boolean describing if the patient has seen abodmen swelling
date	The current date and time (hours & minutes) when response is submitted

### Example Response

json: {'message': 'Received and added'}

Parameter	Description
message	Describes if submission was successful (depends on if the JWT token is valid)

## POST /login

receives login credentials for patients and sends a JWT token if valid else a message describing why it was unsuccessful

**Resource URL:** http://18.117.70.181/login

### Request Information

- Request format: Json

### Resource Information

- Response Format: Json
- Requires Authentication: No
- Rate Limited: No

**Example Requests:** http://18.117.70.181/login

The header of the request must have

```
{  
  'Content-Type': 'application/json; ',  
}
```

Json: {'firstname': 'jane', 'lastname': 'doe', 'mrn': '1234', 'dob': '2000-01-01T12:00:00'}

Parameter	Description
firstname	A string describing first name of patient
lastname	A string describing last name of patient
mrn	A ID assigned by Grady to each patient
dob	DOB of patient with Date & Time (Time is ignored by Flask)

### Example Response

Success:

```
json: {'access_token': 'abcd'}
```

Failure:

```
json: {'message': 'Bad username or password or dob'}
```



Parameter	Description
Message (upon failure)	Describes if submission was successful (depends on if the JWT token is valid)
access_token (upon success)	A JWT token for patient to use in future requests

## GET /get\_weight

Gets weight data for patient to display in app

**Resource URL:** [http://18.117.70.181/get\\_weight](http://18.117.70.181/get_weight)

### Request Information

- Request format: Json

### Resource Information

- Response Format: Json
- Requires Authentication: Yes (JWT Token issued to patient when logging in)
- Rate Limited: No

**Example Requests:** [http://18.117.70.181/submit\\_survey](http://18.117.70.181/submit_survey)

The header of the request must have

```
{  
  'Content-Type': 'application/json; charset=UTF-8',  
  'Authorization': 'Bearer $token'  
}
```

where \$token represents the JWT token issued to user when first logging in.

Request Body: Nothing

### Example Response

json: [{ 'date': '04-02-2024', 'weight': 130, 'user': 0 }]

Note that response contains list of objects. Each object contains the following data.

Parameter	Description
date	Date on which the entry was recorded
weight	Weight recorded
user	ID of the user whose data is retrieved

## Request-Response Param and Descriptions (Web)

The following is a list of REST API documents for web client (medical staffs) application, Flask routes.

## GET/staff\_logout

logouts a logged in staff user and redirects to login page

**Resource URL:** http://18.117.70.181/staff\_logout

### Request Information

- Request format: No data

### Resource Information

- Response Format: HTTP Response
- Requires Authentication: Yes (Only logged in user will be able to access this endpoint)
- Rate Limited: No

**Example Requests:** http://18.117.70.181/staff\_logout

No data in request body

### Example Response

Redirected to login page if successful

Users that are not logged in currently will get a “Unauthorized” message directly from Flask built-in tools.

## GET/

Redirects to the staff dashboard page if the current user is logged in. Otherwise to the login page

**Resource URL:** http://18.117.70.181/

### Request Information

- Request format: No data

### Resource Information

- Response Format: HTTP Response
- Requires Authentication: No
- Rate Limited: No

**Example Requests:** http://18.117.70.181/

No data in request body

### Example Response

Redirected to staff dashboard if logged in Else to the login page

## POST/staff\_login

logs in staff user or errors if invalid credentials

**Resource URL:** http://18.117.70.181/staff\_login

### Request Information

- Request format: Json

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: No
- Rate Limited: No

**Example Requests:** http://18.117.70.181/staff\_login

Json: {'username':'sai', 'password':'1234'}

Parameter	Description
username	Username of staff
password	Password of staff

### Example Response

Logs in user and redirects to staff dashboard if success, else redirects to same login page with an error message

## GET/staff\_dashboard

**Resource URL:** http://18.117.70.181/staff\_dashboard

### Request Information

- Request format: No body

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: Yes (logged in users only)
- Rate Limited: No

**Example Requests:** http://18.117.70.181/staff\_dashboard

No request body

### Example Response

Shows list of current patients of the staff member so that they can click on someone to go to their detail view. It also shows a list of patients not assigned to that staff so they can add them to their list.

## POST/staff\_create\_patient

allows staff member to create a new patient entry

**Resource URL:** http://18.117.70.181/staff\_create\_patient

### Request Information

- Request format: Json

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: Yes (only logged in staff user)
- Rate Limited: No

**Example Requests:** http://18.117.70.181/staff\_create\_patient

Json: {'firstname': 'sai', 'lastname': 'avunuri', 'mrn': '1234', 'password': '2024-01-01'}

Parameter	Description
firstname	First name of patient
Lastname	Last name of patient
mrn	A id assigned by Grady to patient
dob	Date of birth of patient

### Example Response

Redirects to staff dashboard and shows the new patient on unassigned list.



## GET/staff\_add\_patient

Allows staff member to add a unassigned patient to their tracked list

**Resource URL:** [http://18.117.70.181/staff\\_add\\_patient/<patient\\_name>](http://18.117.70.181/staff_add_patient/<patient_name>)

### Request Information

- Request format: Json

### Resource Information

- Response Format: Json
- Requires Authentication: Yes (only logged in staff user)
- Rate Limited: No

**Example Requests:** [http://18.117.70.181/staff\\_add\\_patient/sai](http://18.117.70.181/staff_add_patient/sai) (1234)

Parameter	Description
patient_name	Part of the URL that tells the name & mrn of patient and used to identify and add that patient to staff's list

### Example Response

The response always returns a json containing the following data. This tells the frontend to send a redirect request to Flask so that after adding patient, the staff dashboard is rendered again.

Parameter	Description
redirect	url_for('staff_dashboard')

## POST/staff\_remove\_patient

Allows staff member to remove a patient from their tracking list

**Resource URL:** [http://18.117.70.181/staff\\_remove\\_patient](http://18.117.70.181/staff_remove_patient)

### Request Information

- Request format: Json

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: Yes (only logged in staff user)
- Rate Limited: No

**Example Requests:** [http://18.117.70.181/staff\\_remove\\_patient](http://18.117.70.181/staff_remove_patient)

```
{'user_id':1}
```

Parameter	Description
user_id	Id of user that we need to remove from the staff's patient list

### Example Response

Redirects to /staff\_dashboard page

## POST/edit\_patient

Allows staff member to edit a patient's details.

**Resource URL:** http://18.117.70.181/edit\_patient/<patient\_id>

### Request Information

- Request format: Json

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: Yes (only logged in staff user)
- Rate Limited: No

**Example Requests:** http://18.117.70.181/edit\_patient/1

Parameter	Description
patient_id	ID of patient whose details are edited

### Example Response

Redirects to /detail page of that same patient once the details are updated

## POST/send\_notifications

Allows staff member to send custom notifications to a patient

**Resource URL:** [http://18.117.70.181/send\\_notifications/<patient\\_id>](http://18.117.70.181/send_notifications/<patient_id>)

### Request Information

- Request format: Json

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: Yes (only logged in staff user)
- Rate Limited: No

**Example Requests:** [http://18.117.70.181/send\\_notifications/1](http://18.117.70.181/send_notifications/1)

```
{'notification':'hi!'}
```

Parameter	Description
patient_id	ID of patient to whom to send notifications (part of URL)
notification	The actual message to send

### Example Response

Redirects to /detail page of that same patient once the notification is stored and later sent to client when client pings for it

## GET/detail

Allows staff member to view details of a single patient

**Resource URL:** [http://18.117.70.181/detail/<patient\\_name>](http://18.117.70.181/detail/<patient_name>)

### Request Information

- Request format: No request body

### Resource Information

- Response Format: HTTP Response Flask object
- Requires Authentication: Yes (only logged in staff user)
- Rate Limited: No

**Example Requests:** <http://18.117.70.181/detail/sai> (1234)

Parameter	Description
patient_name	Contains patient name and mrn that are used to identify them and fetch their details

### Example Response

Redirects to /detail page of the patient and sends list data for graph plotting as part of Flask context

## Appendix B: Team Collaboration Roles

Throughout the development of the Detailed Design document, each team member played a crucial role, contributing their unique skills and expertise. The details are provided below.

Team member name	Role(s) performed for each draft (e.g., 1st, 2nd, final)
Jaehoon Song	Pre-Writer/Brainstormer, Diagram Design/Production, Document Design, Drafter
Michael Fu	Copyeditor/Proofreader
Edward Storey	Pre-Writer/Brainstormer, Copyeditor/Proofreader, Delivery Coordinator, Drafter
Jacob Rogers	Pre-Writer/Brainstormer, Copyeditor/Proofreader
Sai Anoop Avunuri	Pre-Writer/Brainstormer

**Additional Information:** N/A