RU Healthy?

Report 3: Full Report

Class

ECE 567 – Software Engineering 1

Semester

Fall 2017

Group #2

Tahiya Chowdhury, Tina Drew,

George Koubbe, Aymen Al-Saadi,

Himabindu Paruchuri and Ramya Tadepalli

Github Repository

RU Healthy?

Website

https://ruhealthy.github.io/ruhealthy/

TABLE OF CONTENTS

EFFORT BREAKDOWN

Report 1	Aymen	George	Tahiya	Tina	Himabindu	Ramya
Customer Statement of work						
Problem Statement (5)		100%				
Glossary (4)						
System Requirements						
Functional Requirements (2)		100%				
Non-Functional Requirements (2)						
UI Requirements (2)						
Functional Requirements Specs						
Stakeholders Actors/Globals (2)					100%	
Use Case Casual Description (8)	33%	33%		33%		
Use Case Diagram (5)	33%	33%		33%		
Use Case Full Description (10)	33%	33%		33%		
System Sequence Diagram (5)					100%	
User interface						
Preliminary Design (11)			50%			50%
Effort Estimation (4)			50%			50%

Domain Analysis						
Concepts (10)	50%	50%				
Associations (5)			100%			
Attributions (5)						100%
Contracts (5)					100%	
Mathematical Model						
Plan of Work (5)				100%		
Project Management (10)	16%	16%	16%	16%	16%	16%
References						

Report 2	Aymen	George	Tahiya	Tina	Himadindu	Ramya
Interaction Diagrams						
UML Diagrams (10)	50%	50%				
Prose description of diagrams (10)					50%	50%
Alternate Solutions (10)			50%	50%		
Class Diagram and Interface Specification						
Class Diagrams (4)				100%		
Data Types and Operation Signatures (5)	50%	50%				
Traceability Matrix (1)				50%	50%	
System Architecture and System Design						
Architectural Styles (5)						100%
Identifying Subsystems (2)			100%			
Mapping Subsystems to Hardware (2)			100%			
Persistent Data Storage (3)					%100	
Network Protocol (1)	50%	50%				
Global Control Flow (1)	50%	50%				
Hardware Requirements (1)	50%	50%				
Algorithms and Data Structures						
Algorithm	20%	20%	20%		20%	20%
Data Structures	20%	20%	20%		20%	20%

User Interface and Design						
User Interface and Design	20%	20%	20%		20%	20%
Design of Tests						
Design of Tests	20%	20%	20%		20%	20%
Project Management						
Merging Contributions	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%
Project coordination and Progress report				100%		
Plan of work				100%		
Breakdown of responsibilities				100%		
References	100%					

Report 3	**Aymen**	**George**	**Tahiya**	**Tina**	**Himadindu**	**Ramya**
	%100 **ALL TEAM MEMBERS CONTRIBUTED EQUALLY**					

1 SUMMARY OF CHANGES

Based on the feedback we received from Professor Marsic and Grader Huancan, during our demos and previous report submissions, we made a few changes. We tried to implement as many uses cases as we can, but we have not edited out the un-implemented sections from the report. Below is the list of changes:

- In Customer Statement of Work, we added "Recovery Time" to the Glossary of Terms.
- We implemented a new use case UC-14 'Check Recovery time' and updated the functional requirements, Traceability Matrix and domain model accordingly.
- We updated our project from local database to Firebase Online database and accordingly updated the section 'Persistent Data Storage'.
- We added a mathematical model for anomaly detection in heart rate recovery time using Bayesian Prediction.
- Updated the class Diagram with BayesianPrediction, ToDo, RecoveryRate, AlarmReceiver classes.
- Added the Data Types and Operation Signatures for the new classes added.
- We added the Prediction using Bayesian Curve Fitting in the Algorithms Section.
- We also added the updated UI images in the User Interface and Implementation Section .
- We added User Effort Estimation using Use case Points.
- We added two new pages in the Android app for 'Heart rate recovery time' and 'Scheduler' feature in the main dashboard of the application.
- We considered publisher-subscriber design patterns for our design choices and explained them in interaction diagrams and class diagram section.
- We added a 'Dashboard' page in the Android app UI to minimize user effort and allow access to different functionalities of the app with minimum click.
- We considered fail case post conditions for the use cases in Fully dressed descriptions.
- We added titles to each reference in Reference section.

2 GENERAL PROJECT INFO

2.1 Customer Statement of Work

2.1.1 PROBLEM STATEMENT

We live in a technology driven, quick paced, gluttonous society. We know that exercising has various benefits including: weight management, decreasing health conditions and diseases, improving mode, and boosting energy [2]. The issue is that we don't always receive these benefits because we don't regularly exercise. It is easy to get caught up in online profiles, games or social statuses and neglect physical exercise. It is also extremely convenient to purchase fast food from a local McDonald's rather than making a healthy home-cooked meal. Unfortunately, these habits lead to obesity and multitude of health issues. According to the 2016 Health Study by the United States Department of

Health and CDC (Centers for Disease Control and Prevention), over 70% of Americans over the age of 20 are overweight [10]. Another study from the CDC showed that 80% of Americans don't get the recommended amount of exercise [3].

The similarities in these numbers between those that are overweight and those that do not get the recommended amount of exercise is not surprising. According to the Merriam Webster dictionary obesity is defined as: "a condition characterized by the excessive accumulation and storage of fat in the body" [6]. We know that body fat is essentially excess calories that have not been burned. Exercise provides a way to burn calories at a rapid rate. So the more the exercise, the more calories you burn; the more calories you burn, the more fat you lose.

According to the CDC, each adult should do a minimum of 150 minutes of moderate intensity activity or 75 minutes of vigorous activity each week [8]. Put simply, this amount is about 22 minutes of moderate active per day or 11 minutes of vigorous activity per day. A moderate activity is one that is able to raise your heart rate enough for you to sweat. Examples of moderate aerobic activity include [8]:

- Walking fast
- Doing water aerobics
- Riding a bike on level ground or with few hills
- Playing doubles tennis
- Pushing a lawn mower

Vigorous activity raises your heart rate moderately and involves heavy breathing. Some forms of vigorous activity include [8]:

- Jogging or running
- Swimming laps
- Riding a bike fast or on hills
- Playing singles tennis
- Playing basketball

Additionally, muscle strengthening activity is advised [8].

2.1.2 CURRENT MARKET SOLUTIONS

As a way to promote activity and reduce obesity, various fitness devices and applications have been developed. Some of the fitness applications on the market include: Jawbone, Fitbit, Johnson and Johnson's 7 minute workout, FitStar, Lose It, Workout Trainer, JEFIT, Sworkit, and Strong Workout

Tracker [7]. Most of the fitness applications offer activity tracking, calorie counting, and step tracking. Many of these apps, such as Fitbit, include web interfaces that allow you to join in with other workout groups. Some apps such as JEFIT and workout trainer, demonstrate how to do certain exercises [7]. Additional apps such as JEFIT also allow you to map workout routines and offer workout timers [7]. These advancements in technology have made is easier to track activity progress while also encouraging people to exercise by the positive cognitive reinforcements of social media groups.

Figure 2.1.2 - Applications of same category that are currently available in market [13]

2.1.3 RU HEALTHY? APP SOLUTION

The "RU Healthy?" app wants to continue this trend of using technology to motivate people to exercise. It will implement the basic fitness features such as activity tracking and step counting by using the motion sensors and camera of the smartphone. The app will also detect whether you are exercising according to your schedule. Furthermore, the app will store these statistics to an online database that both the physician and user can access.

No one enjoys the problems that come with being overweight and we know that exercise reduces body fat. So, why don't we exercise more? For many of us there are three main reasons that we don't exercise: lack of accountability, lack of motivation, or scheduling inconsistencies. The RU Healthy? app addresses all these issues.

To address the issue of the lack of motivation, let's consider the impact of accountability. Accountability motivates people to move forward. There is a friend who recently started a 30 day challenge to burn 1000 calories a day. The challenge was started because she wanted to buy a special shirt from her friend. The friend would not offer the shirt to her but stated that if she was able to complete the 1000 calories a day challenge, he would give it to her. He then opened up the challenge to other friends in their group in a fitness app. The one who does the best on the challenge gets the shirt and a $200 cash price. What is interesting about this group of people is that many of them are not very active and are not usually motivated to exercise. However, because they are part of the group and their progress is reported to the group in this fitness application, they are intentionally exercising. The group's accountability motivated the non-active members to get up and move. The RU Healthy? app would essentially do the same thing except the user would be accountable to their physician.

Another reason that we don't exercise consistently is because we have a very busy lifestyle. We never have the time and we simply "forget" or don't put it in our schedule. The RU Healthy? app offers a possible solution for this issue. The app has a schedule reminder that sets off an alarm when it is time to exercise. The user can set up their own workout schedule, or be advised a schedule by a physician or personal trainer. This schedule will look different for each patient. For some, it may be 5 to 10 minutes of activity at certain points of the day. For others it may be a block of time setup specifically to work out. Once the schedule is set, it will sync with the phone calendar and timer to send an alert notifying the user that it is time to exercise. This alarm or alert gives the user a reminder to get up and move. The chances that we will forget to exercise are reduced. Additionally, because the schedule will be set up and agreed upon by the user, it should be at a pace that they can adhere to and follow.

From a physician's perspective, the RU Healthy? app makes it easier to hold their patient accountable to the exercise regiment. Think about it, everytime we go to the doctor they ask the question: are you exercising? The answers to this question often frustrates physicians because their patients are either not following their suggestions regarding exercise or not being honest about the exercise routine. What if the physician already had the answer to that question prior to the appointment? This would save time from the doctor trying to decipher the patient's honesty and commitment to the exercise. Furthermore, this helps the patient to develop an action plan, a technique that increases the probability that the person will exercise and meet the goal [9]. In short this makes it easier for patients to be accountable to their physicians. This is one of the concepts that makes our app unique. Other fitness applications do not promote allowing physicians to access the patient's activity results.

Additionally, the doctor would have access to the patient's basic health statistics like heart rate and analysis of "recovery rate", which is a useful tool to track a patient's progress. For the patients who are at risk this information could be extremely useful because it could make the physician aware of anomalies such as irregular heart rates. Since an irregular heart rate is often an indicator of other health issues, having this information could help physicians diagnose problems in early stages. The physician could also examine the heart rate levels of patients and detect possible issues such as thyroid problems or heart palpitations.

Figure 2.1.3 - A Short comic from Glasbergen.com [18]

2.1.4 GLOSSARY OF TERMS

Table 2.5.1: Glossary

Term	Description
Accelerometer	Handles the axis-based motion sensing and is used to count the step taken or distance walked.
Account	Entity in database that contains specific information about the user including profile details, body measurements, progress, etc.
Alarms	User can set his personal alarms for exercise.

BPM	Beats Per Minute is the number of times the heart beats in one minute.
Calorie Counter	It will track the number of calories the user burnt based on the distance covered.
Database	Place to store information. Stores the data of the user as a structured set.
History	Shows the previous data of the calories burnt/steps covered and if the user is working as per schedule.
Messages	Users can check some links of various exercise available in this option.
Profile	This includes the user's name, age, height, weight, profile picture(optional), email, gender etc.
Register	User can give username and password and register to use the mobile application.
Scheduler	Feature in the app to display the schedule of exercises for each month for ease of user.
Settings	User can change the profile settings like steps/calories target, distance measurement units, etc.
Smartphone	Portable personal computer with a mobile operating system containing features useful for handheld use.
Step Tracker	A feature in app that tracks the number of steps completed and steps left to reach target.

Web Application	Web interface which the physician can use to access the patient data.
Recovery Time	Required duration of time taken by a process to restore its normal state after an action. In our case, time taken for the user to restore his/her normal heart rate after exercise.

2.2 How will ‘RU Healthy?’ accomplish this?

Let’s say the patient, Mike, visits a physician and creates a user account, logs into the app and links it with the doctor’s account. Then, Mike will have to do an initial one-time configuration to enter his age, gender, height, weight. With the app, he will have the ability to check: when he started and stopped walking, and the total distance that he walked. Moreover, he will be able to see and keep track of the calories he burnt during the week and his activity history, along with a heart beat rate history.

At the other end, the physician will be able to see Mike’s activities in a web UI. The web UI will provide him/her with all the necessary information about the patient status. It will also contain the patient’s personal data such as:

- Name
- Age
- Gender
- Weight
- Height
- Activity history
- Heart rate history and average heart rate
- Number of the steps he/she walked
- How many miles he/she walked
- How many calories he/she burnt during his activity

The physician will have the ability to control all the patients from the web UI, and review their information. The data of this UI will be obtained from a database which regularly stores user activity (say, weekly). We can use Django or a similar platform to make the web UI for the doctor. Data from the Android smartphone can be sent to a server/database with the help of Java or Android libraries or this can be synced to the cloud. We can further scale this up to ensure that the physician is able to see all of his patient's records, if they have installed the app.

Every day, the person's data will be stored. The weekly update will be sent to his/her physician. This way, the physician will get to know whether Mike is following his schedule or not. The doctor will have a web page with login information, and when he/she logs in, he/she can see the information updates coming from all their patients. And whichever patient is failing to follow the schedule, it will be mark by red and the device will give the patient a "warning" or a notification. The physician will set up an appointment with those patients to talk about it.

2.2.1 HOW DOES THE PROPOSED SYSTEM COUNT THE STEPS?

The user just needs to carry his smartphone with him/her or leave it in their pocket. It will detect the unusual movement of the user and try to calculate the acceleration, based on three variables (x, y and z). The accelerometer sensor is found in every Android device (old and new). It is a device that measures the acceleration (or rate of change of velocity) of a body in its own instantaneous rest frame. Single and multi-axis models of accelerometer are available in Android devices to detect magnitude and direction of the proper acceleration, as a vector quantity, and can be used to sense orientation (because direction of weight changes), coordinate acceleration, vibration, shock, and falling in a resistive medium. That is one of the features that will give the advantages for the app , so it will not need any extra costs for the user to use this feature.

Our Footsteps Detector will collect the data from the sensors (x, y and z), velocity and device acceleration which is measured by m/s^2. After collecting these data we will process them and store the data in arrays. These arrays will try to help us to filter the data and normalize them based on normalization algorithms. These algorithms have a specific limit (which we will call sensor sensitivity) to detect the abnormal data that is coming from the sensors and store them in single dimension array or vector. We can retrieve this data, calculate the amount abnormal data obtained, and view it as a step.

Below is a specific description of the classes that we used for the Footsteps Detector:

Application Classes

- Step Listener Class Interface: This class-interface will listen to number of alerts about steps being detected.

- Step Detector Class: A class which will accept updates from accelerometer sensor and deploy the filter to detect if a step has been covered by the user.

- Sensor Filter Class: A class with an algorithm to filter out values that have a close approximation to steps sensitivity.

- Main Activity Class: This class will contain all the buttons on click activities and all the data returns from the main classes.

Application Functions

- On click Listener: This function will activate the Sensor Manager Listening to the abnormal acceleration from the Step Listener Class.

- On Sensor Changed: This function will update the sensor values from the Step Detector Class.

2.2.2 HOW DOES THE PROPOSED SYSTEM MEASURE HEART RATE?

For now, we would like to provide a general idea about how we intend to measure the heartbeat rate. We plan on using the smartphone's flash and camera. All the user has to do is open the app and hold his/her index finger over the camera lens to start measuring. The camera is used to track color changes on the fingertip that are directly linked to the pulse. This is the same technique that medical pulse oximeters use. The whole process should not take more than 30 seconds.

3 SYSTEM REQUIREMENTS

3.1 Functional Requirements

A requirement is defined as "a condition or capability to which a system must conform" [4]. A system may have a variety of requirements and these are often categorized to ensure better focus on each.

Functional requirements may be calculations, technical details, data manipulation, data processing and other specific functionality that define *what* a system is supposed to accomplish. These type of requirements are expressed in the form "system shall do <requirement>" for all the features that must be implemented, whereas features that are an optional addition are expressed in the form "system should do <requirement>". The priority weight assignation is based upon the customer's essential requirements that must be satisfied. A higher number indicates that a requirement is more crucial for the project.

Table 3.1.1: Enumerated functional requirements for 'RU Healthy?'

ID	Priority Weight	Requirements
REQ-1	5	The system shall be able to obtain and process the information from the Android device sensors.
REQ-1a	5	The system shall be able to obtain motion sensor data and count how many steps the user walked during his/her last activity.
REQ-1b	5	The system shall be able to obtain the data from camera and compute BPM.
REQ-2	4	The system shall be able to store the data collected from the

		sensors in an offline database.
REQ-3	4	The system shall be able to compute how many miles the user walked, how fast their paces were, and the duration of travel.
REQ-4	4	The system shall be able to compute how many calories the user burnt based on his/her traveled distance and speed.
REQ-5	4	The system shall allow the user to add/edit/remove accounts.
REQ-6	2	The system should be able to give the user alerts and notifications.
REQ-6a	2	The system should be able to send notifications about current progress.
REQ-6b	2	The system should send alerts when it is time for the user to exercise.
REQ-7	3	The system should be able to communicate to an online database.
REQ-7a	3	The system should be able to do online backups.
REQ-7b	3	The system should be able to do online restores.
REQ-8	5	The system shall be able to send a weekly activity summary to the physician.
REQ-9	4	The system shall be able to deliver the user all the important notes that the physician wants the user to follow.
REQ-10	2	The system should allow a workout schedule to be set by the physician in conjunction with the user.

REQ-11	1	The system should allow the user to manually turn off the alerts.
REQ-12	4	The system shall allow users to start and stop monitoring information upon the user's request.
REQ-13	5	The system shall allow the users to check the recovery time after each exercise.
REQ-14	5	The system shall run the Prediction algorithm in the background and based on that inform the user whether his/her Recovery time is in normal range.

The first thing to do in RU Healthy? is to create a user profile (REQ-5). However, when it is time to link it with a doctor, how will the patient do that? Does he/she know a doctor? Does he/she prefer someone in particular? We thought about it, and we have come up with the following policy:

RUH-BP01: A prior connection between the patient/physician will need to be established through a physical visit that will allow the initial configuration of the app. That is, setting up initial information and linking the user's profile to the doctor's account.

We also analyzed the option in which the patient does not have a physician, and he/she could select the area and choose a physician of preference, or even the system could get the patient's current location and match him with a physician nearby. But what if after the blind selection, the user actually gets to know his/her physician and does not like him/her?. The user would have given the doctor records and private information that could expose him/her. So for now, we will be staying with our current business policy.

Once the physician selection through physical visit is complete, the primordial function of the system will be to obtain and process information from the camera and motion sensor every time the patient exercises. In order to satisfy this, we have separated REQ-1 into two different processes: activity

and heart rate, that is REQ-1a and REQ-1b. Without this, the system has no point and no future development. The patient can turn this functionality on and off at will, so we need to introduce REQ-12.

Because the software will only use the internal sensors of the Android device, there is no way of continuously monitoring the heart rate. Therefore, we need to introduce our second policy:

RUH-BP02: The heart monitoring won't be a continuous feature. Instead, the patient will need to measure it manually every time he/she needs to.

The system will also provide additional metrics about the exercise, and we need REQ-3 and REQ-4 for this.

An optional feature that the system could have is to provide the user with updates on his/her workout every now and then while he/she is exercising. This is accomplished with REQ-6. Also, it sounds like a good idea to alert the user when is time to exercise, so he/she does not forget, as well as allow the same notification functionality to be turned off at any time. REQ-6 and REQ-11 go hand to hand.

This workout schedule should not be something to be set by the physician only. It should have feedback from the user regarding exercises, duration, days of the week, etc. For this, we introduce REQ-10.

In order for the system to be successful, a communication between patient/physician is crucial. The physician needs to be able to monitor the patient's activity, so he/she can recommend or give him/her further indications. For this, we have REQ-8 and REQ-9. But what if the patient, besides sending the data to the physician, wants to be able to backup everything online? Imagine he/she gets robbed, or ends up losing their smartphone? We have come up with REQ-7, which makes sure the patient feels safe in this matter.

One more thing. Because the system will have no administrators, nobody will control the creation or removal of user profiles. The physician needs to be able to access the user profile for reviewing purposes. The following needs to be established:

RUH-BP03: the Physician can terminate his participation at any point if he feels that the Patient does not need it anymore. The Patient can choose if he/she wants to keep using the app.

3.2 Non-Functional Requirements

Non-functional requirements are based on the environment and quality of the system-to-be. The acronym 'FURPS+' represents a widely used model that classifies software quality attributes. This set of attributes includes: Functionality, Usability, Reliability, Performance and Supportability. The '+' symbol denotes the requirements to include design constraints, implementation, physical and interface requirements [4, 5].

The attributes related to 'Functionality' of our proposed system is enlisted in Section 3.1. Based on our study on existing applications, previous related works and informal survey on target customer population, we specify the non-functional requirements for our proposed system 'RU Healthy?' in Table 3.

Table 3.2.1: Enumerated non-functional requirements for 'RU Healthy?'

ID	Priority Weight	Requirements
REQ-15	5	All information from sensors will be stored in the system database.
REQ-16	4	The interface of the mobile app will be easy to navigate. The patient shall be able to change between menus with minimal effort even when working out.
REQ-17	3	The system response shall be prompt to user's command. Patient shall not feel delay between action (click) and

		reaction (system response).
REQ-18	2	The app should not crash if running for a long period of time.
REQ-19	1	Both the mobile and web application shall be intuitive; so that general experience with Android app will suffice to use the basic features effectively.
REQ-20	1	The app (mobile and web) should provide a help document to guide the patient and physician into initial registry of the system.
REQ-21	2	The app should disclose the degree of access it will have to patient's phone and his information before actually starting to use it.
REQ-22	3	The patient information stored in the database of the application shall have a high-level security. Each patient will have access to his/her information only.
REQ-23	4	The physician will be required to protect the privacy of the patient. This means he/she will not disclose login information to his account to web application with others.
REQ-24	5	The system will prohibit manipulation of work-out data by the patient or the physician. All data stored in the database and presented by app will be collected from mobile sensors.
REQ-25	3	The patient should be able to stop the alarm/reminder from the app by starting his scheduled work-out on that time.
REQ-26	2	The app will stop running after sensing inactivity of the patient for a certain period to save battery life.

Usability Requirements

The concept of ‘RU Healthy?’ focus on providing a user-centered design and a satisfying user experience. The user interface, with its easy to navigate features and prompt response to command, will mark consistency for the user experience (REQ-16, REQ-17). The proposed requirement for an intuitive design on interface will ensure that anyone with experience of using an Android app will be able to accomplish their goal (REQ-19). Alongside that, the help document will be able to help a novice user (REQ-20).

Reliability Requirements

The app, with the amount of data processed during work-out, will be required to elicit adequate reliability. While the app is running for a long period of time, it should not crash with the data for the session being lost (REQ-18). Also, with all data obtained from the sensors and not from manual input, the integrity of the data will be ensured (REQ-24).

Performance Requirements

The performance of the system should be efficient, quick and accurate. The app thus needs to provide quick reaction to user command and operate as such (REQ-17). Once the user start working out, it should provide explicit feedback for the user to confirm response and availability (REQ-25).

Supportability Requirements

All information from the sensors will need to be stored for processing and analyzing of data. The processed data will be used for fathoming the progress of the user for a period of time (REQ-15). In addition to that, the resource usage by the app during idle time should be restricted (REQ-26).

Design and Implementation Restrictions

This portion of requirements sets the dimension for policies of database integrity, resource limit, etc. As a large amount of private information about the user will be available in the database, each user should only have access to their own information only (REQ-22). The physician is restricted to maintain confidentiality of the information regarding his/her patient as part of his professional confidentiality agreement (REQ-23. The integrity of available data in the database should be maintained by avoiding direct manipulation from any of the users (REQ-24)

3.3 On-Screen Appearance Requirements

The section below show hand-drawn sketches that demonstrate the user interaction.

3.3.1. MOBILE APPLICATION

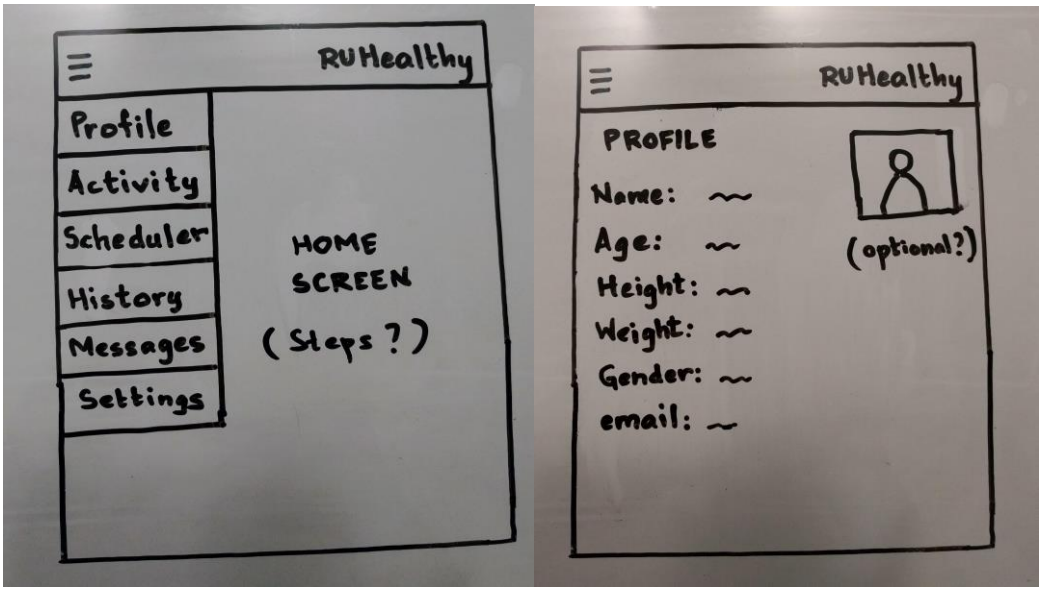


Fig 3.3.1.1: Options and User Profile

Figure 2 shows the available options on the mobile application. The user will be able to register and login. User's details will be stored to a database. Once the user logs into their account, they will be able to add details to their profile. This details include: age, height, weight, and gender and will be used to set a reasonable target for their activity. They can also provide their email for further communication.

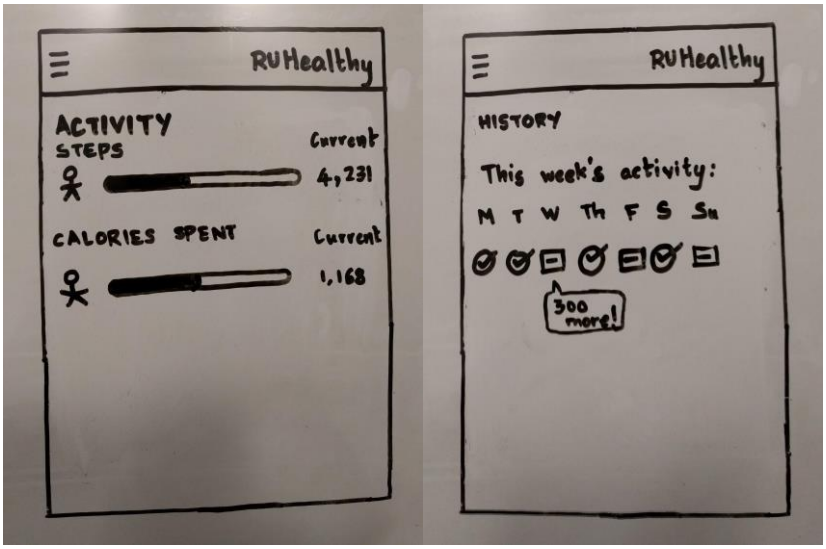


Fig 3.3.1.2: User Activity

Figure 3 shows how the User Activity will appear. The Users will be able to see their current progress for the day as a number and also as a graphic in comparison to their target. This includes the number of steps taken in miles or steps, the number of calories burnt (which will be calculated based on the profile details provided). Users will also be able to view a history of their activity as a timeline view, which highlights whether or not they have reached their target for each day.

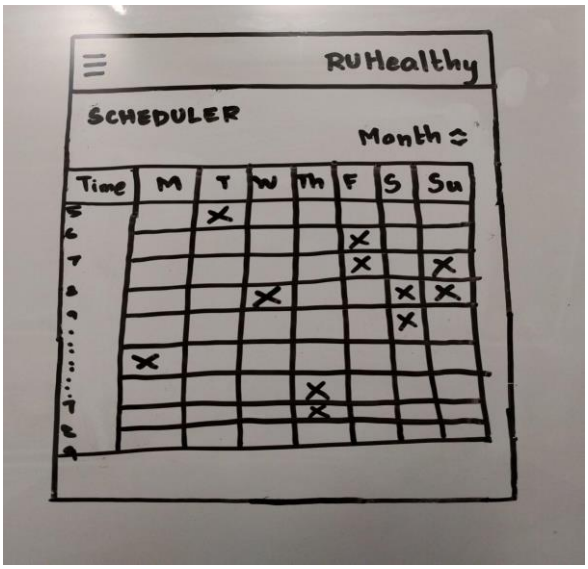


Fig 3.3.1.3: Scheduler

Figure 4 shows the Scheduler option. The Users will be able to mark some time of the day for exercise. The app then detects whether or not the User is performing any activity at that specified time. Since, we assume that the phone is on the user for the app to fully function, if the app detects the user being idle or lack of activity, it will remind the user that it's time for exercise.

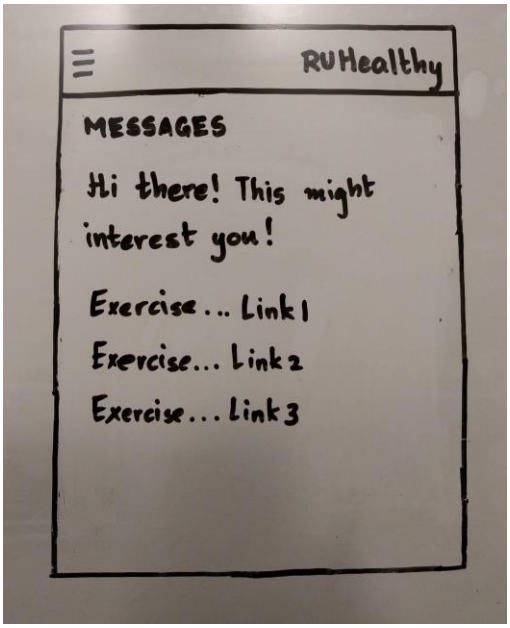


Fig 3.3.1.4: Messages

Figure 5 shows the messages option for the app which could include messages that have been auto generated based on the activity the user performs or it could be messages sent to the user by their physician/personal trainer. This may include articles or links to some related exercise options or those advised for that particular user.

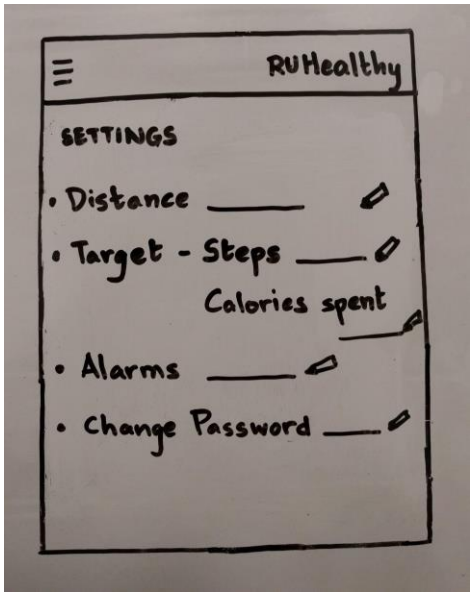


Fig 3.3.1.6: Settings

Figure 6 shows the Settings page of the app users may set their targets, reminder options etc.

3.3.2. WEB APPLICATION

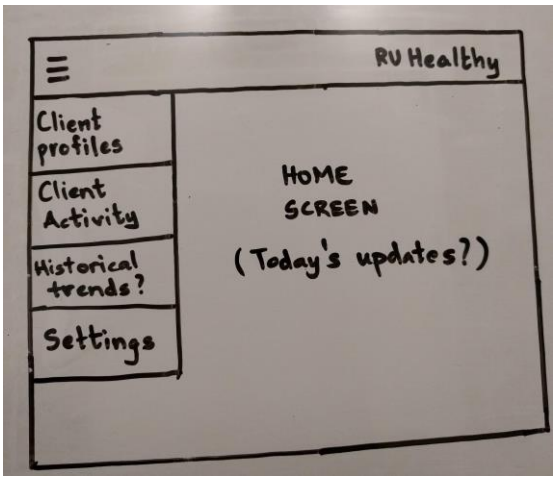


Fig 3.3.2.1: Options

We have also proposed a Web based application which will help the Users' physicians or personal trainers view their activity status. Let us call them admins. The admin will be able to register and login to the Web application. They can add or delete Client profiles i.e., details of their clients like, Name, age, gender, height, weight, BMI and possibly their health record. They will also be able to view Clients' activities. We envision that this information will help in better diagnosis and treatment and/or help clients lose weight.

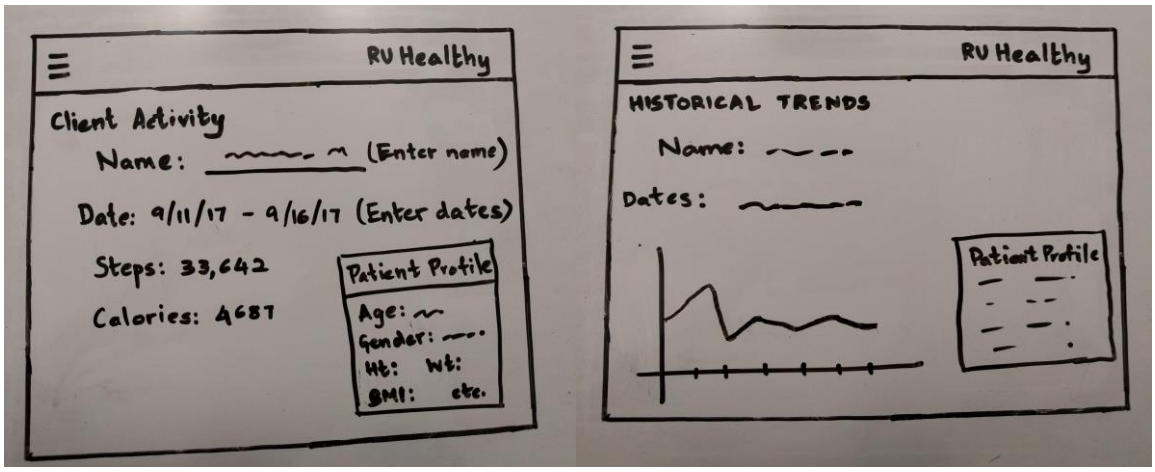


Fig 3.3.2.2: Client activity

Figure 8 shows how the admin will be able to view Client activity. This includes 2 pages, one to view activity and one to view trends in their activity. We may merge this into one page if that seems like a better and more user friendly view of the client's' activity. The admin is allowed to choose a specific set of dates and the User activity for that period is displayed on screen.

4 FUNCTIONAL REQUIREMENTS SPECIFICATION

4.1 Stakeholders

Stakeholders are those with any interest in project's outcome. Stakeholders are people who are invested in the project and who will be affected by your project at any point along the way, and their input can directly impact the outcome. Accordingly, the stakeholders in the project

RU Healthy? would include the following categories of people:

End Users or Customers: This project targets the people who aim to lose weight through exercising regularly and need some sort of push to achieve their goals. They can use this system to keep track of their health conditions and get easy way to keep healthy. So, the customers/end users would be the largest group of stakeholders using this system.

Physician/Personal trainer: This project will help the physicians /personal trainers to always keep track of their patient exercise routine. This would help save time in unnecessary appointments since he/she would have all the patient statistics (like heart rate) and it can be helpful in finding anomalies early.

Project team members: The team members also form a major stakeholder for the project who came up with an idea for it and have a major role in developing and executing it. They are also affected by the outcome and can, thus, be listed as stakeholders for the project.

4.2 Actors and Goals

Actors can be defined as people or devices that will directly interact with the product or the system. All these actors would have specific goals by interacting with the system. For this project, the following would be the actors:

User/Patient: The patient's goal is to access the mobile application to keep track of his/her exercise schedule and health conditions.

Physician: The physician's goal is to access the web application to keep track of his/her patient statistics for easy and early diagnosis.

Local Database: The local database (SQLite) would be used in the initial stages of the project to store the records of all patient data(like profile data). It will also store the health data for each patient separately.The mobile application will access the local database to retrieve all the required information.

Online Database: The online database would be used in the later stages of project for the same purpose as the local database. Firebase Real Time Database (cloud) would be used for the same which can be accessed by both the mobile application as well as the web application.

Mobile Application: The mobile application allows the user/patient to interact with the system and transfer of information between both. It acts as a participating actor in most of the use cases.

Web Application: The web application allows the physician to interact with the system and keep track of all patient records, manage their schedules.

4.3 Use Cases

Use case modeling is a technique used to represent system requirements. Each use case describes how the user and system interact to achieve business goals. [15] The sections below contain descriptions of our use cases and identify how they align with the system requirements.

4.3.1 CASUAL DESCRIPTION

Below is a brief summary of each of the use cases.

UC-1: Toggle Activity - Allows the Patient to start and stop the app's activity and/or heart rate monitoring.

Derived from REQ-12, REQ-13.

UC-2: Health Data - Allows the Patient to check and monitor activity using the smartphone app.

Derived from REQ-1.

We analyzed the idea of allowing the patient to monitor his/her activity without being logged in. That means not having/using an account to save all the information, and by implication, not communicating with the physician. There will be no point in doing that, since there are many apps that already have this functionalities. Because of this, it is implicit that the patient must be signed in to start using and monitoring his/her activity.

UC-3: Get Vital Signs Phone - Allows the Patient to retrieve his/her activity log, heart rate, recovery time, calorie count, and progress information through the smartphone application interface.

Derived from REQ-2, REQ-3, REQ-4, REQ-13 and REQ-14.

UC-4: Get Vital Signs Web - Allows the Physician to retrieve the user activity log, heart rate, recovery time, calorie count, and progress information through the web interface.

Derived from REQ-2, REQ-3, REQ-4, REQ-7 and REQ-13.

UC-5: Enable Notifications - Allows the Patient to receive notifications about his/her progress. (optional sub use case, «extend» UC-2: Health Data).

Derived from REQ-6.

UC-6: Get Notes - Allows the Patient to receive notes posted by the Physician. (mandatory sub use case, «include» from UC-3: Get Vital Signs Phone).

Derived from REQ-9.

UC-7: Manage Schedule - Allows the Physician to setup and control exercise schedule in conjunction with the Patient (optional sub use case, «extend» UC-2: Health Data).

Derived from REQ-10.

UC-8: Toggle Alarm - Allows the Patient to receive an alarm when it's time to exercise, or to turn it off (optional sub use case, «extend» UC-2: Health Data).

Derived from REQ-6 and REQ-11.

UC-9: Enable Weekly Updates - Allows the Physician to receive automatic weekly updates about the Patient's activity and heart rate status.

Derived from REQ-8.

UC-10: Send Notes - Allows the Physician to update the system with notes for the Patient.

Derived from REQ-9.

UC-11: Physician Sign-in - Allows the Physician to manage users accounts, doing operations like add/delete accounts.

Derived from REQ-5.

UC-12: User Sign-up - Allow the Patient to create his/her own account using her/his own personal information.

Derived from REQ-5.

UC-13: User Sign-in - Allow the Patient to access his/her own account using her/his own personal valid credentials, and will have the ability to modify his/her personal information.

Derived from REQ-5.

UC-14: Check Recovery Time - Allow the user to check his/her recovery time after exercise. Also, inform the user whether the recovery time is normal or high based on the prediction algorithm.

4.3.2 USE CASE DIAGRAM

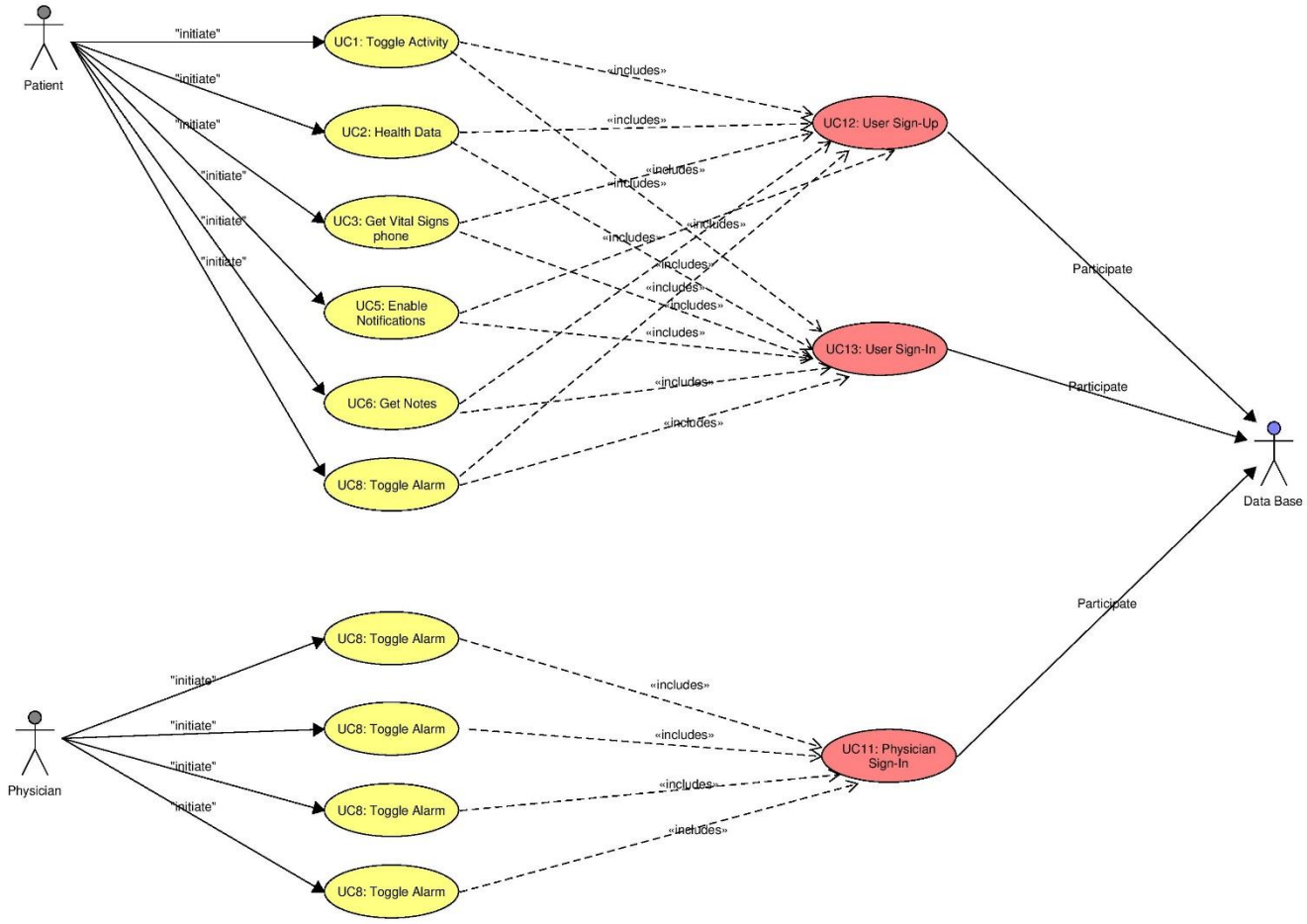


Figure 4.3.2.1- Use cases diagram showing the System Use case

4.3.3 TRACEABILITY MATRIX

Table 4.3.3.1: Traceability matrix

Req't	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC-14
REQ-1	5		X												
REQ-2	4			X	X										
REQ-3	4			X	X										
REQ-4	4			X	X										
REQ-5	4											X	X	X	
REQ-6	2					X			X						
REQ-7	3				X										
REQ-8	5									X					
REQ-9	4						X				X				
REQ-10	2							X							
REQ-11	1								X						
REQ-12	4	X													
REQ-13	5	X													X
REQ-14	5	X													X

Max PW		5	5	4	4	2	4	2	2	5	4	4	4	4	5
Total PW		14	5	12	15	2	4	2	3	5	4	4	4	4	10

3.4 FULLY-DRESSED DESCRIPTION

Below are the most important use cases. Without them, the system will be no longer functional.

Use Case UC-2	**Health Data**
Related Requirements	REQ-1 stated in the Table 4.3.3
Initiating Actor	Patient
Actor's Goals	To monitor his/her own activity
Participating Actors	Mobile Application, Local Database
Preconditions	• Smartphone has sufficient battery charge. • The Patient has a valid registration credential (signed-in) in order to store/access his/her own data.
Post condition - Success	● Activity record was stored in Local Database.
Post condition - Failure	● Activity record failed to store in Local Database, can fail in following condition: Sensor (accelerometer and camera) not working

Flow of Events for the Main Success Scenario	→ **1.** (a) **Patient** opens the app, (b) initiate the tracking function and (c) starts exercising. ← **2.** (a) **Mobile Application** counts the steps, (b) provides relevant data and (c) sends eventual notifications to the **Patient.** → **3. Patient** stops the tracking function. ← **4.** (a) **Mobile Application** shows the Patient his/her current Session and (b) stores it in Local Database.
Flow of Events for Extension (Alternative Scenario)	2a. **Patient** stops moving for a very long time. ← **1. Mobile Application** detects it. ← **2. Mobile Application** stops tracking activity.

Use Case UC-3	**Get Vital Signs Phone**
Related Requirements	REQ-2, REQ-3 and REQ-4 stated in Table 4.3.3
Initiating Actor	Patient
Actor's Goals	To retrieve the patient information and activity logs from the database using the Mobile Application.
Participating Actors	Mobile Application, Local Database
Preconditions	● Smartphone must have sufficient battery charge.
Post condition - Success	● **Mobile Application** displayed only the chosen record.
Post condition - Failure	● Data not found/ not displayed, can happen in following condition: Data storage was full and data was not saved properly
Flow of Events for the Main Success Scenario	→ **1.** (a) **Patient** opens the app and (b) presses the button corresponding to history logs. ← **2.** (a) **Mobile Application** retrieve information from **Local Database** and (b) shows the list of available activity records. → **3. Patient** selects the desired record.
Flow of Events for Extension (Alternative Scenario)	2b. **Mobile Application** shows an empty list because **Patient** has no previous activity and vital sign records. ← **1. Mobile Application** prompts **Patient** to start exercising for the first time.

Use Case UC-4	**Get Vital Signs Web**
Related Requirements	REQ-2, REQ-3, REQ-4 and REQ-7 stated in Table 4.3.3
Initiating Actor	Physician
Actor's Goals	To retrieve the Patient information and activity logs from the Online Database using the Web Page.
Participating Actors	Web Page, Online Database
Preconditions	● **Physician** has a valid registration credential (signed-in) in order to access the **Web Page**. ● **Patient** must have previous activities and vital signs record in order for the **Physician** to view his log.
Post condition - Success	● **Web Page** displayed the desired **Patient's** record.
Post condition - Failure	● Record not found/ not displayed, can happen in following condition: Data was not saved properly Internet connection is not available
Flow of Events for the Main Success Scenario	→ **1.** (a) **Physician** accesses the **Web Page** and (b) selects the desired **Patient**. ← **2.** (a) **Web Page** retrieves information from **Online Database** and (b) displays the **Patient's** records. → **3. Physician** selects a specific record to see.
Flow of Events for Extension (Alternative Scenario)	2b. **Web Page** shows an empty list because **Patient** has no previous activity and vital sign records.

Use Case UC-11	**Physician Sign-in**
Related Requirements	REQ-5 stated in the Table 4.3.3
Initiating Actor	Physician
Actor's Goals	To manage Patient accounts, doing operations like add/edit/remove
Participating Actors	Physician, Online Database
Preconditions	● **Physician** has a valid registration credential (signed-in) in order to access the **Web Page**.
Post condition - Success	● Patient was added/edited/removed to/in/from the database. ● In case of removal, **Online Database** no longer has Patient's information.
Post Condition - Failure	● Add/Edit/Remove request was denied; Patient's information remain unchanged, can happen in following conditions: Internet connection not available
Flow of Events for the Main Success Scenario	→ **1.** (a) **Physician** accesses the **Web Page** and (b) selects the desired **Patient**. ← **2. Web Page** prompts the **Physician** with the available options. → **3. Physician** selects one of the options (add/edit/remove) ← **4. Web Page** updates the **Online Database**.
Flow of Events for Extension (Alternative Scenario)	3a. **Web Page** cannot add/edit/remove a specific **Patient** ← **1. Web Page** alerts **Physician** saying that **Patient** already is/is

	not in **Online Database**

Use Case UC-14	**Check Recovery Time**
Related Requirements	REQ-13 and REQ-14 stated in Table 4.3.3
Initiating Actor	Patient
Actor's Goals	To allow the user to check his/her recovery time after exercise. Also, inform the user whether the recovery time is normal or high based on the prediction algorithm.
Participating Actors	Mobile Application, Cloud Database
Preconditions	● Smartphone must have sufficient battery charge. ● User must be logged into his/her account
Post condition - Success	● **Mobile Application** displayed the recovery time.
Post condition - Failure	● **Mobile Application** failed to display the recovery time, can happen in following condition: Camera sensor not working Insufficient light, heavy moisture on fingertip
Flow of Events for the Main Success Scenario	→ **1.** (a) **Patient** opens the app and (b) presses the recovery rate Icon, keeps his/her finger on the camera and checks time needed for the user to restore to Normal state. ← **2.** (a) **Mobile Application** retrieve past information from **Cloud Database** and (b) runs the prediction algorithm in Background.

	← **3. Mobile Application** displays the recovery time for the user and also the result of the prediction algorithm showing Whether the recovery time is normal or too high.
Flow of Events for Extension (Alternative Scenario)	2b. **Mobile Application** doesn't show proper recovery time Because Online database doesn't have enough dataset to Predict whether the recovery time is normal.

4.4 System Sequence Diagrams

A **system sequence diagram** (SSD) is a sequence diagram that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events. System sequence diagrams are visual summaries of the individual use cases.

Below are the System Sequence Diagrams for few important Use cases:

Use Case UC-2: Health Data

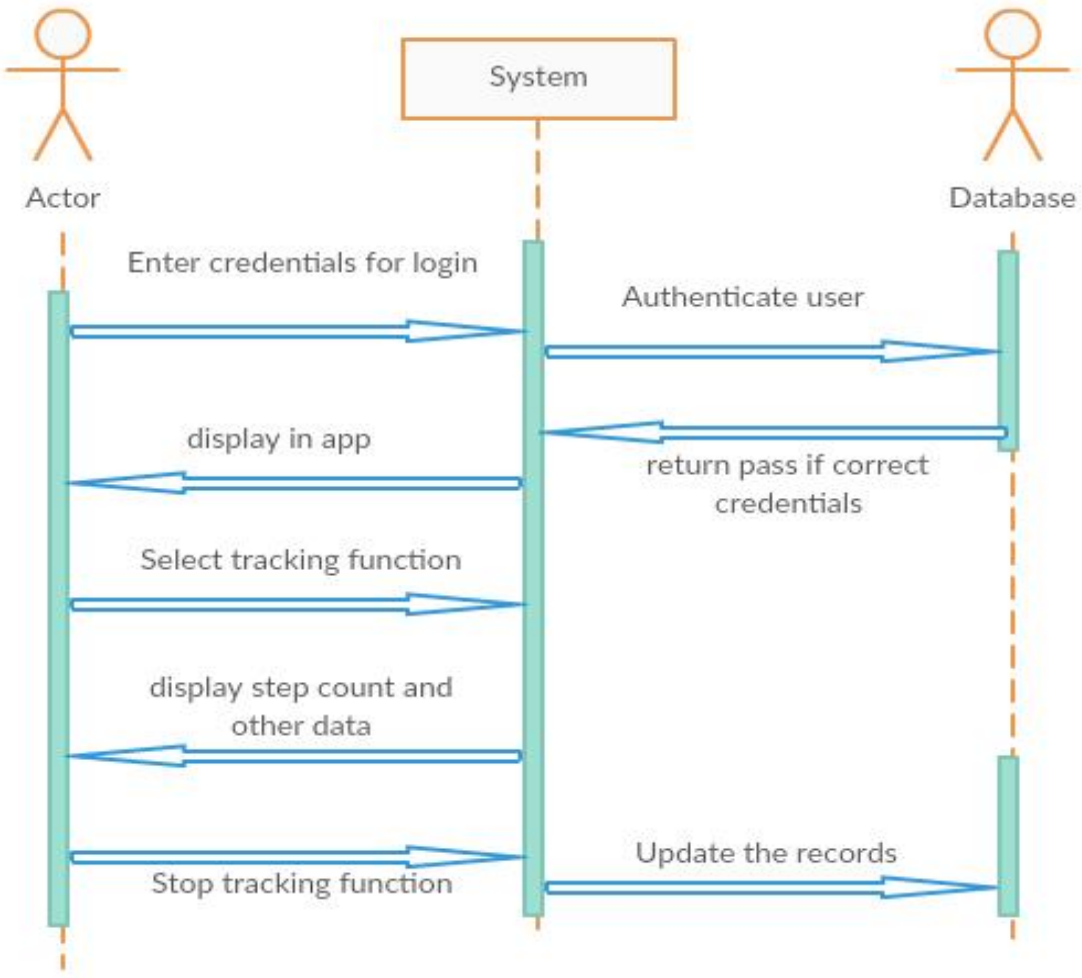
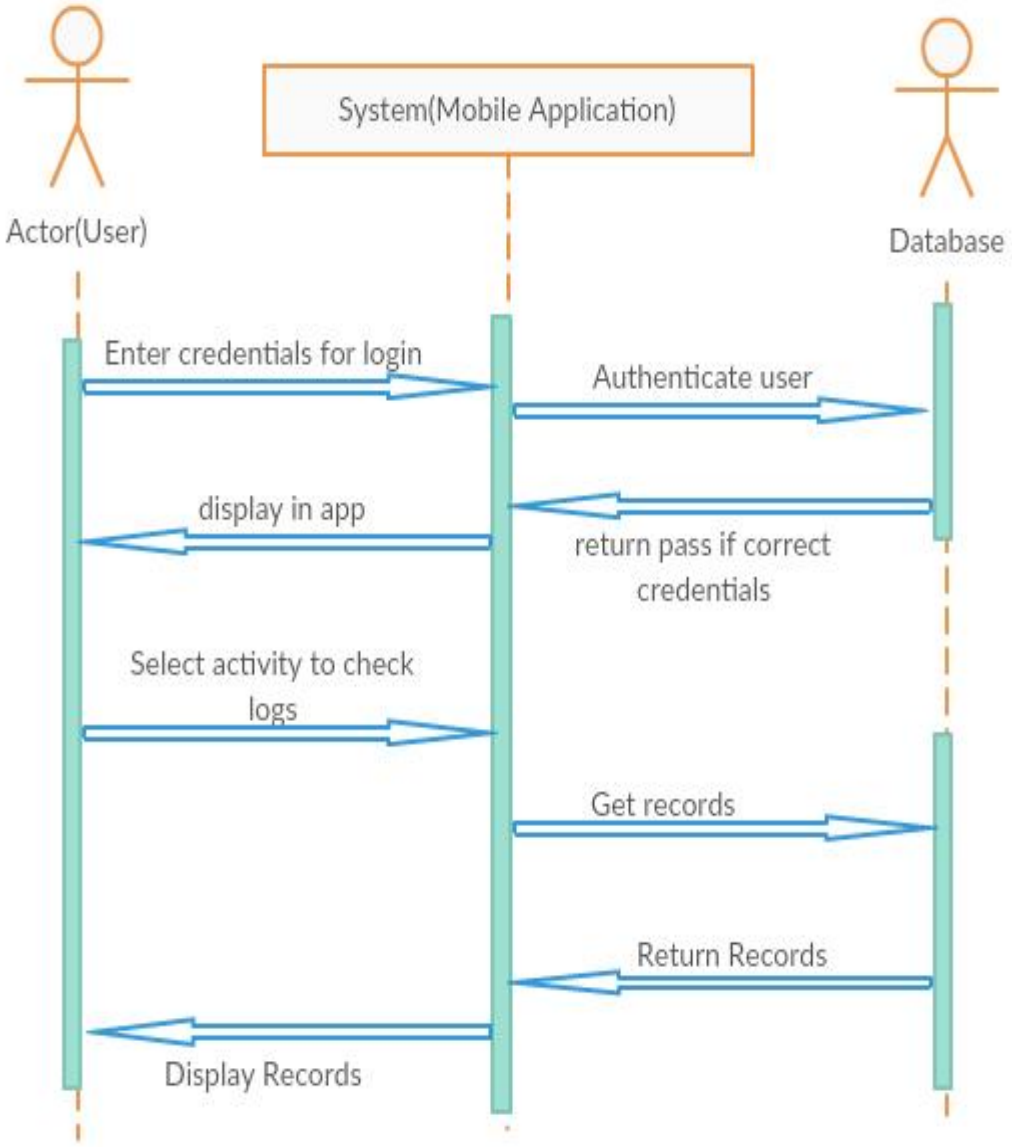


Figure 4.4.1- Sequence diagram for Use case UC-2:Health Data

Use Case UC-3 : Get vital signs Phone



Digure 4.4.2- Sequence diagram for Use case UC-3:Get Vital Signs Phone

Use Case UC- 4: Get Vital Signs Web

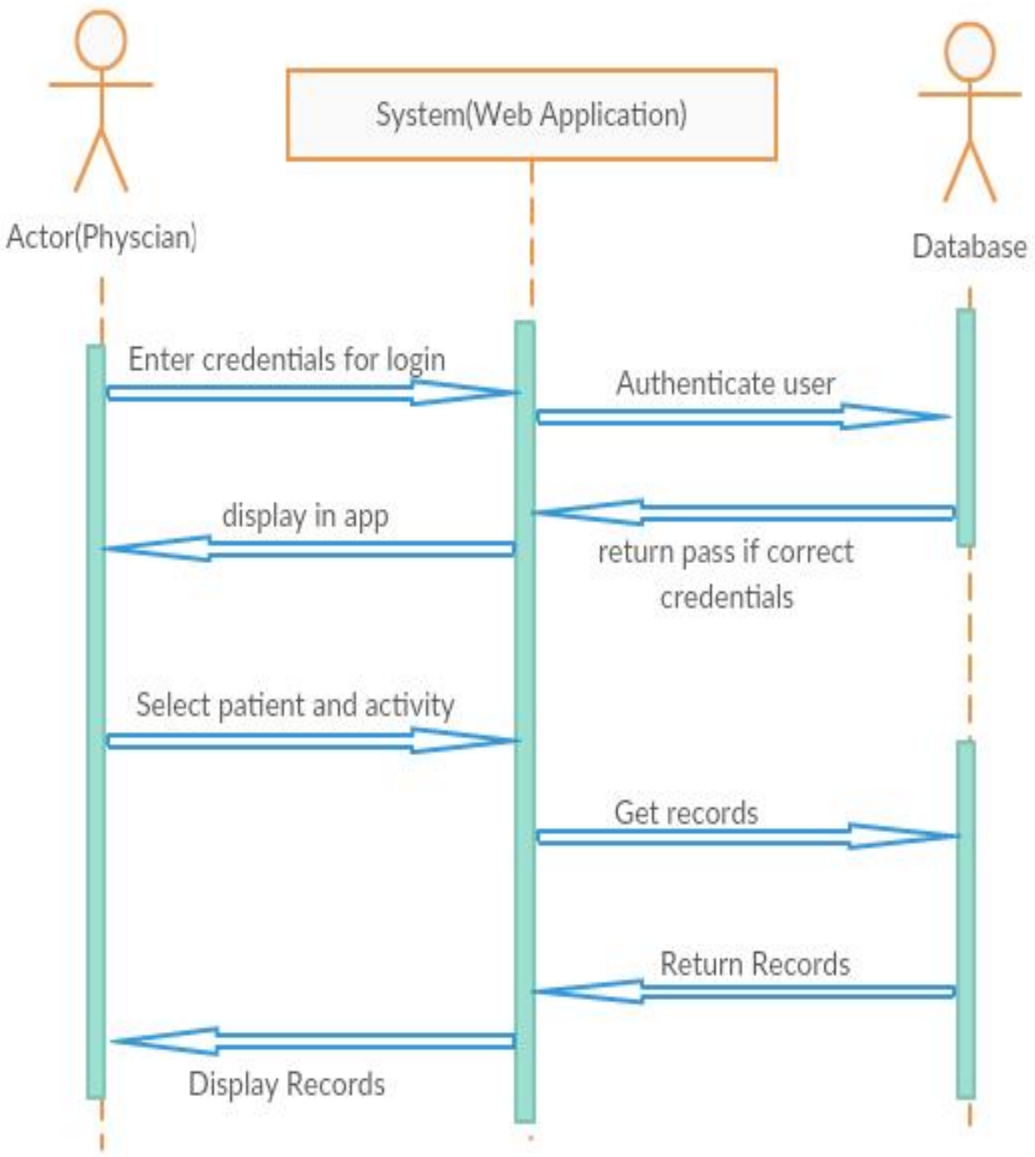


Figure 4.4.3- Sequence diagram for Use case UC-4:Get Vital Signs Web

Use Case UC-11: Physician Sign-in

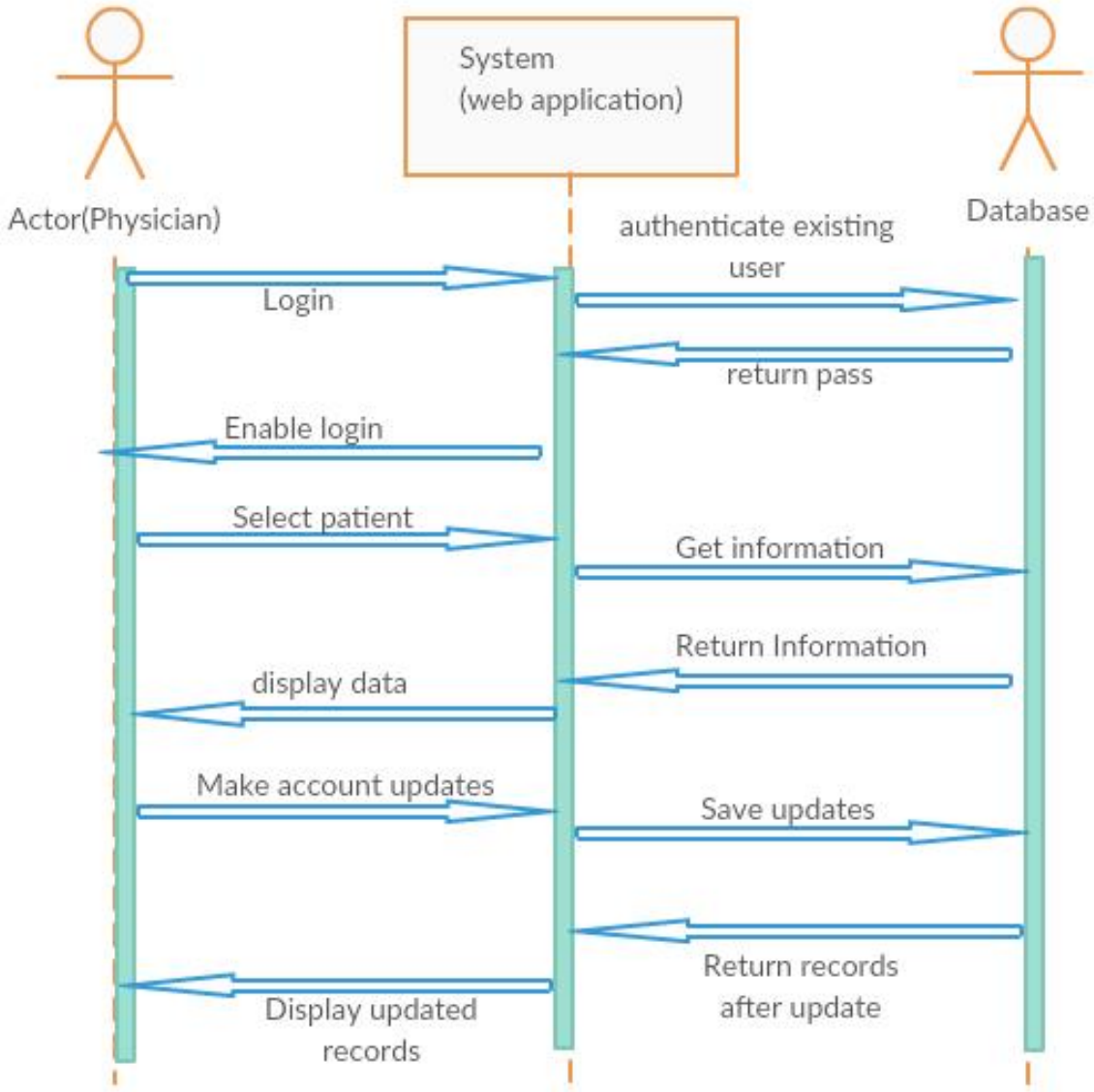


Figure 4.4.4- Sequence diagram for Use case UC-11:Physician Sign-In

Use Case UC-14: Check Recovery Time

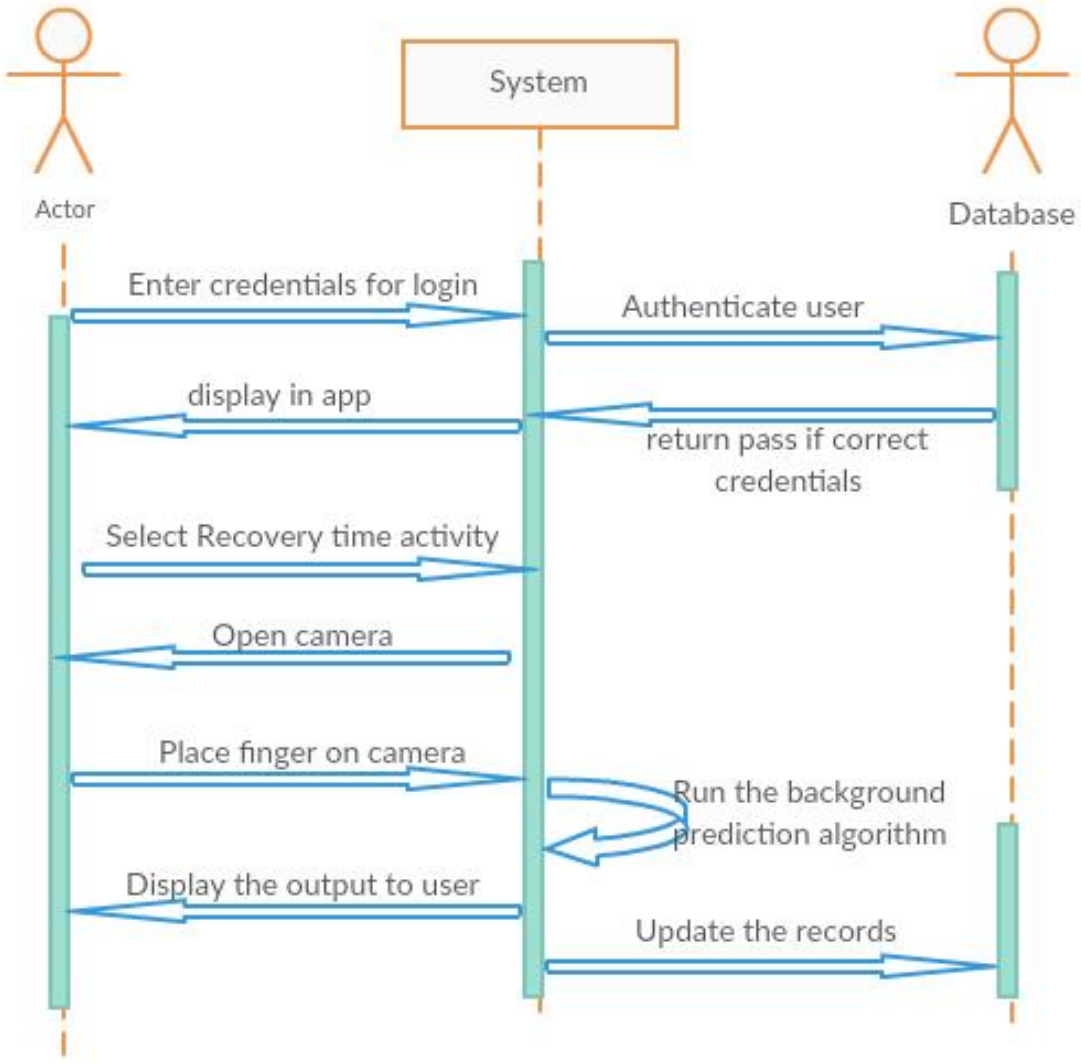


Figure 4.4.5- Sequence diagram for Use case UC-14:Check Recovery Time

5 USER INTERFACE SPECIFICATION

5.1 Preliminary Design

5.1.1 REGISTRATION AND LOGIN (UC-12 AND UC-13)

When the mobile app is installed, the user will be able to Register with minimal information. The details will be stored in a database which will help with authentication when the user tries to login. The login page will then verify the credentials provided and grant access if information is present in the database. All icons used for design are based on stock photos available online for reuse in non-commercial purpose.

Figure 5.1.1- Registration and Login UI

5.1.2 PROFILE INFORMATION (UC-13)

Once the User is logged into the application, they will be asked to enter some basic information. There will also be an optional page to enter the user's physician/personal trainer's information which will serve as authentication for the physician to be able to view their clients' information and activity. The activity is stored in an online database which will be shared in the admin's web application. This will then display the information to the authenticated physician.

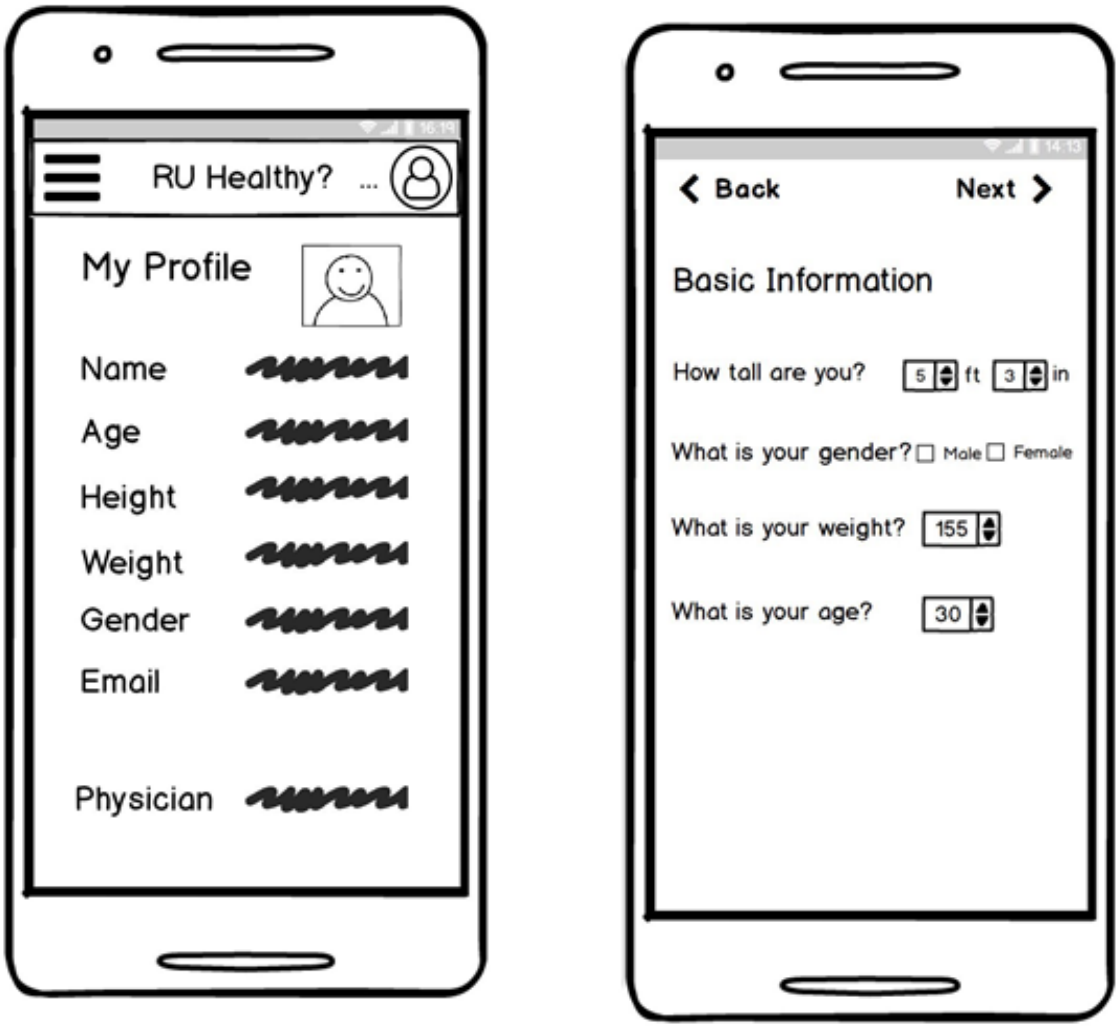


Figure 5.1.2- Profile Information and Basic Information UI

Figure 5.1.3 - Add Physician App UI

5.1.4 ACTIVITY SCREEN (UC-3)

User's current activity is displayed on the home screen and includes steps taken and calories burnt. Additionally, we will also display heart rate information. These details are regularly stored in the online database.

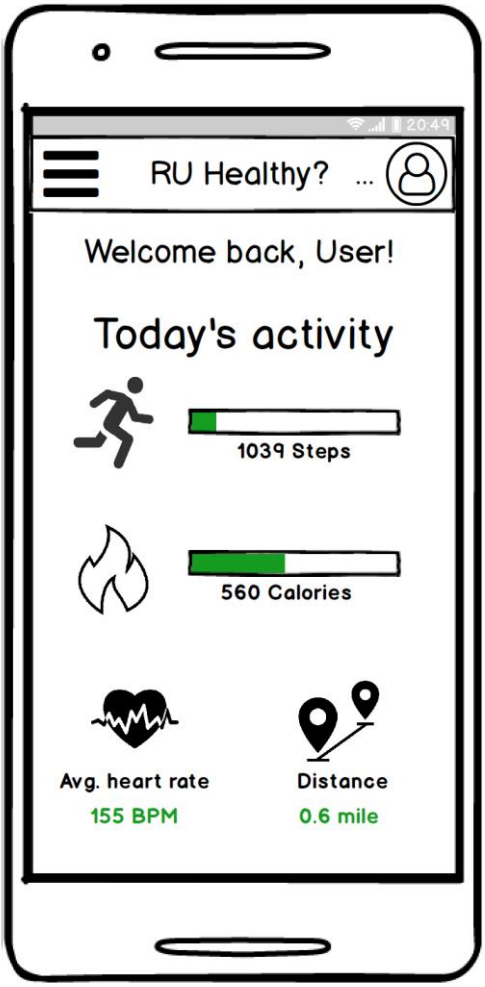


Figure 5.1.4 - Activity Monitor App UI

5.1.5 SCHEDULER (UC-7)

The User will be able to schedule or allocate some time for exercise every day. The app will then wait for the user to perform some kind of activity at the allotted time. When the app detects inactivity, using the mobile phone sensors, it will remind the User that it is time to exercise by sending a notification to the User.

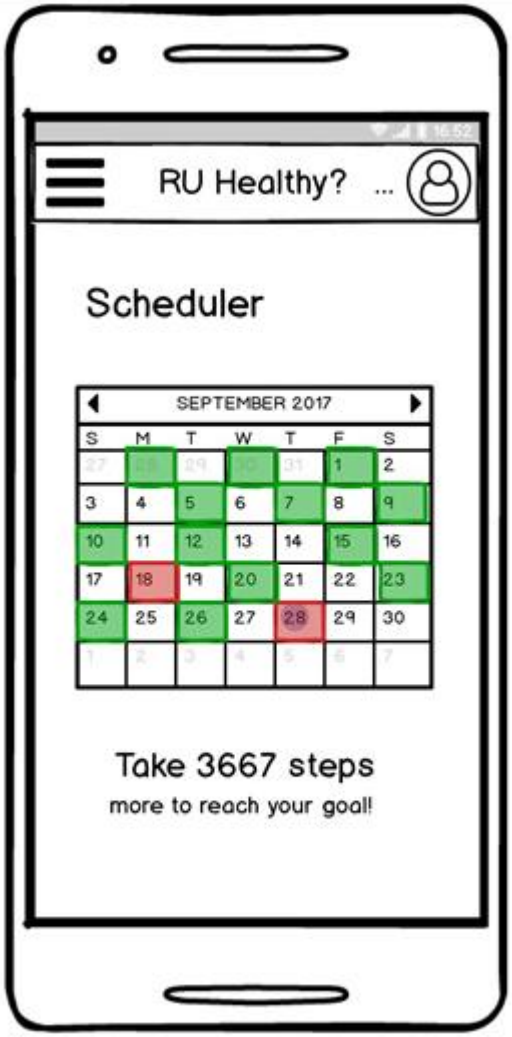


Figure 5.1.5 - Scheduler App UI

5.1.6 NAVIGATING BETWEEN PAGES (UC-3, UC-7)

When the user logs in to the app on his/her phone, the app welcomes the user with a screen that looks like figure 5.1.4. This screen allows the user to view the current summary of activity for that day. However, the user may require to navigate to the other pages of the app (e.g. Profile, Scheduler) or may just want to log out of the account. The user thus needs a navigation drawer to other pages to access the functions the app offers. The app allows the user to do so by clicking on the hamburger menu at the top left corner of the screen. Once clicked, a menu containing links to all pages of the app slides rightward and the screen looks like following:

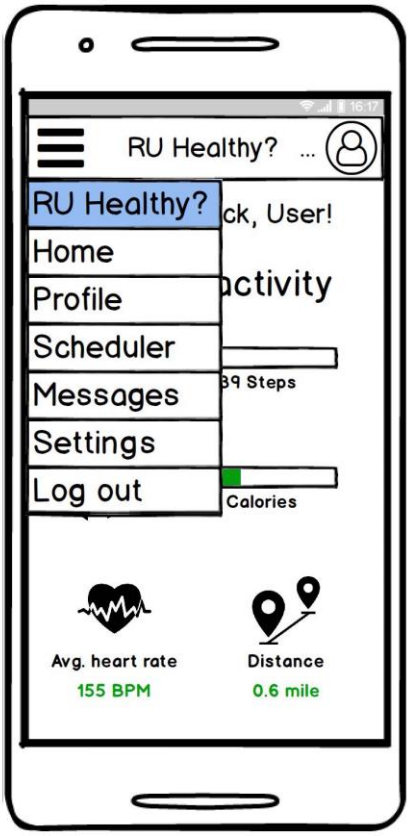


Figure 5.1.6 - Navigation Drawer

The admin (physician or personal trainer) will be able to view client profiles after sue authentication. Admin with these privileges can view client profiles and their activity on a particular day or set of days.

The User can login using his credentials to ensure privacy of data.

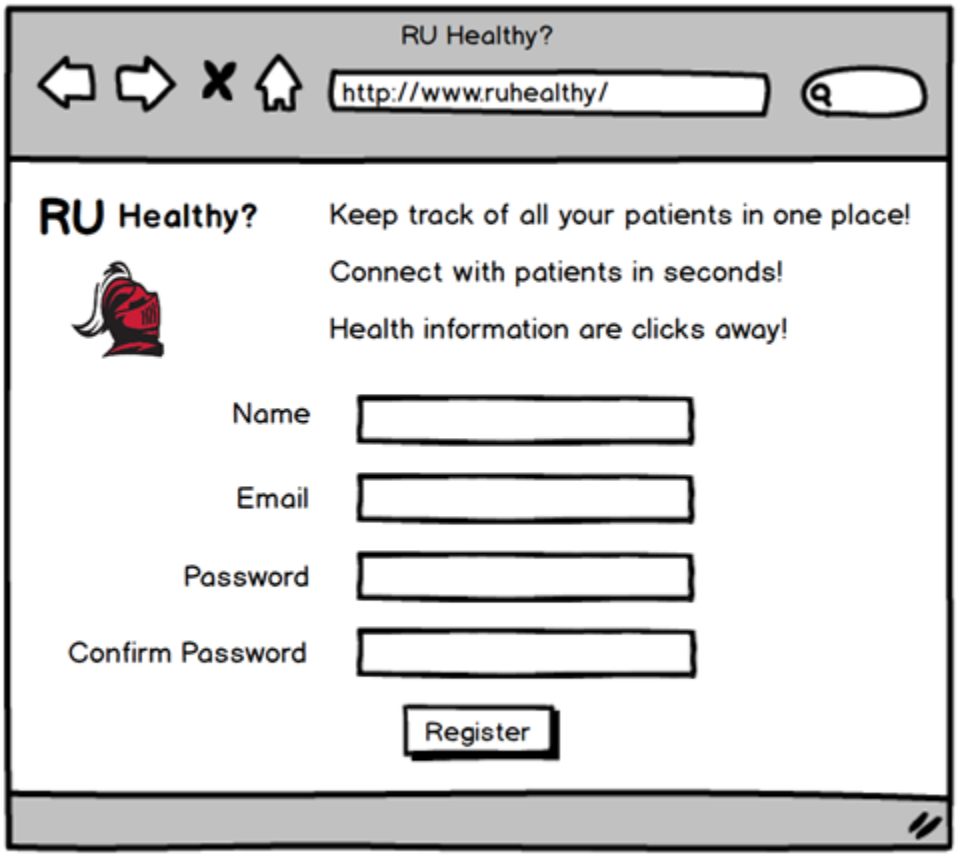


Figure 5.1.6 - Register Web UI

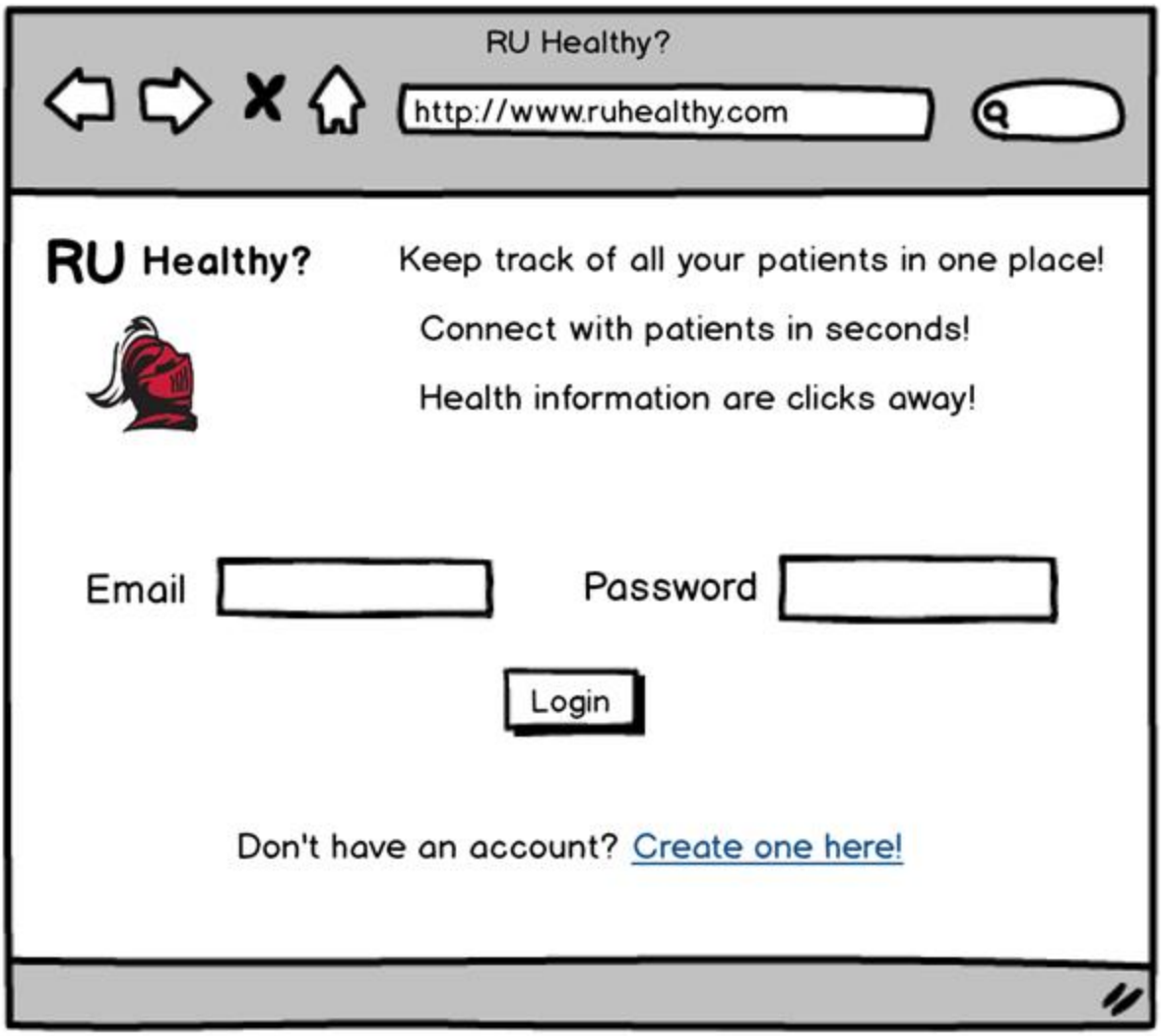


Figure 5.1.7 - Login Web UI

The admin will be able to individually view each client's profile which will include a number of options such as:

- the client's basic information
- the client's current activity

- a health record
  (This section will allow the admin to get a quick understanding of the client's status)
- a message tool
  (This section will let the admin send important messages to that particular client)
- an analysis section
  (This section will let the admin see a graphical representation of the client's

  activity status for a particular period of time. Furthermore, it will include some notes for future reference.)

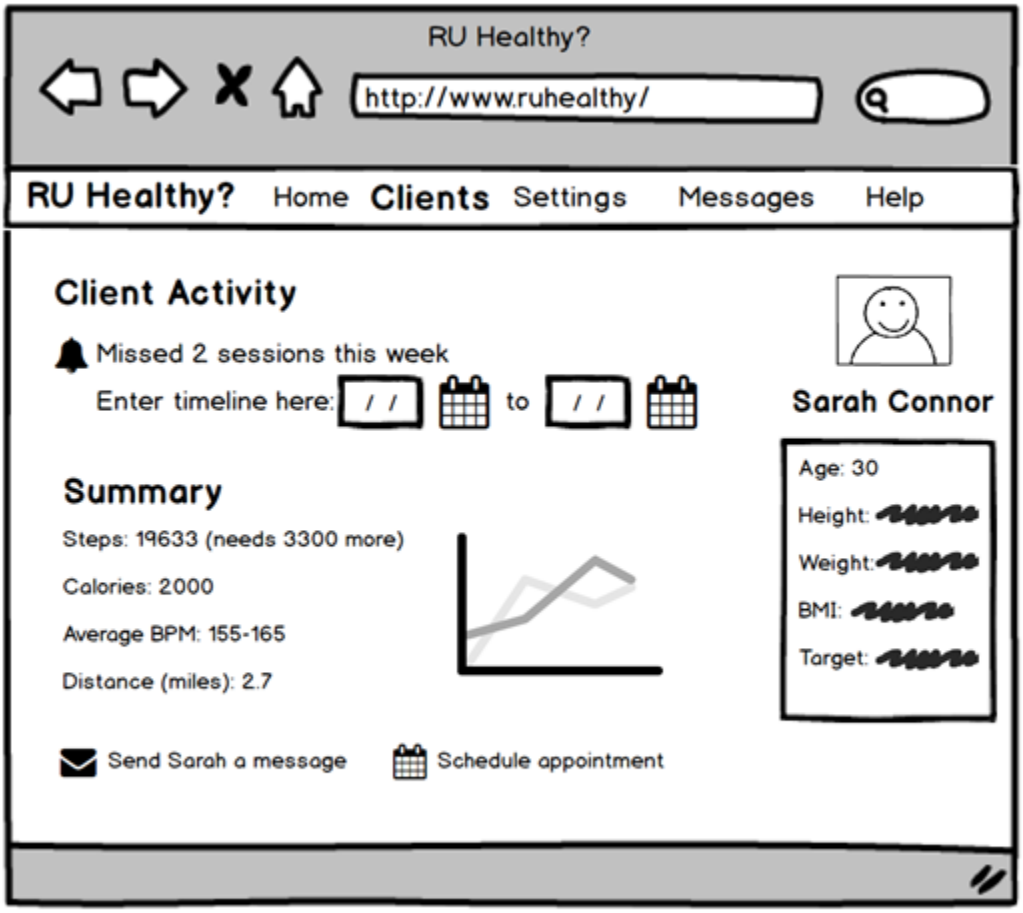


Figure 5.1.8 - Summary Web UI

5.2 Effort Estimation

USER EFFORT ESTIMATION

This section helps us understand how much average effort a user has to expend in order to access a functionality. While it is important to have a software that achieves a desired functionality in an optimized solution, it is also very important that we minimize user effort estimation.

Mobile Application:

In order to let the physician have access to the user's information, the user needs to register on the mobile app. Data entry may be a little high for this part of the app, but since it is a one-time activity, its effect on daily user effort is quite low.

Registration:

2 mouse clicks - "Register" button, "Submit" button

Data entry - 2 fields - email, password - At least 20 keystrokes (enter email ID, tab, enter password)

Login:

1 mouse click - "Log In" button

Data entry – 2 fields – email, password - At least 20 keystrokes (enter email ID, tab, enter password)

Profile:

1 mouse click – "Submit" button

Data entry – 7 fields – Name, Age, Gender, Height, Weight, Target heart rate, Target step count - At least 50 keystrokes (enter Name, tab, enter Age, tab, enter Gender, tab, enter Height, tab, enter Weight, tab, enter Target Heart Rate, tab, enter Target step count)

After Registration and login, user is asked to enter his Profile as shown above. This takes the user directly to the dashboard the next time he logs in. The Dashboard is nothing but a menu of available app

functionalities. The effort estimation to access any of the following activities is minimal and has been greatly reduced by adding this view of the list of functionalities.

Step count:

1 mouse click – "Step counter" icon – No mouse click needed to start detecting step count. But in order to view the step count, user needs to select "Step counter" from the Menu. The user can also see a Graph view of his progress so far.

Heart Rate:

1 mouse click – "Heart Rate" icon – By placing the finger on the camera, the user will see his heartbeat.

Recovery Rate and Prediction:

1 mouse click – "Recovery Rate" icon - By placing the finger on the camera, the user will see his heartbeat recovery rate after exercise and whether this falls in the "Normal" range or is "Higher than expected"

Scheduler:

3 mouse clicks – "Scheduler" icon, "add new" button to add a new exercise on the calendar, "save" button

Data entry – 2 fields – Name and Place of exercise

Select Physician:

2 mouse clicks – "Select Physician" icon, click on the physician's name to save the user details to the corresponding Physician's database.

Web Application:

The web application is exclusively for Physician's use. It is required that he/she registers to the application so that the authentication details are stored in the database.

Register:

2 mouse clicks - "Register" button, "Submit" button

Data entry - 2 fields - email, password – At least 20 keystrokes (enter email ID, tab, enter password)

Login:

1 mouse click - “Log In” button

Data entry – 2 fields – email, password - At least 20 keystrokes (enter email ID, tab, enter password)

After doing the above, the physician has access to client records. This is again done using minimal number of clicks and data entry.

Access Patient List:

1 mouse click – By clicking on this button, the physician can access a list of patient names and IDs

Access Patient details:

1 mouse click – By clicking on this button, physician can access that particular patient’s medical records as obtained from the mobile app. This will be real-time data and we are using Firebase to make this happen.

Overall, we hope to present the applications to the user so they can use them with minimal effort. We can see that Registration and Login are the main data entry points but since they’re a one-time activity, they don’t contribute much to the overall effort estimation. The everyday use of the app is done with minimal effort. We estimate the User Effort Estimation to be 90% navigation clicks and only 10% data entry.

EFFORT ESTIMATION USING USE CASE POINTS

5.2.1 Unadjusted Use Case Points (UUCP)

We calculate UUCP by taking a sum of Unadjusted Actor Weights and Unadjusted Use Case Weights.

ACTOR	COMPLEXITY	WEIGHT
Web Application	Medium	2
Database	Medium	2
RU Healthy Mobile App UI	Simple	1
Users	Complex	3

Table 5.2.1.1- Unadjusted Actor Weight

UAW = 2+2+1+3 = 8

We allocate points to each of our Use Cases using the following table:

Use Case complexity	Weight
Simple	5
Medium	10
Complex	15

Table 5.2.1.2 - Unadjusted Use Case Weights

Based on our casual description of Use cases, we allocate points to each of our Use Cases (13 in number.)

We then calculate the UUCW by summing up points to all Use cases.

Weights have been allocated in the order of mention is the section 4.2.3 Casual description of Use cases. (i.e., from UC-1 to UC-13 in that order)

UUCW = 5 + 10 + 15 + 15 + 5 + 5 + 10 + 5 + 10 + 5 + 10 + 5 + 5 = 105

Therefor, UUCP = UAW + UUCW = 8 + 105 = 113

5.2.2 Technical Complexity Factor (TCF)

TECHNICAL FACTOR	DESCRIPTION	WEIGHT	COMPLEXITY	TOTAL
T1	Interface that User finds easy to use and less cluttered	3	1	3
T2	Ease of access to historical activity record- Database	4	2	8
T3	Accurate tracking using sensors	3	2	6
T4	Web app Security is important as it concerns health records	3	1	3

T5	Adjustments to cater to battery saving, memory management - minor	1	2	2

Table 5.2.2- Technical Complexity Factor

Technical Factor Total (TFT) = 3+8+6+3+2 = 22

TCF = C1+ C2 * TFT where C1 and C2 are constants with values, C1= 0.6, C2 = 0.01

TCF = 0.6 + 0.01 * 22 = 0.82

5.2.3 Environmental Complexity Factors (ECF)

ENVIRONMENTAL FACTOR	DESCRIPTION	WEIGHT	IMPACT	TOTAL
E1	Familiarity with health monitoring and defined problem	1	1	1
E2	Beginner knowledge of Android development approach	2	2	4

E3	Knowledge of Web application development	1.5	2	3
E4	Teamwork and motivation	2	1	2
E5	Meeting specified requirements	3	2	6

Table 5.2.3- Environmental Complexity Factors

Environmental Factor Total (EFT) = 16

ECF = C1 + C2*EFT

Where C1 and C2 are constants with the values C1=1.4, C2=-0.03

ECF = 1.4 + (-0.03 * 16) = 0.92

5.2.4 Total Use Case Points and Final Effort

Finally, we calculate the total use case points by the equation,

UCP = UUCP * TCF * ECF = 113 * 0.82 * 0.92 = 85.25

Final Effort = UCP * Conversion factor

We assume a conversion factor of 28 hours

Therefore, Final Effort = 85.25 * 28 = 2387 hours

6 DOMAIN ANALYSIS

6.1 Domain Model

For the most important fully-dressed use cases specified above, we first start extracting the responsibilities of each one, and we proceed to assign concept names. The following tables contain this information.

6.1.1 CONCEPT DEFINITIONS

Table 6.1.1.1 - Concepts And Responsibilities Extraction For UC-2: Health Data

Responsibility Description	**Type**	**Concept Name**
Controls and coordinates of all other concepts associated with use cases.	D	Controller
Contains all the Patient's data that is, was, and will be detected during the Patient's activity.	K	Data Container
Operates the process of user authentication with the Database.	D	Database Authenticator
Operates the device tracking functionality by disabling and enabling it when it is necessary.	D	Tracking Enabler
Operates the process of managing Local Database entries and update them with the Online Database.	D	Database Manager

Operates the process of managing notifications	D	Notifications Enabler
Operates the process of data detecting from the device sensors with the help of “Tracking Enabler” concept.	D	Sensor Operator

Table 6.1.1.2 - Concepts And Responsibilities Extraction For UC-3: Get Vital Signs Phone

Responsibility Description	**Type**	**Concept Name**
Controls and coordinates of all other concepts associated with use cases.	D	Controller
Manages and specifies the Patient search parameters and the Patient’s activity logs retrieval.	K	Search Manager
Provides the connections between the Patient search inputs and the Offline Database and retrieve the matched records from said Database	D	Database Operator
App interface shows available activity records of patient.	K	Interface
Renders and projects the retrieved records from the Database and sends it to the User Application GUI	D	Pagemaker

Filters and applies the specific Patient search criteria for the retrieved data from the Database .	D	Database Postprocessor
Operates the process of managing and sending notifications to Patient if he does not have valid vital signs.	D	Activity Notifier

Table 6.1.1.3 - Concepts And Responsibilities Extraction For UC-4: Get Vital Signs Web

Responsibility Description	**Type**	**Concept Name**
Control and coordinates of all other concepts associated with use cases.	D	Controller
Manages and specifies the Physician search parameters and the Patient's activity log retrieval.	K	Search Manager
Provides the connections between the Physician search inputs and the Online Database and retrieve the matched records from said Database	D	Database Operator
Renders and projects the retrieved records from the Database and sends it to the Web Page UI	D	Pagemaker
App interface shows available activity records of patient.	K	Interface
Filters and applies the specific Physician search criteria for the retrieved data from the Database .	D	Database Postprocessor

Operates the process of managing and sending notifications to Physician if Patient does not have valid vital signs.	D	Activity Notifier

Table 6.1.1.4 - Concepts And Responsibilities Extraction For UC-11: Physician Sign-in

Responsibility Description	**Type**	**Concept Name**
Control and coordinates of all other concepts associated with use cases.	D	Controller
Manages and specifies the Physician search parameters	K	Patient Searcher
Manages and specifies the operation (add/edit/remove) to be performed by the Physician	K	Patient Operator
Compares search input with Online Database to verify if user is/is not in said Database	D	Patient Comparator
Keeps the Online Database synced accordingly to the operation specified by the concept “Patient Operator”	D	Database Manager
Alerts the Physician if Patient already is/is not in the Online Database	D	Patient Notifier

Table 6.1.1.5 - Concepts And Responsibilities Extraction For UC-14: Check Recovery Time

Responsibility Description	Type	Concept Name
Controls and coordinates of all other concepts associated with use cases.	D	Controller
Manages and specifies the Patient search parameters.	K	Patient Operator
Provides the connections between the Patient search inputs and the Online Database and retrieve the matched records from said Database	D	Database Operator
App interface shows the recovery time and the prediction output to the user	K	Interface
Opens the camera for measuring the recovery time	K	Camera Operator
Filters and applies the specific Patient search criteria for the retrieved data from the Database .	D	Database Postprocessor
Takes the records from the Database and outputs the processes those records for final output display to user.	D	Prediction Function

6.1.2 ASSOCIATIONS

Associations are used to *which concept* needs to work together and *why* do they need to work together. In the following tables, we derived associations for principal use cases of our project (UC-2, UC-3, UC-4, UC-11) based on the fully dressed descriptions in section 4.3.4.

Table 6.1.2.1 - Identifying associations for use case UC- 2: Health Data

Concept pair	**Association description**	**Association name**
Controller ↔ Tracking Enabler	Controller passes request to Tracking Enabler to start tracking user motion via sensor and receives confirmation on Tracking initiation.	Conveys tracking request
Tracking Enabler ↔ Sensor Operator	Tracking Enabler sends signal to sensor for activation and receives confirmation.	Conveys sensor activation request
Sensor Operator ↔ Database Manager	Sensor sends sensor data to Database Manager for processing and storage.	Provides data
Database Manager ↔ Data Container	Database Manager sends data received from sensor to Data Container for local data storage.	Stores data
Controller ↔ Notification Enabler	Controller passes request to Notification Enabler for notifications based data stored in Data container.	Requests notification
Notification Enabler ↔ Data	Notification Enabler requests	Requests processed data

Container	for processed data from Data Container.	
Controller ↔ Database Authenticator	Controller requests access to Online database for updating data and receives access permission.	Requests access
Controller ↔ Database Manager	Controller sends signal to Database manager to send current to Online Database for storage.	Requests save

Table 6.1.2.2 - Identifying associations for use case UC- 3: Get Vital Signs Phone

Concept pair	**Association description**	**Association name**
Controller ↔ Pagemaker	Controller passes request to Pagemaker and Pagemaker prepares available activity records for display.	Conveys display request
Pagemaker ↔ Database Operator	Pagemaker requests for available records and receives data retrieved by Database Operator.	Requests data
Pagemaker ↔ Interface	Pagemaker prepares the received data for display on the App Interface.	Display data
Pagemaker ↔ Search Manager	Pagemaker conveys patient's search request parameters to Search Manager and receives retrieved results for display.	Requests search operation
Search Manager ↔ Database	Search Manager sends	Conveys specified search

Postprocessor	specified search query based on search criteria to Database Postprocessor.	query
Pagemaker ↔ Activity Notifier	Pagemaker requests notification on patent's vital sign and receives them based on retrieved data from database.	Requests notification

Table 6.1.2.3 - Identifying associations for use case UC- 4: Get Vital Signs Web

Concept pair	**Association description**	**Association name**
Controller ↔ Pagemaker	Controller passes request to Pagemaker and Pagemaker prepares available activity records for display.	Conveys display request
Pagemaker ↔ Database Operator	Pagemaker requests for available records and receives data retrieved by Database Operator.	Requests data
Pagemaker ↔ Interface	Pagemaker prepares the received data for display on the Web Interface.	Display data
Pagemaker ↔ Search Manager	Pagemaker conveys Physician's search request parameters to Search Manager and receives retrieved results for display	Requests search operation
Search Manager ↔ Database Postprocessor	Search Manager sends specified search query based on search criteria to Database	Conveys specified search query

	Postprocessor.	
Pagemaker ↔ Activity Notifier	Pagemaker requests notification on patient's vital sign and receives them based on retrieved data from database.	Requests notification

Table 6.1.2.4 - Identifying associations for use case UC-11: Physician Sign-in

Concept pair	**Association description**	**Association name**
Controller ↔ Patient Searcher	Controller sends search query to database to retrieve information of specific Patient.	Requests search operation
Patient Searcher ↔ Patient Comparator	Patient searcher compares search query to available patient list to verify and sends update.	Requests verification
Controller ↔ Patient Operator	Controller requests selected operation (add, remove or delete) on patient information.	Requests editing permission
Patient Operator ↔ Database Manager	Based on Physician's requested operation, Patient operator sends editing commands to database manager.	Provides editing command
Patient Comparator ↔ Patient Notifier	Patient comparator sends notification to Patient Notifier in case of not finding desired patient in the available list.	Send notification

Table 6.1.2.5 - Identifying associations for use case UC-14: Check Recovery Time

Concept pair	Association description	Association name
Controller ↔ Patient Operator	Controller sends search query to database to retrieve information of specific Patient.	Requests search operation
Patient Operator ↔ Database Operator	Patient Operator sends the request to the Database Operator for retrieving specific records.	Requests verification
Controller ↔ Camera Operator	Controller requests the camera operator to open the camera for checking the recovery time	Requests camera permission
Controller ↔ Data Postprocessor	Based on the data returned to the controller from the camera operator, the data is sent to Data PostProcessor to do necessary processing.	Provides data for processing
Data Processor↔Prediction Function	After processing the data , the prediction function runs the prediction algorithm.	Runs the prediction algorithm.
Prediction Function↔ Interface	After the algorithm is run, the result is sent to Interface to display to user.	Sends result to display.

6.1.3 ATTRIBUTE DEFINITIONS

Table 6.1.3 - Attribute Definitions

CONCEPT	ATTRIBUTE	ATTRIBUTE DEFINITION
Controller	Central coordinator	Controls and coordinated all operations performed for a certain action performed by users
Sensor Operator	Mobile Application UI and functions	Enables and collects tracking data via phone sensors
Search Manager		Allows user to find specific physician/trainer
Tracking Enabler		Allows user to enable tracking-GPS
Notifications Enabler		Allows user to enable and change notification settings
Data Container	Database Operations	Stores mobile user data – profile and activity
Database Authenticator		Verifies user authenticity
Database Manager		Ensures continuous update of user data

Database Operator		Provides a connection between the database and the Web user
Database Postprocessor		Allows for specific data outputs based on search criteria – querying
Interface	Web Application UI	User Interface for the Web user
Pagemaker		Data processing and display for Web user
Patient Searcher	Web Application functions	Allows physician to search for specific records
Patient Operator		Allows physician to add or delete patient records
Patient Notifier		Alerts the Physician of incorrect input/search criteria

6.1.4 Traceability Matrix

Below is the traceability matrix which specifies how the different cases map to the domain concepts:

DOMAIN CONCEPTS	UC1	UC2	UC3	UC 4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14
Controller	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Sensor Operator	X	X												
Search Manager							X							
Tracking Enabler	X	X	X											
Notifications Enabler				X	X				X	X				
Data Container		X	X		X							X	X	
Database Authenticator													X	
Database Manager		X	X	X	X	X	X	X	X	X	X		X	
Database Operator				X			X		X	X	X			X
Database Postprocessor		X	X	X	X	X	X	X	X	X	X		X	X
Interface				X			X		X	X	X			X

Pagemaker				X			X		X	X	X			
Prediction Function														X
Patient Searcher				X			X		X		X			
Camera Operator														X
Patient operator											X			X
Patient Notifier											X			

6.2 System Operation Contracts

System Operation Contracts are a way of documenting complex system sequence designs. Each operation contract describes what each message will need to accomplish.

CO-1: Register User

Cross References: UC-2, UC-3

- **Preconditions:** None
- **Postconditions:** User profile is linked to database and all user information is stored in database.

CO-2: User Login

Cross References: UC-2, UC-3

- **Preconditions:** User is already registered.
- **Postconditions:** The account features are accessed by the user.

CO-3: View Records:

Cross References: UC-3, UC-4, UC-11

- **Preconditions:** User has an account.
- **Postconditions:** None.

CO-4: Edit User Account

Cross References: UC-4, UC-11

- **Preconditions:**

1. Physician is logged in his/her account.
2. User to be edited/deleted has an account.

- **Postconditions:** Database updated according to made changes.

CO-5: Feedback by Physician

Cross Reference: UC-4

- **Preconditions:**

1. Physician is signed into the web application.
2. Physician has received data of requested user from the online database.

- **Postconditions:**

1. Doctor writes comments for the user based on his/her report.
2. User is notified about the comments.

6.3 Mathematical Models

6.3.1 CALORIE CALCULATIONS

The System will have the capability to find the number of calories burned by the user based on certain factors:

1. Gender
2. Body Weight.

3. Average Heart Rate

4. Time

One of the more standard and most accurate ways to calculate the equation is to use the calorie expenditure formula below. It comes from the Journal of Sports Sciences and provides a formula for each gender.

We use the following equations to calculate calories burnt by the user:

Men:

$$Calories\ Burned = [(Age * 0.2017) — (Weight * 0.09036) + (Heart\ Rate * 0.6309) — 55.0969] * Time / 4.184.$$

Women:

$$Calories\ Burned = [(Age * 0.074) - (Weight * 0.05741) + (Heart\ Rate * 0.4472) - 20.4022] * Time / 4.184.$$

Note : Heart Rate = average heart rate.

A less accurate Formula would be:

$$Calories\ Burned = 0.75 * your\ weight\ (in\ lbs.)$$

These equations are based on the article titled “Energy Expenditure of Walking and Running” published in Medicine & Science in Sport & Exercise in 2004 [20].

6.3.2. EQUIVALENT DISTANCE CALCULATION

The System will be able to calculate the distance in miles based on the number of Steps that the user did perform ,On the basis that a person of average height has a stride length of about 2.1 to 2.5 feet we can say that approximately 2,000 steps for 1 Mile so we can say that

*Number of Miles = Number of Steps performed *0.0005. []*

These calculations was published on 2004 by the ACSM(American College of Support and Medicine) and was certified by the American Nutrition Association.

6.3.3 HEART RATE

Normal heart rate can be in the range of *50% * Maximum heart rate – 85% * Maximum heart rate.*

Where Maximum heart rate is approximately equal to 220 – *Age.*

6.3.4. HEART RECOVERY RATE

Heart rate is an important factor to assess the health of the user. We plan to use the heart rate data collected by the app to detect anomalies in heart rate after exercise. It is a documented medical fact that heart rate, once up during exercise, will to come down to the user's normal heart rate as a sudden drop before stabilizing. This drop is usually about 20 beats per minute in the first minute and about 15 beats per minute in the later couple of minutes. Anything less than 12 beats in the first minute, is a potential heart risk.

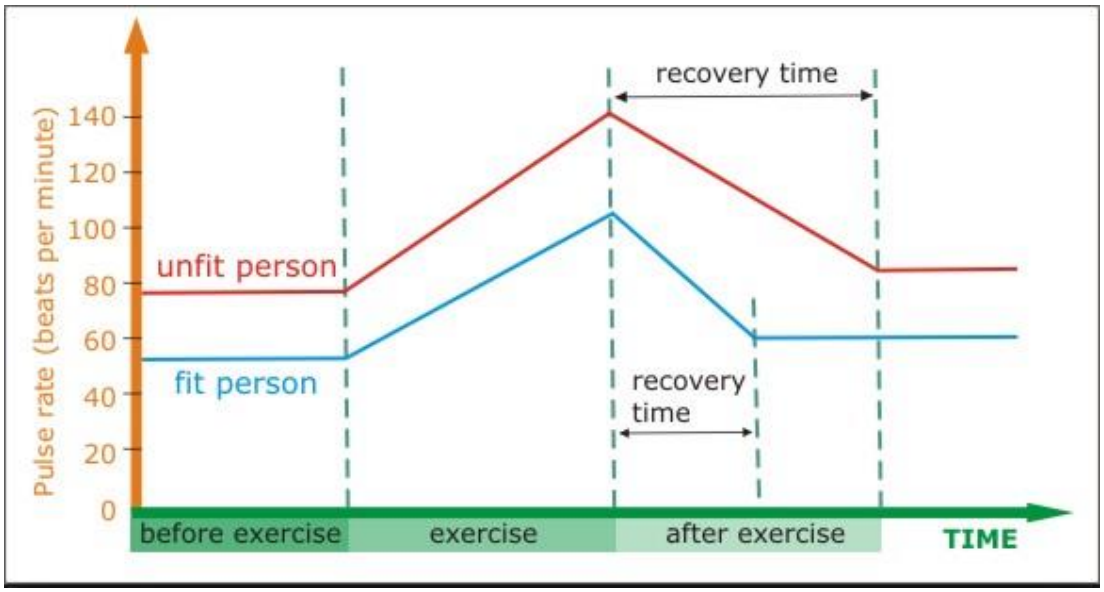


Fig.6.4.1 Heart rate pattern during and after exercise

The more fit a user is, faster his heart rate will get back to normal after exercise. Since the amount and intensity of exercise is an important factor in this recovery rate (the more you exercise, more time it takes to recover to normal rate), we make the assumption that the User's exercise pattern will indicate his heart recovery rate. Moreover, if a physician wants to monitor a patient who is recovering from a surgery or illness, this will be a useful analytical tool to show the user's recovery rate pattern and alert the doctor/user when there has been a series of abnormalities. We plan to use Bayesian curve fitting to obtain this pattern and predict the next heart rate. Bayesian Prediction is a useful statistical tool for short term prediction.

Bayesian prediction takes into account a trend that is being followed, and predicts the next values.

We have a data set of n points, $\mathbf{X} = \{x_1, x_2, \ldots\ldots, x_n\}$, Bayesian prediction will fit the given data points into a curve and predict the next data point x_{n+1} based on the pattern. Since we assume that the user's exercise patterns are different every day, Bayesian prediction would help predict the next day's recovery rate.

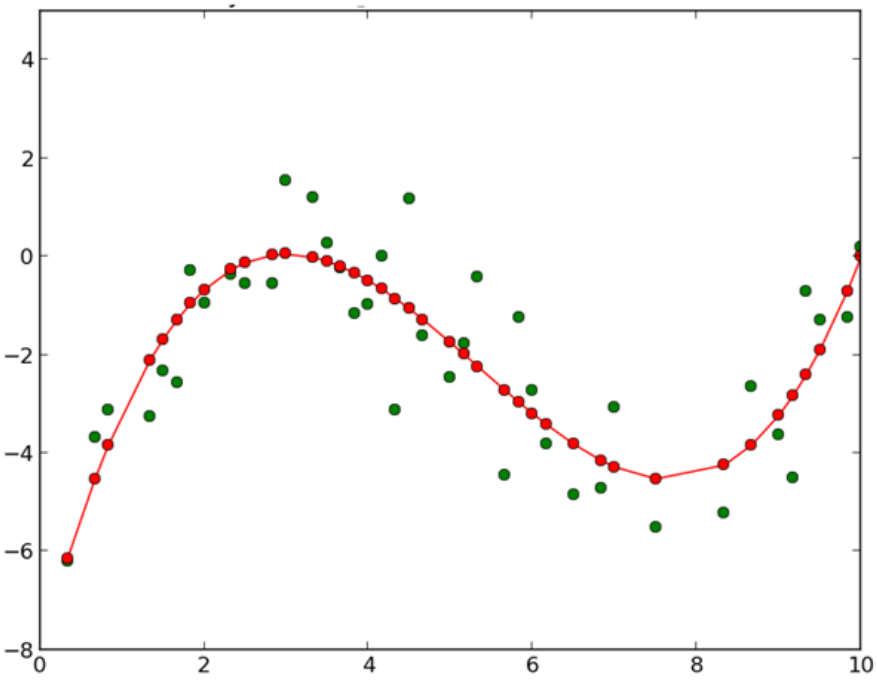


Fig.6.4.2 Bayesian curve fitting with a set of data points

Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a "likelihood function". Bayesian inference computes the posterior probability according to Bayes' theorem:

$P(Y \mid X)$ = , where $p(X) = \sum p(X|Y)\, p(Y)$

The physician can then use this analysis to determine whether the patient’s progress is as expected.

7 INTERACTION DIAGRAMS

7.1 UML Diagrams

When drawing these more detailed diagrams, we always tried to follow the most important design principles at the objects level: *Expert Doer*, *High Cohesion* and *Low Coupling*.

UC-2: Health Data

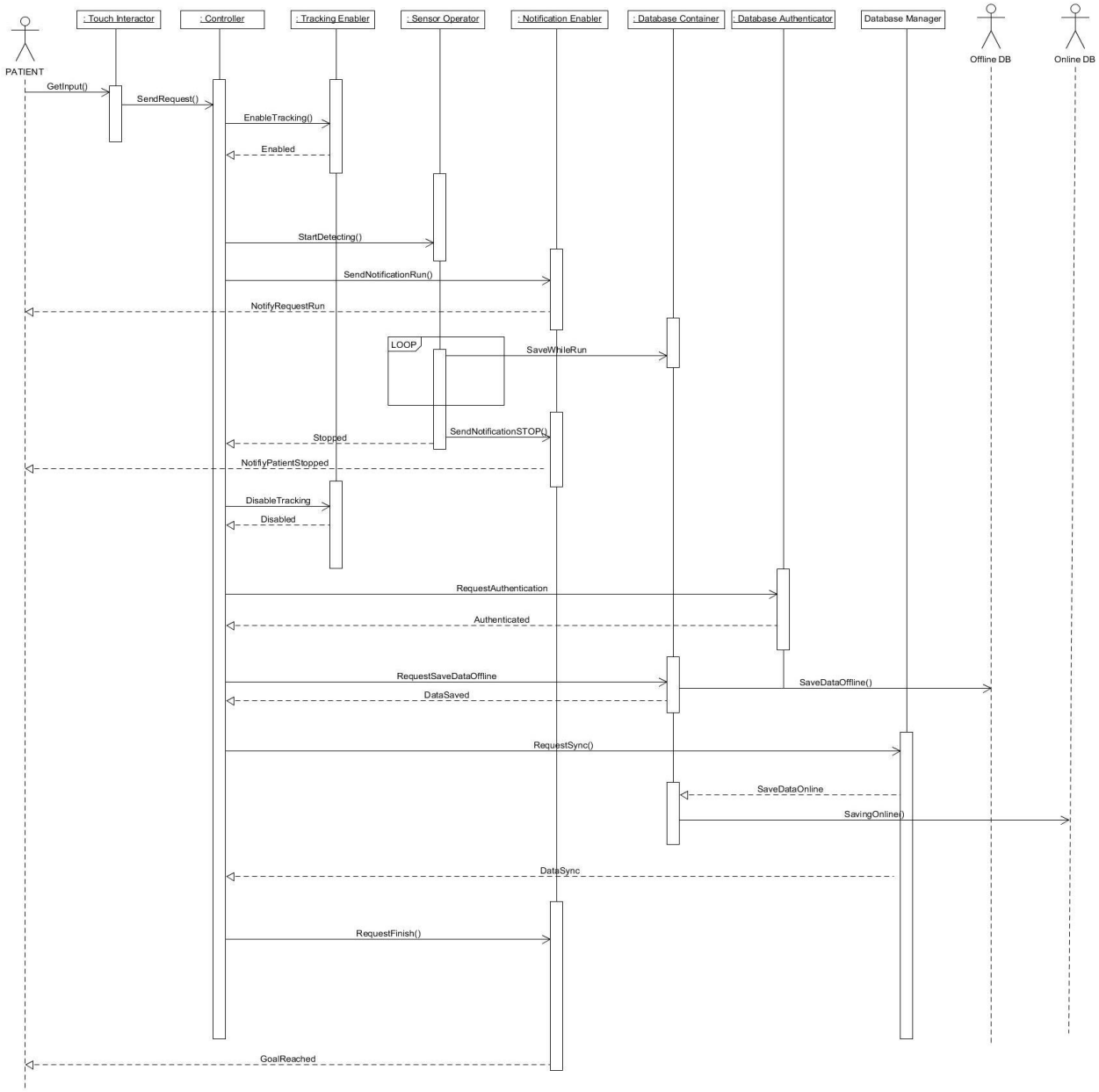


Figure 7.1.1 Health Data

The figure above shows the diagram for the main use case UC-2: Health Data, which monitors the Patient's activity. While sketching the design, we realized that we were missing a conceptual software object that allows direct interaction with the Patient. Thus, we added the concept *"Touch Interactor",* and assigned it the said responsibility. The Patient then touches the screen to start, and *Touch Interactor* communicates this to *Controller*. Even though *Controller* seems to handle several communications, we favored the Expert Doer principle for it because after all, Controller needs to know everything at all times to be able to delegate tasks effectively. If the Patient wants to start monitoring his activity, *Controller* asks *Tracker Enabler*, who acts like a switch, giving permission for *Sensor Operator* to start detecting. *Controller* then allows *Notification Enabler* to start sending notifications to the Patient about his current activity.

By the Expert Doer principle, we keep observing that every concept has its specific task, and they only communicate through *Controller*, in accordance with the Low Coupling principle. We analyzed the option of letting *Controller* perform the *Tracking Enabler* task, but in order to keep a High Cohesion, we ended up using the former.

Sensor Operator is going to continuously obtain the data and send it to *Database Container*. When the exercise finishes, the loop is over and Controller is notified, along with the Patient. Another option here is to let Notification Enabler communicate directly with *Tracking Enabler*, but *Controller* needs to know that the activity finished anyway, and to favor the design principles, so we decided not to go with the option.

Finally, *Controller* starts the process of authenticating and storing the data to the Offline Database, and request syncing online. When this is done, the Patient gets notified for the last time that all of his information was saved online.

UC-3: GetVitalSigns - Phone

The following figure explains the interaction between concepts for the next important use case, UC-3: Get Vital Signs Phone. Here, the main goal is to allow the Patient retrieve their information from the local database, so he/she can access his past activities and records.

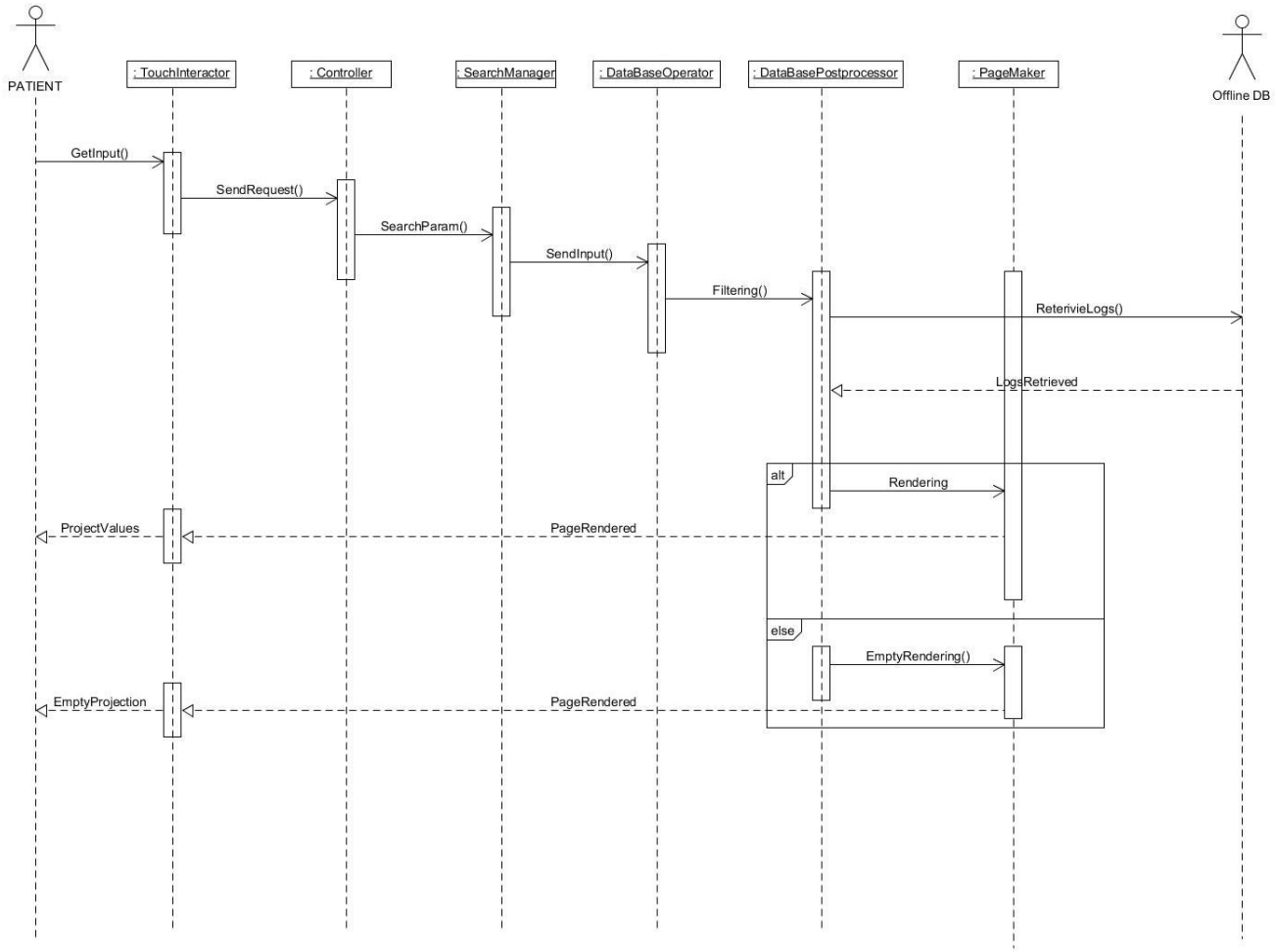


Figure 7.1.2 Get Vital Signs - Phone

. Please note that in order for a physician to select a patient, they choose they enter the patient ID in the patient ID field and select search. This is fulfill the SearchManager portion of the domain concept .

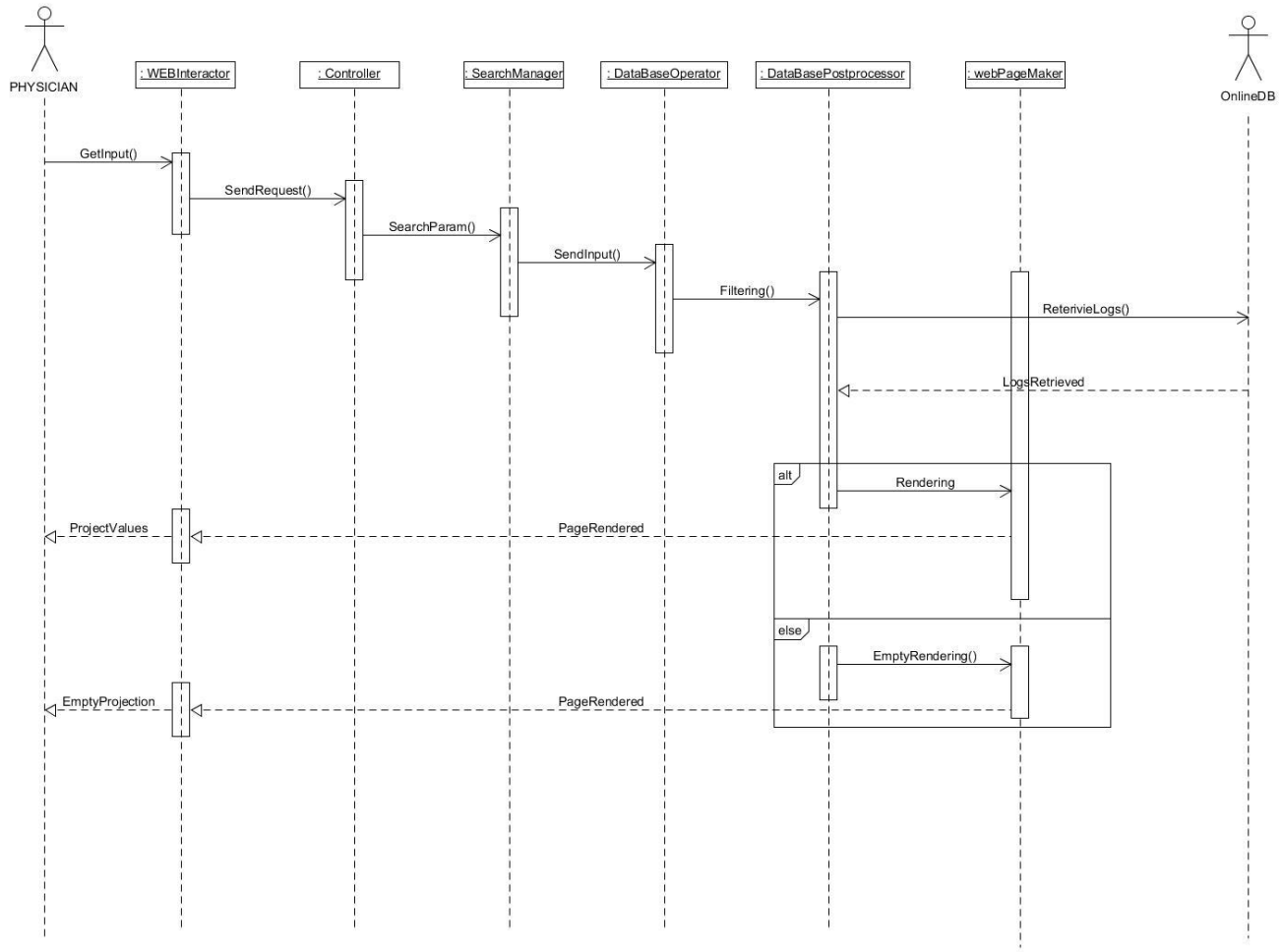


Figure 7.1.3 Get Vital Signs - Web

UC-11: Physician Sign in

The following figure explains the interaction between concepts for UC-11: Physician Sign In. Here, the main goal is for the doctor to sign in and edit the database by adding or removing patients.

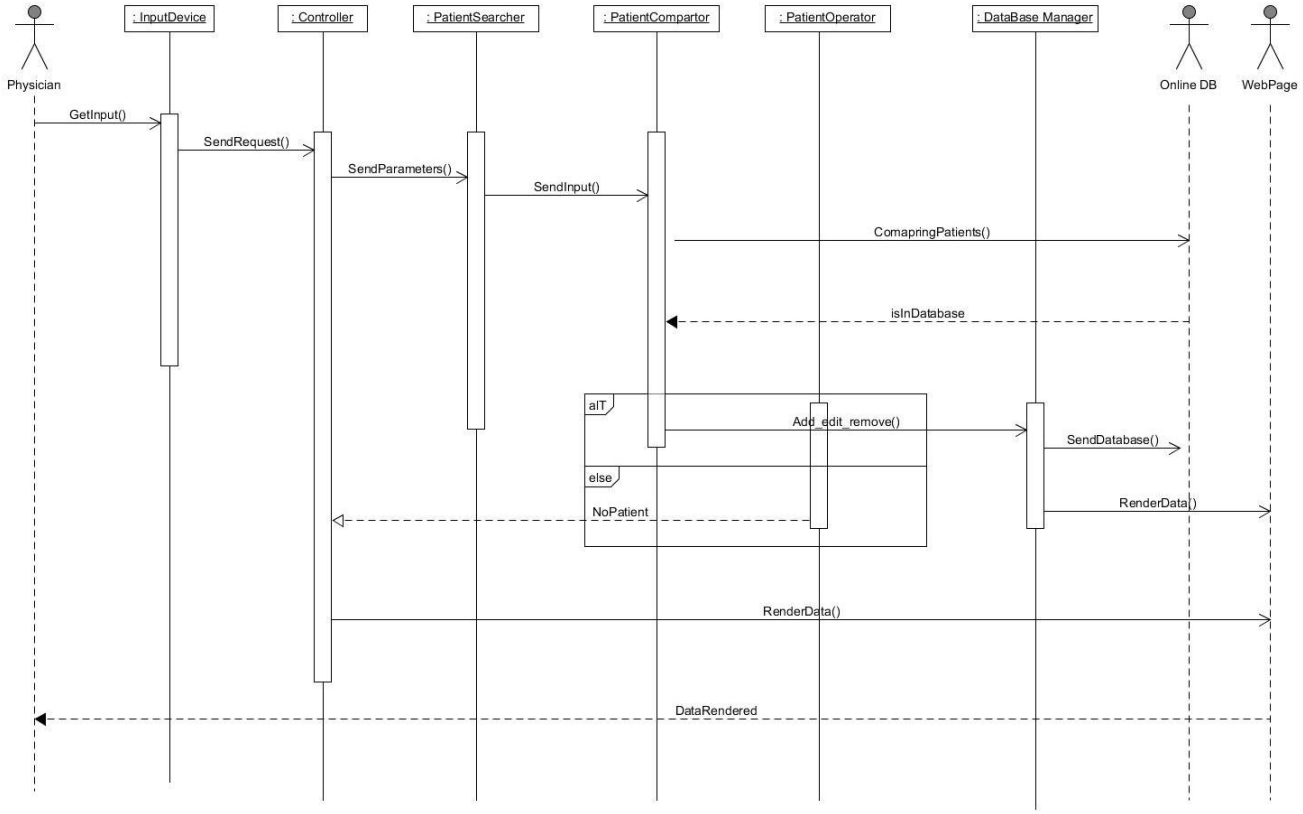


Figure 7.1.4 Physician Sign In

Delegating Responsibilities

We aim to implement software that is reusable as individual components and each component is more or less independent, in a way that changes made in one component do not heavily affect a lot of other components. In other words, we plan to minimize dependencies between objects. This helps us in not only dividing the work between members of the group, it also helps us when we put together all the components, and when Software Testing is performed. Since, we will also continue modifying and

refactoring the code to provide all the functionalities we proposed, having this kind of design principles will be more convenient.

The most important components of our software are the Controller and Database Manager. This does not mean that most work is done by these two components; it simply means that since the controller initiates most functionalities and since the database manager is required for interaction with our Database, and helps in communication between our Web and Mobile app, they have some key roles and most number of interactions with other objects. We have, however, decentralized the control flow and have assigned specific roles to other components. For example, the Sensor Operator performs the function of getting information from the sensors and passes it onto a Data Container object to store it. This in turn talks to the Database Manager to help store this information in the Online Database. Similarly, we have assigned specific roles to other objects. With such requirements, we have planned to implement our software mainly using the design principles of Expert Doer - to have short communication links between objects, High Cohesion - decentralized flow of control with each component having its own specific role which in turn supports Low Coupling - having minimum number of interactions for each component.

7.2 Prose Description of Diagram

Use Case-2 :Health Data

The user can monitor his/her current activity using the mobile application. The user/patient opens the application and initiates the tracking function. When initiated, the Touch Interactor gets the input from the user and sends corresponding request to the Controller. The Controller then notifies the Tracking Enabler to start tracking. It also requests the Sensor Operator to start detecting using the sensor and simultaneously sends request to Notification Enabler to notify the patient to start running. The Sensor Operator saves the data in the Data Container in loop. When the patient stops running, the sensor operator sends Stop notification to Notification Enabler and the Controller send disable tracking request to the Tracking Enabler. The Controller sends authentication request to the Database Authenticator to verify user authenticity and after the authentication is done, the controller requests to save data offline from the Data Container in the Offline Database. The Database Manager ensures that the user data is synced and updated in the Online Database, when it receives a sync request from the controller. Also, The Controller sends a finish request to the notification Enabler , when a specific goal is reached and displays the corresponding message to the patient.

UC-3 : GetVitalSigns - Phone

The User may wish to see his current health data or history of his previous activity. This Use Case refers to the historic data of the user. The User opens the app and presses the History button to view his previous activity details. The Touch Interactor then talks to the Controller to retrieve the User's records. The Controller sends in basic User information which serves as a Search criteria in the database. The Database Operator then queries the Database for the User's records using this search criteria. After the information has been obtained from the database, the information is assigned to specific components by the Postprocessor and the PageMaker is an xml file that determines how the Display page will look like. This is then available for the user to view. We have used a Publisher-Subscriber Design Pattern here with no specific central controller. The control flow traverses from object to object. Even though we do have a controller, it does not have direct communication with all components and each part of our code is reusable.

UC-4: GetVitalSigns - Web

This Use Case is similar to UC-3, in that most components interact in the same way. The Physician must be signed in to use this functionality. The Physician can only view records of patients he is authorized to access. Hence, his login is authenticates and provides access to certain records in the Online database. The Physician requests for patient history by clicking on the appropriate button. The Controller then sends the basic patient data (for example, ID) serving as a search criteria to obtain information from the Online database, which is then displayed to the user. The design pattern and control flow are the same as described in UC-3.

Use Case 11: Physician Sign In

This use case describes how the physician can manage patient records by performing operations like add/edit/remove. The physician is assumed to be signed into his/her account and view the web page.The Physician sends the input to the Input device which in turn sends the corresponding request to the Controller. The Controller sends the input parameters to the Patient Searcher. The Patient Searcher in turn sends the input to Patient Comparator which compares the input with the Online Database and returns the required response back to the Comparator. After checking the Database for the patient records, the Controller sends the add/edit/remove request to the Patient Operator. Then Patient Operator sends the commands to the Database manager and renders the data to the Web Page for display.

7.3 Alternate Solution Description

7.3.1 BASIC DESIGN PROCESS

While designing the flow of interactions between the concepts of the system, it is a common practice to follow responsibility-driven approach. This gives a way to assign the responsibilities among the elements of the design in a balanced, fluid and optimal way. While designing ours, we considered a design to be optimal that manages to articulate the interaction flow while maintaining the following design principles [15]:

- Expert Doer Principle - Short communication chains between the objects
- High Cohesion Principle - Balanced workload across the objects
- Low Coupling Principle - Low degree of connectivity among the objects

At global design level, we also considered additional implicit principles for several segments of our design to give it more flexibility for transition to higher fidelity design [21]. Some of them are:

- Rigidity - It is hard to change because every change affects too many other parts of the system.
- Fragility - With changes made in one part of design, unexpected parts of the system break.
- Immobility - It is hard to reuse in another application because it cannot be disentangled from the current application.

In the initial process, we crafted multiple design solutions for the use cases of our system that achieve the corresponding design goal. However, by employing design principles and design heuristics from developer's point of view, we compared the alternatives, check whether they violate any of the design principles, considered necessary trade-offs and decided to abandon the ones that failed to achieve the benchmark. That being said, in cases with conflicts, we reached a conclusion by compromising with our reasoning and a design solution has been chosen based on the developer's judgement for this context. While it can be argued whether our chosen design is the best one or not, we do state that within the design constraints and in our judgement, we have tried to conclude with the optimal choice.

7.3.2 DESIGN SOLUTION DESCRIPTION

UC-2: Health Data

We considered alternate solutions for the responsibility SendRequest() to the Controller for starting the tracking ability. We considered assigning this responsibility directly from the initiating actor (Patient) to Controller. However, following Expert Doer and Low Coupling principle, we decided to incorporate this interaction between Patient and Controller via an additional UI element. Hence, Touch `was included to receive the request to start tracking from the Patient and then convey this to Controller [figure 7.1.1].

Another decision we had make was regarding the responsibility of notifying the tracking activation (EnableTracking). Initially we considered conveying the StartDetecting responsibility directly from Controller to Sensor Operator. However, in order to incorporate feedback in the process for the Controller (Controller needs to be notified of the success in enabling the tracking operation), we decided to include Tracking Enabler that notifies Controller. Adding a new object in the system had the risk of including longer chain of communication between the design objects. As the current design promised higher cohesion, improved readability and maintainability, we decided that it was a necessary choice. We employed this logic process to reach design decision for responsibility of notifying the patient that running stopped (NotifyPatientStopped) and reaching the running goal for the day (GoalReached).

One more instant where we considered alternate choice was assigning the responsibility of saving the data. In initial design prototype, we considered saving the data to the Offline database directly via *Database Manager*. In that case, Offline Database needs to save the real-time data sent from the sensor and later after the session ended, it needs to produce session data and send to online database. In later iteration, we discovered this introduces two potential problem:

- For longer session, there is a chance of missing data when data being sent to Offline Database in real-time.

- *Controller* should be assigned the responsibility to make command for requesting to save data online or offline.

Considering these two drawbacks of our design, we decided to save the real-time data in *DatabaseContainer*. Once a session ended, Controller sends request to *DatabaseContainer* to save data in the Offline Database. This design ensures each element has only one responsibility at a time (Expert Doer) and *DatabaseContainer* lightens the responsibility of Online Database (High Cohesion).

UC-3: GetVitalSigns - Phone

In this particular use case, we needed to employ our better judgement to make a choice of optimal design from the alternatives we have. As shown in the interaction diagram [figure 7.1.2], we needed to assign the responsibility of sending request from *Patient* to *PageMaker* for displaying the *VitalSigns*. The initial and final point of the flow of interaction were known: Controller and PageMaker. The decision of responsibility on receiving the data on *VitalSigns* was straightforward: *PageMaker* directly send the related page to the UI element (*TouchInteractor*) and allows the *Patient* to interact.

However, we had alternatives in the case of sending the request. We had the following alternative design choices:

Option 1: Request from *Patient* will be directly conveyed to *PageMaker* via *Controller* and *PageMake*r will return the page containing the most update report of *VitalSigns* (received from Offline Database).

Option 2: Request from *Patient* will come to the *Controller*. *Controlle*r will delegate the task to several additional elements to assign one responsibility per element to extract specific information requested by *Patient*.

In option 2, in later iteration, we discovered that we need to incorporate further objects to make process functional and balanced. We incorporated *SearchManager*, that will search for the specific parameters sent by *Controller* (additional object for managing the search operation). Based on the input

in *SearchManager*, *Database Operator* was assigned the responsibility to send filtering request to the Database Postprocessor. Finally *Database Postprocessor* executed the task of rendering the filtered data to the PageMaker (to be conveyed for display in the UI).

When we employed the design principles on the two options, it was obvious that option 1 violated Expert Doer and High Cohesion principle. On the other hand, option 2 had a long chain of communication between the newly incorporated objects. We decided to compromise in this case and chose the design that offers optimal solution with less severe violation. This leads us to choose option 1. We believe that the current design ensures that each element knows what it is performing (Expert Doer), doing one task at a time (High Cohesion) and communicates effectively while staying within the design constraints.

UC-4 Get Vital Signs Web

A possible alternate design strategy for the "Get Vital Signs Web" use case, (UC-4), was to exclude the *SearchManager* object. Instead, *Controller* handles these responsibilities. The controller not only communicates with the database, but it also determines and manages the search parameters. The advantage of this solution is that there is a shorter communication distance between the Controller and the Database

We choose not to go with this design option for a few reasons. First of all, this design does not meet the high cohesion principle because *Controller* has too many responsibilities. This version of the use case requires that the controller to designate the search parameters in addition to handling all communication responsibilities. The search request definition and communication are not related attributes. Thus, *Controller* would be responsible for two functions instead of one.

Additionally, this solution reduces the design's demonstration of the expert doer principal. This principle means that "the one who knows should do the task". Although *Controller* knows the user's input for the request they are not the expert in how to arrange this input as a search request to the

database. Adding *SearchManager* as an object, allows this object to be the expert in delegating search request parameters. This in turn allows the controller to continue to be the expert doer in communication only rather than trying to be an expert in another task.

UC-11 Physician Sign In

One design consideration for the physician sign-in use case was to add an object, *PatientEditor* to the design. In our interaction diagram this object would reside between the *PatientOperator* and *DatabaseManager* objects. It responsibilities would include adding and removing patients. We considerd this option because we thought it would reduce the number of responsibilities for the *PatientComparator* object and thereby promote the high cohesion principle.

However, the following communication interactions would have to be added to support this object: 1) *PatientComparator* → *PatientEditor* and 2) *PatientEditor* → *OnlineDatabase*. Additionally, the *PatientComparator* object would still have to communicate with the online database. Therefore, this would increase the overall number of communication interactions among objects.These additional communication responsibilities defy the "low coupling principle". Furthermore, since the *PatientComparator* already has the information required to perform the add/remove patient request, it makes sense for it to do so.

7.4 DESIGN PATTERNS

To ensure that change in one object affects its responsibilities towards other objects to the minimal, we had tried our best to effectively utilize the delegation of work into knowing, doing and calling methods. Initially we tried to design in a way so that direct communication lies in each association between the objects, as we assumed that would enable the system to respond faster. However, after Demo 2, we realized that indirectly notifying the objects by dispatching event occurrence are also important. Hence, in later iterations we decided to use a combination of direct and indirect communication based on the state of the event is currently in. In several cases, we also considered Publisher-Subscriber design patterns, especially in the case of event occurrence based on click and user interaction with the device or browser. In the following paragraphs we briefly describe what factors we took into consideration for these design decisions.

For example, in UC-2, where the patient health data is obtained via sensors on database, we utilized direct communication for events like EnableTracking, StartDetecting or DisableTracking. This allowed the controller to directly communicate with the object who has the responsibility for responding to the event. This was repeated for Authentication or Sync events as well.

However, in UC-3, for retrieving data on the phone from the database, we used indirect communication instead as the responsibility of filtering the query/request is delegated to different objects of the system to ensure loose coupling. We used the design logic for similar requests from the patient or physician in UC-4 and UC-11.

While we preferred indirect-direct communication for request-based occurrence, we decided to use publisher-subscriber pattern for several event based occurrence. A good example for such events are Saving data while running (SaveWhileRun) in UC-2 where SensorOperator sends data for sving in the DataContainer while the user is running. The Data container acts as a subscriber to this event and saves data whenever an update to the event occurs. This operation is performed in a loop so that data is constantly saved as long as sensor detects.

Another type of Doer we used in our publisher-subscriber design choice asks for information and acts or dispatch this information for the responder to act. This type of subscriber pattern was used where an object will express to get information and acts upon it once it is received. They only operate once they get their desired information. This is true for SendData in UC-11 to send data in database for storing. We employed same design logic for other similar cases in our design. In several cases we also utilized Decorators in our design pattern to ensure that actor does not interact with too many objects at the same time directly rather uses the decorator (e.g Controller) to pass requests. Another advantage for this kind of design pattern is ease of incorporating net responsibilities into the system. As each object knows its responsibility and has minimal number of responsibilities, adding new responsibilities require only to add new classes that is specifically assigned with that particular responsibility.

While we were mapping our design in the first iteration, we only tried to ensure that our design provides a good demonstration of Expert Doer, High Cohesion and Low Coupling. In the later iteration

we realized that we needed to consider different design pattern to come up with an optimal design. However, we also understand that sometimes considering design patterns in a specific portion of the system may result unrealistic and more complex design. It is more important to first accomplish the desired goal with the system under design, and then optimize it by reducing complexity. While it can be argued whether the current status of our design optimally addresses these issues, we do state that we have tried to our best effort maintain this principle in our design.

8 CLASS DIAGRAM AND INTERFACE SPECIFICATION

8.1 Class Diagram

8.1.1 ANDROID APP

The class diagrams are separated into several sections so that the diagram would not appear cluttered. Each diagram displays the classes that are closely related to main activities. This includes the classes with function related to Login, Registration, and profile management. The Health Monitoring Activity Class diagram displays classes related to the detection of activity and health information. This includes the classes with functions related to activity monitoring (step counter) and heart rate detection.

Main Interface

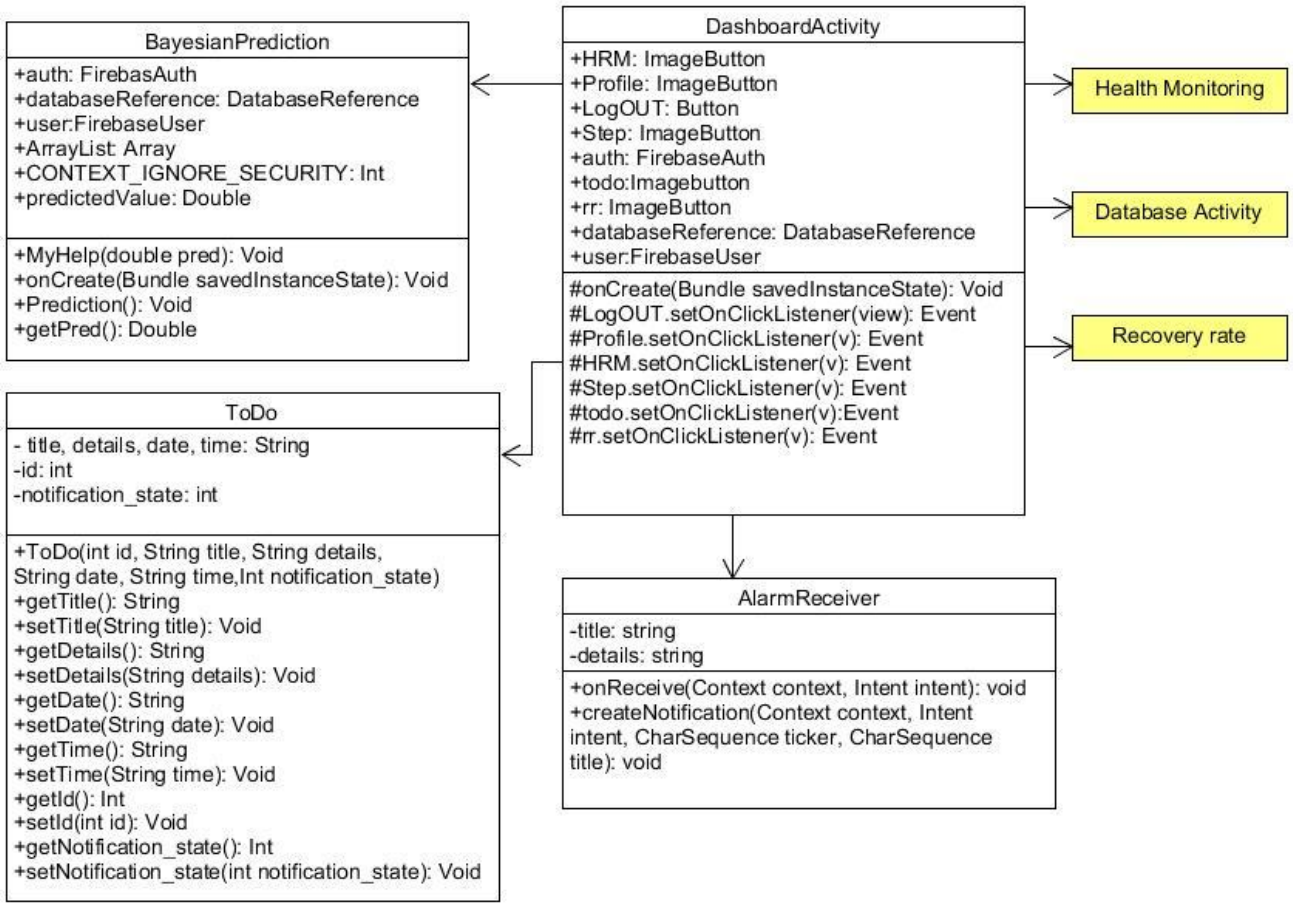


Figure 8.1.1: Main Access Class Diagram (Android)

Health Monitoring Activity

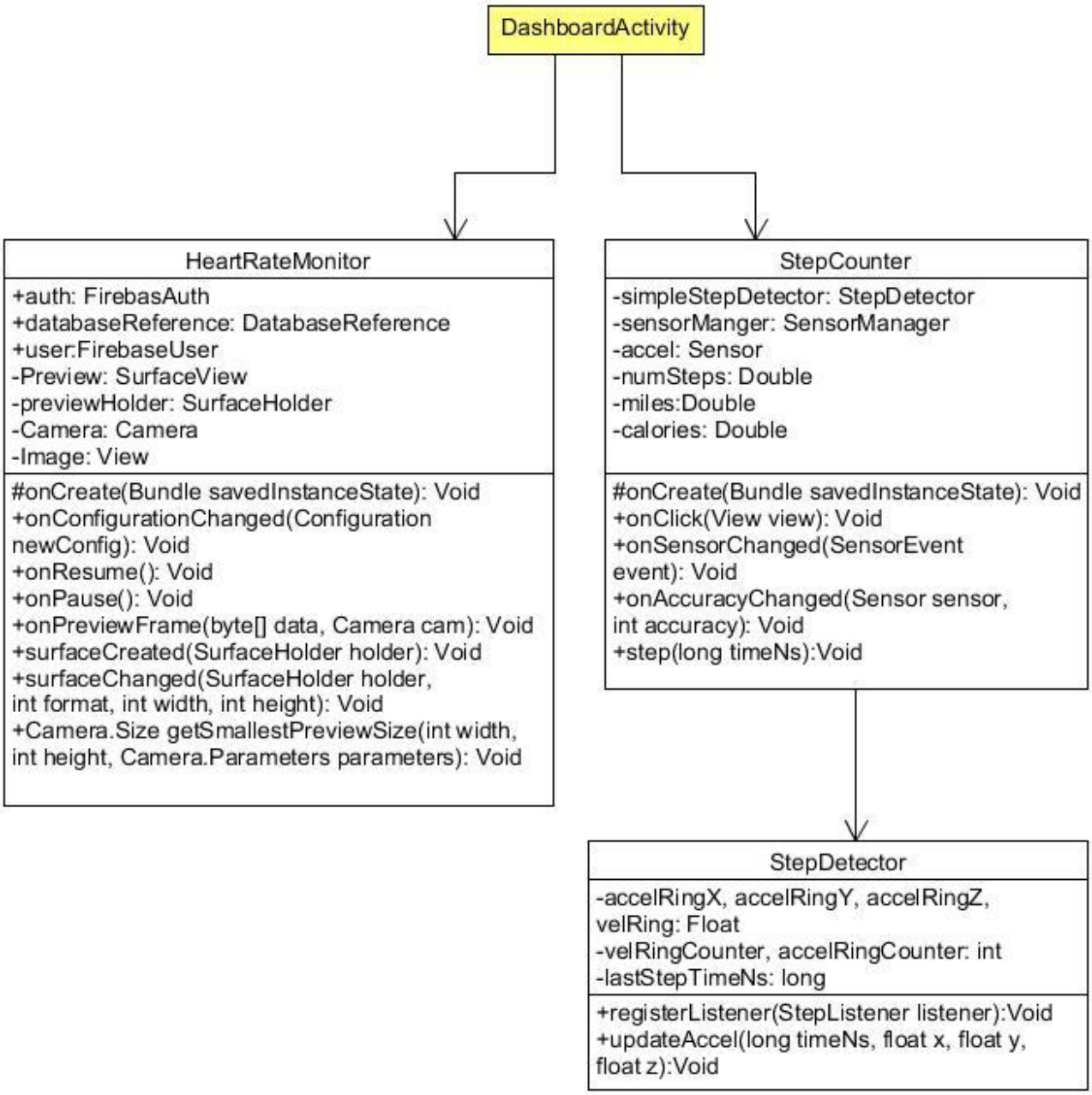


Figure 8.1.2: Health Monitoring Class Diagram

Database Activities

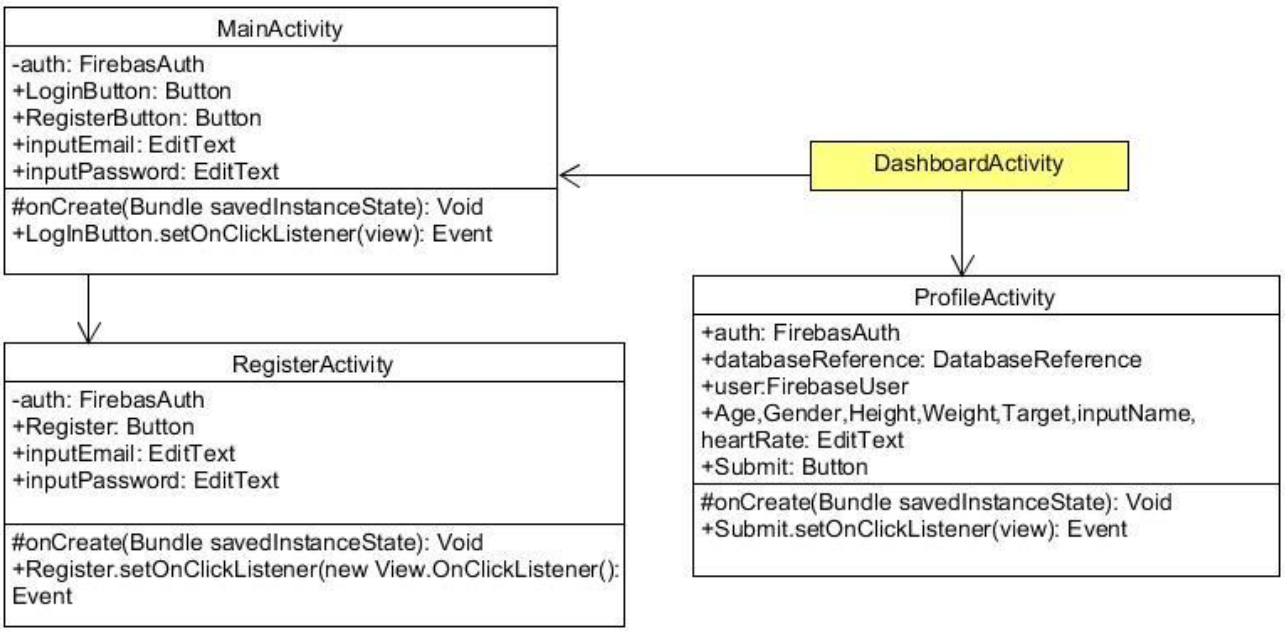


Figure 8.1.3: Health Database Activities Class Diagram (Android)

Recovery Rate

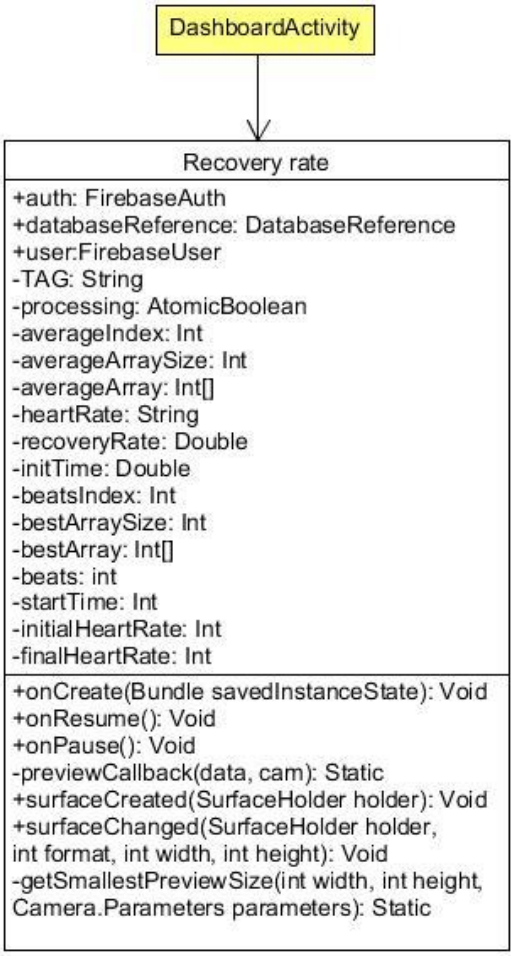


Figure 8.1.4: Recovery Rate Class Diagram (Android)

8.1.2 WEB APP

The class diagrams for the web app are also separated into several sections so that the diagram would not appear cluttered. Each diagram displays the pages that are closely related to main activities. The main diagram includes items related to Login, Firebase Manipulations, and updating the profile. The registration includes pages and files associated with registration. The patient information sections included pages and files that are related to searching for and displaying patient information.

Main Class Diagram

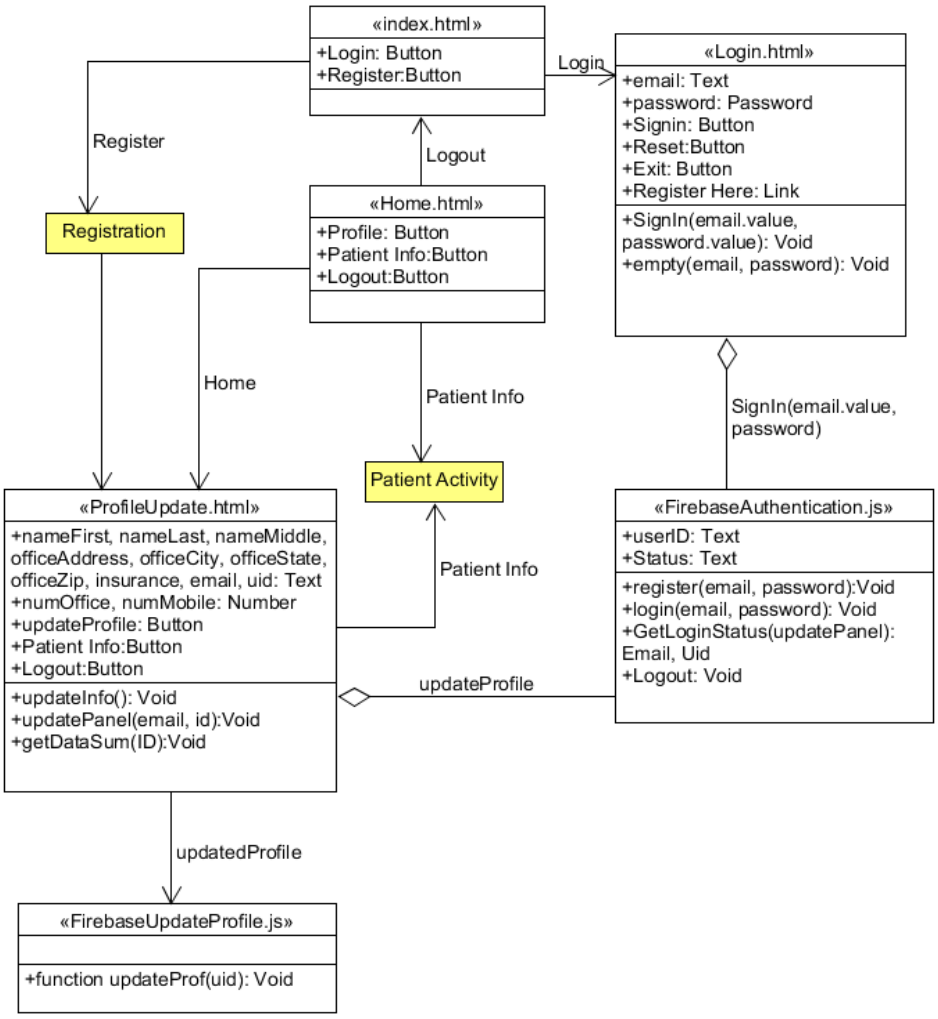


Figure 8.1.5: Main Class Diagram (Web)

Registration Class Diagram

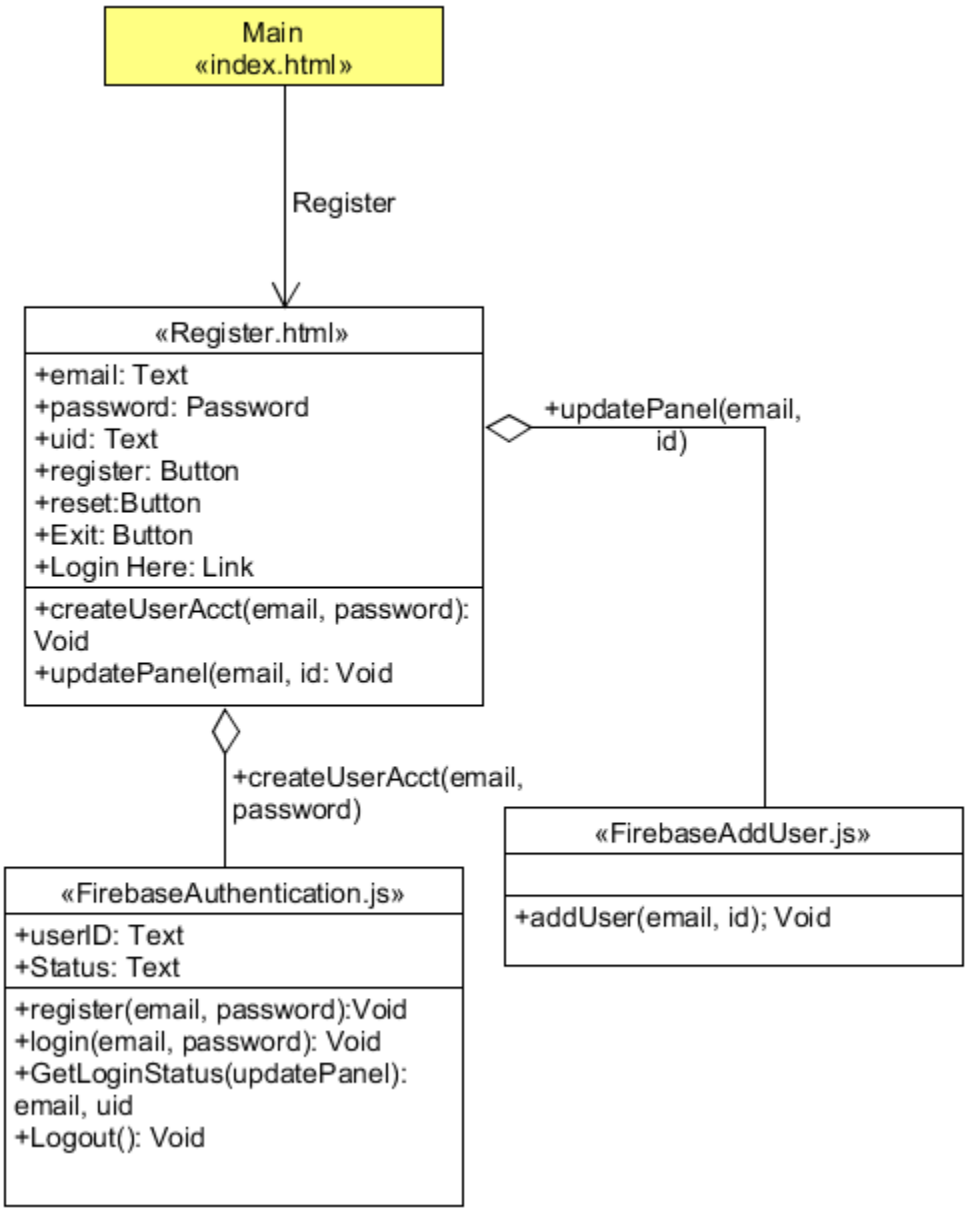


Figure 8.1.6: Registration Class Diagram (Web)

Patient Information Class Diagram

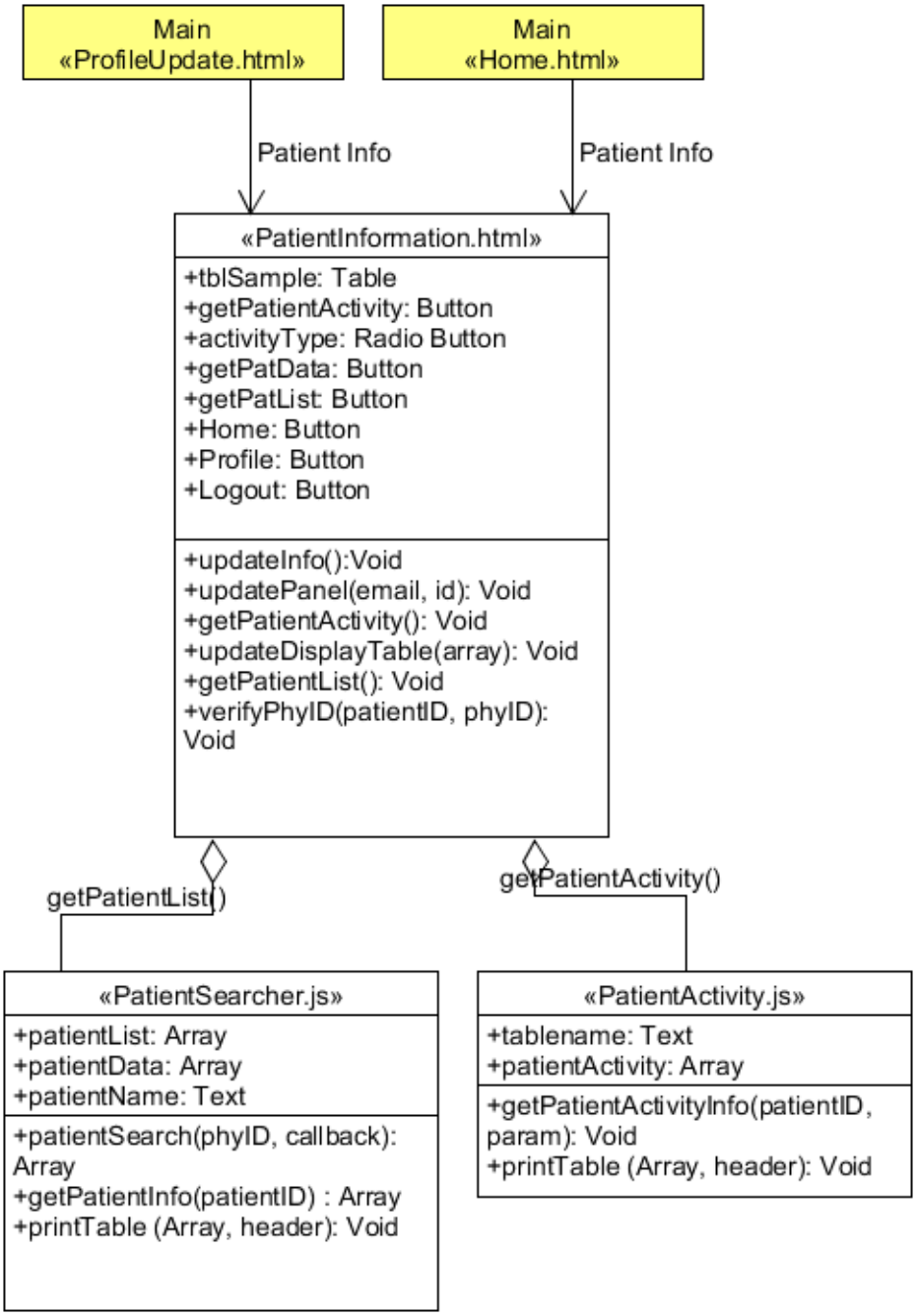


Figure 8.1.7: Patient Activity Class Diagram (Web)

8.2 Data Types and Operation Signatures

8.2.1 ANDROID APP

Registration

For the registration part of our code, the classes are as follow:

MainActivity
This class starts the user instance by initializing login objects. It hosts relevant object calls from different files
Attributes: + **LoginButton**:Button. **+RegisterButton**:Button. **+InputEmail**:EditText that will receive the current user email input (Main Activity) **+InputPassword**:EditText that will receive the current user password input (Main Activity) . **+auth**: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class. [22]
Operation: **#onCreate(Bundle savedInstanceState): Void.** This method gets called when the activity is first created and runs only once. **LogInButton.setOnClickListener(view): Event.** When the login button is clicked, this method uses the firebase signInWithEmailAndPassword class to sign into an account in the firebase system.

RegisterActivity
In this class is where the user creates a new account. The database is opened for entering new data.

Attributes:

-auth: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class. [22]

+**inputEmail, inputPassword,**: EditText.Stores the information that the user enters

+Register: Button

Operation:

#**onCreate(Bundle savedInstanceState)**: **Void**. This method gets called when the activity is first created and runs only once.

+Register.setOnClickListener(new View.OnClickListener():Event. When the register buttons is clicked, the method verifies the password credentials. If the password is too short example, it outputs a message to the user. If the credentials are appropriate it registers the user with the createUserWithEmailAndPassword Firebase class.

ProfileActivity

The Patient creates his/her profile. All of the details and information are stored in the database.

Attributes:

+**Age,Gender,Height,Weight,Target, inputName, heartRate:** EditText. Stores the information that the user enters.

+auth: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class. [22]

+**databaseReference: DatabaseReference**. An object that allows reference to a firebase database through the Firebase DatabaseReference class. This allows the software to read and write information in the database. [22+]

+**user:FirebaseUser.** An object that allows the user profile information to be represented in the database via the FirebaseUser class. [22+]

Operation:

#**onCreate(Bundle savedInstanceState)**: **Void**. This method gets called when the activity is first created and runs only once. It references the firebase database to retrieve current information.

+Submit.setOnClickListener(view): Event When the submit button is clicked this method updates

profile information if the submit button is clicked.

DashboardActivity

Here is where all the application options are located: Profile, Heart Rate Monitor, Step Counter, Recovery Rate, Scheduler, Physician Selection.

Attributes:

+**HRM:** Button. Opens HeartRateMonitor class when clicked.

+**Profile**: Button. Opens ProfileActivity class when clicked.

+**LogOUT:** Button.Triggers the log out event when clicked.

+**RecoveryRate:** Button triggers the Recovery Rate class and consequently Bayesian Prediction class when clicked.

+**PhysicianSelect:** Button triggers the Physician class when clicked.

+**Step**: Button. Opens StepCounter class when clicked.

+**ToDo**: Button. Opens Scheduler class when clicked.

+auth: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class. [22]

+**databaseReference: DatabaseReference**. An object that allows reference to a firebase database through the Firebase DatabaseReference class. This allows the software to read and write information in the database. [22+]

+**user:FirebaseUser.** An object that allows the user profile information to be represented in the database via the FirebaseUser class. [22+]

Operation:

#onCreate(Bundle savedInstanceState): Void. This method gets called when the activity is first created and runs only once.

#LogOUT.setOnClickListener(view): Event. When the logout button is clicked this method logout out of the system using the Firebase authSignout class.

#Profile.setOnClickListener(v): Event When the profile button is clicked this method opens the user's profile.

#HRM.setOnClickListener(v): Event When the HRM button is clicked this method starts the heart rate monitoring activity.

#PhysicianSelect.setOnClickListener(v): Event When the Physician Select button is clicked this method starts the physician select activity.

#Step.setOnClickListener(v): Event When the Step button is clicked this method starts the step monitoring activity.

#todo.setOnClickListener(v):Event When the ToDo button is clicked this method opens the the ToDo (calendar) activity.

#rr.setOnClickListener(v): Event When the rr button is clicked this method starts monitoring the patient's recovery rate.

Step Counter

For the step counter functionality, the implemented classes are as follow:

StepCounter
This class starts StepDetection by calling relevant objects.
Attributes: **-simpleStepDetector:** stepDetector. Stores the value of the accelerometer sensor. **-sensorManager:** SensorManager. Variable used to manage the smartphone's sensors. **-accel:** Sensor. Temporary variable used to store the initial accelerometer values. **-numSteps:** Double. Stores the actual number of steps **-miles:** Double. Stores the miles traveled so far. **-calories:** Double. Stores the calories burned so far.
Operation: **#onCreate(Bundle savedInstanceState):** Void. This method gets called when the activity is first

created and runs only once.

+onClick(View view): Void. This method just adds click listener the the buttons.

+onSensorChanged(SensorEvent event): Void. Updates the accelerometer values every time the sensor detects a change.

+step(long timeNs): Void. Calculates the steps, miles and calories.

StepDetector

The sensor values are read and the steps calculations are made.

Attributes:

-**accelRingX,accelRingY,accelRingZ,velRing**: Float identifiers for the Accelerometer sensor.

-velRingCounter,accelRingCounter:integer identifiers for the Accelerometer sensor.

-**lastStepTimesNs**: long identfiers to hold the last number of steps that detected from the user.

Operation:

+registerListener(StepListener): void. This Method will register the unusual behaviour between the sensor data and millisecond.

+updateAccel(long timeNs, float x,float y, float z); Void. This method will update the the old Accelerometer values with new upcoming ones .

Heart Rate Monitor

For the Heart Rate monitoring functionality, the implemented classes are as follow:

HeartRateMonitor

This class initializes all necessary sensors: camera, wake screen, flash, etc. The, it gets the preview frame and process red values using rolling average filter.

Attributes:

-Preview: SurfaceView. SurfaceView takes care of placing the surface at the correct location on the screen.

-PreviewHolder:SurfaceHolder. Is identifier for The holder of the surface.

-Camera:Camera. provides an interface to individual camera devices connected to an Android device.

-Image: View. preview images are sent to SurfaceView or TextureView.

+auth: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class. [22]

+**databaseReference: DatabaseReference**. An object that allows reference to a firebase database through the Firebase DatabaseReference class. This allows the software to read and write information in the database. [22+]

+**user:FirebaseUser.** An object that allows the user profile information to be represented in the database via the FirebaseUser class. [22+]

Operation:

#onCreate(Bundle savedInstanceState): Void. This method gets called when the activity is first created and runs only once.

+OnConfigurationChanged(Configuration newConfig): Void. This method will identify any new configuration and will make new configured environment based on the received configuration.

+**onResume()**: Void. This method is called everytime the MainActivity is hidden and try to resumed.

+**onPause()**: Void This method is called everytime the main activity is stopped and switched to another activity.

+onPreviewFrame(byte[] data,Camera cam): Void. This is a Callback interface used to deliver copies of preview frames as they are displayed.

+SurfaceCreated(SurfaceHolder holder , int format , int width , int height):Void .This method will hold and store the current Surface with its dimensions

+SurfaceChanged(SurfaceHolder holder , int format , int width , int height): Void. This method will update the current surface with new one (dimensions of the surface).

+Camera.Size getSmallestPreviewSize(int width, int height, Camera.Parameters parameters):

Void. This method will sets the dimensions of the picture.

Scheduler (ToDo class)

For the Heart Rate monitoring functionality, the implemented classes are as follow:

ToDo
This class allows the accesses calendar and scheduling activity.
Attributes: - **title, details, date, time: String:** String inputs that allow the user to input information for the title, details, date, and time of the task. **-id, notification_state: int:** Integer variables that give item id, notification state.
Operation: **+getTitle():**String: Lets the user set a Title. +**setTitle(String title):**Void: Assigns the title to the respective object. **+getDetails():**String: Lets the user enter exercise description and details. +**setDetails(String details):**Void: Assigns the details to the respective object. **+getDate():**String: Lets the user set a Date. +**setDate(String date):**Void: Assigns the date to the respective object. +**getTime():**String: Lets the user set a Time +**setTime(String time):**Void: Assigns the time to the respective object. +**getId():Int:**Lets the exercise get an Id

+setId(int id): Void: Assigns the Id to the respective object.

+**getNotification_state():**Int: This denotes the notification state for the alarm.

+**setNotification_state(int notification_state):**Void: Assigns the state to the respective object.

Recovery Rate (rr class)

For the Heart Rate monitoring functionality, the implemented classes are as follow:

Recovery Rate This class allows user to access the Recovery Rate and Prediction activity
Attributes: - **+auth**: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class. - +**databaseReference: DatabaseReference** An object that allows reference to a firebase database through the Firebase DatabaseReference class. This allows the software to read and write information in the database. - +**user:FirebaseUser.** An object that allows the user profile information to be represented in the database via the FirebaseUser class. - **average index, averageArraySize, beatsIndex, beatsArraySize, beats, startTime, initialHeartRate, finalHeartRate:int:** Variables used to calculate the redness in the images and periodically take the preview and process the image. The beats are calculated using arrays. - **processing: Boolean:** Indicates whether the image is currently being processed - **averageArray, beatsArray: Int[]:** These integer arrays allow for the processing of preview image and output the beats as processed. -**recoveryRate, initTime, endTime: Double:** gives the final value of recovery rate by getting a difference between initial time and time when heart beat reaches normal rate.

Operation:

+onCreate(Bundle savedInstanceState): Void. This method gets called when the activity is first created and runs only once.

+**onResume()**: Void. This method is called everytime the MainActivity is hidden and try to resumed.

+**onPause()**: Void This method is called everytime the main activity is stopped and switched to another activity.

+onPreviewFrame(byte[] data,Camera cam): Void. This is a Callback interface used to deliver copies of preview frames as they are displayed.

+SurfaceCreated(SurfaceHolder holder , int format , int width , int height):Void .This method will hold and store the current Surface with its dimensions

+SurfaceChanged(SurfaceHolder holder , int format , int width , int height): Void. This method will update the current surface with new one (dimensions of the surface).

+Camera.Size getSmallestPreviewSize(int width, int height, Camera.Parameters parameters): Void. This method will sets the dimensions of the picture.

Bayesian Prediction class:

The Bayesian Prediction class is triggered with the recovery rate class to predict a recovery rate value based on the previous 10 days' values and indicates to the user whether the actual value is within ramge or higher than expected.

BayesianPrediction
This class uses Bayesian curve fitting within input dataset of 10 values and predicts a value.
Attributes:

- **+auth**: **FirebaseAuth** A firebase object that allows entry into the firebase authentication SDK. via the FirebaseAuth class.

- +**databaseReference: DatabaseReference** An object that allows reference to a firebase database through the Firebase DatabaseReference class. This allows the software to read and write information in the database.

- +**user:FirebaseUser.** An object that allows the user profile information to be represented in the database via the FirebaseUser class.

- **predictedValue: Double:** The final predicted value is obtained from any class by using a getter method.

Operation:

+onCreate(Bundle savedInstanceState): Void. This method gets called when the activity is first created and runs only once.

+**MyHelp:** Helper method to calculate the curve fitting values.

+**Prediction:** Main class used for prediction. Database is accessed and values of 10 days is obtained this array list is used to calculate the curve fitting parameters and make a prediction.

+**getPred():Double:** This method allows access to the predicted value from any of the classes in the application. We use this in the recovery rate class.

8.2.1 WEB APP

Main Activity

For the main activity part of the website code, the scripts are as follow:

index.html
This is the pages that allows the user to enter into the RU Healthy Site
Attributes: **+Login: Button** Redirects to the Login page when clicked. **+Register:Button** Redirects to the Registration page when clicked..

Login.html
This page allows the user to access and update their profile information.
Attributes: +**email: Text** Stores user input data for email. +**password: Password:** Stores user input data for password. +**Signin: Button** Activates the SignIn function when clicked. +**Reset:Button** Resets fields in the login table to default values when clicked. +**Exit: Button** Initiates the logout function. +**Register Here: Link** Redirects to the registration page when clicked.
Operation: +**SignIn(email.value, password.value): Void** This method calls the login and GetLoginStatus functions. +**buffer(email, password): Void** Upon a successful sign in, this method waits for a designated time

period and then redirects to the Home.html page.

FirebaseAuthentication.js

This Javascript file contains functions for the authentication user signin and registration for the firebase interface. The functions in this script use objects and methods from the Firebase class.

Attributes:

+userID: Text. Stores value for the userID

+Status: Text. Store value for function status.

Operation:

+register(email, password):Void. This method registers a user in the RU Healthy firebase system.

+login(email, password): Void. Upon receiving the username and password credentials**,** this method logs a user into the RU Healthy firebase system.

+GetLoginStatus(updatePanel): Email, Uid. This method attains the current login status an returns Email and UID information. It also outputs a status windows if the user is not signed in.

+Logout(): Void This method logs the user out of the firebase system.

FirebaseUpdatedProfile.html

This javascript file allows contains a function that allows the user their profile information in the Firebase Database.

Operation:

+function updateProf(uid): Void. This method retrieves input data from ProfileUpdate.html page and update the Firebase parameters accordingly.

Home.html

This page serves a the user homepage and connection point to additional pages.

Attributes

+Profile: Button. Redirects to the ProfileUpdate.html page when clicked.

+Patient Info:Button Redirects to the Patient Information.html page when clicked.

+Logout:Button Redirects to the index.html page when clicked.

Registration

Registration.html

This page allows the user to register for the RU Health system.

Attributes:

+**email: Text** Stores user input data for email.

+**password: Password:** Stores user input data for password.

+**uid: Text:** Is populated with uid upon successful registration.

+**Register : Button** Activates the createUserAcct function when clicked.

+**Reset:Button** Resets fields in the login table to default values when clicked.

+**Exit: Button** Calls the logout function. Redirects to the index.html page.

+**Login Here: Link** Redirects to the Login.html page when clicked.

Operation:

+createUserAcct(email, password): Void This method calls the register, login, and GetLoginStatus functions in sequential order.

+updatePanel(email, id): Void This method, updates the uid value value in the front panel with the current user's ID and calls the add user function.

FirebaseAddUser.js

This Javascript file allows users to be added to the firebase database.

Operation:

+addUser(email, id); Void. This method adds a new user to the RU Healthy Firebase database

Patient Activity

PatientInformation.html

This page allows the physician to retrieve patient information.

Attributes:

+tblSample: Table Displays patient list.

+getPatientActivity: Button Calls the getPatientActivity() when clicked.

+activityType: Radio Button Allow user to select which type of activity information that they would

like to receive.

+getPatData: Button Calls the verifyPhyID function when clicked.

+getPatList: Button Calls the getPatientList function when clicked

+Home: Button Redirects to the Home.html page when clicked.

+Profile: Button Redirects to the ProfileUpdate.html page when clicked.

+Logout: Button Redirects to the index.html page when clicked.

Operation:

+updateInfo():Void This method calls the GetLoginStatus and updatePanel functions sequentially.

+updatePanel(email, id): Void This method, updates the uid value value in the front panel with the current user's ID and calls the add user fun

+getPatientActivity(): Void This method calls the getPatientActivityInfo function.

+updateDisplayTable(array): Void. This methods updates the table on the front panel with the list of patients retrieved from the search criteria.

+getPatientList(): Void This method calls the patientSearch function

+verifyPhyID(patientID, phyID): Void. This method queries the selected patient and determines the current physician matched their id. Upon a successful match the method calls the getPatientInfo function. If IDs do not match this method output an alert message and ends the operation.

PatientSearcher.js

This Javascript files includes function to query the Firebase Database and output result accordingly.

Operation:

+patientList: Array Stores the data array for the patient list.

+patientData: Array Stores the data array for the patient data.

+patientName: Text Stores the text for the patient name.

Operation:

+patientSearch(phyID, callback): Array This method queries the Firebase database for patients with the physician listed and returns the results.

+getPatientInfo(patientID) : Array This method queries the Firebase database and return information for a specific patient.

+printTable (Array, header): Void This method outputs the array data from search to a table

PatientActivity.js

This Javascript files includes function to query the Firebase Database and output result accordingly.

Operation:

+tablename: Text Store the table name.

+patientActivity: Array: Stores the array data with patient activity.

Operation:

+getPatientActivityInfo(patientID, param): Void This method queries the Firebase database and returns information about the specific activity of a patient.

+printTable (Array, header): Void This method outputs the array data from search to a table

8.3 Design Patterns

During the initial implementation phase, we structured the design of our system based on the domain model that we identified. However, in the later iterations, our domain concepts evolved on a large scale and newer objects were included in each domain. Based on these extended concepts, we developed the classes for different activities in our system. We followed general design principles (Expert Doer, Low Coupling and High Cohesion) along with design patterns discussed in Chapter 4 and Lecture 17 - 18 [14].

8.3.1 ANDROID APPLICATION

We used a combination of direct and indirect communication for the activities of our system as we mentioned in design pattern section of interaction diagrams. As our preliminary target was obtaining functionality, we first ensured that we covered the use cases of our system through the activity we designed. As we decentralized the use cases by assigning maximum one use case to each activity, this enabled incorporating of later use cases and functionalities into our system much more conveniently and efficiently.

Our design mostly used publisher-subscriber pattern for the following reasons:

1) To ensure loose coupling in our system as objects were delegated specific responsibilities to increase cohesion.
2) To avoid complex design logic in the system and rather introduce simplified design flow.
3) To incorporate additional classes new responsibilities with less effort.

Our design mostly followed indirect communication. However, as mentioned in Interaction Diagram section, we employed direct communication in several cases when necessary.

In publisher subscriber pattern, Eventdetecting object is referred as Publisher and the doer who responds based on the event received is called subscriber. The role of the publisher is to dispatch event notifications and subscriber, upon receiving this dispatch, performs necessary action as instructed. The

advantage of this design is new functionality can be easily implemented by just adding new doer type (subscriber).

As main functionalities of our mobile app is active when user is interacting with the phone, so occurence of these events depend on the click interaction of user. The doer of these responsibilities wait for the event of click (subscriber) and the clickable buttons dispatch the events. However, StepCounter functionality is an exception in this case as in the latest implementation we ensured it runs in the back of the application even when the user is not directly interacting with the phone application. Thus, this activity is performed by ensuring sensor sends continuous data to the data container, even without receiving explicit command for it. This function is implemented by setting subscriber for EventListener and EventSensorChanged, which detects the change of event and acts accordingly. The buttons act as publisher (in some cases decorator) and corresponding subscriber (accelorometer, camera sensor, etc) perform the responsibility of the Doer. We used the the same design logic for main interface, database, recovery rate and heart rate activities.

8.3.2 WEB APPLICATION

Although the design of the web app is different in terms of user interaction and functionalities, we mostly used publisher-subscriber pattern. However, the purpose of the web app is to present information from the database based on user's click interaction. THere is no need for a subscriber to constantly receive and update information. When a button is clicked or a user is selected, it acts as an event source and sends out a query for the corresponding information to the database. When response from database is received, the information is parsed and fills the corresponding field in the interface. While this can be accomplished using direct communication as well, we chose publisher subscriber pattern as this is convenient when we want to include new classes. We will just need to update the subscriber list and add a new event source for the class.

8.4 Object Constraint Language OCL (Contracts)

1. Main Activity UML/OCL

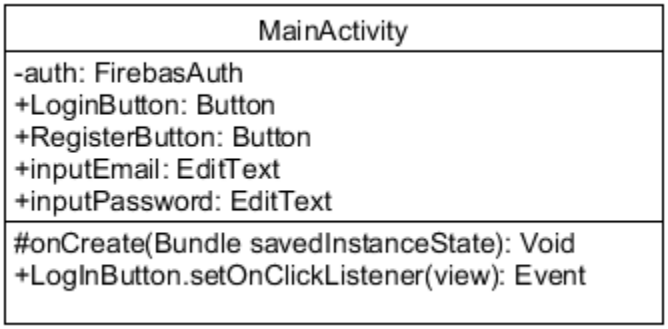


Fig. 8.4.1: UML/OCL specification for MainActivity

context	Main Activity
pre	• User Must Have Valid Internet Connection. • User Must Have Functional Account to Access App Activity. • User Must Register to Access App Activity.
post	• User will gain Access to the App Main features and Activities.

2. Register Activity

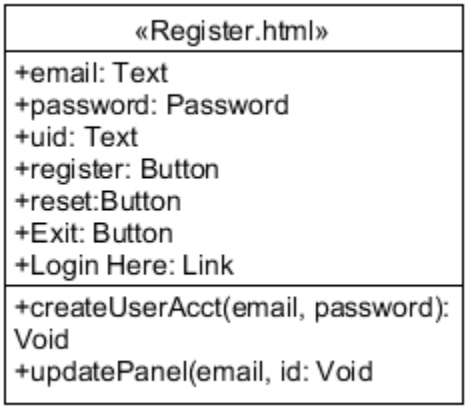


Fig. 8.4.2: UML/OCL specification for RegisterActivity

context	RegisterActivity
pre	● User Must Provide Valid Information (Name, Email, Password) ● User Must Have Valid Internet Connection. ●
post	● User will Have a valid Account and Active Session. ● User will gain Access to the App Main features and Activities.

8. Profile Activity

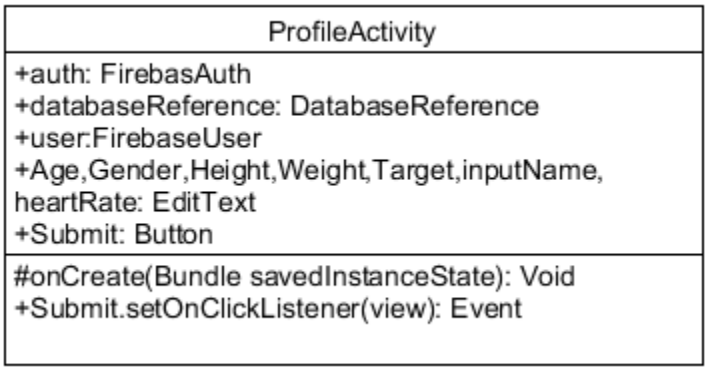


Fig. 8.4.3. UML/OCL specification for RegisterActivity

context	ProfileActivity
pre	● User Must be logged in to the System to Access the ProfileActivity
post	● User Information will be Added to the System Database or ● User information will be updated.

4. HeartRate Monitoring

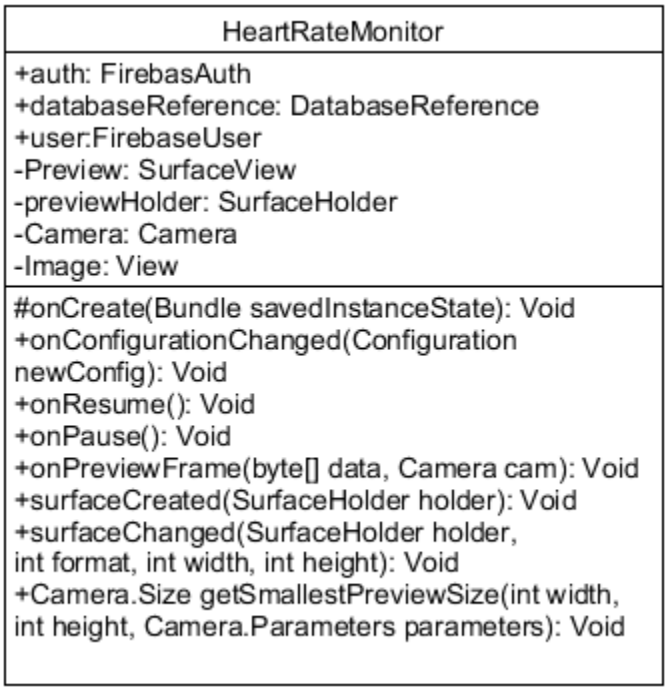


Fig. 8.4.4: UML/OCL specification for Heartrate Monitor

context	HeartRateMonitor
pre	● User Must be logged in to the System to Access the Heartrate Monitor ● Valid Permission to Camera Application, ● Valid Permission to Flash Services.
post	● The User will have a result about his Heart beats. ● The User data will be stored in the Database.

5. StepCounter

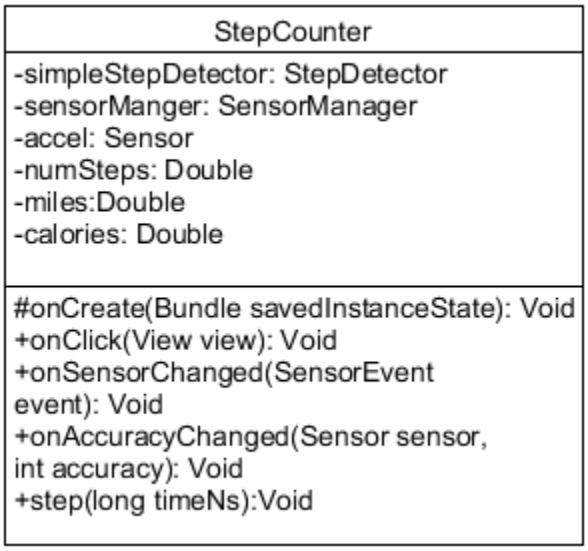


Fig. 8.4.5: UML/OCL specification for StepCounter

context	StepCounter
pre	● User Must be logged in to the System to Access the StepCounter ● Valid Permission to Camera Application, ● Valid Permission to Flash Services.
post	● The User will have a result about his Heart beats. ● The User data will be stored in the Database.

6. ToDO

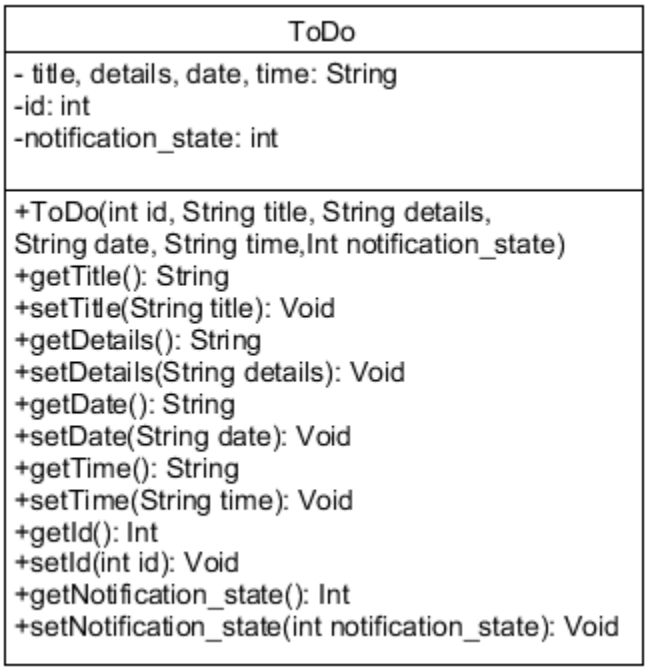


Fig. 8.4.6: UML/OCL specification for TODO

context	ToDO
pre	● User Must be logged in to the System to Access the Heartrate Monitor ● Valid Permission to calendar Services, ● Valid Permission to Alarm Services.
post	● The User will have a Functional Alarm ● The User data will be stored in the Database.

7. RecoverRate

Fig. 8.4.7: UML/OCL specification for RecoverRate

context	RecoverRate
pre	● User Must be logged in to the System to Access the Heartrate Monitor ● Valid Permission to Camera Application, ● Valid Permission to Flash Services.

post	• The User will have a Predicted Value with HeartRate Recovery Time. • The User data will be stored in the Database.

8.5 Traceability Matrix

Please see Appendix A at the end of this document for the traceability matrix.

Our classes differ from our domain model because we decided to take a different approach in the code implementation. Our domain model was originally based on a *pipe and filter* approach. We centered the model around how one object would process the data before presenting it to the next object. So for example, the SearchManager filtered and processed the search parameters from the controller and passed the data to the DatabaseOperator. Similarly, the DatabaseOperator processed the data from the SearchManager and output the filtered information to the DatabaseOperator.

However, in implementing the code we choose to "divide and conquer" and addressed the more complex tasks of the project first. Thus, we made data collection from sensors (used for the step counter) and database communication a priority. As a result, we shifted the software architectural style to a more task oriented approach. This leads to different class objects and different naming conventions. We wanted to name the classes according to their associated task or activity. Thus we have the classes names such as StepCounter and RegistrationActivity.

Although the class names and functionality differ from the domain model, we are ensuring that the vital domain is/will be addressed. Due to time constraints, some of the optional use cases will not be implemented. Thus these domain concepts will not link to a particular class. The domina model concepts that were not implemented highlighted in gray.

9 SYSTEM ARCHITECTURE AND SYSTEM DESIGN

9.1 Architectural Styles

Our design is comprised of three main components. A mobile application, a database and a web application. The Architectural styles that we have used to design our web application is two-tier Client-Server model. In this type of Architectural style, clients can communicate with a server over a computer network. To ensure data consistency, we are using an online database as part of the server. The client which is user running the web application can initiate a communication session to this server and retrieve patient health information as required. This is done by connecting to the server using TCP/IP socket connection. The advantage with this kind of model is that we have a centralized control and easy maintenance as changes to server will not affect the client end. In our system, the web application will display patient details to the doctor/admin. The online database stores patient information. This includes basic profile information in addition to step activity and heart rate which we get from the mobile application.

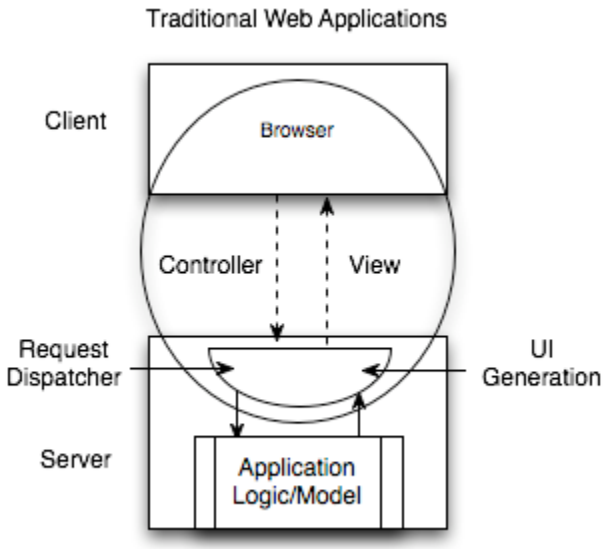


Fig 9.1.1: Client server style for a simple web application

The mobile application uses Model View Presenter (MVP) architectural pattern. The biggest advantage this model gives us is code organization. The Model layer controls how data is stored and modified in accordance to the business logic. View (Android Activity) is a passive interface that displays data and user actions to the presenter and defines flow logic. Presenter retrieves data from Model and show it in the View. It also processes user action forwarded to it by the View. This kind of

abstraction lets us define clear roles for each component. Object oriented design is being used as an integral design style in coding. In addition, we have defined clear responsibilities for each Activity so that changes to one component will not affect another component. This is especially useful for big applications which involve multiple contributors. As mentioned before, an online database is used for data consistency. This also helps when the app is used by multiple users and on multiple devices.

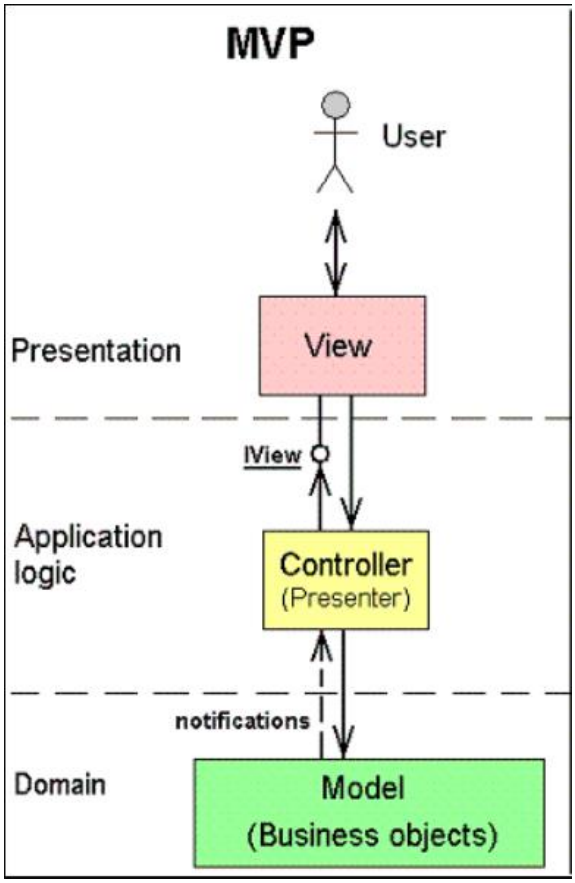


Fig 9.1.2: **Model-View-Presenter Pattern for Mobile Application**

9.2 Identifying Subsystems

Design Considerations

We followed OMG UML instructions to identify all subsystems and packages for our system [22].

Package

A package is the grouping of model elements of different kinds including other packages to hierarchy of model. Packages are mostly used to create an overview of a large set of model elements, to organize a large model, to group related elements and offer separate namespaces to avoid confusions among multiple libraries.

In our system, when identifying packages for UML package diagram, we followed the following principles:

1. Gather model elements with strong cohesion in one package
2. Keep model elements with low coupling in separate packages
3. Minimize associations between model elements in different packages.

Subsystem

Subsystem is a grouping of model elements that represent a certain technique used for decomposing a large system into several smaller systems. Subsystems are very useful to express how a set of components are composed into a large system for component based project development. A subsystem generally offers two aspects-

Specification elements - It offers an external view showing the services provided by the subsystem. It mainly describes the interface of the subsystem.

Realization elements - It offers an internal view showing the realizations of the subsystems. It contains the actual contents and functions of the subsystem.

There are multiple approaches for performing the specification techniques: Use case, logical class, and operations approach. A specific or a mixed approach may work for different systems. Mapping between specifications and realizations defines the role to be played when a task to be performed.

In our system, when identifying subsystems, we followed the following principles:

1. Defining a subsystem for each separate part of the large system
2. Choosing specification technique depending on the kind of the system and its subsystems. In this case, we chose use case approach.

Identifying subsystems involves backtracking, evaluation and revision of possible solutions. The heuristics we used to identify subsystems are:

- Creating a dedicated subsystem for objects in one use case into the same subsystem
- Creating a dedicated subsystem for objects used for moving data among subsystems Ensuring all objects in same system are functionally related
- Minimizing number of association or interaction between subsystems

We further considered the following criteria:

- The kind of services provided by the subsystems
- The contents of the subsystems
- High Cohesion - Each subsystem will perform only one activity
- Loose Coupling - Minimum dependency between the subsystems

In general, objects that are identified during requirement analysis were grouped into subsystems. The degree of cohesion and coupling between subsystems are used to perform subsystem decomposition.

The Subsystems

As we observe the system sequence diagram of our system design, we noticed that our system has a web application on the physician end and a mobile application on the patient end. The mobile application only interacts with the local database, while web application interacts with online database. Furthermore, we observed that local database has access to online database for updating and storing information.

Considering the heuristics mentioned in the above section, we decomposed our system into following subsystems:

1. Mobile application
2. Local database
3. Web application
4. Online database

Based on our observation, the diagram of the subsystems and the containing packages looks following:

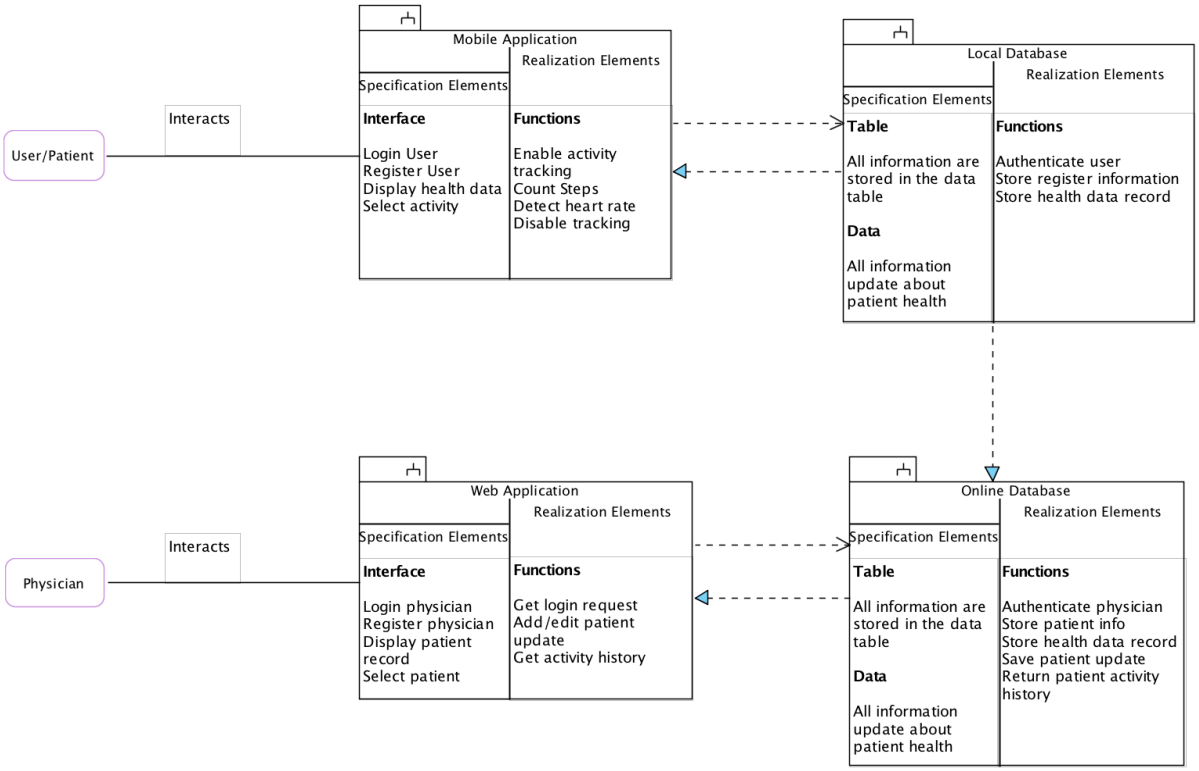


Figure 9.2.1: UML package diagram of subsystems

9.3 Mapping Subsystems to Hardware

Mobile application

The mobile application initiates the vital parts of the patient end of the system: tracking user activity, detecting user's steps, detecting heart rate of the user and disable activity tracking. The accelerometer sensor and camera sensors are crucial for the functions performed inside this subsystem. The interface allows the user to enter information for login and register purpose. It also displays the log of health vitals during particular activity and retrieved record of health data from the local database.

Our application has been developed for Android operating system based handheld (mobile devices). The hardware for this subsystem will be any Android mobile device that has API level 15 or higher. It should also have its accelerometer sensor working and functional medium resolution back camera for the step detecting and heart rate monitoring service.

Web application

The web application initiates the vital parts of the physician end of the overall system: receiving login and register request, updating patient information (add/edit) and retrieving the activity history of a particular patient. The web page interface allows the physician to enter information for login and register purpose. It also allows physician to choose any particular patient and displays health data and information of all patients retrieved from the local database.

The web application has been developed to be displayed via any web browser (e.g. Safari, Mozilla, Chrome, etc.). Any computing device (e.g. Desktop, laptop, tablet or mobile phone) with internet connection is capable of providing the hardware support for the web application subsystem. The web application does not provide offline service in the current design.

Local Database

The real-time functionality of the mobile application depends on the performance of local database. In our case, we used SQLite, the built-in database offered in Android because it is a highly reliable, public- domain and self contained relational database. Data tables are used to to store two types of information regarding the patient: the basic profile information and the health vitals during activity. In each session of activity, the local database fills out the data table with information of the most recent session and stores this updated information the database. It also sends the information requested by the user about a selected activity to be displayed on the mobile UI. As we are using a built-in database within Android, our hardware for this subsystem is also the Android mobile device used by the user (patient).

Online Database

The functionality of the web application depends on the performance of the online database. It receives activity information of different patients from the local database residing in the mobile device of each patient and stores them. From the physician side, it stores the basic information of each patient that are entered by the doctor. It also the registration-related information of the patient that is requested for authentication each time the physician logs in. Finally, the online database is also responsible for storing the patient information updated by the physician. This database is crucial for implementing the activity scheduling feature of our system, which is yet to be implemented. We plan to use Google Firebase for this subsystem as it offers realtime syncing and storing of data.

The following diagram gives an overview of the current mapping of the subsystems of our system into hardware:

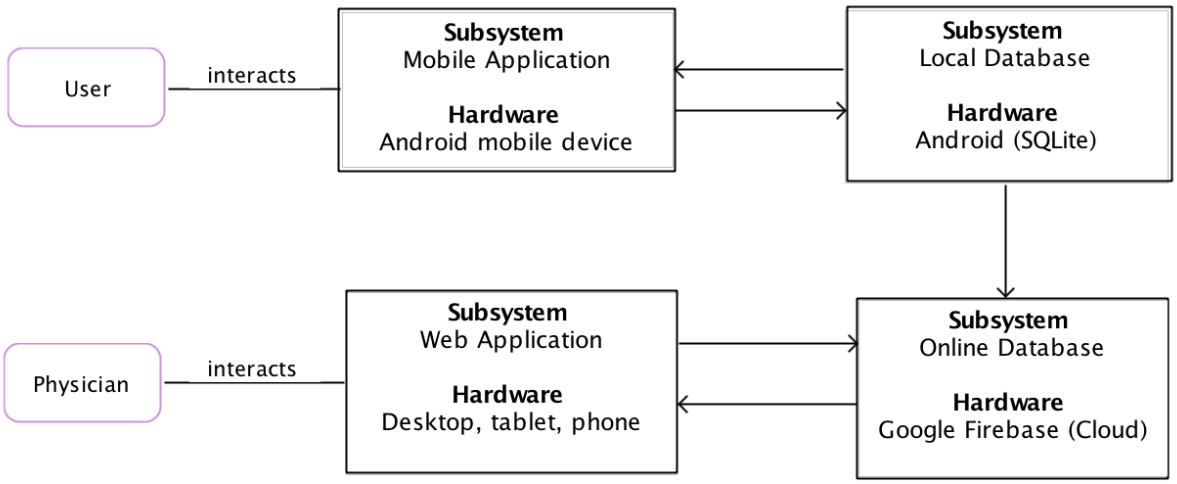


Figure 9.3.1: Mapping of subsystems into hardware

9.4 Persistent Data Storage

For the initial stages of our project, we used SQLite Database which is preferred for Android applications. The SQLite database is simple, convenient and robust. But, we later changed it to Firebase database which is online Cloud database to access the patient and physician data using both the mobile and web application. The data is stored as JSON and can be used by cross-platform apps whether Android , iOS or web applications.

In our project, the users reg0ister and login into the mobile application using authentication methods pre-defined in Firebase. The users will be added to Firebase with a unique User UID as shown in figure below.

Identifier	Providers	Created	Signed In	User UID ↑
adsf@sers.cpm	✉	Dec 12, 2017	Dec 12, 2017	445dssXu4vZ0Gh2Jf6jeXnj1oug2
tina@test.com	✉	Dec 13, 2017	Dec 13, 2017	4U3YLIHpBWQAaYtn0lbCa71WyC...
hima.bindu@rutgers.edu	✉	Dec 8, 2017	Dec 15, 2017	7OwbI8fsHBMVeiKvUJxsiIntE9C3
john.smith@email4.com	✉	Dec 12, 2017	Dec 12, 2017	7zCtByC9uvVtDc5Ydrpob51vSJq2
tina.drew@yahoo.com	✉	Dec 13, 2017	Dec 13, 2017	9K9wWQBC4mZYZO1nsp8dQXbRf...
himap2009@gmail.com	✉	Dec 8, 2017	Dec 11, 2017	DBN90pZfaUdxBqBkY8CVZAPxXE...
aj@yahoo.com	✉	Dec 3, 2017	Dec 14, 2017	GISf33wx9uOU1cBKo4WSR7NEm...
john.jones@email.com	✉	Dec 12, 2017	Dec 12, 2017	IgudXKBTAHcM2GZQo7yPchB8el92

Figure 9.4.1: Firebase User Authentication

Also, since the data is stored in JSON format, there are key-child pairs in a tree structure.

The main parent Firebase has two direct children - Patient and Physician. Each patient/physician is stored as a child with key as user UID. The Patient data includes all the profile data of the patient along with heart rate, daily step count and recovery time. The Physician data includes all the profile details of the physician. The below figure clearly describes how data is structured in Firebase database.

Figure 9.4.2: Firebase Example

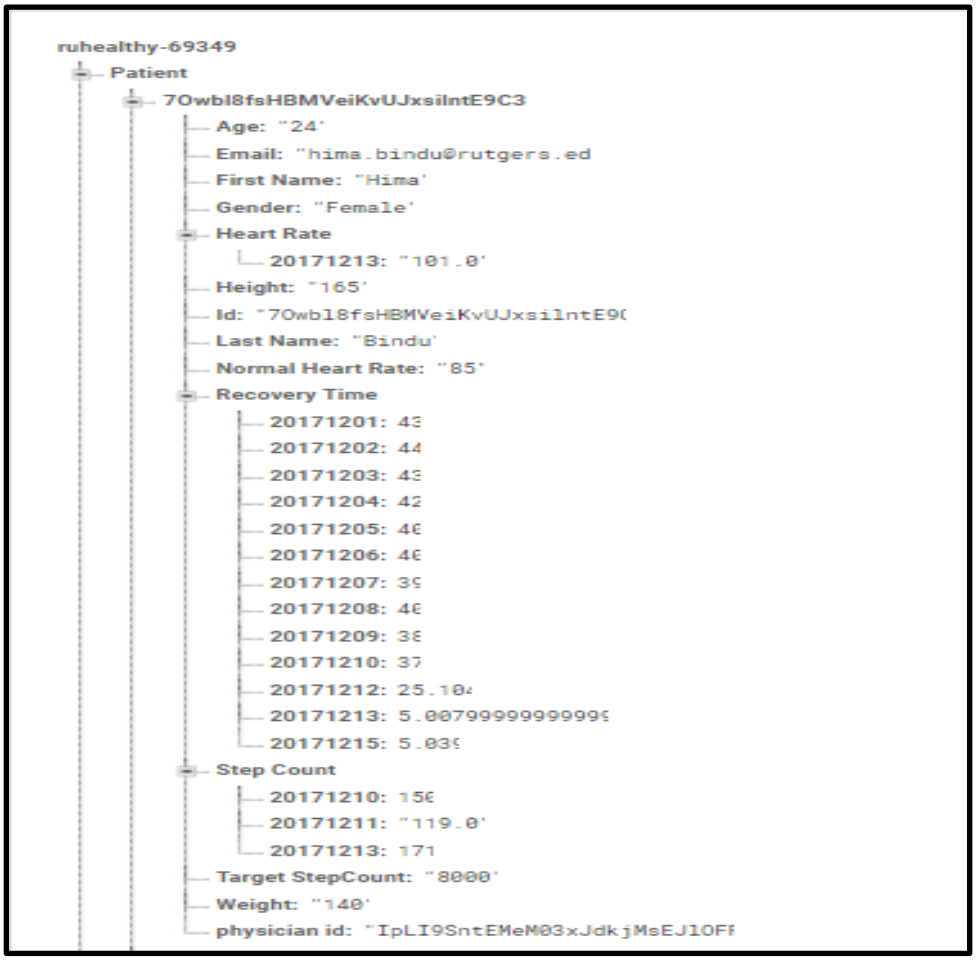


Figure 9.4.3: Firebase Database structure

9.5 Network Protocol

Several network protocols were implemented for our project:

1. **HTTP Protocol:** We used this type of protocol (Hypertext Transfer Protocol) in order to support the connection for the login/sign up in a web page and to support the connection between the user and the physician part, to stream the data request between the user terminal and physician terminal.

2. **SSH Protocol:** SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application. This protocol is uncensored because we are using the SQLite which is based on the SSH client – server stream tunneling.

3. **TCP/IP Protocol:** Nearly all computers and smart devices today support TCP/IP. This is not a single networking protocol, and will also be uncensored because this protocol will be used on the Physician (Computer) end and the Patient (Android Phone) end as a TCP/IP4 (Wi-fi, 4G, 3G, 2G).

4. **Named Pipes Protocol:** This protocol is used only on SQL Database communications**,** it will allow network access to SQL Server Express by supporting numerous network protocols, including NetBEUI, TCP/IP.

9.6 Global Control Flow

9.6.1 ANDROID APPLICATION

The system for Android Application is event-driven which is based on the user interaction with the device. When the user interacts with the smartphone and initiates an event, execution of event he/she will be able to choose the activity to perform. Also, once the user wants to check his/her previous activity log history, he/she will need to press the specific activity so the corresponding data will be retrieved from database and displayed to the patient.

There will be no need for timer in this current system. This means that it will be concerned about real time as it is a real-time system. However, it is not periodic. The data is processed on real time basis, but it only does so based on user-initiated event.

9.6.2 DATABASE

The database has some periodic properties as the heart rate recovery time of the user is stored in the database on a daily basis. This is done for the prediction of heart rate recovery time. Also, the anomaly of heartrate detection needs data from last ten days to estimate the predicted recovery rate. The database also operates in real-time in the sense that the step count, heartrate, etc. data are stored on real time basis.

9.6.3 WEB APPLICATION

The web application is also event-driven in the sense that physician needs to interact with the interface to initiate data display. As the purpose of the web application only display data processed and retrieved from the mobile app, like the user of mobile application, web app user interacts with the controls of the app and response to those events are executed.

9.7 Hardware Requirements

For the Physician Part

1. **Operating System:** it can be Microsoft Windows (7, Vista,8.1,10), Mac OS, Linux.

2. **Display:** Any output Source can perform well (not requiring any special sources), with any resolutions.

3. **Minimum Memory :**1 GB of RAM (in order to open the Physician web page and performs the job correctly without any glitches).

9. **CPU:** Single core CPU and UP**.**

5. **Storage:** no need for Local storage because all the information will be stored on online Database for the Physician so it will be retrieved from the server based on the Physician Request.

6. **Network:** 56 Kbps (minimum) in order to perform the request in sufficient way (less will take longer and will cause data lost sometimes).

7. **Browser support:** supported by latest versions of Google Chrome and Firefox .

For the Patient Part

1. **Operating System:** Android (based on Linux OS) with minimum SDK (2.2).
2. **Display:** it will require colored Screen with at least 640 x 480 pixels and more.
3. **Minimum Memory:**1 GB of RAM to perform the OS and Application internal requests to be performed well without glitches.
9. **CPU:** Dual Core CPU and UP.
5. **Storage:** it will need sufficient amount of space (100MB-500MB) to store the Application resources and temporary files and also to store the local backed-up database.
6. **Network:** Wifi-4G-3G-2G.

10 ALGORITHMS AND DATA STRUCTURES

10.1 Algorithms

10.1.1 STEP COUNTER

The step counter uses the phone's accelerometer to count the steps. The accelerometer uses the standard 3-axis sensor coordinate system to see the data values. This is defined relative to the screen of the phone in default orientation. The step detector triggers an event each time the user takes a step. The latency or delay is expected to be around 2 seconds. The accelerometer sensor measures the force in m/s^2 , which is applied to the device on all the three axes(x,y and z), including the force of gravity. These values are normalized so that the value is close to zero. And then the estimated velocity can be checked with a given threshold. If the estimated velocity is more than the threshold, it will be counted as a step. After getting the step count, we have used the below formulas to calculate the miles runs and the calories burnt.

Miles = number of steps * 0.0005, Calories = Miles* 0.0128

10.1.2 HEART RATE MONITOR

We have used an open source algorithm for determining heart rate. The algorithm uses the phone camera to capture the image of user's fingertip and calculates the heart rate based on redness of the image. A brief description of the algorithm is as follows:The flash and the wake screen lock are acquired when the camera is turned on. The preview frame is taken as an image of the YUV420SP format. This is then converted to the known RGB image format. This is done by a set of equations:

*R = 1192*y + 1634 * u ; G = (1192 * y - 833 * v - 400 * u); B = (1192 * y + 2066 * u);*

Where R,G,B correspond to the RGB components and y,u,v correspond to the YUV420SP format.

After the redness amount is obtained as explained above, this is stored as the average red value. This is done repeatedly for as long as the fingertip is placed on the camera. A rolling average (or moving average) filter is used to smooth out this data. When the average is higher or lower than the average, heart beat can be obtained and displayed.

10.1.3 RECOVERY TIME

As part of the final application, we will be adding a feature to determine the "recovery rate" of the patient. As our app focuses on the exercise part of the user's activity, this is a very good indicator of the user's fitness. When the user is exercising, their heart rate is spiked up. The number varies on exercise time and intensity of exercise. When the user stops exercising, heart rate returns to normal heart rate in about 3-5 minutes, with a majority of drop in the first minute. A lot of studies show that this "recovery time" is an important indicator of the user's fitness. Giving this information to the physician will help in better diagnosis. It is also especially useful for patients who have undergone surgery and are in their recovery period or for personal trainers to track their client's fitness levels. Since this requires learning the user's heart rate values every day, we had originally planned to use Artificial Neural Networks (ANN). We wanted to use the feed forward, back-propagation algorithm to predict future recovery time values. When constructing ANN models, one of the primary considerations is choosing activation functions for hidden and output layers that are differentiable. This is because calculating the back-propagated error signal that is used to determine ANN parameter updates requires the gradient of the activation function gradient. We use the sigmoid function for this purpose.

$$a(x) = \frac{1}{1+e^{-x}}$$

This is used for the feed-forward network, along with our input values and random weights assigned to each link. We use the actual value for the final data point as the target and compare with our predicted value. This lets us determine the error. We can now determine delta values by multiplying this error with activation function values. The weights are then adjusted using the back-propagation algorithm.

*derivative = output * (1.0 - output)*

*error = (expected - output) * transfer_derivative(output) (for a neuron)*

*error = (weight_k * error_j) * transfer_derivative(output) (for the hidden layer)*

Where error_j is the error signal from the jth neuron in the output layer, weight_k is the weight that connects the kth neuron to the current neuron and output is the output for the current neuron.

Network weights are updated as follows:

*weight = weight + learning_rate * error * input*

The network is trained like explained for daily values till we can accurately determine the target value. This is used to predict the next day's value. When the patient is consistently having lower time than predicted value, the physician can be alerted to check on him.

Prediction using Bayesian Curve Fitting:

Al0though we had originally planned to implement neural networks for our prediction, we didn't do so for a couple of reasons. Simply coding a neural network in the android application was beyond our scope as our efforts were required for various other parts of the project. The simpler solutions to neural networks were not good enough and could not give us good results for the prediction. We could use Tensorflow or other open source Neural Networks available online. But the data set we have is not sufficient to train the neural networks and use it for prediction. Neural networks was also not the best predictor for short term predictions and pattern recognition, which was more appropriate to our case here.

Upon research, we chose **Prediction using Bayesian curve fitting**. The advantages of this prediction are that the coded predictor gives good prediction and more importantly, a small dataset is enough to fit the curve. A point to be noted here is that we plan to use recovery time from previous days to predict future recovery time. Since regular exercise can benefit the user by decreasing or maintaining recovery time, this is more of a pattern recognition and prediction problem, which makes Bayesian Prediction a great choice. Since we have no hard and fast rules about what recovery time is ideal and also since it varies for every individual, we can use this prediction value and compare with the actual value to detect an anomaly.

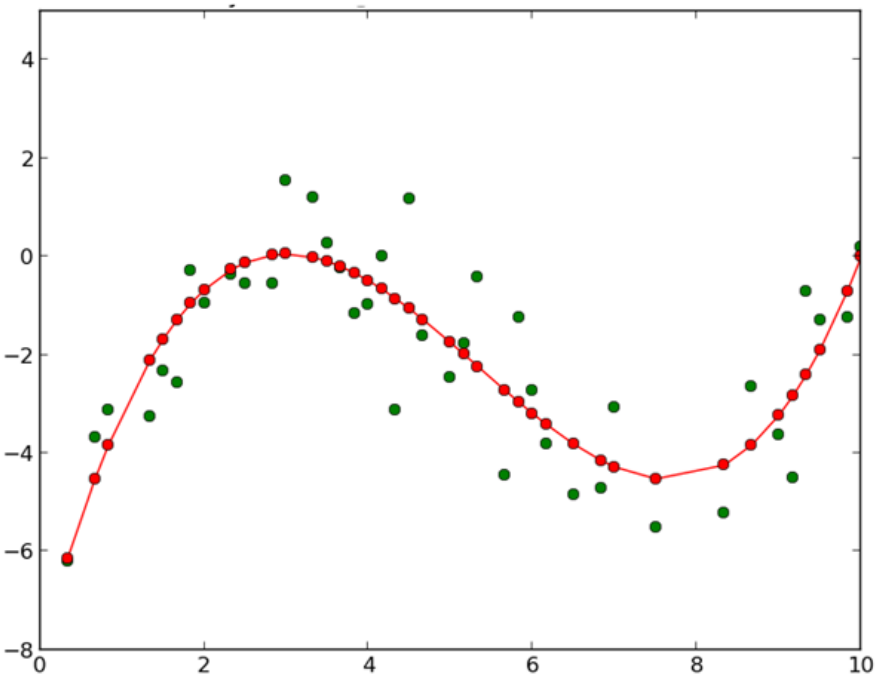


Fig 10.1.1: Bayesian Curve Fitting

The goal in the curve fitting problem is to be able to make predictions for the target variable t given some new value of the input variable x on the basis of a set of training data comprising N input values $x = (x_1,...,x_N)^T$ and their corresponding target values $t = (t_1,...,t_N)^T$. We can express our uncertainty over the value of the target variable using a probability distribution. For this purpose, we shall assume that, given the value of x, the corresponding value of t has a Gaussian distribution with a mean equal to the value y(x, w) of the polynomial curve. Thus we have,

$$p(t|x, w, \beta) = N\ t|y(x, w), \beta-1$$

where, we define a precision parameter β corresponding to the inverse variance of the distribution. We now use the training data {x, t} to determine the values of the unknown parameters w and β by maximum likelihood. Having determined the parameters w and β, we can now make predictions for new values of x. Because we now have a probabilistic model, these are expressed in terms of the predictive distribution that gives the probability distribution over t.

The above equations when translated to code will fit the points in the dataset to a curve approximation. This is a good way of prediction for us because we would like to observe this pattern in the user's exercise schedule and regularity of exercise.

10.2 Data Structures

Currently, we use **integer arrays** to store intermediate values in step counter and heart rate monitor. In Step Counter, we use arrays to store acceleration values for the three axes x, y and z. The steps counted, the calories burnt and miles walked are of integer, decimal and decimal data types respectively. In Heart Rate Monitor, we use arrays for storing average image redness, conversion to RGB and beats count since we have a set of values to store and process. Similarly, for Bayesian Prediction, we have used arrays to store data points and for the calculation of all intermediate values. We will use 2 dimensional integer arrays for weights of links between input and hidden, and hidden and output nodes. The rest of the intermediate variables are almost all of type **integer** or **float**. We also will use list or array data type to obtain values from the database to the web application for analysis and display, and android application to display history. Lastly, we have used **Hashmaps** to store and retrieve data from Firebase since we required to store our main data points (Steps, Heartbeat, Recovery rate) for each day. We used date and value as key-value pair for the Hashmap.

11 USER INTERFACE DESIGN AND IMPLEMENTATION

During brainstorming session prior to start working on the implementation, we analyzed several similar apps that are available in the market and identified what are the likeable features popular among the users. Based on these observations, In the first iteration of report, we identified UI requirements for our system which are trivial for the functional purpose of our system, but crucial for the user to succeed in achieving user goals with ease. The requirements were following:

The interface of the mobile app will be easy to navigate so that patient can change between menus with minimal effort even when working out.

Both the mobile and web application shall be intuitive; so that general experience with any Android or web app will suffice to use the basic features effectively.

We designed the initial mock-ups for the mobile and web application user interface keeping these requirements in mind. The mock-ups were designed for the use cases that are most important for

the system to be functioning to serve user's needs. We tried to include in these mock-ups the features that are not only essential for functional purpose but also make the system easy to use for the human user.

11.1 Modifications made during Implementation

During the implementation phase, we realized that though our mock-ups were well thought out and were designed with user-centric focus in mind, several changes were required to be made. This was basically required due to repeated revision made on our initial idea as we went deeper into the implementation phase.

11.1.1 MOBILE APPLICATION

Information required for registration and login *:* One significant change we made was the information required for the initial account information for the user. In our mock-up screen for registration, full name of the user, a username and password were sufficient for a new user to open an account with the mobile application. During implementation, we realized that it is crucial for the authentication purpose to require the user enter his/her e-mail address during account initiation. As our system had a web application on the physician end and we plan to sync the mobile and web

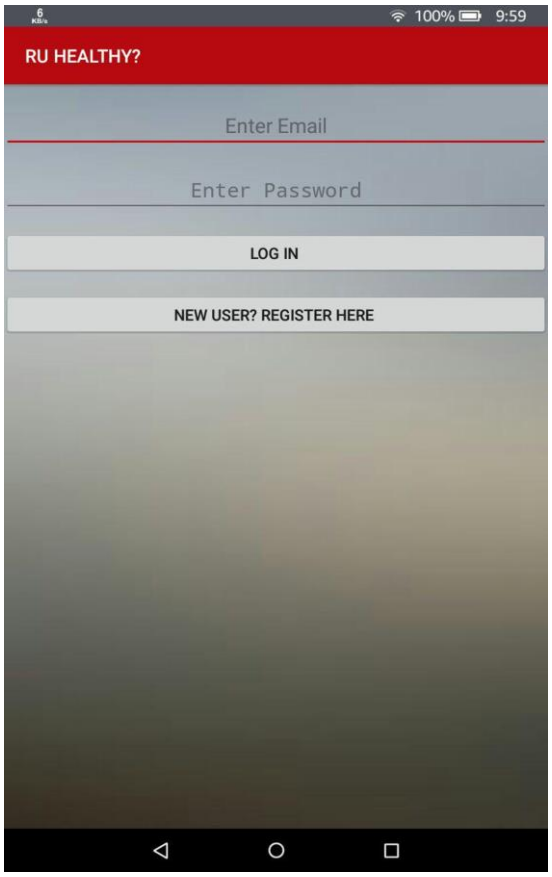


Figure 11.1.1 Login Page (Mobile)

applications to operate in harmony with the online database, inclusion of the e-mail address seemed a wise choice for security purpose. We also decided to include a field for the user to confirm their password to avoid cases where the user may have pressed a wrong key when typing the password. Confirming the password a second time will ensure that the user entered the password they intended to.

Inclusion of Dashboard page: Furthermore, in our initial mock-up design, when user logs into his/her account with credentials, the user is directed to profile page containing the basic information like age, height, etc. In our current design, once logged in, user gets to the Dashboard page, which contain tabs like Profile, Heart rate monitor, Step counter, Log out, etc. The idea is that instead of directing the user to the profile, we will allow the user to choose wherever he/she wants to navigate from the Dashboard page.

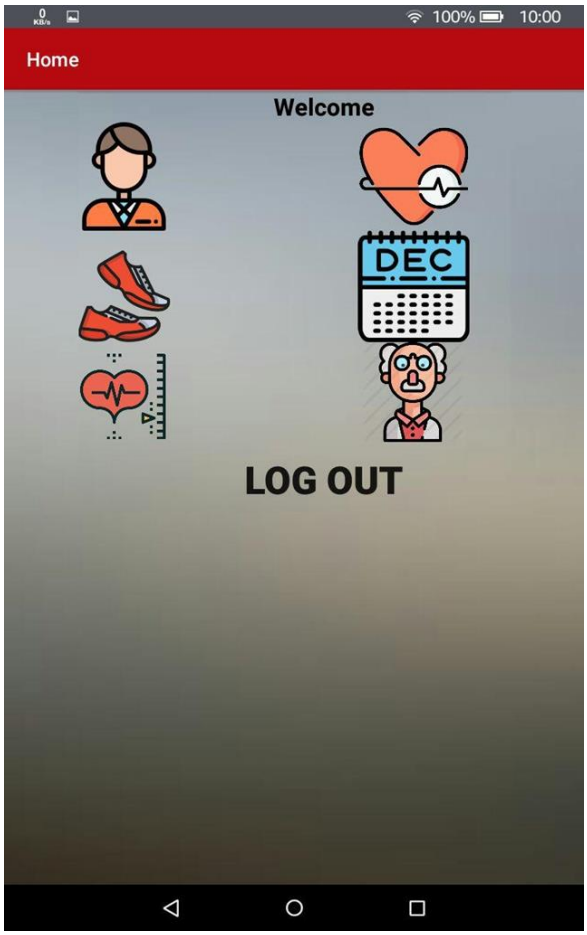


Figure 11.1.2 Dashboard

Inclusion of additional page for heart rate monitor, scheduler and select physician: Another major change we have brought to our UI design is a separate page for the heart rate monitor. In the initial mock-up we excluded this feature as it was not among the use cases we were focused on. However, towards the end of first iteration, we decided to include the heart rate monitoring feature along with step counting feature. The dashboard was a vital part for this decision, as by including the additional page called Dashboard, we allowed user to navigate to all the basic features of our app as needed. This certainly reduces user effort, as user can view the feature they need to one at a time with a single click on the corresponding button.

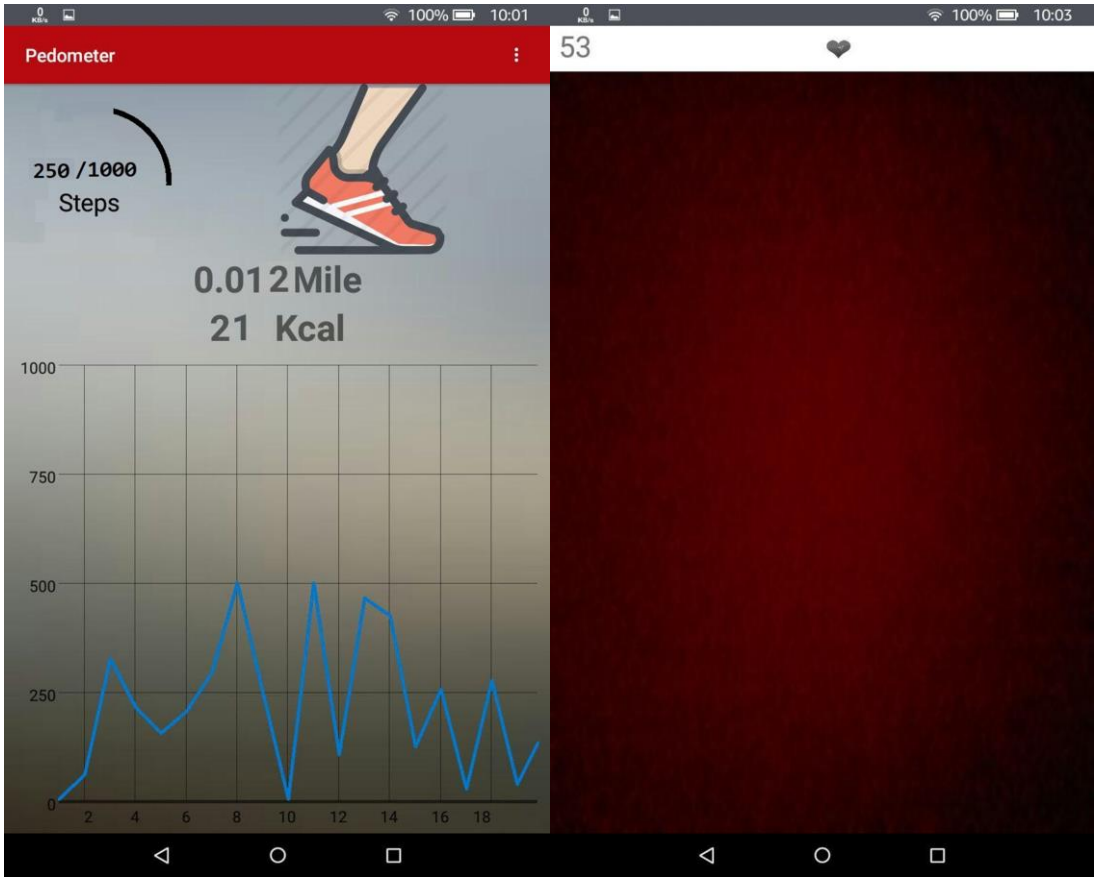


Figure 11.1.3 and 11.1.4 Step counter (left) and Heart rate monitor (right)

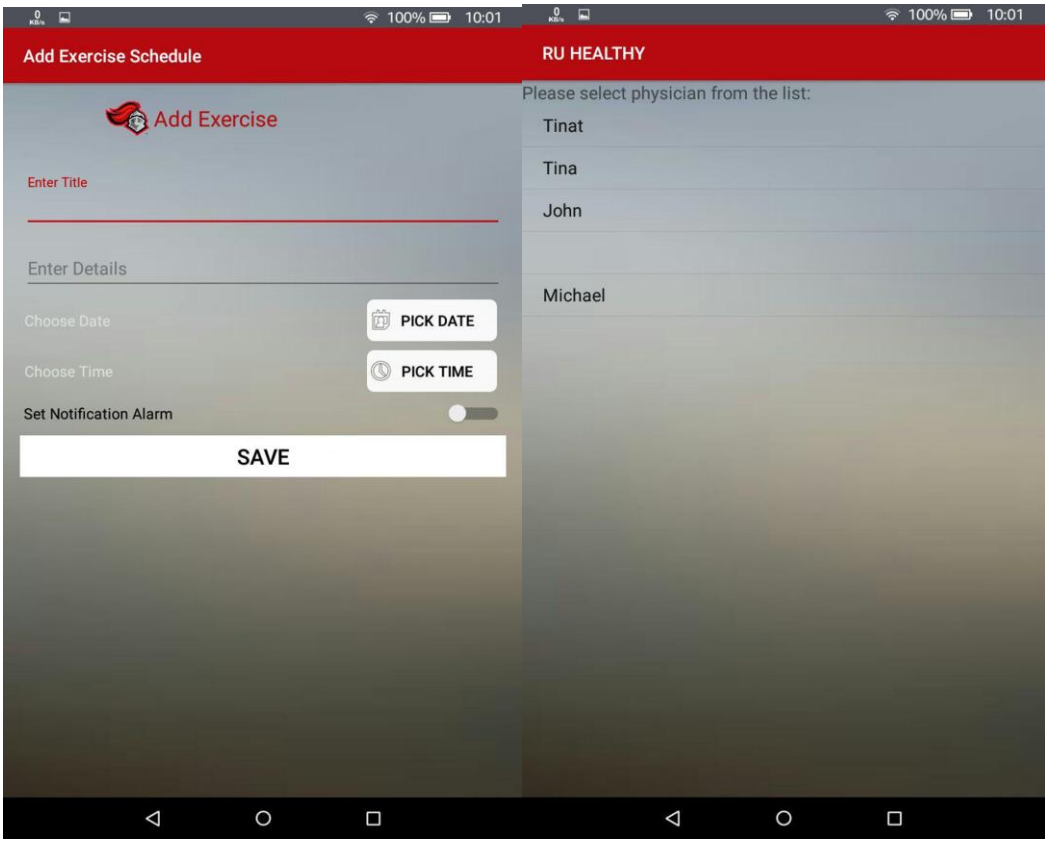


Figure 11.1.5 and 11.1.6 Exercise Schedule (left) and Select Physician (right)

11.1.2 WEB APPLICATION

Webpage theme: After receiving feedback from reports 1 and report 2 we decided to update the webpage design so that the pages had a consistent theme. We also tried to match theme closer to that of the android design. The basic UI design includes a red bar at the to the page that allows access to other pages. The background color is a gray instead of white. Most pages also include the new Rutgers logo so that the user can recognize the application.

Information required for registration*:* Similar to the mobile application, a significant change for the web application was the initial information required for registration. In our mock-up, we only included user's name, email and password as the required information. We decided to go back to this approach. So now a username, password and User ID are required for registration. Once and account has been

created in the firebase database, the user is the directed the profile page where they can enter in additional information. This includes their full name, address, etc. Furthermore, we implemented the web application in such a way so that user can choose the type of role they play in the system; they can select whether they are a patient or a physician. In initial mock-up, the web application was specifically designed for physicians only. This enables the user to use RU Healthy services from a web browser along with the mobile application used during workout.

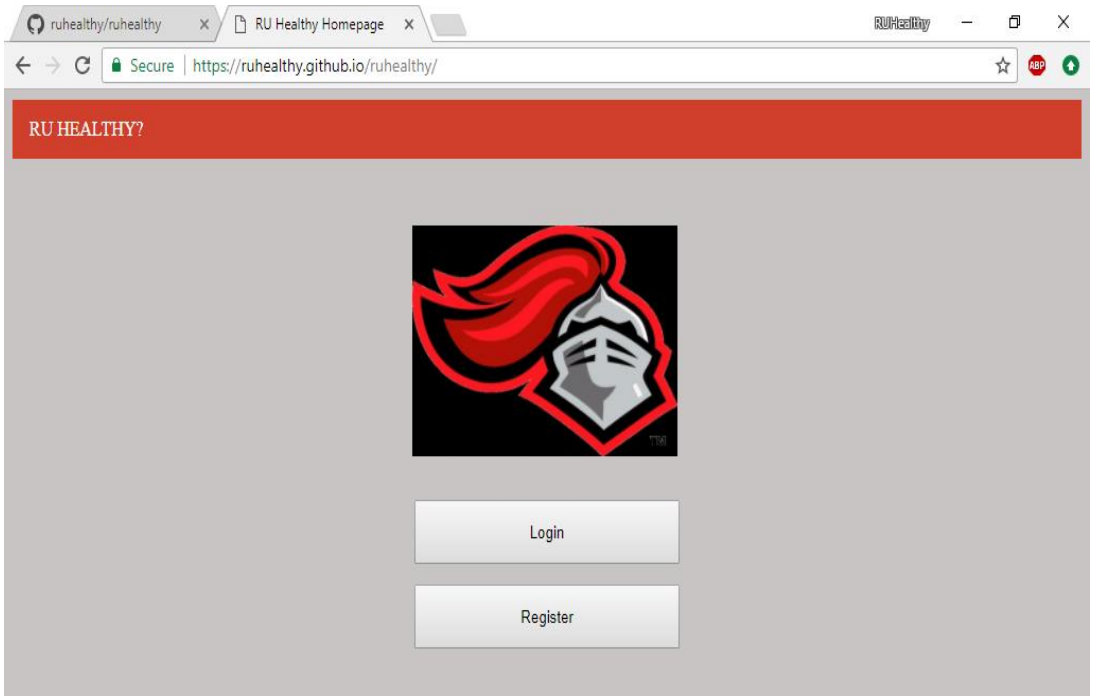


Figure 11.1.7 RU Healthy Homepage (Web)

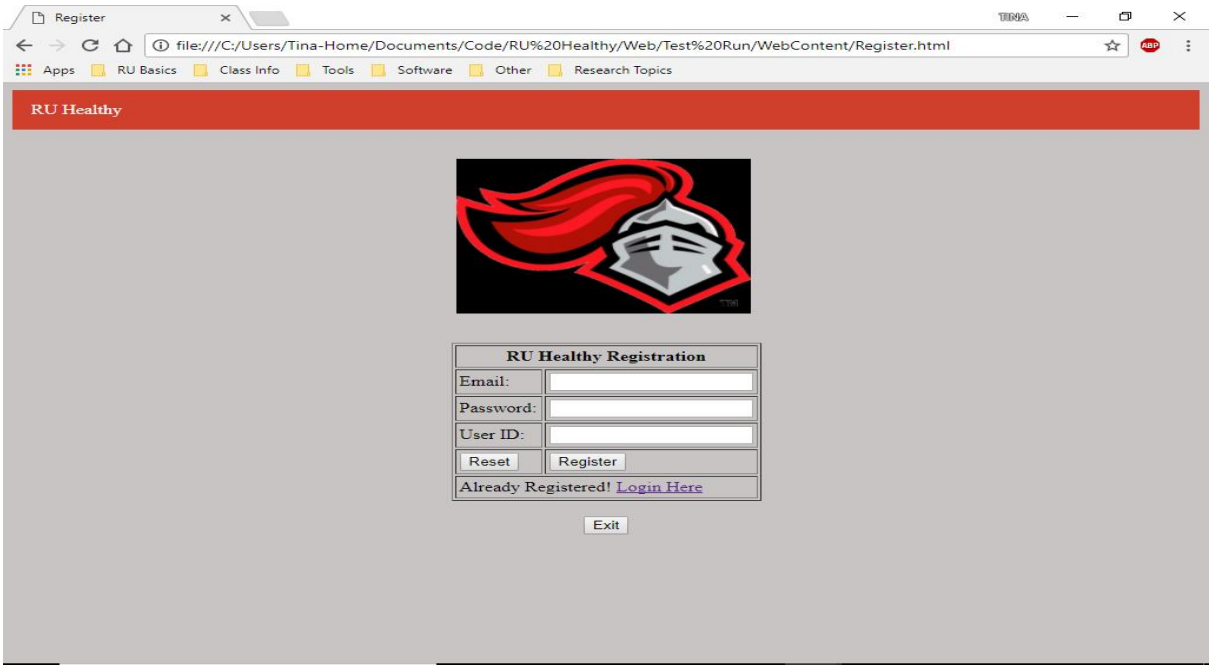


Figure 11.1.8 Registration (Web)

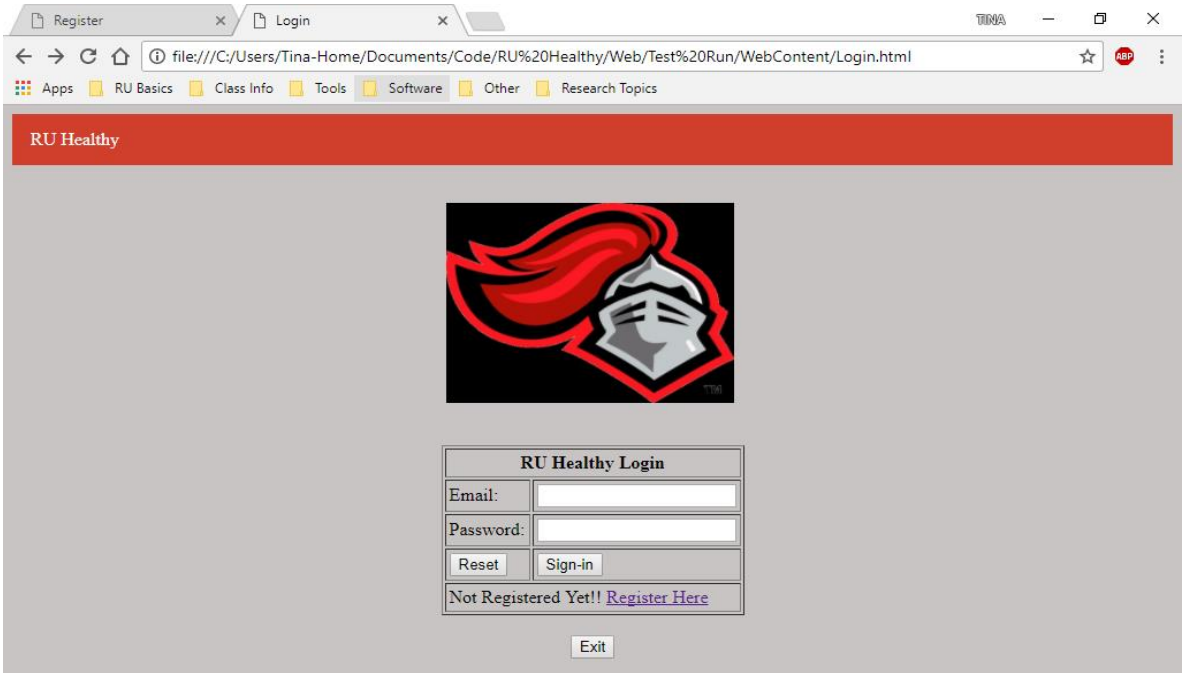


Figure 11.1.9 Login (Web)

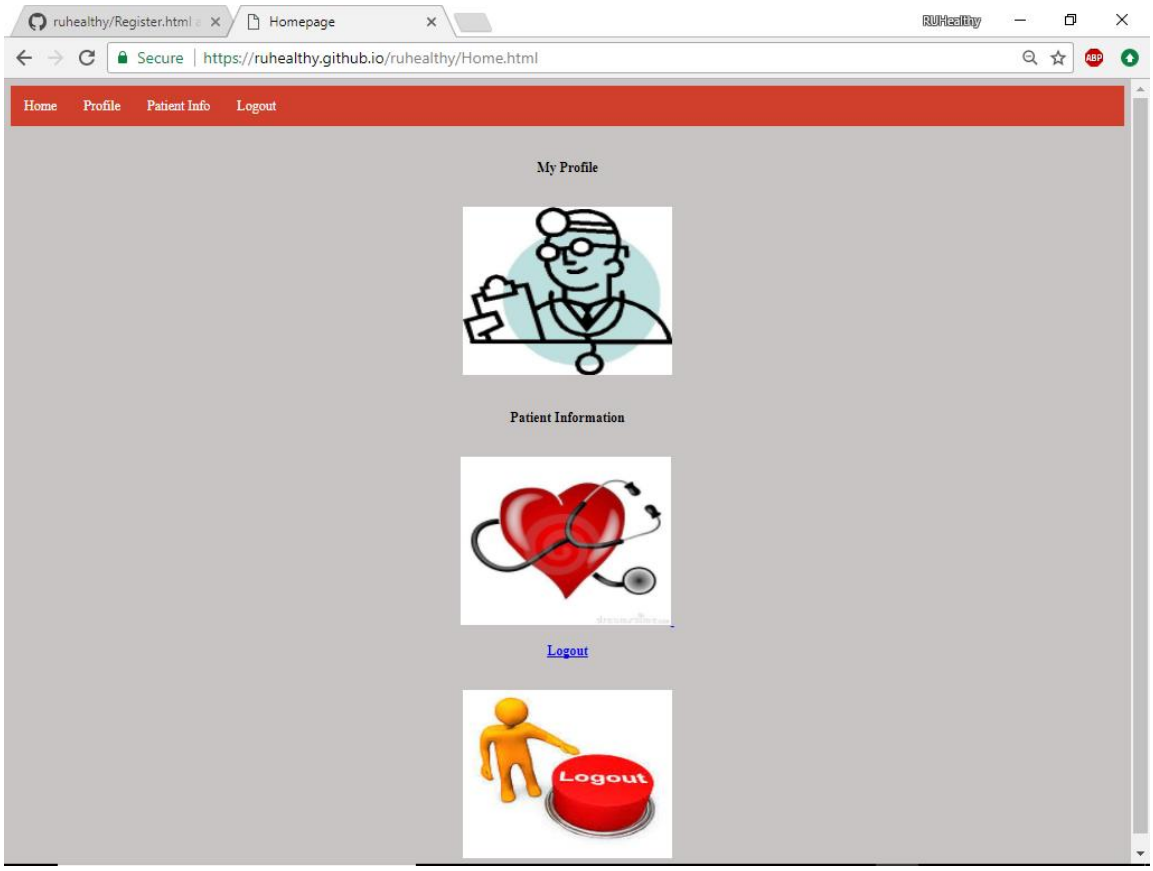


Figure 11.1.10 User Homepage (web)

Revised Patient summary page: Additionally, we customized the visual look of the patient summary page. In the initial mock-up, the patient's weekly activity summarized patient's performance in a graph and displayed a one-line summary of the missed sessions for the week. We later realized that the physician would require a summary on a daily basis to provide more accurate health advice. In the current version of implementation, patient activity for the day is summarized in a table which contains step count, heart rate, weight, etc.

Selecting a Patient page and menu bar at the top: We received review for our mock-up mentioning that the navigation between pages are not intuitive, which was mentioned as a non-functional requirement. In the current version of the web app, the website has a menu bar which contains quicklinks for Home, Patient Infor, Profile, etc.

Selecting Patient Data and Patient activity: Furthermore, in case the physician has multiple patients registered with him/her, the physician is able to select a particular patient using the unique patient ID assigned to them when they register with the physician. By clicking on 'Get Patient Data', patient can access the patient's profile and other information. The physician can also select each activity from the dropdown menu labeled 'Get Patient Activity' and receive corresponding exercise data as well.

When the physician navigates to patient information page it will be blank as shown below:

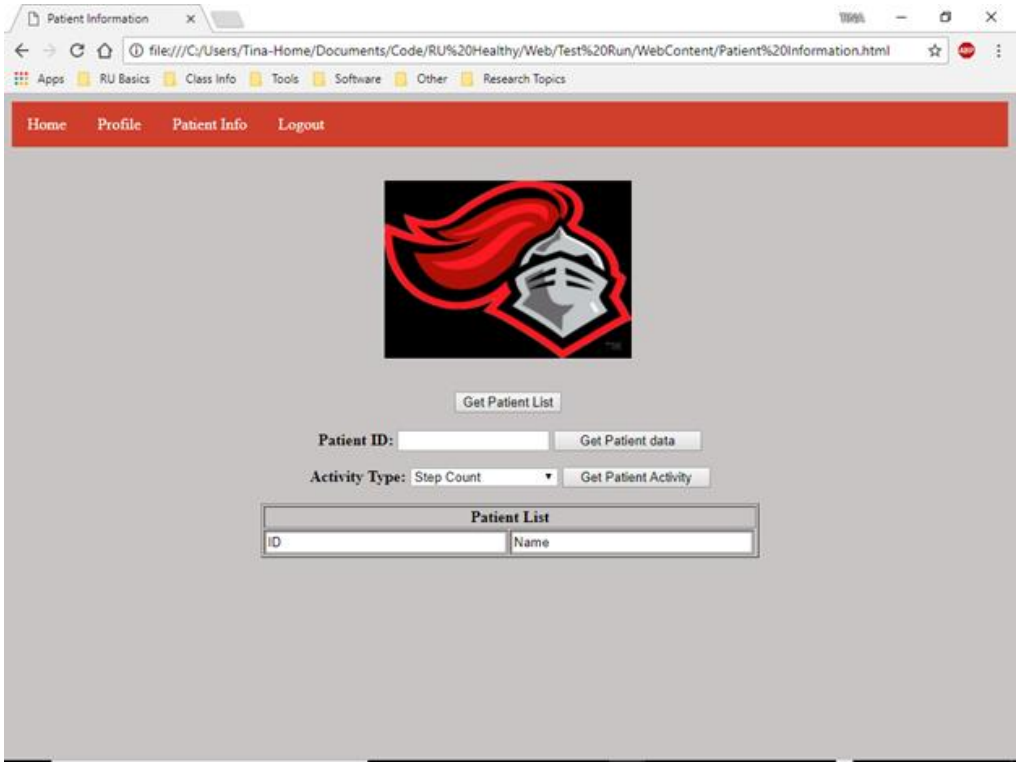


Figure 11.1.11 Initial Patient Information Page

From the patient information page, the physician should select the **"Get Patient List"** button in order to populate the patient list.

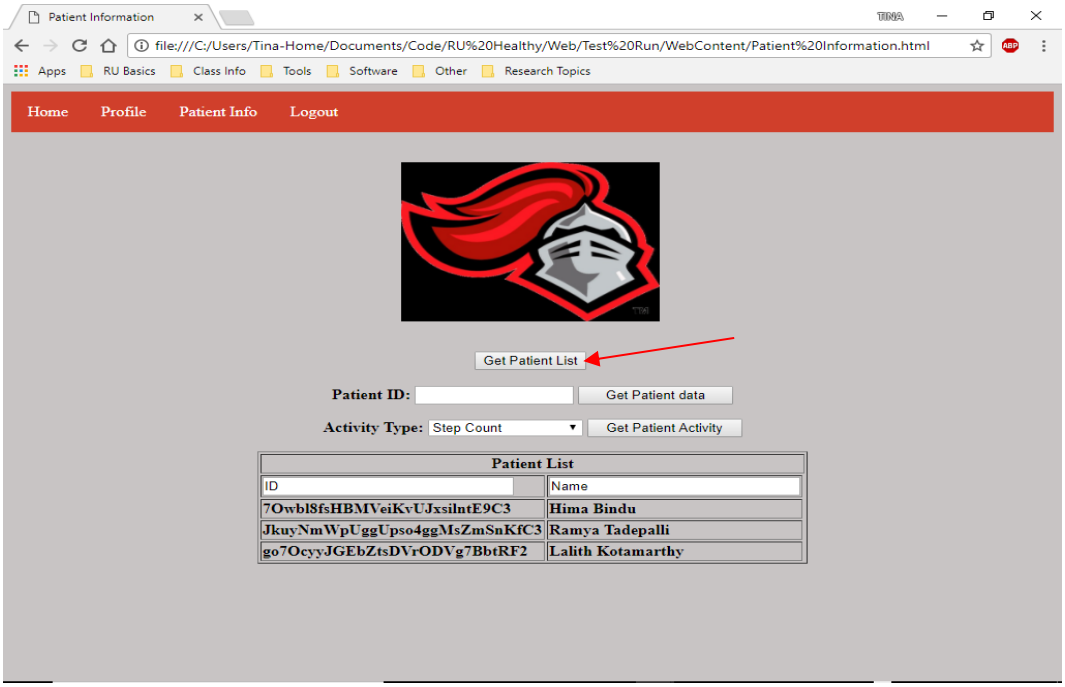


Figure 11.1.12 Get Patient List

Then the physician can enter in the correct patient ID and select get “Patient Data” button.

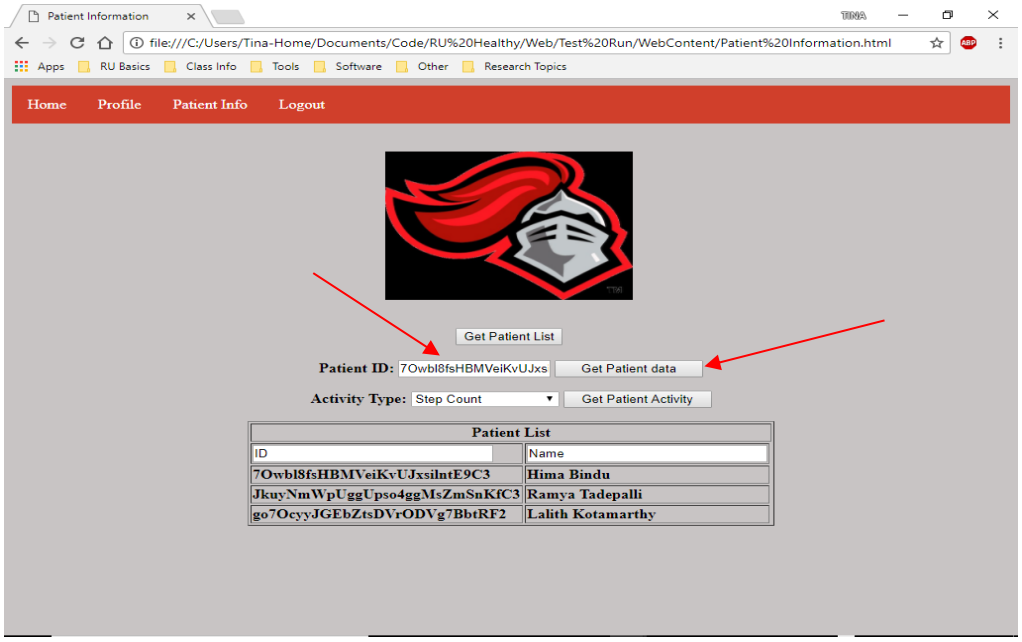


Figure 11.1.13 Get Patient Data Selector

This will redirect the physician to a patient data summary page.

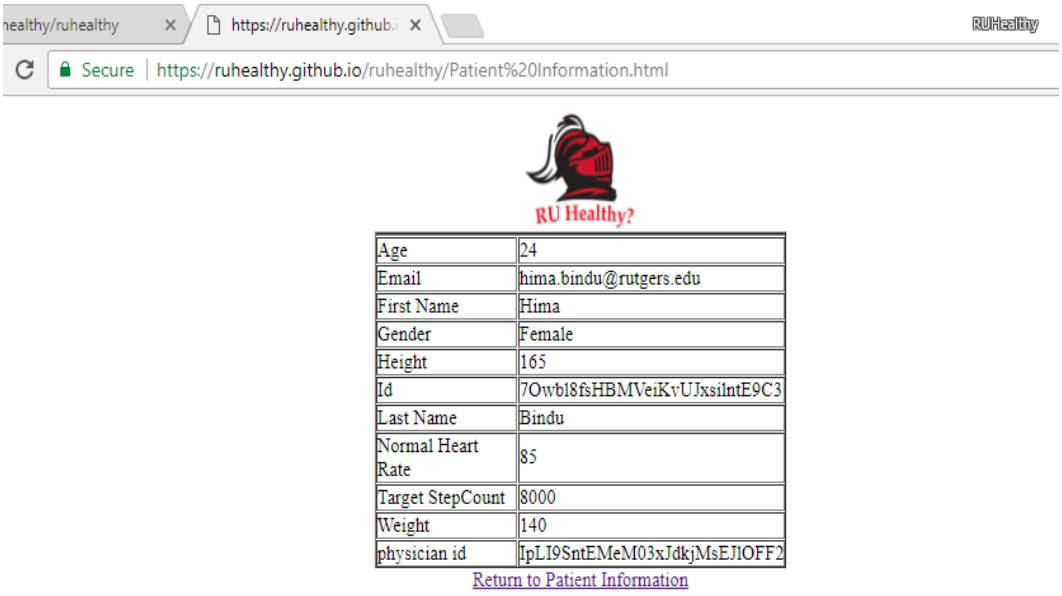


Figure 11.1.14 Patient Data Summary Selector

If the Patient has not selected the physician an alert will come up as shown below:

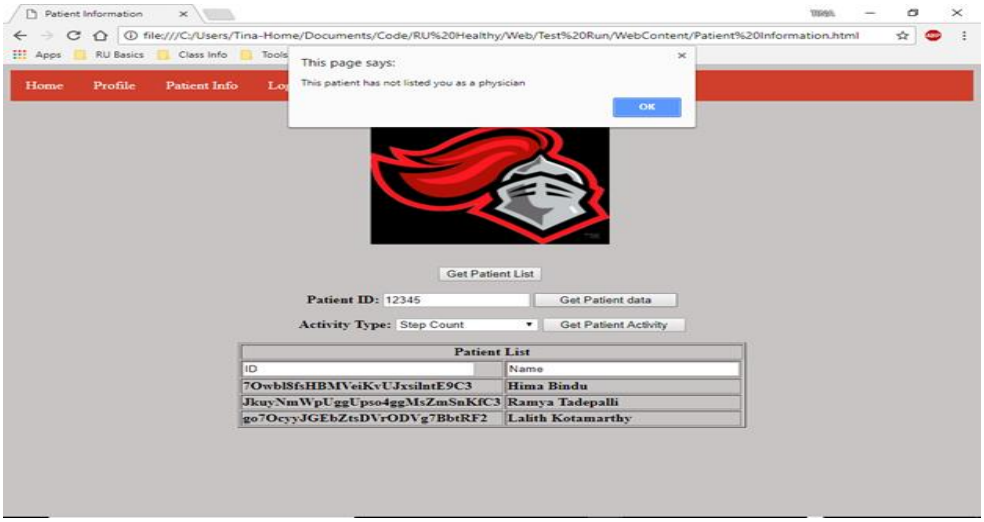


Figure 11.1.15 Invalid Patient Alert

A similar approach is also used to navigate to the patient activity information. The pages redirect to a activity page summary that is similar to the one shown below:

Figure 11.1.16 Step Count Activity Summary

11.2 Thematic approach for Graphical User Interface

In our initial mock up, we developed a basic theme for our interface based on the three colors: Red, Black and Grey. We further used green for the scheduling page, which is yet to be implemented. This decision was influenced by our logo which is taken from the logo of Rutgers University.

In keeping with this theme, our current UI for the mobile app ‘Black’ for text inputs and ‘Grey’ for visual or textual cues. In the Step counter page, we used ‘Green’ and ‘Red’ for the detecting buttons to give visual cue for the type of function (green denotes start, red denotes stop). We employed the same principle for the Heart rate monitor page. The Android icon and screen remains green initially. As the

user puts a fingertip on the camera sensor, the icon on the screen turns red to denote that sensor data is processing.

We further plan to use this color-themed cue for the activity page. In the initial mock up we used horizontal progress bar to visually represent user's accomplishments for the day. The ides was: progress bar will represent daily target (for steps, distance or calories) for the user while the already accomplished portion will be in green and the remaining portion of target for the day will be displayed in red. However, in our current design, we used round progress bar as we find this will provide more intuitive feedback for the user.

We plan to work towards making the interface more easy to navigate, easy to use and learn for the rest of the semester. A simple, yet functional and intuitive user interface is crucial for the mobile application of our system, as it involves user operating on it during workout. It is not easy to perform simple tasks on a touch screen when a person is walking fast or jogging. While it can be argued whether the design approach we are taking is the best way for minimize user effort and maximize ease of use, we do state that within the resource constraints and trade-offs and in our better judgement, we have tried to provide the optimal user-centric design solution.

12 DESIGN OF TESTS

Testing design is a number of techniques that add certain testability features to the system. Testing design is often viewed as executing a program to see if produces the correct output for a given input which wrongly implies that testing should be postponed until late in the lifecycle[1]. Actually, errors are introduced during the early stages of the software lifecycle, hence, testing activity should be started as soon as possible.

Testing uses different combinations of inputs to detects faults, but trying all possible inputs exhaustively will incur in a huge cost. In order to detect enough faults within a limited cost, we decide to make the testing design to follow the hierarchical structure of the system. We will start the test from individual components, unit testing, then the composition of these components, integration testing, and finally the whole system, system testing, ensuring the system suits the function and non-functional requirements.

So there are three different kind of test that should be made here:

1. **Unit testing:** For unit testing, we mainly use the strategy of boundary testing and for several specific cases, we just use the equivalence testing, like the input process in sign-in.

2. **Integration testing:** The phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.

3. **System testing:** It is a level of the software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

We begin with the design of our unit testing cases. Because there are too many, we'll provide the basics on the most important units of each class.

12.1 Unit Testing

1. **Test Coverage: Sign In** (Android App, Web App)

Tested Unit	**LogIn**
Assumption	The application interface showed up and the sign in input screen is waiting for the user input.
Expected Input	User Email or User Name, User Password
Expected Output	Sign in successfully done with proper input, Sign in unsuccessfully done with improper input, and make alter the same time.
Pass	Successful sign in with valid email or user name and valid password

	combination
Fail and The Probable Error	1. Sign in still success with improper input. Error related input checking. 2. Sign in success with proper input but no related data show in database. Error related to connection with database.

2. **Test Coverage: Registration** (Android App, Web App)

Tested Unit	**Register**
Assumption	The Application interface showed up. The signUp input screen is waiting for the user input.
Expected Input	User Email, User Name, User Password
Expected Output	Sign up and Information registration to database successfully done with proper input, Sign up unsuccessfully done with improper, and make alter the same time.
Pass	Successful sign in with valid email or user name and valid password combination
Fail and The Probable Error	1. Sign Up still success with improper input. Error related input checking. 2. Sign Up success with proper input but no functional connection was found , Error related to connection with database.

3. **Test Coverage: Step Counter** (Android App)

Tested Unit	**Step-Counter**
Assumption	User selects Step Counter Activity. Application is waiting for action

Expected Input	Click on start button
Expected Output	Step Counting-Miles Counting and Calories Burnt Calculations starts successfully and turns to other functions.
Pass	Step Counting-Miles Counting and Calories Burnt Calculations shows real and Actual calculations and Numbers.
Fail and The Probable Error	1. Click on start button, but nothing happens. Errors related training start function. 2. Click on start button, but No Hardware Detected Message Showed up..

4. **Test coverage: Heart Rate** (Android App)

Tested Unit	**Heart Rate**
Assumption	User selects Heart Rate Activity. Application is waiting for user's actions.
Expected Input	User's fingerprint.
Expected Output	Number of heartbeats per minute (BPM).
Pass	The BPM corresponds accurately to that of the user.
Fail and The Probable Error	1. Incorrect BPM. Error related to fail in the algorithm. 2. No BPM is given. Error related to absence of finger to read.

5. **Test Coverage: HistoryActivity** (Android App)

Tested Unit	**GetUserVitalSigns**
Assumption	User selects the option to see his/her activity history for a given time.
Expected Output	All of the user activities in the selected period.
Pass	All patient information is showed.
Fail and The Probable Error	1. Some information is missing. Error related to connection with the database.

6. **Test coverage: DashboardActivity** (Android App)

Tested Unit	**LogOut**
Assumption	User is in the Dashboard Activity and selects the Logout function.
Expected Output	User profile is closed. Application shows Main Activity.
Pass	User's profile is successfully logged out of the system. No user's information is left.
Fail and The Probable Error	1. Display shows is logged out but when returning to the application, the profile is still open. Error related to connection to the database.

7. **Test coverage: PatientInfo** (Web app)

Tested Unit	**GetPatientList**

Assumption	User in the main page selects, get patient list.
Expected Output	Webpage navigates to a page that returns a list of patients that that have selected this physician.
Pass	User's profile is successfully logged out of the system. No user's information is left.
Fail and The Probable Error	1. Patient list returns no data. 2. Patient list does not reflect the patients that selected the doctor. 3. Patient lists includes additional patients.

-

8. **Test coverage: PatientInfo** (Web app)

Tested Unit	**GetVitalSignsWeb**
Assumption	User in the patient list page, selects the desired patient
Expected Output	Webpage navigates to summary page with patients health information.
Pass	Summary page returns the correct information.
Fail and The Probable Error	1. Patient health data is not returned. 2. Incorrect health information is displayed. 3. Webpage navigates to incorrect page.

12.2 Integration Testing

For integration testing, the basic hierarchy of levels is described below:

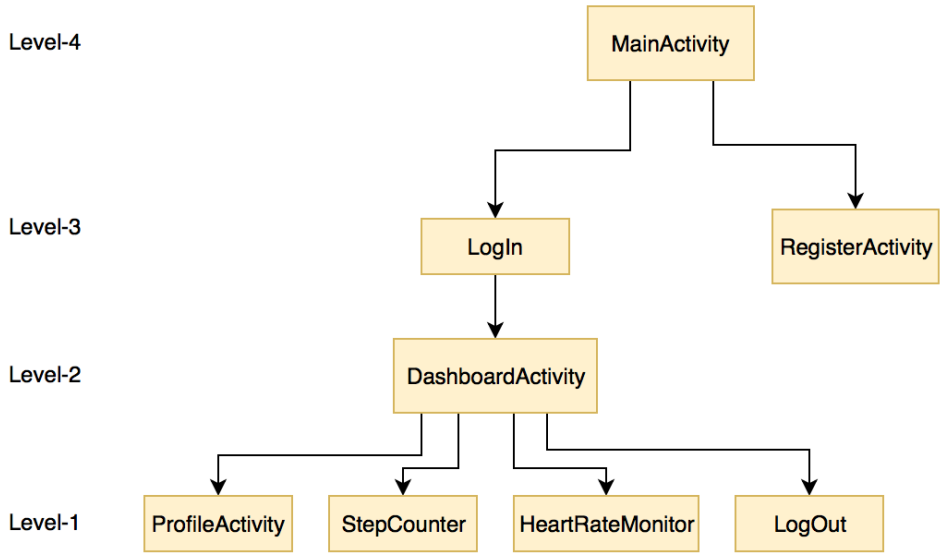


Figure 12.2.1: Testing Hierarchy

12.2.1 STRATEGY

For simplicity, the easiest way of integration testing is going to be “big bang”, in where we link all the units and to try to test it at once. Every unit has been already tested individually, so their integration is expected to produce good results.

12.2.2 PLANS

Unit Testing

For unit testing we completed automated tests. We did this by creating java code to run failure scenarios. The automated code for the registration module tested the following conditions:

1. whether a valid email address (with ‘@‘) is accepted as input and results in pass.

2. whether an invalid email address is rejected as input and results in fail.

3. whether a name that contains an invalid character (‘*’) results in fail.

4. whether a password containing invalid character '^#%' results in fail.

For the remaining unit tests, we intend to continue with this strategy.

Integration Testing

Up to this point, most of our integration testing has been done manually. For manual tests, we tested the functionality of each use case by navigating through the specific pages and monitoring the results. For example for the registration we followed the steps below:

1) Open application main page
2) Navigate through the registration process
3) Complete registration

   Test Criteria

4) Verify page navigation
5) Verify database storage

We intend to extend the test criteria to include more stringent test conditions. For example, we will test the heart rate monitoring under non-optimal conditions. These manual tests will follow the basic steps below:

1) Open application main page
2) Navigate through the appropriate processes process
3) Verify that results meet criteria

13 PROJECT MANAGEMENT

13.1 Merging the Contributions from Individual Team Members

Our report documentation is available in a google drive that to which everyone has access. Each week Tina formats the reports with section headers and outlines so that each group member can edit the document with their contributions. The reports are edited and merged collaboratively using Google

docs. Tina and Tahiya are usually responsible for formatting, style consistency, and modifications to the document appearance.

The report and code distribution are decided collaboratively among the group. One of the challenges that we faced was evenly distributing the report and coding so that everything would be responsible for a relatively equal amount of points. We attempted to divide tasks according to people's strengths while not overwhelming any particular members of the group. We addressed this challenge of point distribution by reviewing the points weekly and assigning sections accordingly.

As far as code distribution we delegated tasks according to each team member's strength. Ramya and Himabindu have experience with socket programming; so they are responsible for code integration. George and Aymen have strong programming skills; therefore they were responsible for various code sections including sensor detection and step counting. Additionally, they have completed some of the project management tasks such as maintaining the Github site. Tahiya is specializes in user interface; so, she has been assigned with the responsibility of the UI pages. Moreover, she has served as the project leader and completed various project management tasks such as coordinating meetings. Tina does not have a strong programming background but has experience in program management. Thus, she has been involved in a lot in project management activities including report formatting and google drive management. She has also been responsible for all of the web development and coding.

13.2 Project Coordination Activities

Basic information

Team Leader: Tahiya

Primary Communication Method: WhatsApp

Meeting Time: Wednesdays, 11:30 am to 12:00 am

Meeting Location: CoRE 100

Document Repository:

https://drive.google.com/drive/folders/0B0NuBl5TDP7_N2RaRk9BaUEyOTA?usp=sharing

Project Repository: https://github.com/karahbit/RU_Healthy

Website: https://ruhealthy.github.io/ruhealthy/

Activities

As a group, much of our time has been devoted to completing documentation required for the class assignments. Additionally, we have worked on the project management portion by mapping out task assignments. We have all contributed to these sections. We meet weekly in person or by Whatsapp to delegate tasks and share information about our current coding status. The bulleted list outlines the non-coding tasks that we have completed:

- Proposal - **All**
- Report 1, Part 1 - **All**
- Report 1, Part 2 - **All**
- Report 1, Full report - **All**
- Report 2, Part 1 - **All**
- Report 2, Part 2 - **All**
- Demo Documents - All
- Coordination of team meetings - **Tahiya**
- Delegation of report sections - **All**
- Delegation of coding assignments - **All**
- Delegation of report updates - **All**
- Delegation of demo assignments - **All**
- Creation of Github Folder - **Aymen, George**
- Organizing Github Folder - **Aymen, George**
- Uploading Documents to Github Server - **All**
- Creation of Github site for webpage - **Tina**
- Management Github website repository - **Tina**
- Gantt chart - **Tina**
- Creation of firebase account for website database management - **Tina**

13.3 Breakdown of Coding Responsibilities

Each group member has contributed to coding, developing, and testing some portion of the project. The delegations of these modules and pages are listed below:

Android Modules

MainActivity.java - **Ramya/Himabindu**
RegisterActivity.java - **Ramya/Himabindu**
ProfileActivity.java - **Ramya/Himabindu**
DashboardActivity.java - **Ramya/Himabindu**
SQLiteHelper.java - **Ramya/Himabindu**
StepCounter.java - **Aymen/George**
StepDetector.java - **Aymen/George**
StepListener.java - **Aymen/George**

SensorFilter.java - **Aymen/George**

ToDo.java - **Aymen/George**

ToDoMain.java - **Aymen/George**
HeartRateMonitor.java - **Ramya/Himabindu**

rr.java - **Ramya/Himabindu**

BayesiaPredictor.java - **Ramya/Himabindu**

Android UI pages

Activity_dashboard.xml – **Tahiya/Aymen**

Activity_main.xml - **Tahiya/Aymen**

Activity_register.xml - **Tahiya/Aymen**

Hrm.xml - **Tahiya/Aymen**

Profile.xml - **Tahiya/Aymen**

Step_count.xml - **Tahiya/Aymen**

Website Code

Home.html - **Tina**

index.html - **Tina**

Login.html - **Tina**

Registration.html - **Tina**

ProfileUpdate.html - **Tina**

PatientInformation.html - **Tina**

FirebaseAddUser.js – **Tina**

FirebaseAuthentication.js – **Tina**

FirebaseOperationsUser.js – **Tina**

FirebaseUpdateProfile.js – **Tina**

PatientSearcher.js – **Tina**

PatientActvity.js – **Tina**

Each member has contributed in some aspect to the code integration. Ramya and Himabindu have been be responsible for socket programming and integrating different functional sections of the android code. Tahiya will be responsible for integrating the user interface with the androd pages. Tina will be responsible for integration the website with the android database. George and Aymen will be integrating the test code with the Android system. Additionally each group member will contribute to the integration testing.

14 HISTORY OF WORK

14.1 Current Status

Currently we have an aesthetically pleasing android application that allows user to login/register for an account, count step activity, detec heart rate, share health information with physicians automatically, schedule workout, and store activity history via an online database. Furthermore we have functioning web application that allows physician access to their patients activity information. Through the web application the physician can login/registers, view a list of all their patients with the RU Healthy web app.

We started by implementing the functionality of the application that are the central part of the project:

- the collecting information from the phone sensors,
- the ability to monitor health data
- granting user access to this health data.

Thus, the initial functionality of the app included getting the heart rate and step information from the phone. This was done in parallel with the user signin and registration information. Following this we implemented the user database capability so that the data could be access by the patient and physician.

We experienced several challenges while working with the database. One of the most prominent issues determining the correct database server to use. Initially for the android app we were storing the information locally on the phone through a sql database. This could not be a permanent solution because it limited physician access and used android memory resources. As for the web application we started by storing the data via php through the XAMPP servers. After several permutations of database servers including, mySQL, mongodatabase, freemysqlhosting.net, we decided to go with firebase. It took some time to become familiar with firebase JSON format. However, we were able to successfully integrate it with both the RU Healthy? android app and website.

We also faced some integration issues and technical difficulties. While attempting to integrate some of the android code, some of of team members experienced technical issues. Also because we were working with different operation systems windows and mac, we experienced some compiling issues. We were able to resolve some of these difficulties by reinstalling the android studio software and/or restarting the computers.

Database integration and completing other milestones, such as the reports and Demo, took more time than anticipated. Thus, some of project milestone dates were extended. The table below summarizes the tasks along with changes in projected dates between reports 1, 2, and 3. (Please note that the dates that changed from one report to another are highlighted in red.)

Projected Dates	Report 1	Report 2	Report 3
Class Assignments			
Proposal	9/17/2017	9/17/2017	9/17/2017
Report 1 - Part 1	9/24/2017	9/24/2017	9/24/2017
Report 1 - Part 2	10/1/2017	10/1/2017	10/1/2017
Report 1 - Full Report	10/8/2017	10/8/2017	10/8/2017
Report 2 - Part 1	10/15/2017	10/29/2017	10/29/2017
Report 2 - Part 2	10/22/2017	11/5/2017	11/5/2017
Report 2 - Full Report	10/29/2017	11/12/2017	11/12/2017
Demo 1	11/1/2017	11/1/2017	11/1/2017
Report 3	12/1/2017	12/10/2017	12/10/2017
Demo 2	12/13/2017	12/13/2017	12/13/2017
Electronic Project Archive	12/18/2017	12/18/2017	12/18/2017
Android App			
Research Similar Apps	9/27/2017	9/27/2017	9/27/2017
Create base structure for app in android studio	10/3/2017	10/3/2017	10/3/2017
Create Mobile app with basic features (Login, registration, etc.)	10/16/2017	10/16/2017	10/16/2017
Create database to store patient info	10/25/2017	10/25/2017	10/25/2017
Add user interface for customer registration	10/25/2017	10/25/2017	10/25/2017
Add profile pages	10/24/2017	10/24/2017	10/24/2017
Add user interface for profile pages	10/31/2017	10/31/2017	10/31/2017
Create functionality to detect motion from phone sensors	10/19/2017	10/19/2017	10/19/2017
Create functionality detect heart rate from phone sensors	10/25/2017	10/25/2017	10/25/2017
Create algorithms to process and store heart rate data	11/7/2017	11/7/2017	12/10/2017
Create algorithms to process and store motion rate data	11/7/2017	11/7/2017	12/10/2017
Add functionality for exercise scheduling	11/15/2017	11/15/2017	12/5/2017
Add functionaility for notifications	11/23/2017	11/23/2017	
Web Application			
Reseach similar applications	10/10/2017	10/10/2017	10/10/2017
Create base structure	10/18/2017	10/18/2017	10/18/2017
Add User Login Page		10/27/2017	10/27/2017
Add Password Credentials for Login Page		10/31/2017	10/31/2017
Add user registration page		10/27/2017	10/27/2017
Add user interface for registration	10/27/2017	10/31/2017	10/31/2017
Link Login and Registration to Database		11/10/2017	11/10/2017
Add profile view page		11/10/2017	11/10/2017
Add data view capability	11/15/2017	11/15/2017	12/10/2017
Link patient information to data on Android server		11/10/2017	11/10/2017
Upload Eclipse Project to GIT server		10/27/2017	10/27/2017
Socket Programming			
Combine code sections for customer registration and motion sensors	10/31/2017	10/31/2017	12/9/2015
Incorporate heart rate	11/7/2017	11/7/2017	11/7/2017
Link web app to android app	11/14/2017	11/14/2017	12/10/2017
Incorporate other code sections	11/23/2017	11/23/2017	12/10/2017
Quality Testing			
Test phone app	11/27/2017	11/27/2017	12/12/2017
Test web app	11/27/2017	11/27/2017	12/12/2017
Make adjustments to the applications	12/1/2017	12/1/2017	12/12/2017
Retest phone app	11/30/2017	11/30/2017	12/12/2017
Retest web app	11/30/2017	11/30/2017	12/17/2017

14.1.1KEY ACCOMPLISHMENTS

Android App

- Created an integrated android application
- Added registration and login capability
- Added step counting capability
- Added heart rate detection
- Added exercise scheduling
- Linked customer data to firebase database

Web App

- Created an web application
- Added registration and login capability
- Incorporated access to patient health information via firebase database.
- Added capability to summarize output of patient health information

Project Management

- Completed report 1
- Completed report 2
- Completed report 3
- Completed demo 1
- Completed demo 2

14.2 Future Work

Due to time constraints, we are not able to implement all of the use cases. As a result, we decided to exclude some of the optional use cases. We believe that these could be implemented for future work. These use cases are listed below:

UC-5: Enable Notifications

UC-6: Get Notes

UC-9: Enable Weekly Updates

UC-10: Send Notes

Furthermore, we believe that additional security can be added to the physician sign up. We have enforced some security by adding a search to patient information part. When the physician accesses the patient information page, they select get patient information button in order to retrieve a patient list. The patient search class ensure that only patients that have selected this doctor as their physician appears in the list. Additionally, when the doctor selects "Get Patient Information" or the "Get Patient Activity" button there is a credential check when ensures that the data is only returned if this physician's ID is stored in their database. If the physician ID is not included in the patient information an alert is shown informing the physician that they patient has not selected as their physician. However, there should be additional security to ensure that they users do not create an account false claim to be physicians. One idea on how to enforce this security is to develop a relationship with hospitals and medical facilities. If the facility agrees to participate with the app, they could be given a special ID or sign-in code. They could then provide the RU healthy admin team with their list of doctors and ID numbers. When the physician registers for the app, they would have to provide, their facility ID code and physician ID code. Upon registration, the app would check in the database and verify that the physician ID and facility ID match.

Additionally, the scope of the project could be broadened to detect and notify people of anomalies. Currently, the physician is responsible evaluating patient information from the database. They can examine the heart rate and activity information the determine if the patient is at risk. This could be expanded by having the system automatically detect anomalies risk factors based on the data retrieved. For example, with the heart rate information, the system could monitor for rapid or slow heart rate activity. This system could then send out notifications to the patient, and possibly physician, indicating the risk.

15 REFERENCES

1) **Google Answers, Q: Calories burned while exercising**
http://answers.google.com/answers/threadview?id=758572%22

2) **Exercise: 7 benefits of regular physical activity**
http://www.mayoclinic.org/healthy-lifestyle/fitness/in-depth/exercise/art-20048389

3) **CDC: 80 percent of American adults don't get recommended exercise**

https://www.cbsnews.com/news/cdc-80-percent-of-american-adults-dont-get-recommended-exercise/

4) **Concepts: Requirements, UPEDU**

http://www.upedu.org/process/gcncpt/co_req.htm

5) **FURPS**

https://en.wikipedia.org/wiki/FURPS

6) **Merriam Webster Dictionary: Obesity**

https://www.merriam-webster.com/dictionary/obesity

7) **25 Best Fitness Apps by John Corpuz**

https://www.tomsguide.com/us/pictures-story/482-best-fitness-apps.html#s10

8) **How much Physical Activity Do Adults Need? Center for Disease Control and Prevention**

https://www.cdc.gov/physicalactivity/basics/adults/index.htm

9) **Sullivan, Alycia N. and Lachman, Margie E. (2017), Kostkova, Patty, ed., "Behavior Change with Fitness Technology in Sedentary Adults: A Review of the Evidence for Increasing Physical Activity", Front Public Health 4: 289**

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5225122/

10) **National Center for Health Statistics. Health, United States, 2016: With Chartbook on Long-termTrends in Health. Hyattsville, MD. 2017.**

http://myphonefactor.in/2012/04/sensors-used-in-a-smartphone/

11) **Health App Android iOS**

https://healthyceleb.com/wp-content/uploads/2012/10/Health-app-android-iOS.jpeg

12) Health App Android iOS Icons
https://www.google.com/search?biw=1240&bih=572&tbm=isch&sa=1&q=health+apps&oq=he&gs_l=psy-ab.3.0.0i67k1l4.73873.76835.0.78238.16.13.1.0.0.0.264.1860.0j5j4.10.0....0...1.1.64.psy-ab..10.5.874.0..0.147.QsrScBdLFfs#imgrc=C6IWjX4kIqVArM:

13) Project Management Guide: The Ultimate Introduction to Project Management Fundamentals

https://www.wrike.com/project-management-guide/faq/what-is-a-stakeholder-in-project-management/

14) Ivan Marsic, (2012). Written at Rutgers University, New Brunswick, New Jersey, Software Engineering

http://www.ece.rutgers.edu/~marsic/books/SE/

15) Vineet Singh Pangtey, Testing Estimation with use case points: An Approach for Testing Size and Effort Estimation
https://www.cmcrossroads.com/sites/default/files/article/file/2013/XUS234033087file1_0.pdf

16) Wikipedia:System Sequence Diagram

https://en.wikipedia.org/wiki/System_sequence_diagram

17) Randy Glasbergen Cartoons

http://photos-ak.sparkpeople.com/7/0/b707669348.jpg

18) Dr Micheal Eichberg, System Sequence Diagram: Introduction to Software Engineering

http://stg-tud.github.io/eise/WS14-EiSE-12-System_Sequence_Diagrams.pd

19) C. Hall et al., (2004), "Energy Expenditure of Walking and Running", Medicine & Science in Sports & Exercise.

https://www.ncbi.nlm.nih.gov/pubmed/15570150

20) Design Principles, Object Oriented Design

http://www.oodesign.com/design-principles.html

21) UML Tutorials

www.uml.org

22) Firebase Object Descriptions

https://firebase.google.com/docs/reference/android/com/google/firebase/auth/FirebaseAuth

23) Heart Rate Monitor Source code

https://github.com/phishman3579/android-heart-rate-monitor

APPENDIX A

This Appendix contains the traceability matrix for the class diagrams.

Class Diagram Traceability Matrix

	Activity Notifier	Camera Operator	Controller	Data Container	Database Authenticator	Database Manager	Database Operator	Database Postprocessor	Interface	Notifications Enabler	Patient Operator	Pagemaker	Patient Comparator	Patient Notifier	Patient Searcher	Prediction Function	Search Manager	Sensor Operator	Tracking Enabler
ANDROID APP																			
BayesianPrediction						X	X	X	X							X			
DashboardActivity			X	X	X	X	X												
ToDo										X									
HeartRateMonitor		X			X	X	X											X	
StepCounter																		X	X
StepDectector																		X	
MainActivity			X		X														
RegisterActivity					X	X	X	X					X						
ProfileActivity					X	X	X	X	X										
RecoveryRate					X	X	X	X	X							X			
WEB APP																			
index.html			X																
Login.html					X	X	X												
Register.html					X	X	X												
ProfileUpdate.html					X	X	X												
PatientInformation.html					X	X	X	X	X		X	X		X			X		
Home.html			X																
FirebaseAuthentication.js					X	X	X												
FirebaseUpdateProfil.js					X	X	X												
FirebaseAddUser.js					X	X	X												
PatientSearcher.js					X	X	X	X			X	X	X		X		X		
PatientActivity.js					X	X	X	X			X	X	X		X		X		

The Activity Notifier Domain Concept was not implemented