SOFTWARE ENGINEERING 1
REPORT 2

---

napIT

November 12, 2017

Madhura Daptardar
John Grun
Preetraj Gujral
Manasi Mehta
Siddhi Patil
Aishwarya Srikanth

URL: https://github.com/napIT-Rutgers-SE-Fall-2017

CONTENTS

INDIVIDUAL CONTRIBUTIONS

Report topics	Madhura Daptardar	John Grun	Preetraj Gujral	Manasi Mehta	Siddhi Patil	Aishwarya Srikanth
Interaction Diagrams						
	16.67%	16.67%	16.67%	16.67%	16.67%	16.67%
Class Diagram and Interface specifications						
Class Diagrams					50%	50%
Data type and Operations signatures				100%		
Traceability Matrix			100%			
System Architecture						
Architectural Styles	100%					
Identifying Subsystem	100%					
Mapping subsystem to hardware	100%					
Persistent Data Storage	100%					
Network Protocol	100%					

Global Control Flow	100%					
Hardware Requirements	100%					
Algorithm and Data Structures						
Algorithms					100%	
Data Structures	100%					
User Interface Design						
			33%	33%		33%
Design of Tests						
		100%				
Project Management						
Product Ownership		100%				
Gantt Chart						100%

1. INTERACTION DIAGRAMS

UC 1: Login

The interaction diagram for Use Case 1: Login is shown in the figure above. This use case allows a user with an existing account to log into the system and access functions of the app and the account data. The user initiates an action with the login interface by pressing the login button. The user's data is passed to the controller with a call to verify login. The controller then passes a query request to the database manager. The database manager looks in the database for a for a record of the login. The database may return a record from the database or it may return no results. The database manager will forward the database query results to the controller. The controller will compare the login from the user to the record retrieved from the database. If the records do not match in username and password. The controller will return verify login with a fail to the login interface. The login interface will inform the user that their login attempt was not valid. If the user has already exceeded the max number of login attempts then the user must wait for a 2 min timeout to expire before attempting to log in again.If the user has not exceeded the max allowed number of logins they will be allowed to attempt to log in. If on the other hand the user has entered matching login credentials then the login interface informs the user that their login was successful. The controller will then request GUI Generator to display the next screen to the user.

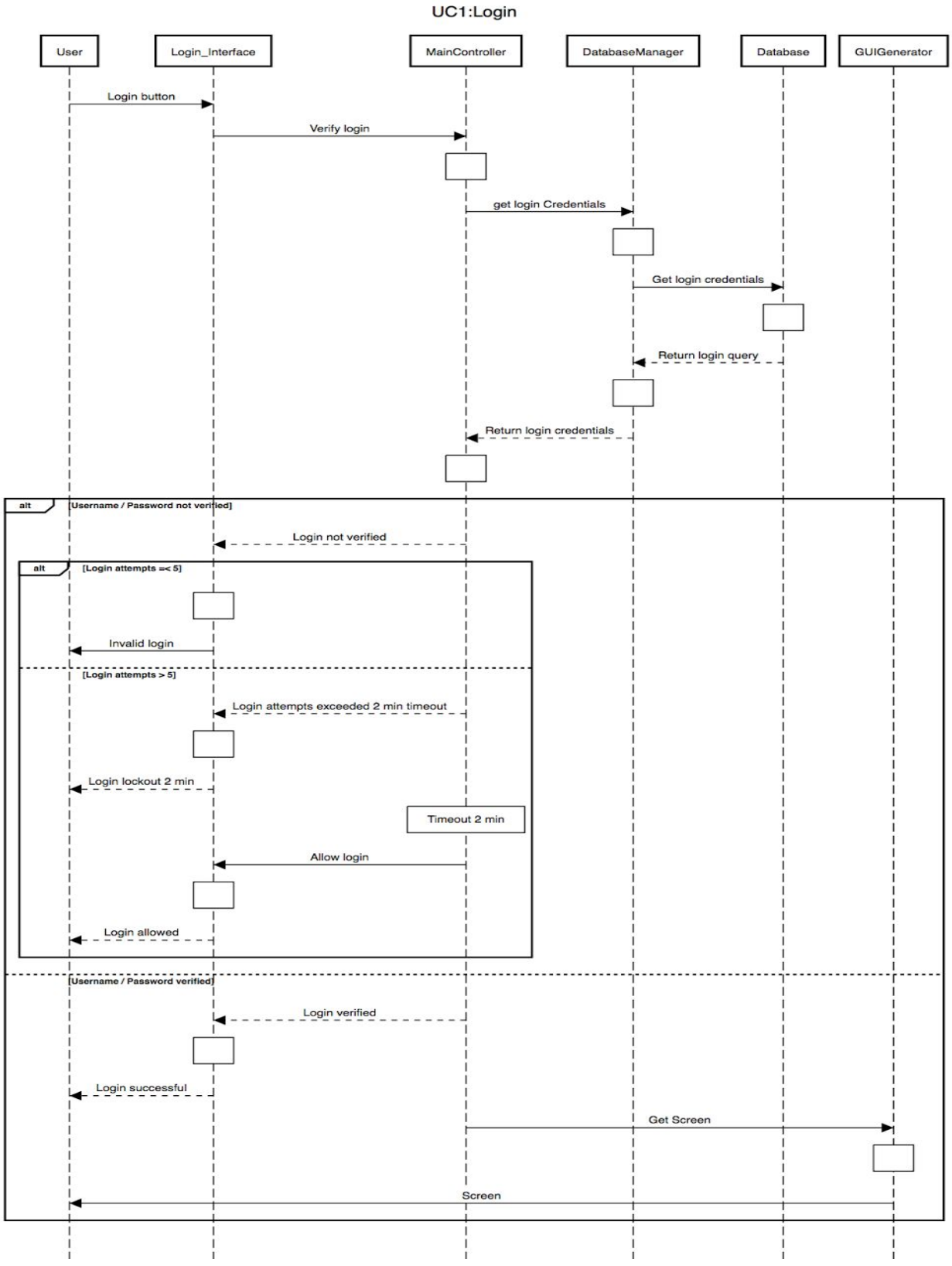

Figure 1: Interaction Diagram for Login

UC 2: Registration

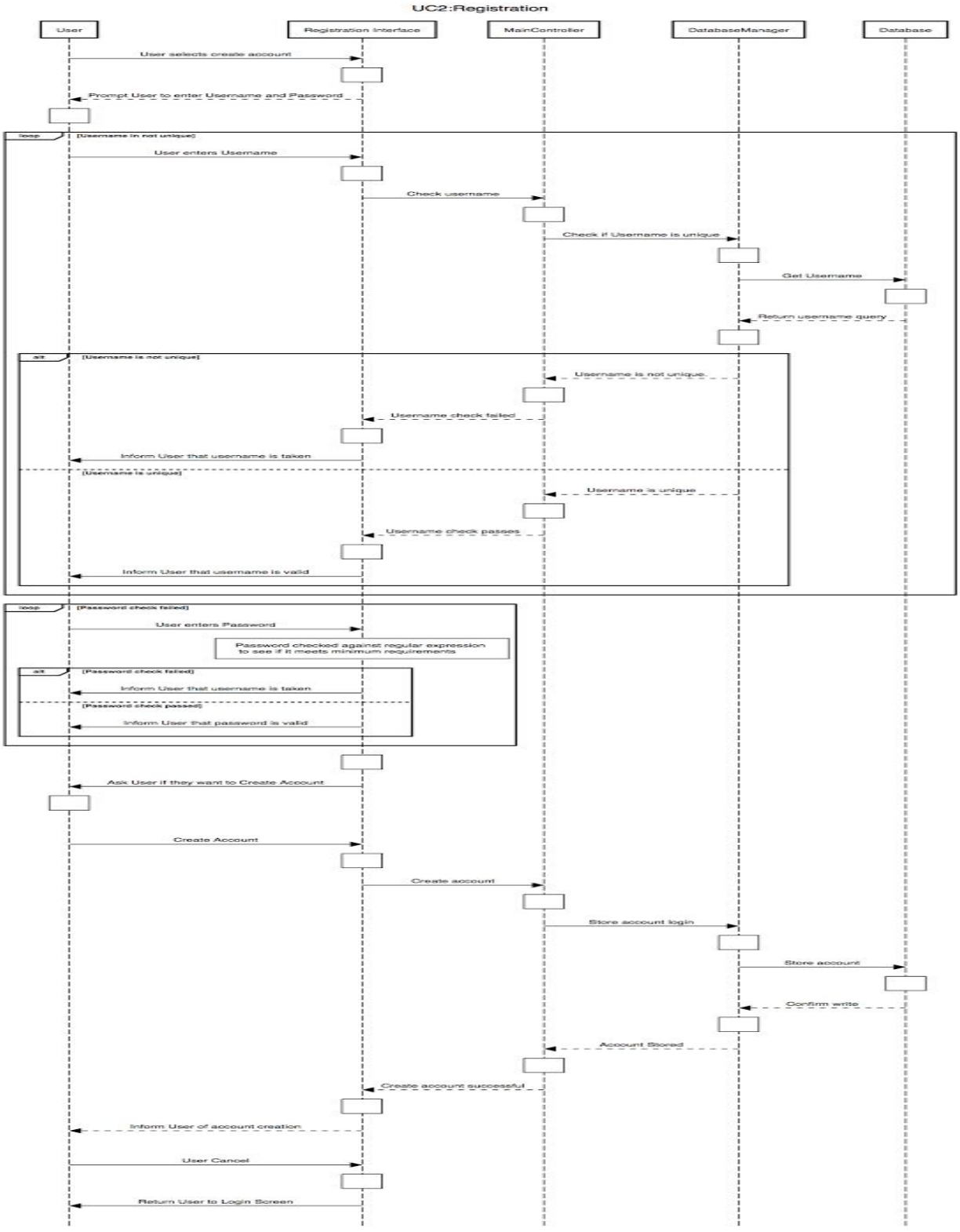

Figure 2: Interaction Diagram for Registration

The interaction diagram for Use Case 2: Registration is shown in the figure above. The user initiates the use case by selecting the create account button. The Registration interface then prompts the user for a new username and password. Once the user enters a username, The registration interface will ask the controller to check for existing usernames. The controller will then ask the database manager to query the database for the username. The database returns the record to the database manager. The database manager returns the query results to the controller. If an existing record was found then the controller tells the registration interface that the username exists. The registration interface will inform the user. This will repeat until the user selects a unique username. Otherwise, if the username was unique the user is informed that the username is unique and will be allowed to continue. The user will then enter a password. The password will be passed to the controller. The controller will test the password against a regular expression to ensure the password meets the minimum security requirements. If the password does not test then the user will be asked to enter a new password. If the password does pass the test then user is informed. The Registration interface will then prompt the user to create account. If the user elects to create an account then the account details are passed to the controller in a call to create account. The controller will ask database manager to store a new account in the database. The database manager writes the account into the database. The database returns a confirmation to the database manager which then informs the controller of the success. The controller informs registration interface. The registration then informs the user of account creation success. If however the user decides not to create an account then no further action will occur.

UC 3: Manage Accounts

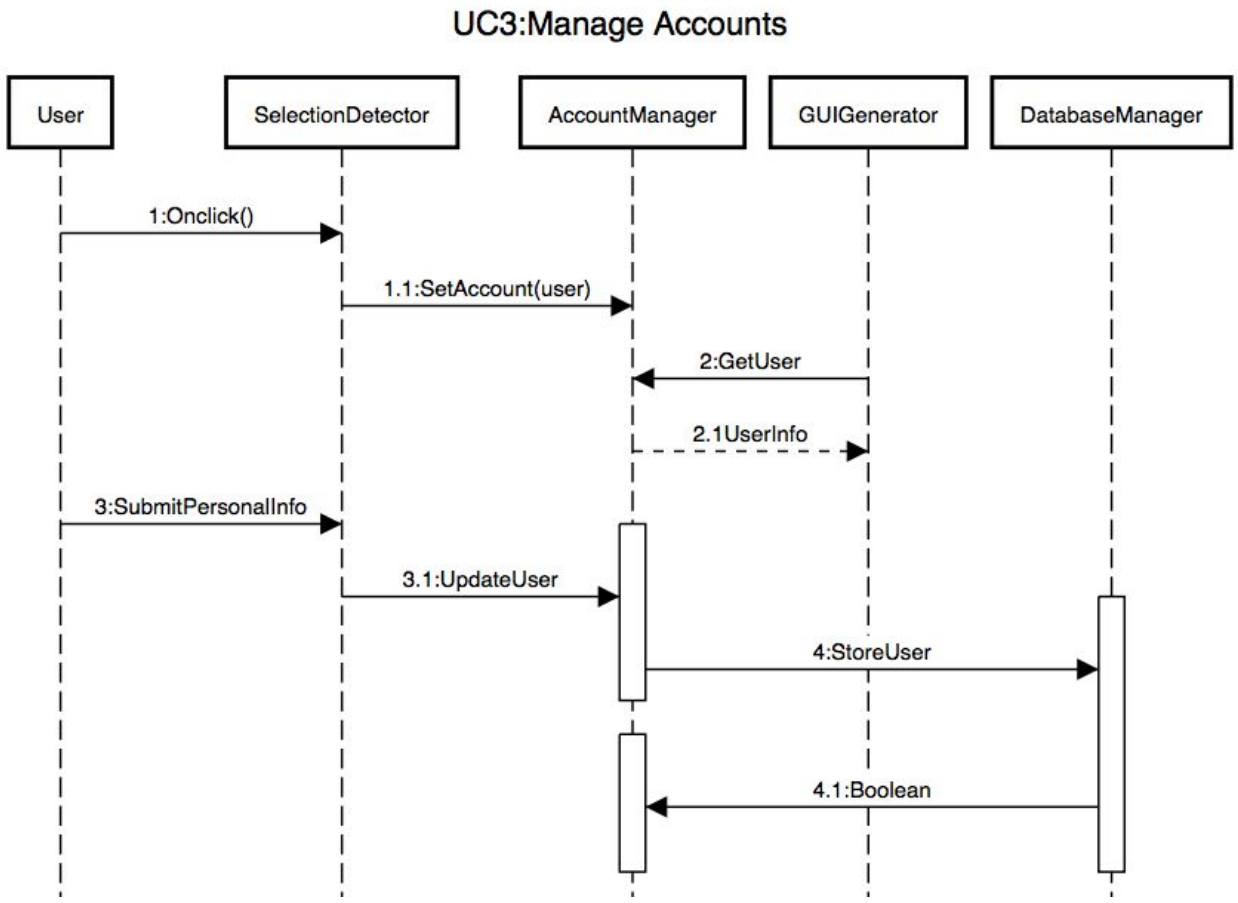

Figure 3: Interaction Diagram for Manage Accounts

The interaction diagram for Use Case 3: Manage Accounts is shown in the figure.The user is the one who initiates the action, by choosing an option out of the ones given on the GUI using the function SelectionDetector, the SelectionDetector then moves to the AccountManager function, where the AccountManager first gets the user's detials needed to sign in and then shows the user's detials on the GUI, which is done by the function GUIGenerator. Now it comes to the user again to chose to submit his/her personal information. The user does by using again, the SelectionDetector on the GUI. This information is then submitted directly to the AccountManager, where the function sends the details to the DatabasManager, where the details that have been entered by the user for their account, gets stored finally.

UC 4: Monitor Sleep

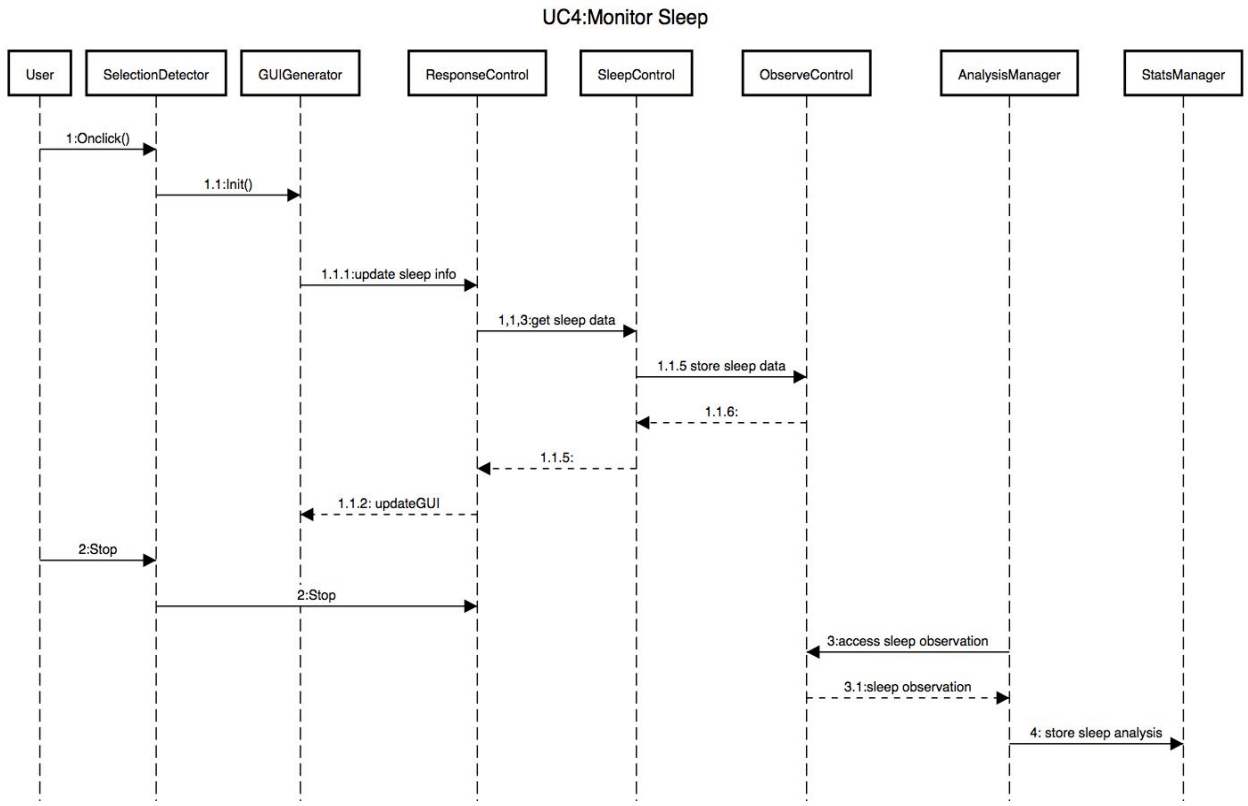

Figure 4: Interaction Diagram for Monitor Sleep

The use case diagram for Use Case 4:Monitor Sleep is shown in the figure.In this use case, the user initiates the function by choosing the option given by the SelectionDetector. If the user selects to monitor sleep, the sleep monitoring starts, and the GUI is updated accordingly via the GUIGenerator. The ResponseControl acts to the input and starts the recording part. It informs the SleepCoontrol function to record the user's sleep, which is done by activating sensors for the operation. The SleepControl records the sleep and updates the ObserveControl function, after which, the GUI is updated, as the process of monitoring keeps going on. This is done until the point the user chooses to stop by choosing the choice given in the GUI by the SelectionDetector. Accordingly, an instruction is sent to the ResponseControl, that stops the recording process.

Once it is done, the AnalysisManager accesses the sleep data from the ObserveControl and performs the analysis on the sleep data. This analysis done, is then stored in the StatsManager.

UC 5: Anomaly Detection

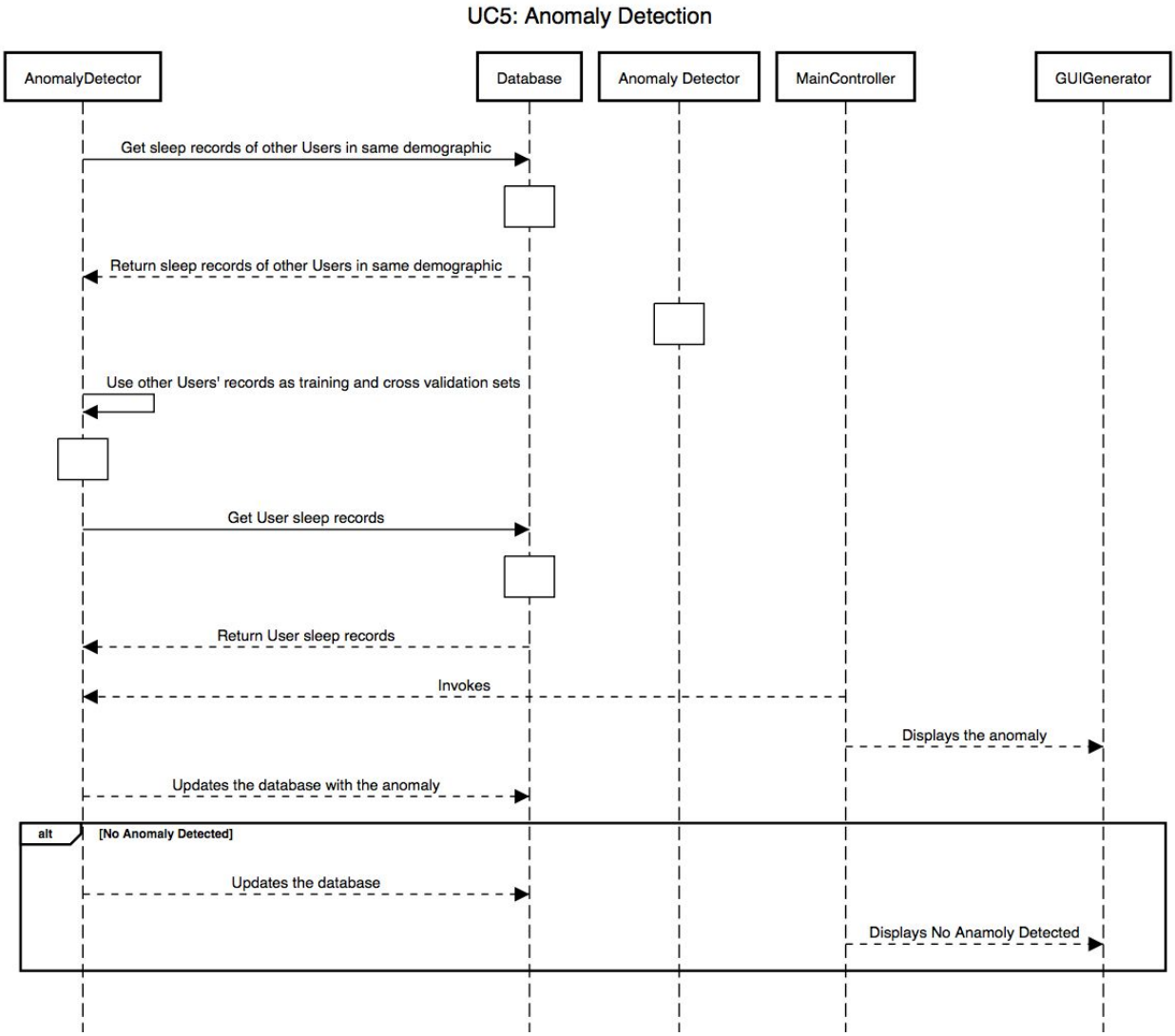

Figure 5: Interaction Diagram for Anomaly Detection

The interaction diagram for Use Case 5: Anomaly Detection is shown in the figure. This use case detects an anomaly based on user's sleeping habits. Initially, the system gets the user's sleep records as well as other user's sleep records in the same demographic. Based on these records,

the AnamolyDetector trains and cross validates the data to detect an anomaly. Once an anomaly is detected, it is displayed to the user using GUIGenerator and updated in the database using DatabaseManager. In an alternate scenario, if AnomalyDetetor detects no anomaly, in a similar manner, the GUIGenerator displays no anomaly detection to the user along with database update using DatabaseManager.

UC 6: View Statistics

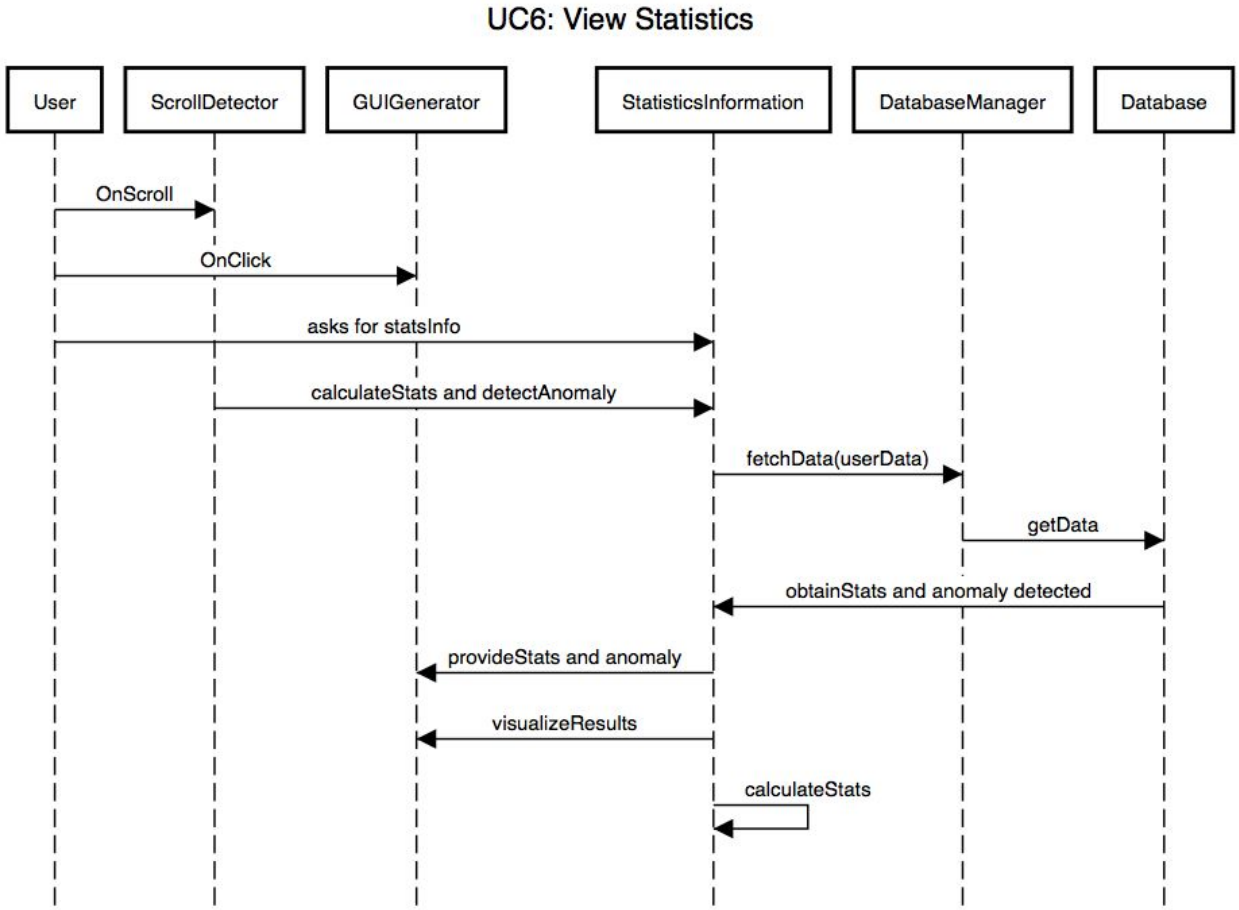

Figure 6: Interaction Diagram for View Statistics

View Statistics is for displaying all the information about the user. After the user clicks on the "View Statistics" button, a page will be displayed showing information from the Sleep, Food and Exercise data. The "StatisticsInformation" will calculate the calories intake for the food, the

calories burnt after Exercise and display it. The data which is used for calculation is fetched from the "Database". Sleep data: Hours Slept, Anomaly and Suggestions is also displayed. The anomaly detected is obtained from the Database and displayed to the user along with the suggestion. The figure shows the Interaction Diagram for Response to the user when he/she clicks the "View Statistics" button displayed on the GUI.

UC 7: Recommender

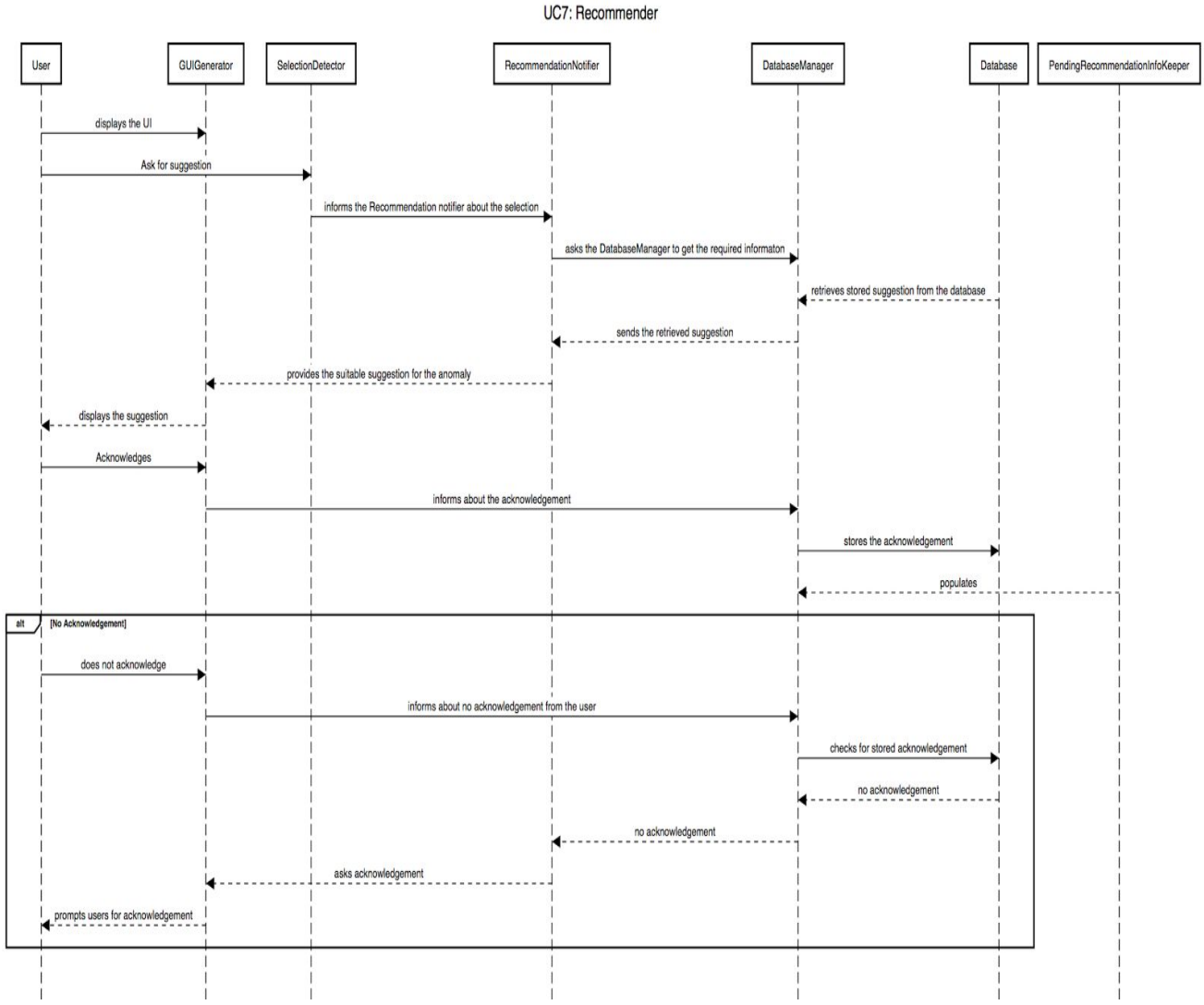

Figure 7: Interaction Diagram for Recommender

The interaction diagram for Use Case 7: Recommender is shown in the figure above. This use case provides the user with certain suggestions pertaining to the detected anomaly. The User asks for the suggestion. The SelectionDetector informs the RecommendationNotifier about the selection. The suggestion for respective anomalies are previously stored in the database. The DatabaseManager retrieves the suggestion from the Database, and provides it to the GUIGenerator, which then displays it on the screen to the User. The User needs to acknowledge that the has read the suggestion, so that the suggestion does not go unnoticed. If the user acknowledges the suggestion, the GUIGenerator informs the DatabaseManager that the user has acknowledged and read the suggestion, and stores it in the database. In the case that the user does not acknowledge the suggestion, the RecommendationNotifier continuously prompts the user to acknowledge.

UC 8: Smart Alarm

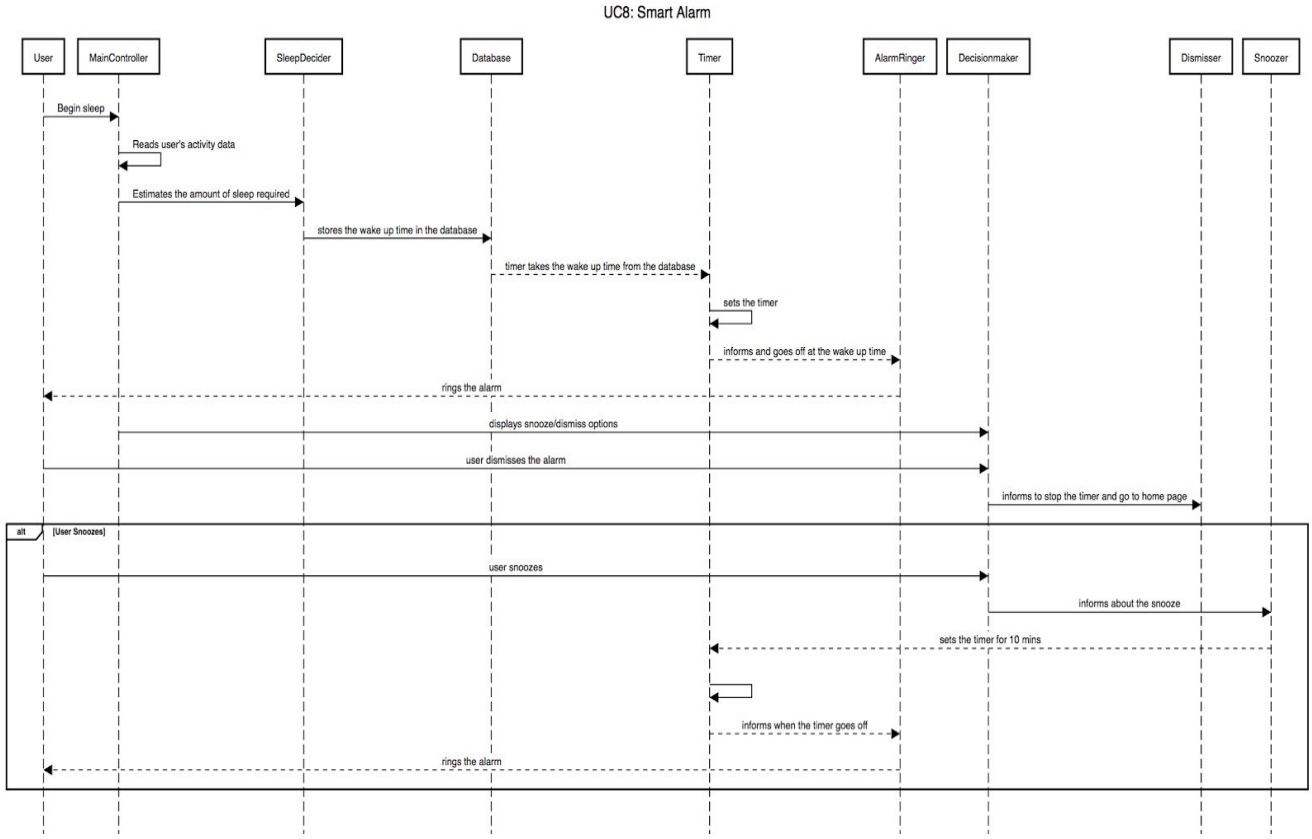

Figure 8: Interaction Diagram for Smart Alarm

The interaction diagram for Use Case 8: Smart Alarm is shown in the figure above. This use case provides the user with a smart alarm, which would calculate the best amount of a sleep the user requires, based on his diet and daily activity. When the user begins sleep, the system estimates the amount of sleep, and provides it to the SleepDecider, which in return stores the wake up in a database. The Timer takes the wake up time from the database, and starts a countdown timer. When the timer reaches 0, it informs the AlarmRinger and the alarm goes off. The user can choose to either Snooze or Dismiss the alarm with the help of the DecisionMaker. If the user dismisses the alarm, the user is redirected to the sleep home page. And if the user snoozes the alarm, the DecisionMaker informs the Snoozer about the snooze, which sets the Timer for 10 mins. Timer goes off after 10 mins and informs the AlarmRinger to ring the alarm, where the user gets an option to either Snooze or Dismiss. This goes on until the user dismisses alarm.

UC 9: Eat

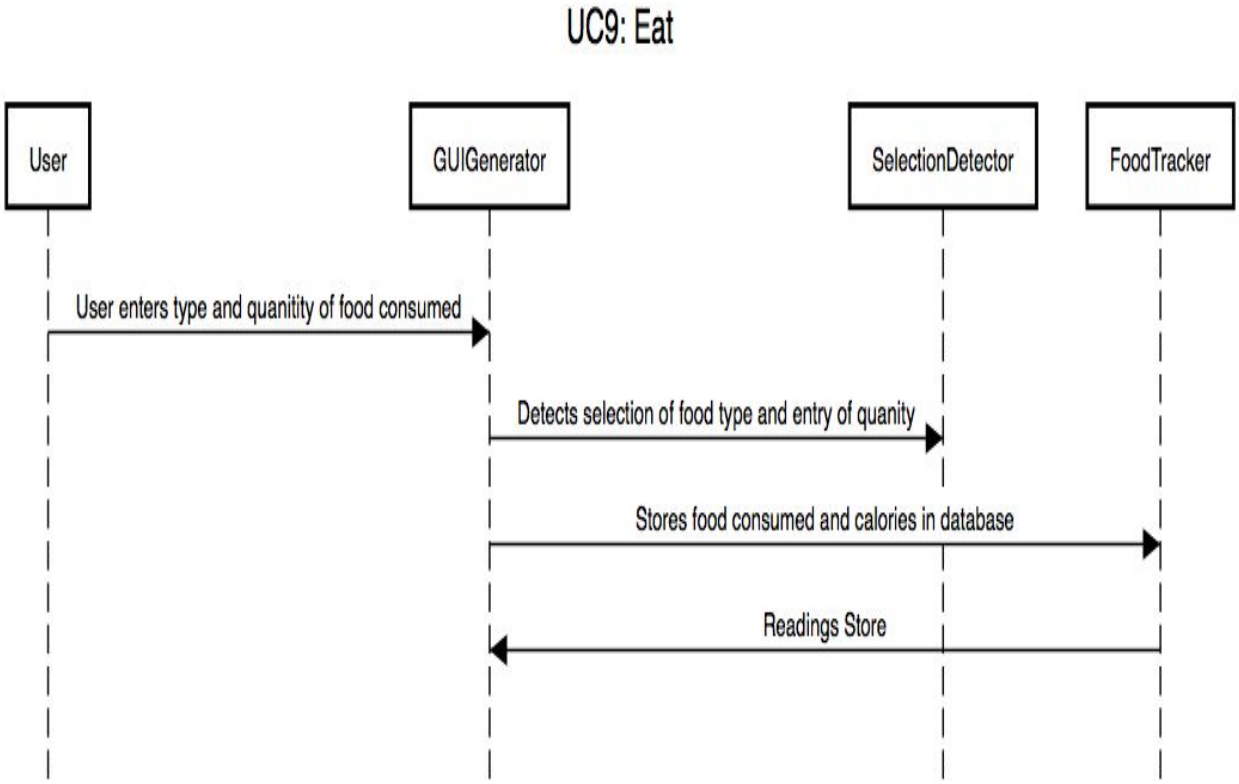

Figure 9: Interaction Diagram for Eat

With respect to the above interaction diagram, the user has to select the food item from the drop down list box in the screen for “Food” (which will appear when the user presses the “Food” button in the “Home” page). Also, the user has to input the quantity of the food consumed in the text box. These details will then be taken as input to our algorithm which will calculate the calories gained and will display that on the screen. All these readings will then be stored in the database through FoodTracker.

UC 10: Exercise

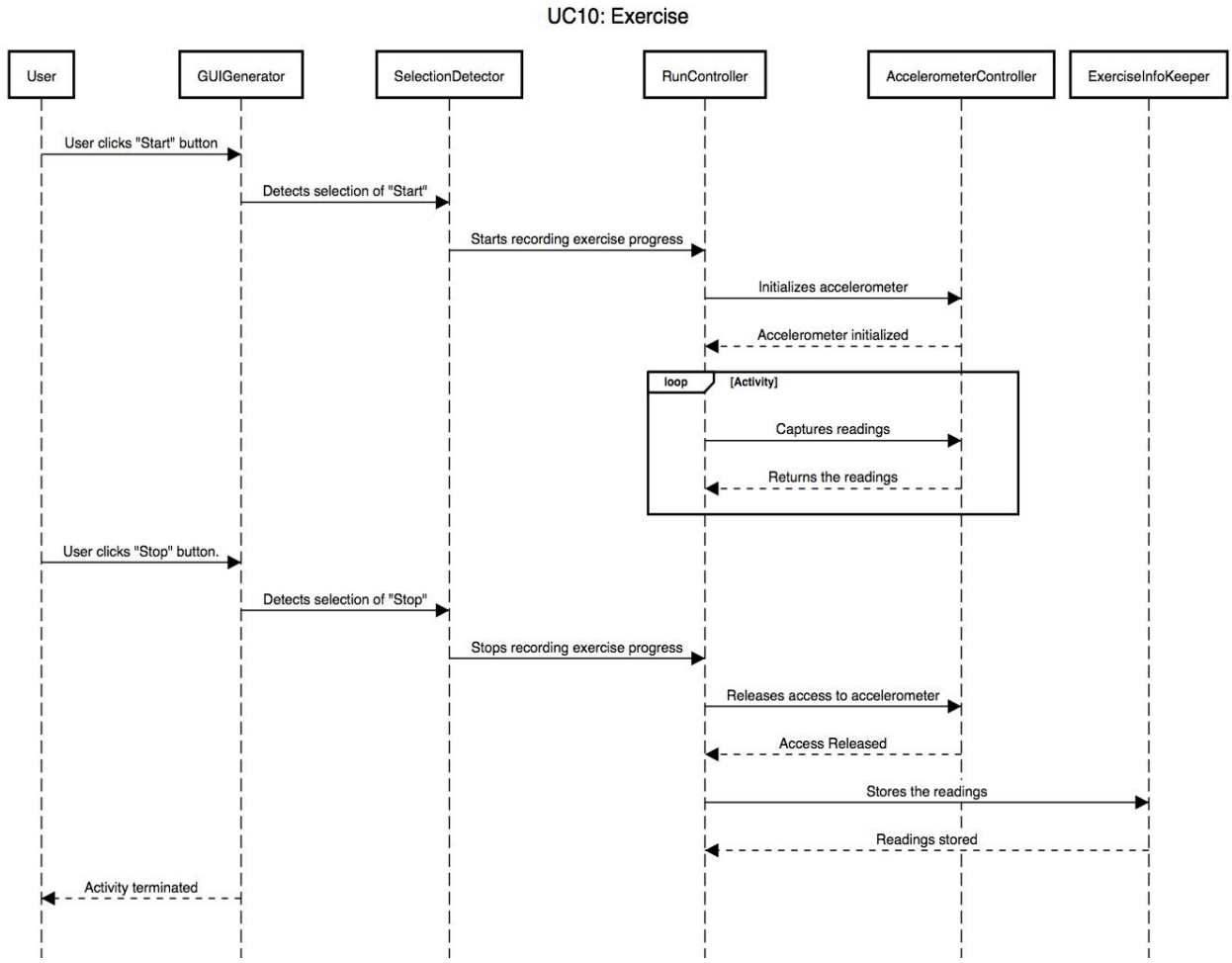

Figure 10: Interaction Diagram for Exercise

According to the interaction diagram above, the user has to first click on the “Start” button in the exercise page (this page will appear if the user clicks on “Exercise” button on the home page). The SelectionDetector automatically senses the touch of the “Start” button and triggers the RunController to automatically record the exercise progress. RunController immediately accesses the accelerometer in the smartphone through the AccelerometerController to measure the miles covered and a timer will be running in the background which will note the time of workout and the approximate calories burnt will be calculated by our algorithm at the end of the workout and will be entered manually by the user. After the user has finished exercising, he/she has to press the “Stop” button in the user interface. Then the SelectionDetector immediately senses the touch of the “Stop” button which in turn makes the RunController to stop recording exercise progress. The RunController will also release access from the accelerometer through the AccelerometerController. The data collected during the workout will then be stored in the database through ExerciseInfoKeeper.

2. CLASS DIAGRAMS AND INTERFACE SPECIFICATION

2.1 Class Diagrams

In Software Engineering, a class diagram in the Unified Modeling Language(UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.[5] The class diagram in figure 11 below, shows the relationship between the major classes in the application. The diagram contains information about the attributes in the class and the functions. It also shows the flow of events between each class. The classes mentioned in the diagram are coded in Java. Diagram list all the functions, attributes in each class and the details like return type.

1. The part of the application that resides on the mobile application uses components of the Android SDK.

- The Android SDK is used to interface the sensor *accelerometer*.The phone accelerometer data is accessed in application through this android SDK.
- Android also provides access to the GUI components on the phone.

2. The SQLite database has been implemented using the Android API to create the table, update them and retrieve the data from them by querying the database appropriately.

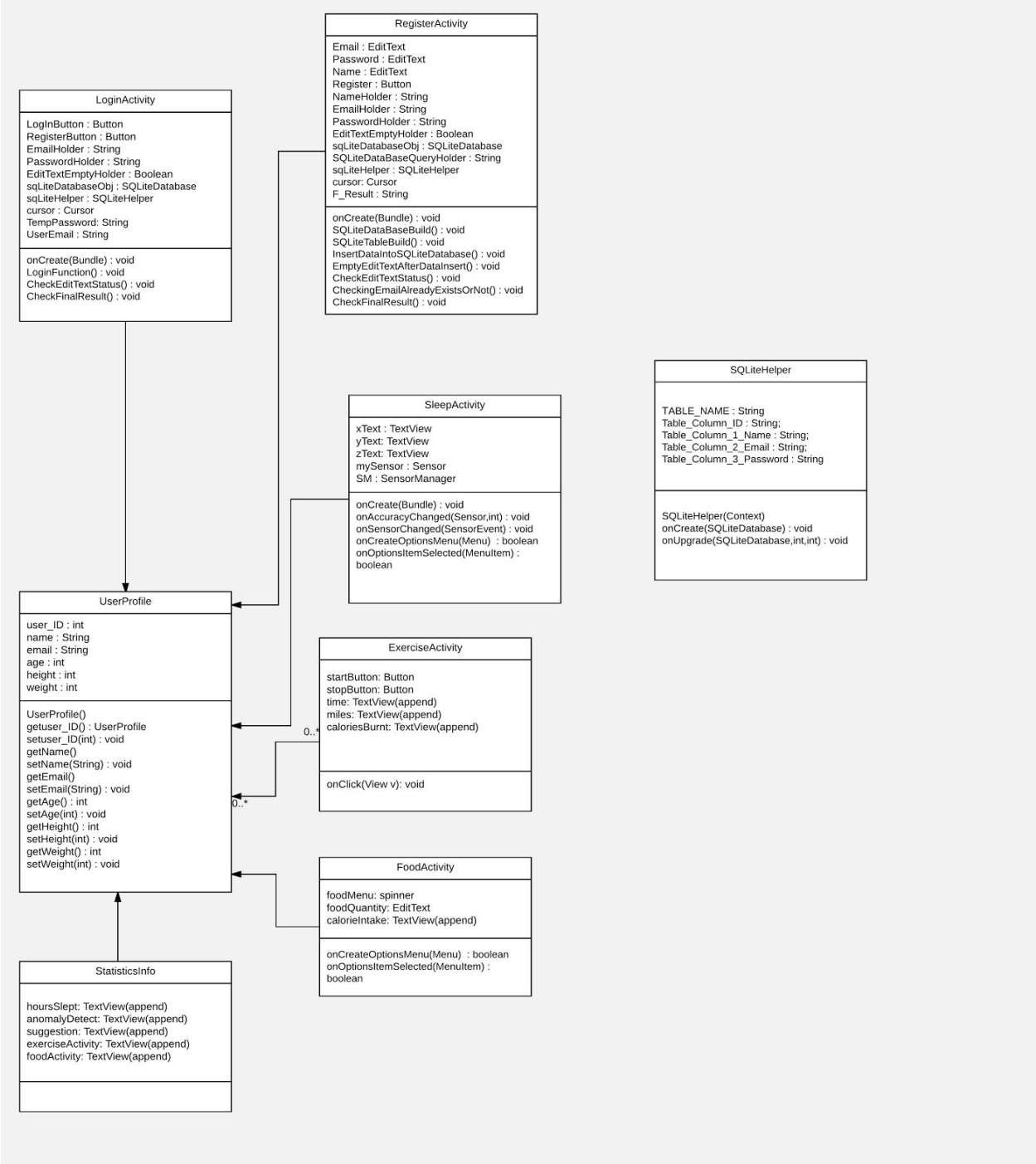

Figure 11 : Class Diagram

2.2 Datatype and Operation signatures

The datatypes, operation signatures with the description has been described below for all the classes.

Variable/Method	Description
LoginButton: Button	When clicked, allow the existing user to login the app
RegisterButton: Button	When clicked, allows a new user to register for the app
EmailHolder: String	Holds the email address of the user
PasswordHolder: String	Holds the password of the user
sqlLiteDatabaseObj: SQLiteDatabase	To access the database
sqLiteHelper: SQLiteHelper	Creates an interface to the database
onCreate(Bundle) : void	This function is used to start the app and set up the XML file
LoginFunction():void	This function verifies the credentials of he user

Table 1: Login Activity

Variable/Method	Description
Email : EditText	Get the email address of the user
Password : EditText	Get the password of the user
Name : EditText	Get the name of the user
Register : Button	When clicked, user is registered in the app.
sqLiteDatabaseObj : SQLiteDatabase	To access the database
SQLiteDataBaseQueryHolder : String	Stores theSQL queries written by the user
sqLiteHelper : SQLiteHelper	Creates an interface to the database
onCreate(Bundle) : void	This function is used to start the app and set up the XML file
InsertDataIntoSQLiteDatabase() : void	Inserts details about the new user in the database

Table 2: Register Activity

Variable/Method	Description
name : String	Name of the user
age : int	Age of the user
email : String	Email of the user
height : int	Height of the user
weight : int	Weight of the user

Table 3 : User Profile

Variable/Method	Description
hoursSlept: TextView(append)	Displays number of hours the user has slept
anomalyDetect: TextView(append)	Displays the anomaly detected
exerciseActivity: TextView(append)	Displays the number of calories burnt by exercising
foodActivity: TextView(append)	Display the number of calories consumed by the user
suggestion: TextView(append)	Gives suggestions to the user

Table 4: StatisticsInfo

Variable/Method	Description
xText : TextView	Readings of the x axis from the accelerometer
yText: TextView	Readings of the y axis from the accelerometer
zText: TextView	Readings of the z axis from the accelerometer
onCreate(Bundle) : void	To access the readings
onSensorChanged(SensorEvent) : void	To get sensor changes in the event of movement during sleep

Table 5: SleepActivity

Variable/Method	Description
startButton: Button	When clicked, it starts taking accelerometer readings
stopButton: Button	When clicked, it stops taking accelerometer readings
time: TextView(append)	Displays the duration of exercise to the user
miles: TextView(append)	Displays miles covered by the user
caloriesBurnt: TextView(append)	Displays calories burnt by the user

Table 6: ExerciseActivity

Variable/Method	Description
foodMenu: spinner	Gives a drop down list of all the food menu items
foodQuantity: EditText	Asks user to enter the food consumed by him
calorieIntake: TextView(append)	Displays number of calories consumed by the user
onOptionsItemSelected(MenuItem) : bool	User selects the food consumed by him

Table 7: FoodActivity

Variable/Method	Description
startButton: Button	When clicked, it starts calculating miles covered by the user while exercising
stopButton: Button	When clicked, it ends calculating miles covered by the user
time: TextView(append)	Displays the duration of the exercise
caloriesBurnt: TextView(append)	Displays the number of calories burnt by the user.
miles: TextView(append)	Displays number of miles covered by the user

Table 8: ExerciseActivity

Variable/Method	Description
TABLE_NAME : String	The name of the database table
Table_Column_1_Name : String	Column for user names in the database
Table_Column_2_Email : String	Column for user's email address in the database
Table_Column_3_Password : String	Column for user's password in the database
onCreate(SQLiteDatabase) : void	To create database tables
onUpgrade(SQLiteDatabase,int,int) : void	To change the database information

Table 9: SQLiteHelper

2.3 Traceability Matrix / Mapping of classes to concepts

- **LoginActivity.java =>** *GUIGenerator, SelectionDetector, DataBaseManager,Disconnector*

  The LoginActivity class is responsible for all the login activity of the user, from logging in to registering a new user. The GUIGenerator takes care of displaying the Interface to the user.The SelectionDetector in this class takes care of selecting the options in the GUI. The DatabaseManager is responsible for storing the data like user account details and email information. Disconnector is used for the user to logout of the application.

- **RegisterActivity.java=>***GUIGenerator, SelectionDetector, AccountManager, DataBase-Manager,UserDetailKeeper*

  The RegisterActivity class is responsible for all the user registration activity. It involves making the user input the details, registering the user in the app, and storing the user details in the database. The GUIGenerator is used for the user interface. The SelectionDetector takes care of providing the user options to chose from, over the interface. The AccountManager is used to pass all information to the UserDetailKeeper and retreiving information from the database. The UserDetailKeeper keeps all the user details and the DatabaseManager stores the data from the UserDetailKeeper to the database.

- **UserProfile.java=>***GUIGenerator, SelectionDetector*

  UserProfile is responsible for handling user's information that are needed in the application like age, height, weight etc, This is done by the GUIGenerator, which takes care of the user interface and the SelectionDetector which takes care of the selections made by the user.

- **StatisticsInfo.java=>***ScrollDetector, GUIGenerator, StatisticsInformation, DataBase-Manager, StatisticsKeeper*

It is used to display all the statistics held in the database to the user for all activities respectively. It makes use of the GUIGenerator, for the interface, DatabaseManager to retreive the stats in the database and populate the StatisticsKeeper, which contains the statistics of user. The ScrollDetector scrolls through all time stats visualization and the StatsInformation gives suggestions based on the user's activities.

- **SleepActivity.java=>***GUIGenerator, SelectionDetector, SleepControl, ObserveControl*
  Used to capture the accelerometer readings while the user is asleep. The SelectionDetector detects if 'Start' has been selected by the user on the interface. The SleepControl takes care of accessing sensors to monitor sleep and release them when the sleep ends. ObserveControl is used to store the observed sleep data.

- **ExerciseActivity.java=>***GUIGenerator, SelectionDetector, RunController, AccelerometerController, ExerciseInfoKeeper*
  The ExerciseActivity class takes care of the exercise activity, involving recording readings as well as displaying details related to exercise. The RunController responds to the selection made by the user through the SelecitonDetector on the interface. The AccelerometerController accesses sensors through the SDK. The ExerciseInfoKeeper is used to hold the details related to the exercise and pass them into the database.

- **FoodActivity.java=>***GUIGenerator, SelectionDetector, FoodTracker*
  The FoodActivity class is responsible for tasks like providing the user with the options from which the user can chose what was consumed. It gives the corresponding calories consumed based on user selection and updates the database. The GUIGenerator and the SelectionDetector are used to provide the user with a dropdown list on the interface and the FoodTracker stores the readings to the database.

- **SQLLiteHelper.java=>***DataBaseManager*

  The SQLLiteHelper class is used to create database tables and update database information. This is taken care of by the DatabaseManager that takes care of all the database in the system.

3. SYSTEM ARCHITECTURE AND SYSTEM DESIGN

3.1 Architectural Styles

The application uses a client-server model. In client-server architecture, one or more clients can communicate with the server, in a request - response pattern. The client sends a request and the server responds back with the requested information. Initially the user logs into the application, by entering his login details (username and password). These entered details are verified with the details stored in database, when the user registered into the application. Once the user is logged into the system, they can request to view statistics, to get the amount of calories consumed or burnt, to start monitoring of their sleep or start the running/ walking activity. Here, the user using the application is the client. The server is the application on the back end which responds to the request by displaying the complete statistics of the user, calculating and displaying the consumed/ burnt calories, activating the sensor and starting the monitoring of sleep or tracking the walk/ run respectively. The user's sleep, activity and food consumption records are also stored in the database. Also, the sleep anomaly is detected by retrieving the sleep data from the database and the system provides corresponding suggestions to the user. Thus, there is a need of constant connection between the user and the backend application, which can be the system, the database or any external API. The client/ server architectural style provides this kind of relationship between the client and the server. In this way, the workflow is divided between various subsystems of the application, which are Accelerometer, Mobile Interface, Server and Database. The major advantage of this type of architecture would be centralization of the data storage, ensuring security (other users cannot view the user's data), better management, scalability and accessibility.

3.2 Identifying Subsystems

The various subsystems of the application are Accelerometer, Mobile Interface, Server and Database.

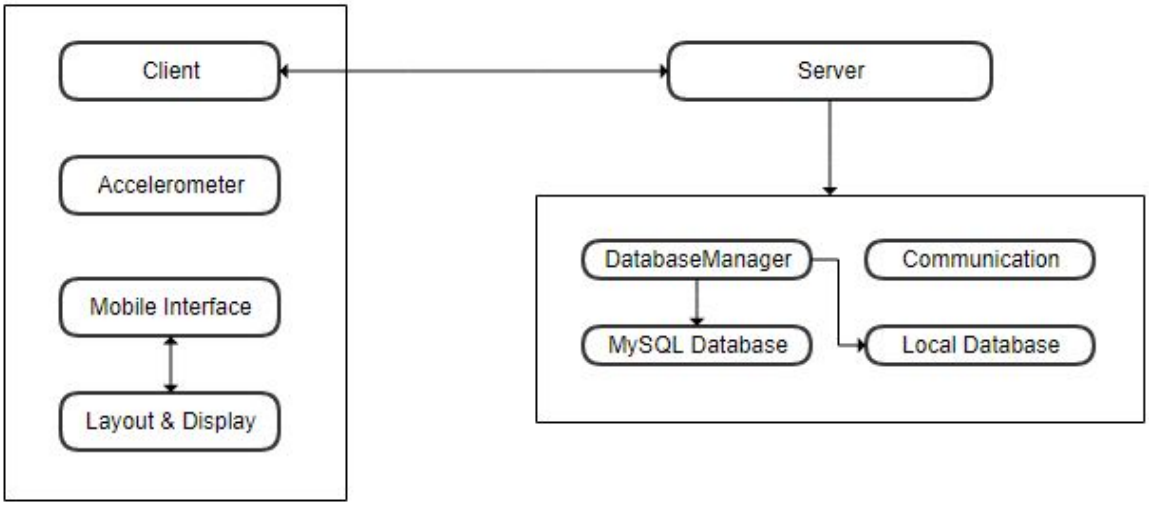

Figure 12: Subsystems of napIT

3.3 Mapping subsystems to hardware

Client: The client is the user's account in the Android/ iOS application. Thus, the hardware used is the Android / iOS Smartphone, on which the application runs.

Server: The server resides on the desktop computer or a laptop of the administrators.

Database: The database used is the MongoDB, for which the hardware used is the local computer or laptop of the administrators.

Accelerometer: The device accelerometer is used as the hardware for calculations requiring accelerometer data.

3.4 Persistent Data Storage

The database we are using is MySQL, but since an Android application does not get access to MySQL, our database is built on SQLite. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. When the user registers into the application, their login details (username and password) are stored into the database. Further when the user uses this detail to log in to the application, they are verified against those stored in the database. Apart from authentication, various components of the application require to store and retrieve data from the database. Thus, the data is separated into certain parts: User, Personal Details, SensorReadings, FoodDetails, ExerciseDetails and SleepDetails. They are stored and are persistent on the Android device.

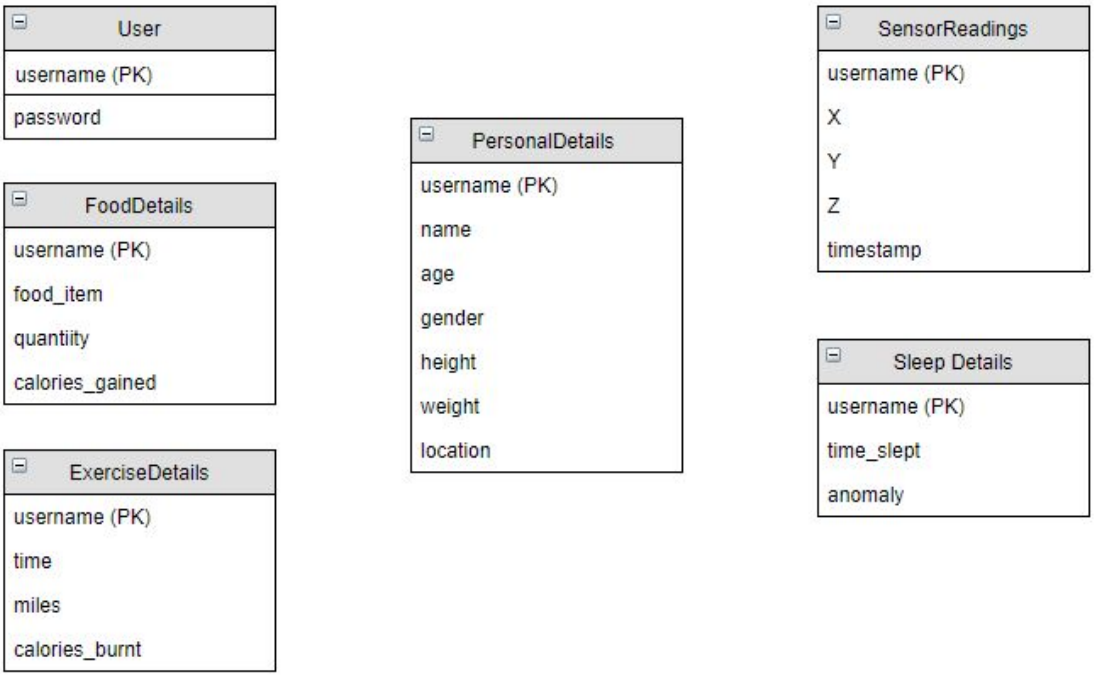

Figure 13: Database Schema

3.5 Network Protocol

Network protocols are formal standards and policies comprised of rules, procedures and formats that define communication between two or more devices over a network. Network protocols govern the end-to-end processes of timely, secure and managed data or network communication. Network protocols include mechanisms for devices to identify and make connections with each other, as well as formatting rules that specify how data is packaged into messages sent and received. Some protocols also support message acknowledgment and data compression designed for reliable and/or high-performance network communication.

Here we have different components of the application which use different network protocols. To accomplish various functionalities in the application, the client has to connect to the server using HTTP protocol. To store and retrieve the data from the database, the system communicates with the SQLite database using queries to create, update and retrieve data. Also, protocols will be required to connect to the Android sensors, through the Android SDK.

3.6 Global Control Flow

1. *Execution orderness:* The system is majorly event-driven, which means it waits in a loop for events.
2. *Time dependency:* The system requires timer to set up the wake up time. The alarm will ring when the timer goes off. Also, the system contains components which are of event - response type.

3.7 Hardware Requirements

The application requires a Smartphone running Android 6.0 platform (Marshmallow) or higher. The smartphone should have a built-in Accelerometer, so that the steps of the user can be retrieved through the Android SDK. The screen resolution should be high around 480x854 pixels. The hard drive storage should be greater than 1GB. The smartphone should have constant access to the internet with minimum network bandwidth of 3.1Mbps. MySQL database is to be used.

4. ALGORITHM AND DATA STRUCTURES

4.1 Algorithm

Our application “napIT” has features like Exercise, Food and Sleep monitoring. Every feature requires some form of algorithm so that it could work efficiently. For example, the Exercise feature should be able to give the user the distance covered, at what speed and time thus giving out the calories burnt. Similarly, in the Food section, after the user enters the food, the application should give out the amount of calories burnt. Sleep monitoring requires sensor readings and to detect whether the sleep is proper or improper, the readings(after analysis and feature extraction) need to be fed into the SVM as input and give out the desired results.

Distance parameter

This parameter will give the distance travelled by the user using the following formula:

Distance= number of steps x distance per step (1)

The distance per step depends upon the speed and height of the user. The step length would be longer if the user is taller or running at higher speed. Our system updates the distance, speed and calories parameter every 2 secs. We use the steps counted in every 2 secs to judge the current stride length.

Steps per 2 s	Stride (m/s)
0~2	Height/5
2~3	Height/4
3~4	Height/3
4~5	Height/2

5~6	Height/1.2
6~8	Height
>=8	1.2 x Height

Table 10: Stride as a Function of Speed and Height

Speed parameter

Our application also displays the speed to the user along with the distance. The formula for speed is given by:

Speed= distance x time (2)

Our system will calculate speed as:

Speed= steps per 2 s x stride/2 s (3)

Calorie parameter

When it comes to Exercising, the most important parameter that the user is interested in is the amount of calories burnt. Though, there is no accurate means for calculating the rate of expending calories, some factors that determine it include body weight, intensity of workout, conditioning level and metabolism. We can thus estimate it using a conventional approximation. The following table shows a typical relationship between calorie expenditure and running speed.

Running Speed(km/h)	Calories Expended (C/kg/h)
8	10
12	15
16	20
20	25

Table 11: Calories Expended vs. Running

From the above table, we have,

Calories (C/kg/h)= 1.25 x running speed (km/h) (4)

Converting the unit of speed from km/h to m/s, we get,

Calories (C/kg/h)= 1.25 x running speed (m/s) x 3600/1000 =4.5 x speed (m/s) (5)

The calories parameter would be updated every 2 secs with the distance and speed parameters. So, to account for a given athlete's weight, we can convert Equation 5 to Equation 6 as indicated. Weight (kg) is a user input, and one hour is equal to 1800 two- second intervals.
Speed = distance/time, so Equation 3 can be used to get the speed parameter, as steps per 2 s and stride

Calories (C/2 s)= 4.5 x speed x weight/1800 =speed x weight/400 (6)

If the user takes a break in place after walking or running, there would be no change in steps and distance, speed should be zero, then the calories expended can use Equation 7 since the caloric expenditure is around 1 C/kg/hour while resting.

Calories (C/2 s) = 1 × weight/1800 (7)

Finally, we can add calories for all 2-second intervals together to get the total calories expended.

SVM(Support Vector Machine) for sleep analysis(whether proper or improper sleep)

The main aim of our application "napIT" is to monitor sleep and perform sleep analysis. We initially gather the sleep data using accelerometer and after processing it (feature extraction), we input it into the Support Vector Machine(SVM) and get the desired classification of sleep as proper or improper sleep.

Support Vector Machine is a supervised machine learning algorithm that analyze data used for classification and regression analysis. Given a set of training example, each marked as belonging

to one or the other of two categories, an SVM training algorithm build a model that assigns new examples to one category or the other.

Therefore, this machine learning algorithm suits our needs perfectly. We take 2 classes of sleep data as proper and improper sleep as our training set. This we implement using our research on sleep analysis as one sleep cycle repeats every 90 minutes and there are 3 such sleep cycles. We will thus have an appropriate data for proper and improper sleep which can be successfully implemented as our training data. We then build a model using SVM and run it on our testing data. Our testing data is the sleep data for various users. The algorithm will classify the sleep as proper or improper using the training data and thus give a result to the user.

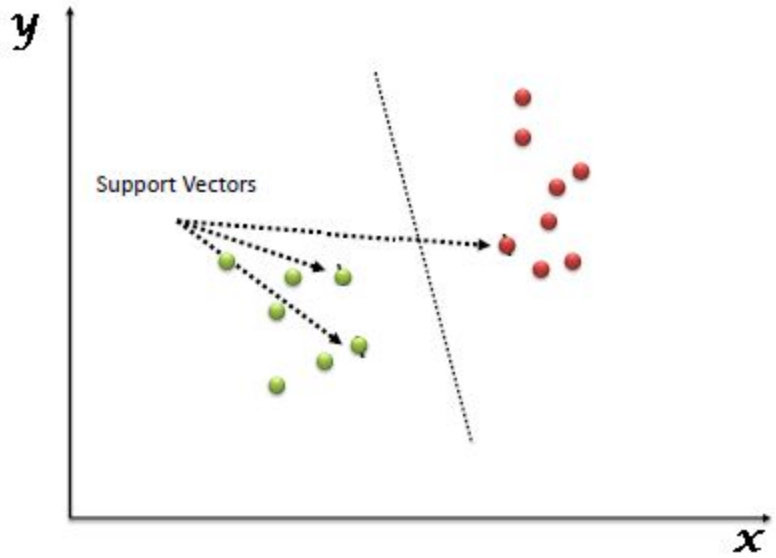

Figure 14: Support Vector Machine Classifier

4.2 Data Structures

The database that we are using to store and retrieve the data is SQLite. The SQLite database will roughly use the following data types:

Data types: The major data types that would be required are integer and string.

1. Sensor Readings (Int): The accelerometer readings will be stored in the data, for every user.
2. User Identification number (user_ID) (Int): It will be auto numbered and auto incremented, and will be used to defined every user.
3. Name, username, email id, location (string): The name, username, email id, location will store the following personal details of the user.
4. Time, miles, calories_burnt, calories_gained (Int): It will be used to store the exercise and food related details of the user.

5. USER INTERFACE DESIGN AND IMPLEMENTATION

The user interface (UI), in the industrial design field of human–computer interaction, is the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operator's' decision-making process. In the instance of mobile application, it is in the form of a Graphical User Interface based on Android SDK.

Following are the designs that were modified to improve the user experience and the implementation uptill now.

- **Login and Authentication:** When the user opens the application, he/she will see two buttons - login and register. If the user already has an account, he/she can login with the existing username and password which they already have and then hit the “Login” button. These details will be compared with the data stored in the database. If there is a match, the user will be authenticated and directed to the home page. If the user is new, he/she can create a new account by pressing the “Register” button. Once it is pressed, the user will be asked to enter his/her personal details. The details entered in this page will be stored in a database once the user hits the “Submit” button.

Figure 14: Main Page and Login

- **Main Menu:** The main menu has different buttons like food, sleep, exercise and view statistics for the user to select. Once the user, selects an option he/she will be directed to the respective page. On the top-left corner, the image button will display a drop down list when it is clicked. The drop down list will have options which will allow the user to manage his/her profile and an option to logout. The user can log out anytime by clicking on the "logout" button.

- **Manage Account:** The user can manage his/her account by clicking on the image button on the top-left corner of the application and then select "Manage Profile". This includes updating or adding a profile picture, height, weight, age and current location of the user.

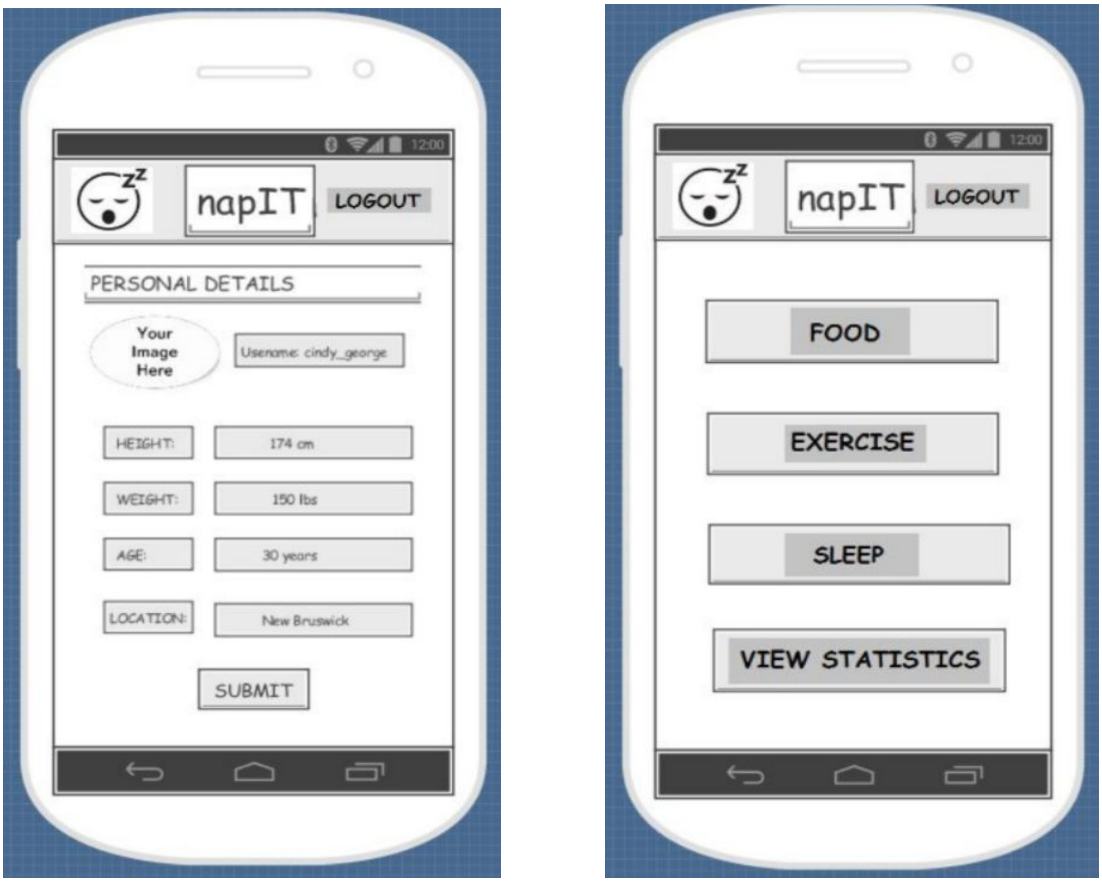

Figure 15: Manage Accounts and Home Page

- **View Statistics:** On clicking this button, the user will be directed to a screen which will provide suggestions and display recorded results. This page will show the number of hours the user has slept, the anomaly detected and the suggestions based on anomaly detected. This will also display the number of calories burnt during workout and the number of calories gained during food intake for that particular day. The user can log out anytime by clicking on the “logout” button.

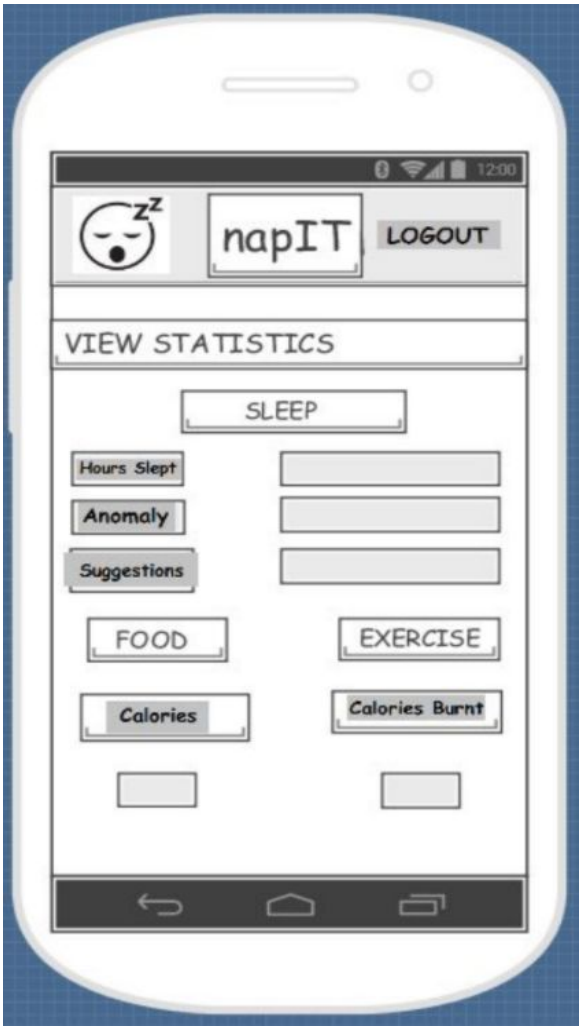

Figure 16: View Statistics

- **Contact Us:** This page has details about the people to be contacted in case the users have any trouble using the application or to clarify any doubts regarding it. It also provides a link to the website associated with the application.

Figure 17: Contact Us

- **Frequently Asked Questions:** This page has some solutions to frequently asked questions. This page is meant to assist the user when they encounter some problem while using the application and to troubleshoot the problem.

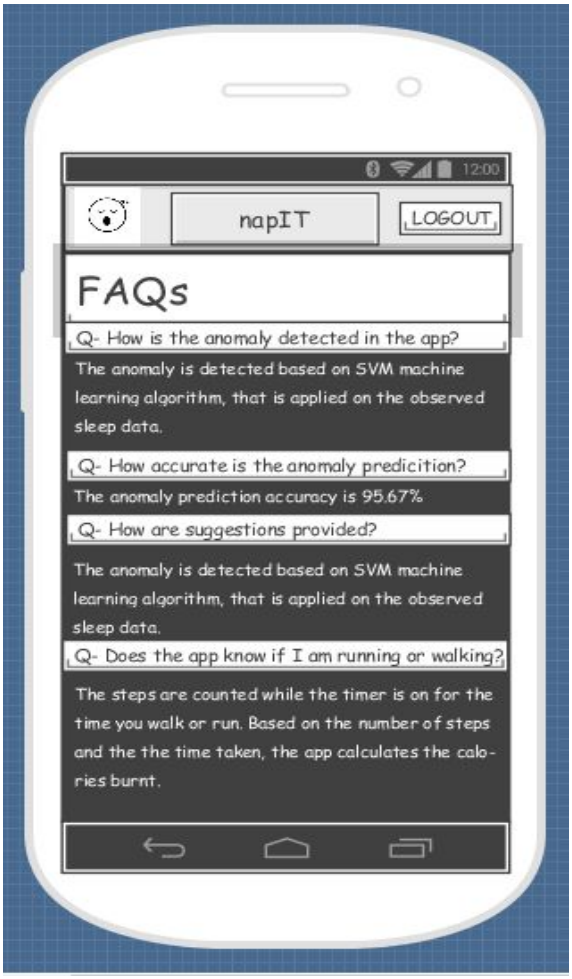

Figure 18: FAQ

- **Feedback and Rating:** This purpose of this page is to collect valuable feedback for the user and the rating they would provide for the application.

Figure 19: Feedback

Additions in design:

We have included the following pages in design

1. Frequently Asked Questions(FAQ)
2. Contact Us
3. Feedback and Rating

We believe that these pages will help improve user experience and will provide guidance for us to make useful updates to the application.

6. DESIGN OF TESTS

6.1 Test cases and Test coverage

Test-case Identifier:	TC1
Use Case Tested:	UC1 Login
Pass/ Fail criteria:	The test passes if the user enters a key that is contained in the database, with less than a maximum allowed number of unsuccessful attempts
Input Data:	Username and password
Test Procedure:	Step 1. Type in an incorrect keycode and a valid door identifier Step 2. Type in the correct keycode and door identifier
Expected Result:	1. User enters invalid credentials. 2. Authentication fails. 3. User is prompted to re-enter credentials. a. User is finally able to login b. User fails more than 5 times in under 2 min

Table 12: Test case for UC1: Login

Test Coverage: The test will cover inputs that are null, empty, correct, incorrect.

Test-case Identifier:	TC2
Use Case Tested:	UC2: Registration
Pass/ Fail criteria:	The test passes if the user can register a username that is not contained in the database
Input Data:	Username and password
Test Procedure:	Step 1. Attempt to register username that already exists in the database. Step 2. Attempt to register username that does not exists in the database.
Expected Result:	1. The User clicks create account 2. The user is asked to create a new username and password. a. The username is checked against existing usernames in the database. Password is checked against regex to ensure password meets minimal requirements. i.) If username exists the user is informed and must pick a new unique username. ii.) If the username does not exists the user is allowed to continue. iii.) The password does not match the minimal requirements. E.g. Does not match regex condition. iv.) The password does match regex and user is allowed to continue. 3. User proceeds to Enter Personal details.

Table 13: Test case for UC2: Registration

Test Coverage: The test will cover inputs that are null, empty, correct, incorrect.

Test-case Identifier:	TC3
Use Case Tested:	UC3: Manage Account
Pass/ Fail criteria:	The test passes if the user can change details about their account.
Input Data:	User details.
Test Procedure:	Step 1. User enters information in system and information is updated.
Expected Result:	1. User clicks settings 2. User enters settings and personal details 3. User entries are validated a. User entries are valid b. User entries are not valid 4. User entries are valid 5. User saves changes

Table 14: Test case for UC3: Manage Account

Test Coverage: The test will cover inputs that are null, empty, correct, incorrect.

Test-case Identifier:	TC4
Use Case Tested:	UC4: Monitor Sleep
Pass/ Fail criteria:	The test passes if the application can collect sensor reading over the sleep cycle.
Input Data:	Sensor readings
Test Procedure:	Step 1. User clicks the monitor sleep button and the application begins recording the sensor data..
Expected Result:	1. User clicks begin sleep monitoring 2. Application starts recording sensor data. 3. Application filters data. 4. App attaches a timestamp 5. App stores data into database.

Table 15: Test case for UC4: Monitor Sleep

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

Test-case Identifier:	TC5
Use Case Tested:	UC5: Anomaly detection
Pass/ Fail criteria:	Anomaly detector gives false positive or false negative
Input Data:	Sensor readings from database
Test Procedure:	Step 1. Cross validation data is feed into the anomaly detection algorithm. Step 2. Data labels are compared to the tags output by the detection algorithm.
Expected Result:	Data labels from the cross validation set match those output by the detection algorithm.

Table 16: Test case for UC5: Anomaly Detection

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

Test-case Identifier:	TC6
Use Case Tested:	UC6: View Statistics
Pass/ Fail criteria:	User is able to view stats about their sleep patterns.
Input Data:	Sensor readings from database
Test Procedure:	Step 1: User selects the view stats button. Step 2: The Stats are displayed.
Expected Result:	The application displays the stats for the user.

Table 17: Test case for UC6: View Statistics

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

Test-case Identifier:	TC7
Use Case Tested:	UC7: Recommender
Pass/ Fail criteria:	Recommend corrective actions to the user which makes sense
Input Data:	Sensor readings from database, output of anomaly detector.
Test Procedure:	Step 1: Trigger recommender system with a known issue. Step 2: See if recommender gives user correct recommendation .
Expected Result:	The recommender gives the correct corrective action.

Table 18: Test case for UC7: Recommender

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

Test-case Identifier:	TC8
Use Case Tested:	UC8: Smart Alarm
Pass/ Fail criteria:	Recommend corrective actions to the user which makes sense
Input Data:	Username database
Test Procedure:	Step 1: Set alarm. Step 2: See if alarm triggers at set time and begins recording sensor data. Step 3: Smart alarm will set best time to wake up.
Expected Result:	1. The system starts recording the user's sleep data with a sensor at the set time. 2. Based on the user's daily activity, the application will set the best time for waking up. 3. At the particular time, the alarm will ring. 4. The user can choose to either snooze or dismiss it.

Table 19: Test case for UC8: Smart Alarm

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

Test-case Identifier:	TC9
Use Case Tested:	UC9: Eat
Pass/ Fail criteria:	App stores the calorie content to the database
Input Data:	Username, database
Test Procedure:	Step 1: User enters calories of food.
Expected Result:	1. Calories are added to the database.

Table 20: Test case for UC9: Eat

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

Test-case Identifier:	TC10
Use Case Tested:	UC10: Exercise
Pass/ Fail criteria:	App stores the calorie burned to the database
Input Data:	Username, sensor data
Test Procedure:	Step 1: User begins recording data from sensors.
Expected Result:	1. The accelerometer senses the distance covered and the calories burnt.

Table 21: Test case for UC10: Food

Test Coverage: The test will cover inputs that are null, empty, out of the valid range, and valid readings.

6.2 Integration testing

Integration testing will be performed between:

- Database and each screen that requires userdata.
- Sensor and screen reading the data
- Sensor screen and the database
- Database and the anomaly detector.
- Database and the recommender.

We will be performing top down integration testing.
Each integration test will first with establishing that communications between each component is possible. Then communication will be check to confirm it is correct. Valid and invalid data will be sent to each component in order to catch edge conditions. Normal operation will be resumed and checked against the sensor readings.

7. PROJECT MANAGEMENT

This project has three major subsystems. Some will be developed in parallel by dividing our team into groups of three. The section below discusses the parts of the subsystems. The group will start working on the functional requirements and the user interface of the project.

7.1 Product Ownership

- Madhura and Manasi will be working on the design of System Architecture, design of data flow of the system, storage of data, detection of anomaly , filtering of sensor, design of exercise recommender.
- John and Siddhi will work on setting up the system to share code, System Architecture design,data flow design, sensor integration, anomaly detection, sensor filtering, backend integrations between components.
- Aishwarya and Preetraj will work on UI design, System Architecture design, data flow design of the system, anomaly detection, sensor filtering, calorie-intake recommender design.

Group	Member 1	Member 2
Subgroup 1	Madhura Daptardar	Manasi Mehta
Subgroup 2	John Grun	Siddhi Patil
Subgroup 3	Aishwarya Srikanth	Preetraj Singh Gujral

Table 22: Division of members

Subgroup	Functionality contributions
Subgroup 1	Setup Document collaboration, System Architecture design, Design data flow of the system, data store, anomaly detection, Research, sensor filtering, exercise recommender.
Subgroup 2	Setup system to share code, System Architecture design,Design data flow of the system, sensor integration, anomaly detection, Research, sensor filtering, backend integrations between components.
Subgroup 3	UI design, System Architecture design, Design data flow of the system, anomaly detection, Research, sensor filtering, calorie-intake recommender.

Table 23: Functionalities of each subgroups

The following table consists of all the deadlines we will meet for this project. The deadlines include report submissions, demos etc. Since we do not have sufficient time, the development phase will be done in parallel.

Milestones	**Description**	**Planned Date**
M0	**Project Proposal**	**Sep. 17, 2017**
	Finalize the project scope after getting feedback	**Sep. 20, 2017**
M1	**First Report**	**Oct. 18, 2017**
	Statement of work and requirements	**Sept 23, 2017**
	Functional requirements Spec and user interface	**Sept 29, 2017**

	Full report submission	**Oct. 6, 2017**
M2	**Second Report**	**Oct 27, 2017**
	Interaction Diagrams	**Oct 13, 2017**
	Class Diagram and System Architecture	**Oct 20, 2017**
	Full report submission	**Oct 27, 2017**
M3	**First Demo**	**Nov 1, 2017**
M4	**Third Report**	**Dec. 8, 2017**
M5	**Second Demo**	**Dec. 13, 2017**
M6	**Electronic Project Archive**	**Dec. 15, 2017**

Table 24: Project Deadlines

7.2 Gantt Chart

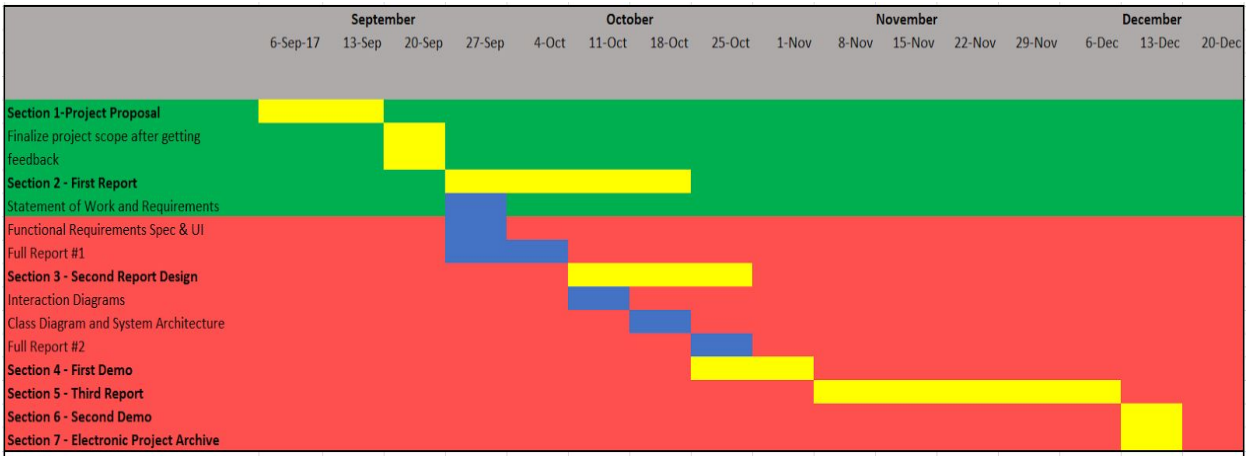

Figure 20: Gantt Chart

REFERENCES

[1] Software Engineering, Ivan Marsic Home Page.
http://www.ece.rutgers.edu/~marsic/Teaching/SE1/

[2] Network Protocol
https://www.techopedia.com/definition/12938/network-protocols

[3] Network Protocol
https://www.lifewire.com/definition-of-protocol-network-817949

[4] SQLite Database
https://www.sqlite.org/about.html

[5]Class Diagram
https://en.wikipedia.org/wiki/Class_diagram

[6] Stride as a Function of Speed and Height
https://www.ijser.org/paper/Personal-Medical-Wearable-Device-for-Distance-Monitoring.html

[7] Calories Expended vs. Running
https://www.ijser.org/paper/Personal-Medical-Wearable-Device-for-Distance-Monitoring.html

[8] Support Vector Machine Classifier
https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/