SOFTWARE ENGINEERING I

REPORT 3

Beat.My.Run

December 10, 2015

Nivetha Balasamy

Careena Braganza

Valia Kalokyri

Thara Philipson

Nirali Shah

Rahul Shome

Rutgers University

Contents

1 SUMMARY OF CHANGES

1. We changed the whole section of System Requirements which was part of report 1. Previously we had only user stories but now we added an expanded functional and non-functional requirements table.

2. We changed the use case diagram which has updated actor names and layout arrangement.

3. In the User Interface Specification session in report 1, we decide to use the latest screenshot of webpages of our project instead of the old ones. And the effort estimation and description are based on the new project.

4. We added a section for Effort Estimation using Use Case Points

5. We added a new section for Design Patterns

6. In the Domain model diagrams section in report 1, we changed all the figures to update the boundary of the domains.

7. In the Class Diagram and Interface Specification session in report 2, we added the Design patterns and OCL Contract Specification.

8. In the Interface Design and Implementation session in report 2, we changed all the screenshots of our application so as to reflect the latest UI implementation.

9. In the Design of Tests session in report 2, we added all the Unit tests, Integration test and test coverage subsections.

10. We added a new section with the history of work

2 CUSTOMER STATEMENT OF REQUIREMENTS (CSR)

2.1 Problem Statement

Physical activity in any form improves physical health and mental well-being. Research[1] indicates that regular exercise can add up to five years to your life. Even a short burst of 10 minute aerobic exercise like brisk walking or running can improve mental alertness, energy and positive mood. Sticking to an exercise regime is difficult and motivation levels are low. Even exercise fanatics find it hard to get motivated from time to time. Thus, when physical activity is kept interesting by having a constant source of motivation it helps to keep the exercise regime firm, because if it's not fun it's not sustainable.

In the present era, things are easily made accessible. The internet of things and proliferation of sensors in handheld mobile devices allow access to various features in minimal clicks and minimal time. Thus, it would be a good option to gain motivation an application running on a mobile device.

Studies[2][3][4] have found that music reduces the perception of how hard you are running by about 10 percent. An external stimulus as exciting as music can actually block some of the internal stimuli trying to reach the brain-such as fatigue-related responses from muscles and organs. When these messages are blocked, reduced physical effort is experienced which encourages to run faster and further. Music also elevates positive aspects of mood[5][6] such as elation and happiness, and reduces negative aspects such as tension, fatigue, and confusion. It is common to find people exercising with their headphones on, listening to the music. Music makes exercise interesting, limits the feeling of compulsion and also makes it more productive. However, studies have shown that not just listening but controlling and creating music in time to one's pace had an even more profound effect on perceived effort during a workout. Synchrony can help the body use energy more efficiently [7] [8]. When moving rhythmically to a beat, the body would have to make as many adjustments to coordinated movements as it would without regular external cues. Music can function as a metronome, helping to maintain a steady pace, reducing false steps and decreasing energy expenditure.

Wanting to be fit but not feeling motivated enough is a common concern. It requires a lot of dedication and time allotment. Making the exercising experience better is essential. Standard music apps do not reason about the state of physical activity.Research shows that the correct music can enhance the exercising experience. User-defined playlists will be adapted to instantaneous rates of activity, ie. music ideal for jogging might not prove pleasurable while sprinting.

Towards this end, an application trying to enhance user's running experience by matching the music playback with the intensity of the workout would maintain the positive effect of music on the running experience. In order to achieve this, the application uses the embedded accelerometer of the phone to monitor the speed or velocity of the running or jogging activity of the user and based on the intensity can playback songs that match user's activity with the tempo of the song. For example, while running fast s/he will be able to listen to a high tempo song. In this solution, the cadence of the run has to be matched with an attribute of the music file selected such as the duration, energy or beats per minute.

In addition, the application can be made more personalized by playing songs that match user's preferences. In order to achieve this, the application can have access to the phone's music library and you-tube's playlists/songs that a user has previously liked or listened to. This way it can playback songs that the user enjoys and not something which is irrelevant to the user's taste.

Also, exercising might seem monotonous, which is mainly due to the lack of knowledge about workout.It leads to unnecessary stressing of the body and mind, thus reducing the time of exercise leading to faster exhaustion and mental stresses. Many a times acquiring a proper training and meeting the trainer's schedules is difficult which again keeps us from exercising due to the fear of not doing it correctly.

A feature that can add to the enhanced user experience would be allowing the user to choose from a predetermined workout template which could be designed to sequentially increase and decrease the intensity of running. For example, the exercise session can be broken down to a five minute jog followed by a short sprint for two minutes and another segment of a jog for three minutes[9]. This can be useful in cases when you don't know how to plan an effective workout where the application would be able to suggest some predefined template workout plans and allow to select one based on one's fitness level.

Again, running alone proves to be a boring and strenuous task and lacks motivation. By creating competitive situations, motivation levels increase and can enhance performance in exercise events[10]. Competitive conditions also augment the cortisol response to exercise, suggesting that enhanced sympatho-adrenal system activation occur in such situations which may be one of the key "driving forces" to performance improvement.

Exercise is generally targeted to maintain fitness but if it is modified to be a game which could challenge friends it would create additional drive and encouragement. In order to achieve that, the application is targeted towards maintaing personal running achievements (the distance travelled and the time taken), allow searching friends that use the same application,send out requests for challenges and also accept a posed challenge. The challenge could

be based on running time, meaning that the person challenegd should run the same distance in less time in order to win the challenge.

Furthermore, it is an undeniable fact that outdoor fitness training comes with many advantages. It provides opportunities to explore new places, breathe in fresh air and eliminates the need of a mundane indoor exercise routine like running or jogging on a treadmill.Hence, a feature which could provide real time suggestions on location of nearby parks and tracks would be an added advantage. It can be in the form of a map which can show the names and locations of the parks and gyms nearby to our current location. This gives enough information to plan a workout depending on the location in which the user is currently in.

However, when exercising outside, weather becomes an important consideration - extreme heat/cold and rain could be a hindrance to a productive outdoor workout. Weather data integration can become an added feature which could help make an informed decision. Temperature, heat advisory and climate changes could be checked and notified while going outdoors for a workout or even exactly at the the time the user starts running.

After all this, analysing a workout is the most effective way of providing encouraging information.It is always necessary to understand the previous performances and keep track of them in order to perform better. Comparisons of the workouts through a period of time can achieve exactly this. Hence, data visualizations of the workout pattern for a duration of time with regard to the distance covered during the run and the time taken can be a great way of keeping track. Last but not least, in a number of studies[11], researchers reported that one out of every 10 premature deaths around the planet are caused by not exercising. In other words, not exercising kills almost the same number of people as smoking does, and Time magazine went so far as to label the situation as a global pandemic. Specifically in the United States, the American Heart Association's "Circulation" research journal reports that approximately 250,000 deaths every year are caused by not exercising. To counteract these startling numbers, the journal highlights that many studies linking a lack of exercise to premature death recommend that people exercise three days per week for 30 to 60 minutes each day. This app would contribute towards helping the general population stick to their exercise regime and stay healthy.

The project proposes a personalized and user-friendly exercise application, supporting music playback that adapts to the intensity of the workout while running. Users can also get contextual information about weather and location, keep track of their workout attributes, compete with their friends and track high scores through leaderboards. The aim of the project is to promote physical well-being by enhancing the running experience and encouraging users to exercise.

2.2 Glossary of Terms

- **Beats per minute**: Beats per minute (BPM) is a unit typically used as a measure of tempo in music and heart rate. The BPM tempo of a piece of music is conventionally shown in its score as a metronome mark. This indicates that every one minute there should be 120 quarter notes.
- **Metronome**: A metronome is any device that produces regular, metrical ticks (beats, clicks) - settable in beats per minute. These ticks represent a fixed, regular aural pulse; some metronomes also include synchronized visual motion (e.g. pendulum-swing).
- **Pulse**: In music and music theory, the pulse consists of beats in a (repeating) series of identical yet distinct periodic short-duration stimuli perceived as points in time occurring at the mensural level.
- **Sprinting**: Sprinting is the act of running over a short distance at (or near) top speed. It is used in many sports that incorporate running, typically as a way of quickly reaching a target or goal, or avoiding or catching an opponent.
- **UI**: Its the user interface that in the industrial design field of human-machine 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 operators' decision making process.
- **Accelerometer**: An accelerometer is an electromechanical device used to measure acceleration forces.Acceleration is the measurement of the change in velocity, or speed divided by time.
- **Data Visualization**: Data visualization is the presentation of data in a pictorial or graphical format. For centuries, people have depended on visual representations such as charts and maps to understand information more easily and quickly.
- **Leaderboard**: A scoreboard showing the names and current scores of the leading competitors. Leaderboards can contain text, images, or even animations.
- **API**: In computer programming, an application programming interface (API) is a set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types.

3 SYSTEM REQUIREMENTS

3.1 User stories

The functional and non-functions requirements have been enumerated from the point of view of the user as user stories

- **Logging in:** The user can log into the application using his *Google* account credentials. Once the login credentials are verified, the user can access all the features of the app. The *Google* account gives access to the *Youtube* account history and the music preferences of the user on *Youtube*.
- **Planning a run:** The user wants to schedule a run.
    1. *Running conditions:* The user can see an UI with the current weather conditions based on current location and a forecast over the next 3 hours.
    2. *Running locations:* The user can look for running locations around the vicinity like jogging tracks, parks and gyms. The UI shows a map along with the names and addresses of the locations.
- **Starting a run:** The user wants to initiate a new run. A new run can also initiated by accepting a challenge. The UI provides the current weather conditions based on location and a forecast for the next 3 hours. In addition, the UI provides a button to start the run.
    1. *Music Playback:* The app chooses a sound track based on the intensity of the user's running. The faster the user runs, the higher the tempo of the music becomes. The running rate is obtained over a window of time from accelerometer data. A change of tempo is reflected in the next song selection. The tracks are selected based on the user's *Youtube* history and playlists. The UI shows the track information and music controls to pause, play, skip and replay the last song.
    2. *Real-time Running Information:* The UI shows the current distance covered during the run and also the current duration of the run.
    3. *Stopping a Run:* There would be a button in the UI to stop the run. Once the run ends, the UI shows a summary of the completed run, including distance and the duration of the run. The UI also shows a map with the route covered during the run.

4. *Challenging a Friend:* After the user stops a run the user can challenge a friend to run the same distance, but in a shorter time. The UI will let the user select friend/s from a list or search for friend/s by username and challenge them.

- **Training:** The app gives the option to exercise according to a predetermined workout template which progressively increases and decreases intensity of running. The workout template includes sprinting and jogging on and off during a run. The UI lets the user select between three inbuilt templates - Beginner, Intermediate and Advanced. For example , the exercise session can be broken down to a five minute jog followed by a short sprint for two minutes and another segment of jog for three minutes

    1. *Starting Training:* The user interface would display weather information based on the current location.
    2. *During Training:* The user interface would be identical to running except for the nature of music playback. The tracks would be selected based on the workout template and can provide audio cues for switching between training modes(jogging and sprinting).

- **Past Exercise Data:** The app aggregates past exercise data and provides an UI to display data visualizations of the data stored for the user. It can show the changes in the running distances and durations over time.

- **Challenges:** Challenges can be sent or accepted between friends.

    1. *Adding a Friend:* An UI lets the user search for the friend using the username and send a friend request. The user can also accept friend requests sent by other users.
    2. *Accepting a Challenge:* The user sees an UI with a list of pending challenges from friends. Once the user accepts a challenge, the user proceeds to the UI to start a run.
    3. *Challenge Result:* At the end of a challenge run, the user either wins or loses the challenge based on how much time the user took to cover the challenged distance. The UI shows the current run statistics and route covered. The UI will also notify the user of this result and also allow the user to use the current run to challenge other friends.

4. *Leaderboards:* The results of challenges update a leaderboard UI which shows the win-lose statistics of the friends of the user, sorted by the number of wins in descending order.

3.2 Enumerated Functional Requirements

Table 1: Functional Requirements

ID	Priority Weight	Description
REQ-1	5	The system should allow the user to enter Google login credentials
REQ-2	4	The system should perform authentication of user's Google account credentials by connecting to Google database
REQ-3	5	The system should show the run options if credentials are verified
REQ-4	4	The system should get the personal details of the user
REQ-5	5	The phone accelerometer should start when the user has selected Run
REQ-6	5	The system should access the user's YouTube song preferences
REQ-7	4	The system should calculate the BPM of the songs using the Echonest API
REQ-8	4	The system should compute the distance run and calories from the steps and height of user after completing the run
REQ-9	3	The system should push all the computed run information corresponding to the user to the database
REQ-10	3	The system should provide the user with the Challenges option after completing the run

Table 2: Functional Requirements.. Continued

REQ-11	4	The system should allow the user to send challenge to another member based on the distance and duration run
REQ-12	2	The system should let the user find a friend to challenge from the database
REQ-13	4	The system should let the user choose if the challenge can be accepted or rejected
REQ-14	2	The system should delete the challenge if the user chooses to reject it
REQ-15	3	The system should allow the user to view all the pending challenges
REQ-16	5	The system should allow the user to monitor the statistics of the exercise
REQ-17	3	The system should let the user view the Statistics according to the time period chosen
REQ-18	3	The system should provide the user with the option to select the time period to view progress
REQ-19	5	The system should plot the distance run , duration of run and calories burnt for the selected period
REQ-20	3	The system should provide the user with the option to view cummulative monthly progess
REQ-21	2	The system should allow the user to select the month to view progress
REQ-22	3	The system should show the entire montly statistics of exercise information

3.3 Enumerated Nonfunctional Requirements

Table 3: Non- Functional Requirements

ID	Priority Weight	Description
REQ-23	5	The system shall be presented with easy to use User Interface
REQ-24	3	The system shall allow the user to create a Google account if one does not exist
REQ-25	4	The system shall update and fetch from database promptly
REQ-26	4	The system shall present the Statistics in a neat and presentable manner
REQ-27	3	The system shall keep the sign in credentials of the user safe
REQ-28	2	The phone should be connected to the Internet
REQ-29	4	The system shall have quick response and screen refresh time
REQ-30	3	The system shall efficiently handle interrupts
REQ-31	3	The phone sensor's accuracy, sample rate and sensitivity is set according to the required operating range
REQ-32	2	The application operates successfully with different bandwidth rates and network types
REQ-33	2	The application operates successfully on a range of screen resolutions and form factors

3.4 On-Screen Appearance Requirements

The following drawings, Figure 38 and 2 show paper prototypes of the way the screens appear to the user while s/he uses the various features of the App.

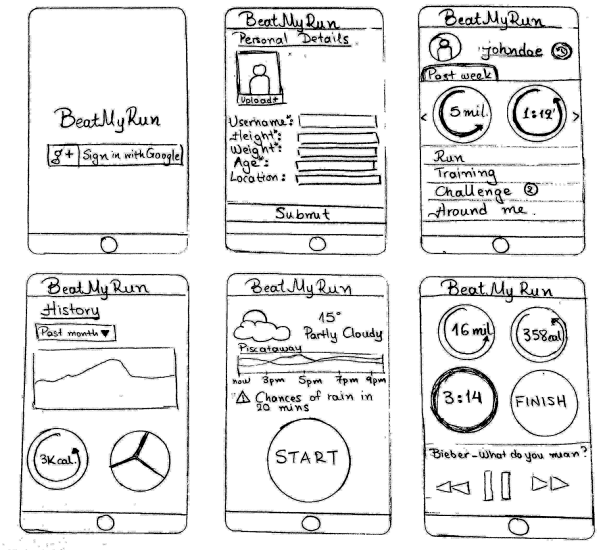


Figure 1: Paper Prototyping for the user stories

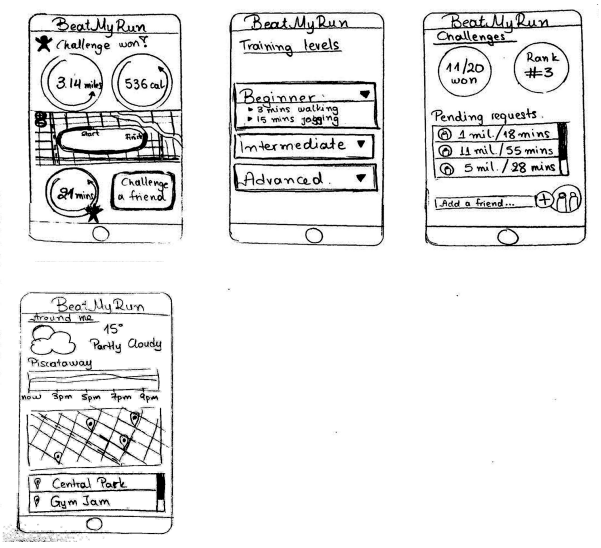


Figure 2: Continued....Paper Prototyping for the user stories

4 FUNCTIONAL REQUIREMENTS SPECIFICATIONS

4.1 Stakeholders

This is a personalized and user-friendly exercise application for a better physical health. This application aims in making exercise more interesting with its features. Every project has stakeholders and they are people who have an interest in the successful completion of the project. There are many different types of stakeholders, and they vary according to the project.The ideal stakeholders of this project are:

1. *End Users*: This application targets people who finds exercise boring and need motivation for the same.Achieving a goal or target with competitive spirit gives many users encouragement and motivation.Many people like to keep track of their workout pattern and compare it with friends.This application is ideal for such kind of users.

2. *Clients:* This application is competitive and can be an investment for many clients to invest or take up the project after its successful completion.

4.2 Actors and Goals

An actor can be human, physical object, or another system external to the application that interacts or participate with the system to achieve a goal. Each actor has a goal or responsibility that contribute to the total working of project.

Actor	Actor Goal	Use Case Name
Users	A logged in user accessing the application and its features	UC1
Friends	User can add friends and challenge them with a task that user has already accomplished	UC6
Echonest	This is used to find the bpm of a song that needs to be selected and played during workout	UC3
Android SDK	This is for collecting phone's accelerometer data in order to calculate user's running rate	UC4
Google and YouTube	Google account is for logging into application and youtube account is for getting user's prefered songs	UC1
Music Player	The song selected is played using music player with artist details and with stop,pause and next options.	UC4
Cloud Infrastructure Component	The details of the user while registering is stored and is used for various calculations.	UC1

4.3 Use cases

In a real world application, the various users communicate with the system to perform various functions. These components which play a role in the application to perform various functions as an initiator or a participant are called Usecases. A description of the Usecases,Diagramatic Representation and the Tracebility Matrix mapping each use case to the Requirements is shown below:

4.3.1 Casual Description

The summary use cases are as follows:

UC-1: Login - Allows the user to login into the system
UC-2: Personal Details - Allows the user to store his/her name, age, weight and height into the Database. («include» UC-1)
UC-3: Song Information Details - It obtains the beats per minutes of songs in the user's playlist.
UC-4: Run - Once the user has completed login and entered his personal details, s/he can begin his Run. In this use case, the app finds the speed of the user and plays a song with an equivalent beats per minute. («include» UC-3, «extend» UC-5)
UC-5: Train - Allows the user to select one of the three training levels of the system.That is beginner, intermediate and expert.
UC-6: Challenge - Allows the user to challenge one of his friends or one of the users of the application.
UC-7: Respond to Challenge - Allows he user to view pending challenge/s and accept/decline it/them.
UC-8 View Statistics - Displays all the information about the user like total miles run,calories burned,time and the challenge results. The history of data can also be viewed in statistics.

4.3.2 Use Case Diagram

The Use Case Diagram is a graphical representation of the broad ways in which the User can interact with the application. Use case diagrams are meant to capture the dynamic aspects of the system from a high level point of view. It is used to represent

- Requirements of a system.

- Present a holistic view of a system.
- Identify factors affecting the system, both external and internal.
- The interactions among actors and requirements

The diagram shown below represents the view of our system and how the user can interact with the Android app that speaks to external agents like the backend database, Google or Echonest.

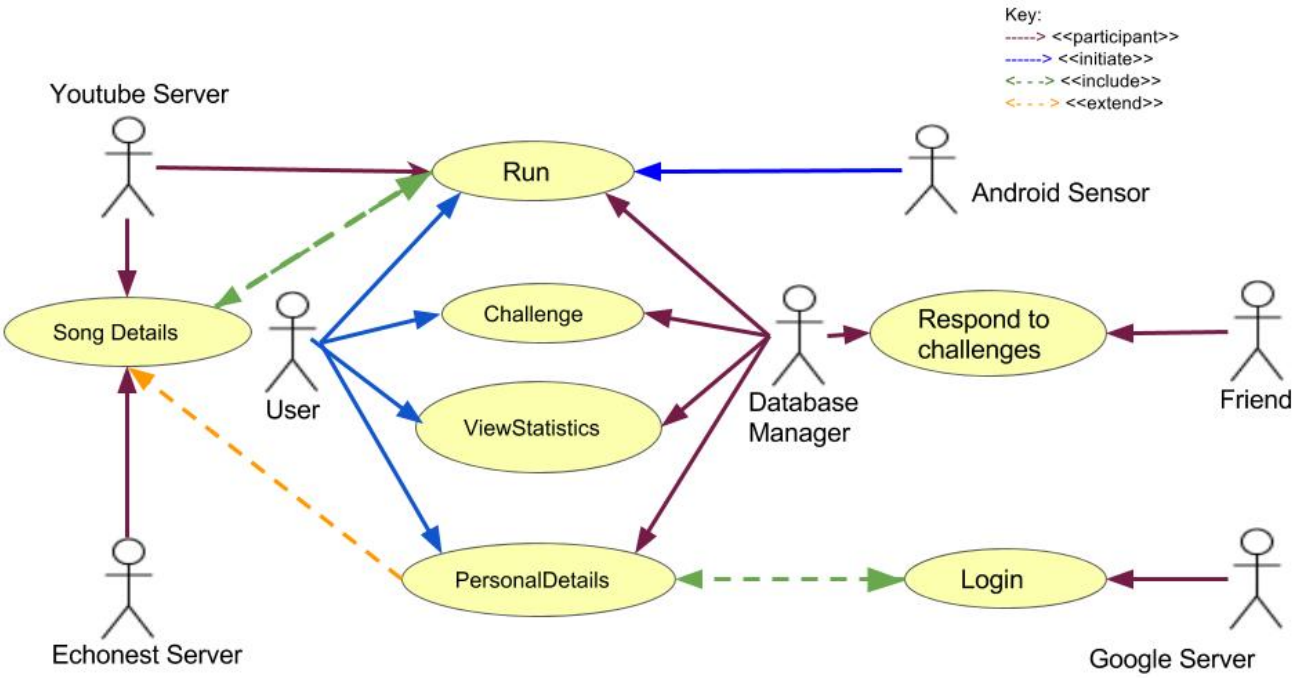


Figure 3: Use Case Diagram

4.3.3 Traceability Matrix

Traceability Matrix maps which Use Case is used to fulfill what Requirement. This way we know the functions of each Use Case and their responsibilities.Following Table shows the Traceability Matrix of the above described Use Cases.

Table 4: Traceability Matrix of Use Cases versus Requirements

REQ-N	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
R1	1	X	-	-	-	-	-	-	-
R2	2	X	-	-	X	-	-	-	-
R3	5	-	-	X	X	X	-	-	-
R4	5	-	X	X	X	X	-	-	-
R5	5	X	-	X	X	X	-	-	-
R6	4	-	-	-	X	-	-	-	-
R7	4	X	-	-	-	X	-	-	X
R8	3	X	-	-	-	-	X	-	X
R9	2	-	-	-	-	-	X	X	-
MAX PW		4	5	4	5	4	3	2	3
TOTAL PW		19	10	19	26	23	10	4	10

4.3.4 Fully-Dressed Description

Below are the Tables giving an elaborate description of each use case. The use cases are elaborated to give a more detailed description on the purpose of the use case, the requirement which it maps to, the actor,the actor's goal, participating actor and the pre and post conditions needed for the particular use case.

Table 5: Use Case 1 : Login

Use Case	**Functionality**
Use Case 1	Login
Related Req	REQ-1, REQ-2, REQ-5, REQ-7, REQ-8.
Initiating Actor	User
Actors Goal	To login and access app features
Participating Actors	Google, Database
Pre conditions	1. The user has a Google account. 2. The UI displays a 'Sign in with Google account' choice.
Post conditions	1. The user's Google credentials validated. 2. A new Beat my Run account is created in the database for the user. 3. The user's Youtube history is accessible. 4. (a) If it is first time login, the user is prompted to enter personal details. (b) All the available features of the app are displayed on the screen.
Flow of events for Main Success Scenario	1. The user opens the app and he clicks on the sign-in button. 2. The user then types his Google account credentials. 3. The system verifies the Google credentials and stores them in the database. 4. The application downloads all the user's song preferences of his youtube account 5. (a) For first time login, the personal details UI is displayed and the user is able to enter his details. (b) The UI displays a menu of features available from the app.
Flow of events for Extensions (Alternate Scenarios)	1. User enters invalid Google credentials. 2. Google authentication fails. 3. User is prompted to re-enter credentials.

Table 6: Use Case 2 : Personal Details

Use Case	**Functionality**
Use Case 2	Personal Details
Related Req	REQ-4.
Initiating Actor	User
Actors Goal	To store his personal details into the Database so that it can be used for 1. accessing songs from his/her Youtube playlist 2. calculating his speed based on no.of steps.
Participating Actors	Database
Pre conditions	1. The user has completed login. 2. The UI displays a form with the required fields of personal details.
Post conditions	The user's personal information is stored in the Database
Flow of events for Main Success Scenario	1. The user enters his/her required personal details into each field. 2. User submits the details. 3. The data is stored into the Database.

Table 7: Use Case 3 : Song Information Details

Use Case	**Functionality**
Use Case 3	Song Information Details
Related Req	REQ-3, REQ-4, REQ-5.
Initiating Actor	System
Actors Goal	To obtain the beats per minute of songs in the user's playlist.
Participating Actors	Echonest, Database
Pre conditions	1. The Google authentication is completed. 2. The song names from the user's Youtube playlist are available in the Database.
Post conditions	The beats per minute of all songs from the user's playlist are available in the Database.
Flow of events for Main Success Scenario	1. Echonest Initial API Connection is established. 2. The beats per minute of all songs from the user's playlist are obtained from Echonest. 3. The beats per minute of each song is stored in the Database.

Table 8: Use Case 4 : Run

Use Case	**Functionality**
Use Case 4	Run
Related Req	REQ-2, REQ-3, REQ-4, REQ-5, REQ-6.
Initiating Actor	User
Actors Goal	To go for a run
Participating Actors	Android SDK, Database, Youtube
Pre conditions	1. The user is logged into the system. 2. The user's personal details are stored in the Database. 3. The beats per minute of songs (obtained from Echonest) from the user's Youtube playlist are available in the Database.
Post conditions	1. The user can start his/her run. 2. The app chooses a sound track based on the intensity of the user's running. 3. The run information is recorded in the database.
Flow of events for Main Success Scenario	1. The user selects menu item 'Run'. 2. The app retrieves the steps per minute of the user through the Android SDK and calculates his/her speed. 3. The app matches the User's speed with a song having an equivalent beats per minute. 4. The UI displays the music controls i.e. controls for play/pause, next song and previous song. 5. The app records the user activity and stores it in the database. 6. The UI also displays a 'STOP' button to stop the run.

Table 9: Use Case 5 : Train

Use Case 5	**Functionality**
Use Case	Train
Related Req	REQ-3, REQ-4, REQ-5, REQ-7.
Initiating Actor	User
Actors Goal	To select one of the three training routines for the run.
Participating Actors	Youtube, Database
Pre conditions	1. The user is logged into the application using his/her Google account. 2. The beats per minute of songs (obtained from Echonest) from the user's Youtube playlist are available in the Database. 3. The UI displays a menu of all available functions.
Post conditions	The user can begin his Run in one of the three training modes.
Flow of events for Main Success Scenario	1. The UI displays a menu with option Train. 2. On selection of train, the UI displays three training options i.e. Beginner, Intermediate and Advanced. 3. When the user selects one of the three options, the pre-decided beats per minute for the routine is retrieved from the Database. 4. Songs are selected from the user's playlist which match the predecided beats per minute. 5. The UI displays the 'STOP' button to end the run and music controls i.e. controls for play/ pause, next song, previous song.

Table 10: Use Case 6 : Challenge

Use Case	**Functionality**
Use Case 6	Challenge
Related Req	REQ-8, REQ-9
Initiating Actor	User
Actors Goal	To compete with his/her friend who is registered on the app.
Participating Actors	Friend, Database, Android SDK
Pre conditions	1. The user completed his Run. 2. The app stored the duration and distance of the run in the database and displays it on the screen. 3. On completion of the run, the app displays an option 'Challenge Friend'.
Post conditions	On clicking 'Challenge Friend' the user is able to type in the username of the person he wishes to challenge.
Flow of events for Main Success Scenario	1. The user goes for a run. 2. The user's run duration and distance is stored in the database. 3. On completion of the run, the app displays an option to challenge friend/s. 4. The user can select friend/s by typing their username into the UI. 5. The selected friend/s is notified about the challenge.

Table 11: Use Case 7 : Respond to Challenge

Use Case	**Functionality**
Use Case 7	Respond to a Challenge
Related Req	REQ-9
Initiating Actor	User
Actors Goal	To accept or decline a challenge from a friend..
Participating Actors	Friend, Database
Pre conditions	The user has pending challenge/s.
Post conditions	1. The user has accepted/denied the challenge. 2. The friend (who initiated the challenge) is notified.
Flow of events for Main Success Scenario	1. The user sees an UI with a list of pending challenges from friends. 2. If the user accepts a challenge a)the display proceeds to the UI to start a run. b) Once the run is completed, app will also notify the user of this result and also allow the user to use the current run to challenge other friends. c) The friend (who initiated the challenge) is notified about the result. 3. If the user declines the challenge, the friend is notified.

Table 12: Use Case 8 : View Statistics

Use Case	**Functionality**
Use Case 8	View Statistics
Related Req	REQ-7, REQ-8
Initiating Actor	User
Actors Goal	To view user's past exercise activity.
Participating Actors	Database
Pre conditions	The user has completed atleast one run.
Post conditions	The UI displays visual information of the User's past running activity.
Flow of events for Main Success Scenario	1. The user selects 'View Statistics' from the menu. 2. The app aggregates past exercise data and provides an UI to display data visualizations of the data stored for the user. 3. It can show the changes in the running distances and durations over time.

4.4 System Sequence Diagrams

The Use Case Description is diagrammatically depicted in the System Sequence Diagrams. They show the step-wise implementation of the functions of the use cases.The sequence diagrams highlight the interaction between the various actors. Following are the System Sequence Diagrams which represent the Use Case Descriptions.

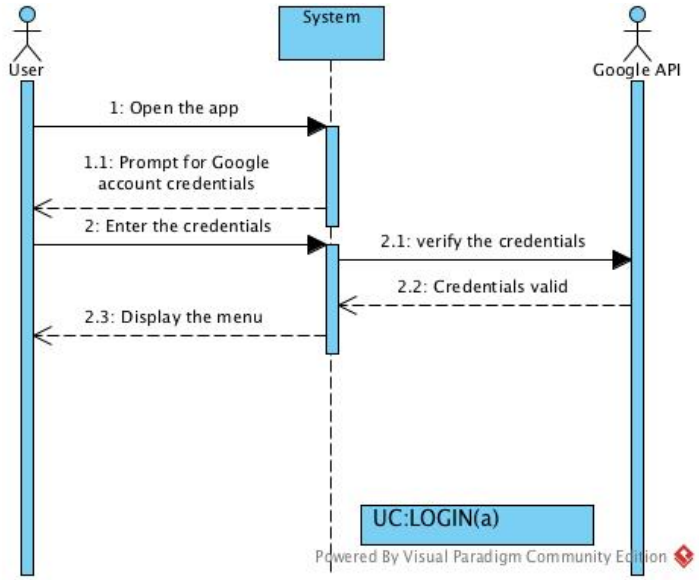


Figure 4: SD:Login(Main Scenario)

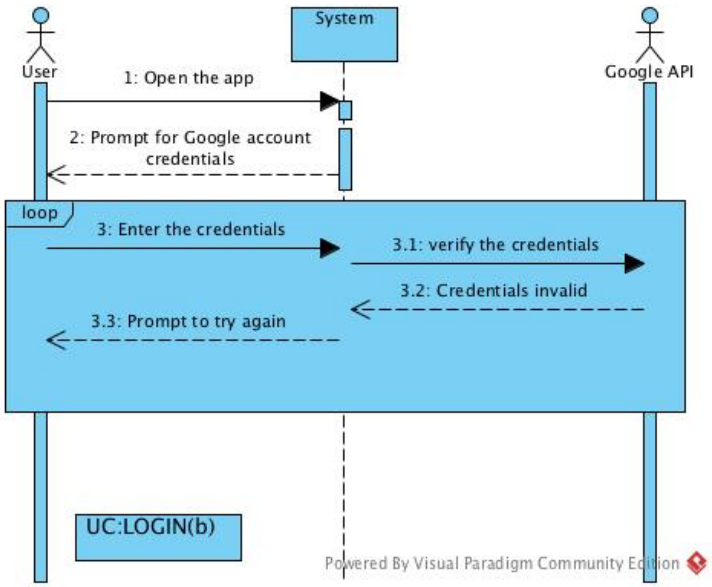


Figure 5: SD:Login(Alternate Scenario)

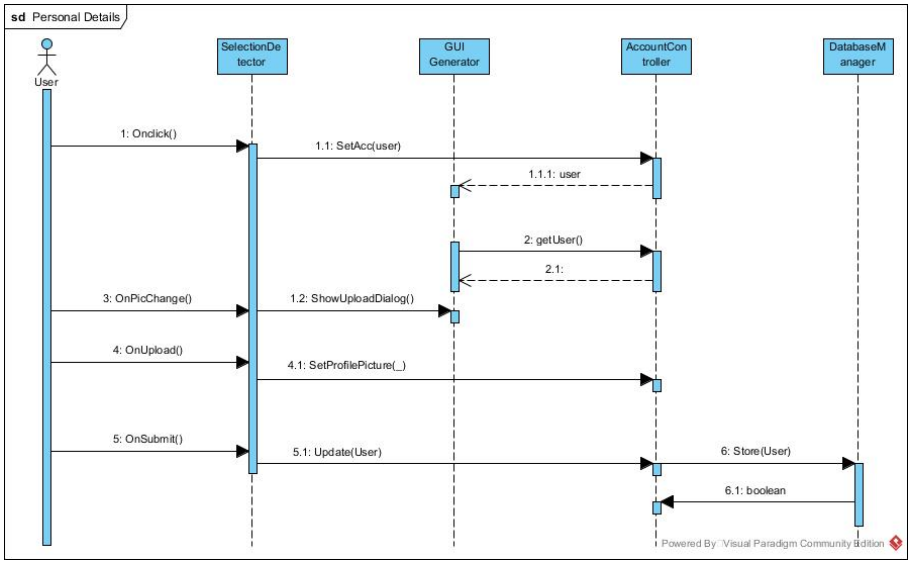


Figure 6: SD:Personal Details

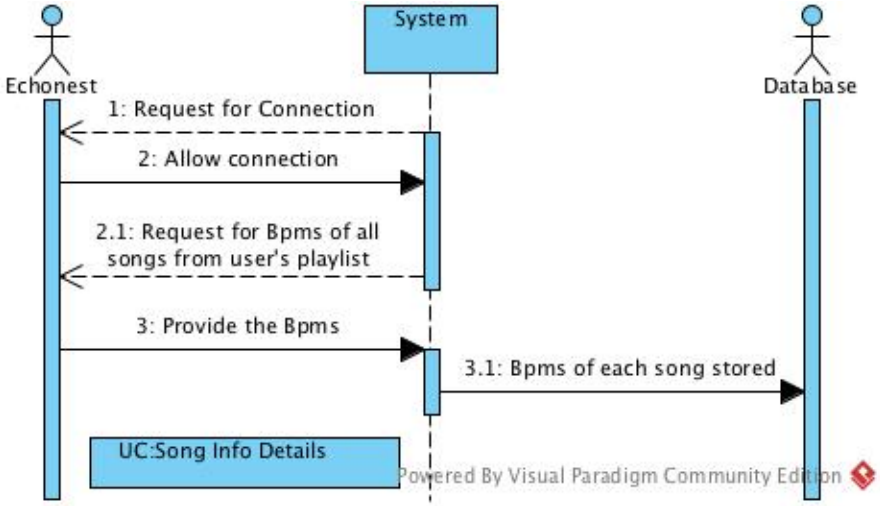


Figure 7: SD:Song Information Details

5 User Interface Specification

The User Interface is the part of the software that interacts with the user. In the instance of a mobile application, this is in the form of a Graphical User Interface based on the Android SDK. Every component of the GUI has to be designed to be intuitive for the user as well as provide ease of use.

The interaction events with the user can be the following for a mobile application:

- **Touch/Click** A screen touch can be used to register the intent of the user to click or activate the GUI component that registers the touch event. If the touch happens on

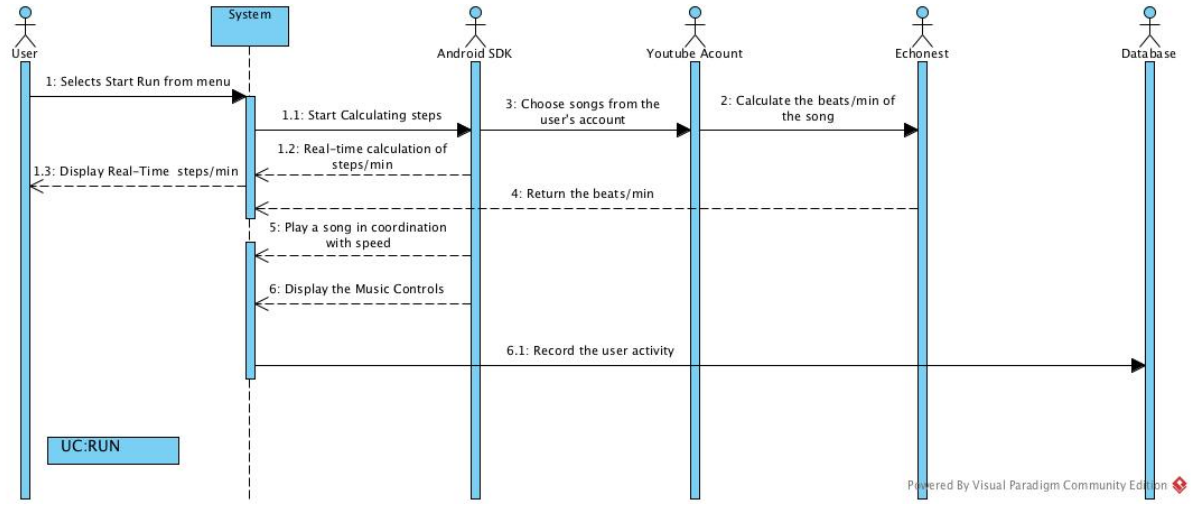


Figure 8: SD:Run

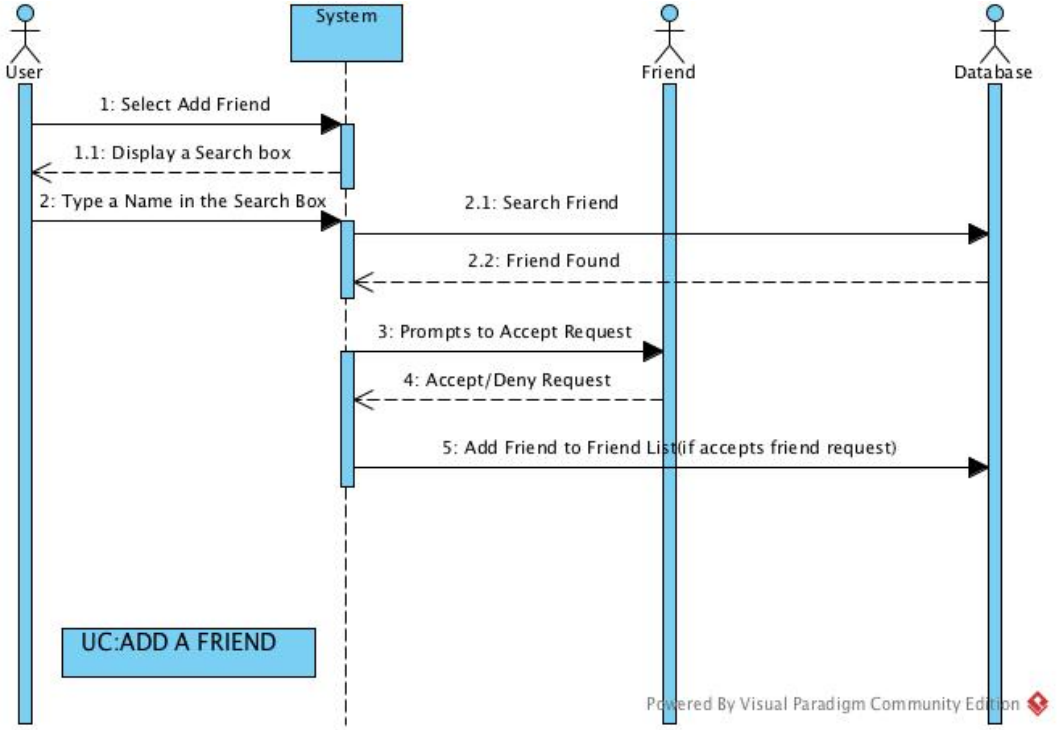


Figure 9: SD:Add a Friend(Main Scenario)

a textbox, the focus of the application switches to the textbox and the contents of the textbox become editable. If the touch happens on a button, the button action is executed.

- **Gesture** Elongated touches can be interpreted as special gestures. Swiping left or right

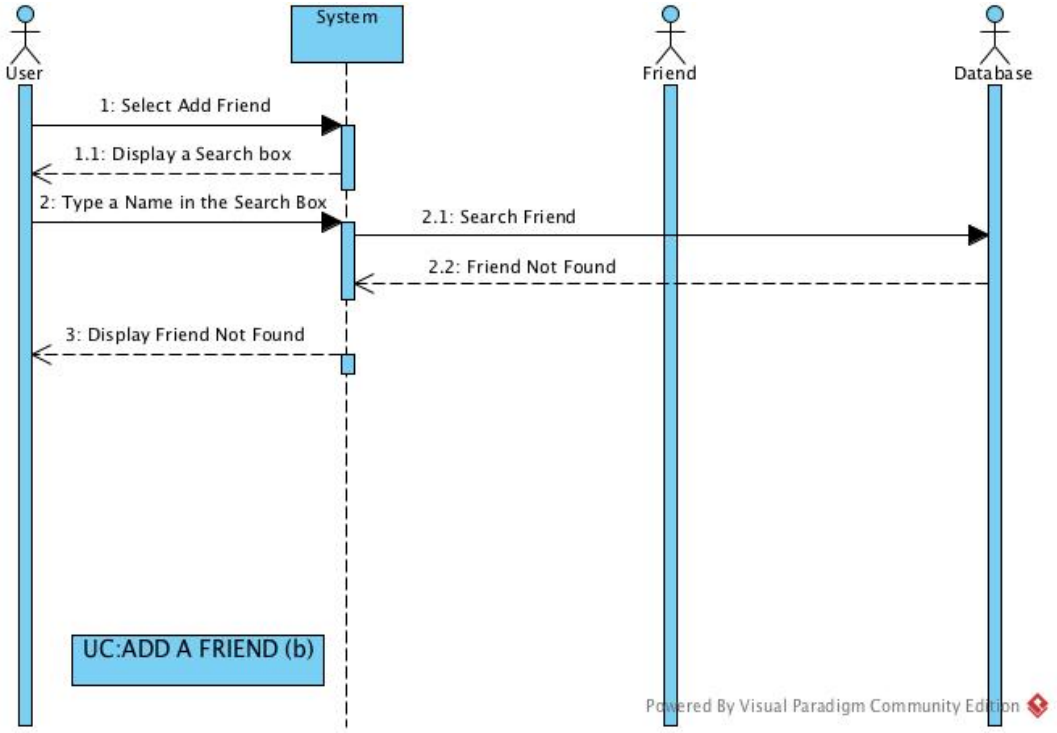


Figure 10: SD:Add a friend(Alternate Scenario)

allows scrolling of horizontal content. Switing up or down allows scrolling vertically.

- **Text Input** Focusing on a textbox allows text input. Text input occurs by typing on the on-screen/physical keyboard of the phone.

Keeping the standard user interactions in mind, the UI attempts to provide maximum, intuitive functionality while minimizing user effort.

5.1 Preliminary Design

The initial design of the GUI has been demonstrated through the use of mockups, Figure 15 and 16.

- **Login:** The top-left in Figure 15 shows the interface to log into the application. The logo is shown in a central and prominent way to draw the user's experience into the app - *Beat.My.Run*. The user can log in using his *Google* credentials. The user presses the *Sign in with Google* button to be either redirected to *Google's* login interface or to be logged in automatically.

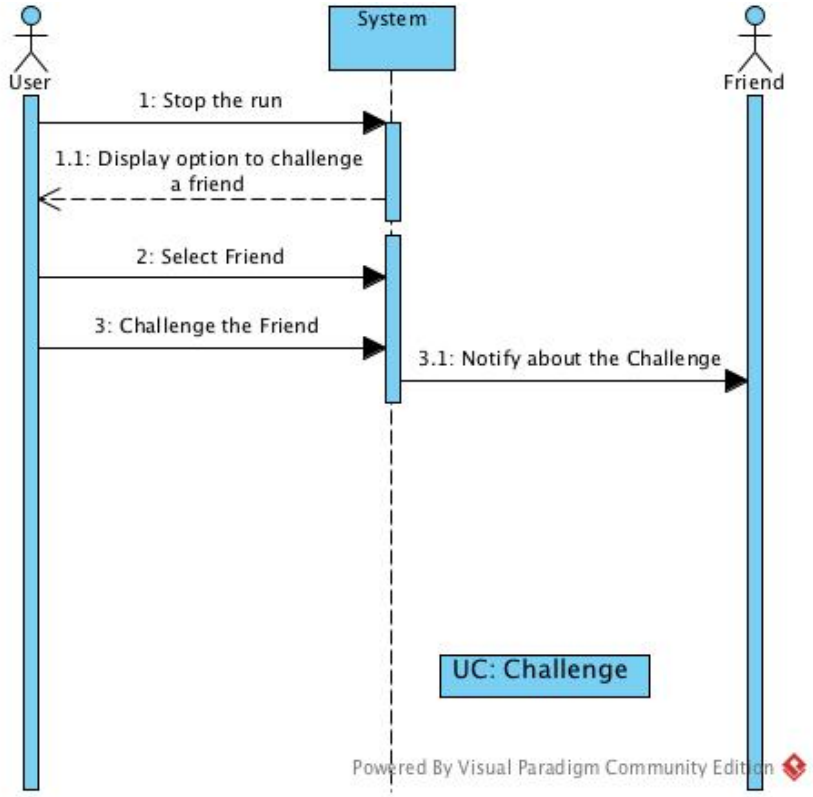


Figure 11: SD:Challenge

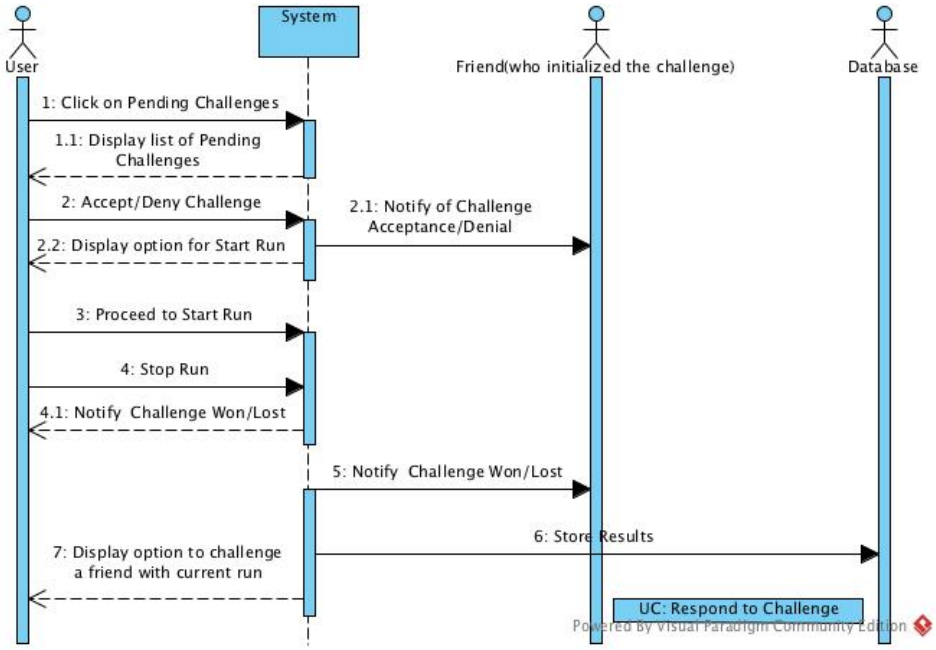


Figure 12: SD:Respond to Challenge

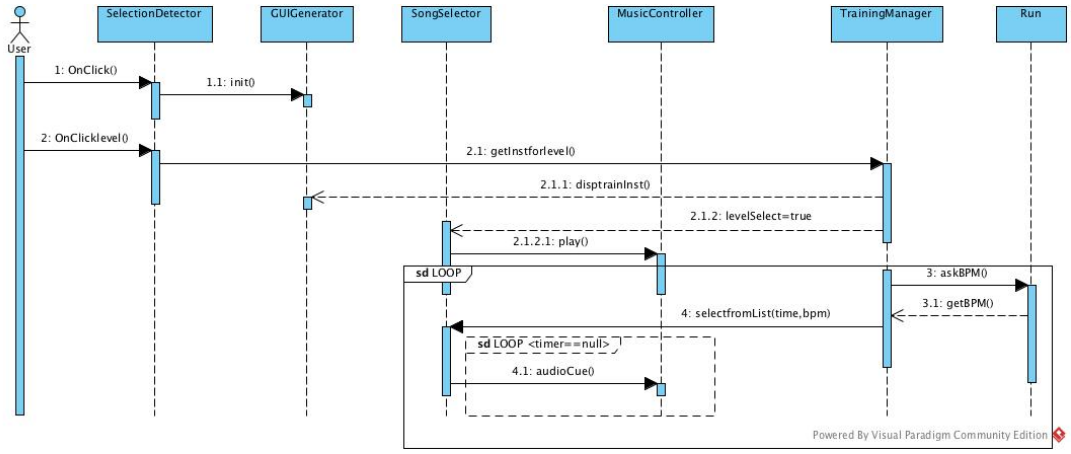


Figure 13: SD:Train

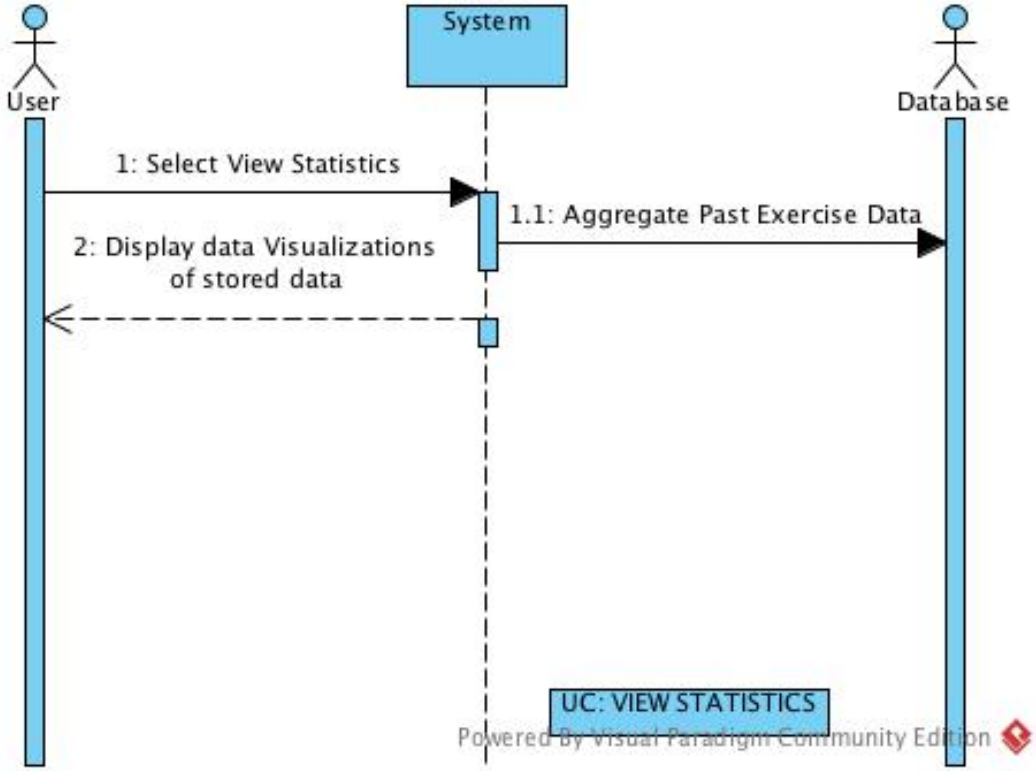


Figure 14: SD:View Statistics

- **Personal Details:** The top-right of the Figure 16 shows the interface for the user to input account details into the application to create the user profile. This screen is shown the very first time the use logs into the app. The picture on the top left can be clicked to upload/change the current profile picture. The *Username* textbox can accept the text input of the user's name. Similarly, the *height*, *weight* and *age* can be entered

into appropriately labeled textboxes. The *location* textbox supports auto-complete to fill in possible location names in the US. By default, this field is populated by the current GPS location. The *Submit* button is pressed to confirm the details and create the account.

- **Main Menu:** The bottom-left of Figure 15 shows the main menu screen. The top of the interface has the picture of the user and the username. Either of these can be clicked to open the *Personal Details UI* Below this is shown the weekly, monthly and all-time statistics of the user. These summaries can be swiped/scrolled left or right, as indicated by the arrows on either side. The clickable *Main Menu* items can be seen below this as *Run, Training* and *Challenge. Challenge* additionally shows the number of pending challenge requests in a small bubble to the right of the text.

- **Starting a Run:** The bottom-right of the Figure 15 shows the interface to start running. The interface shows a summary of the weather information for the current location, obtained from GPS data. The graph shows a hourly forecast and a text warning below it shows precipitation warnings. The *Start* button at the button can be clicked to start the run.

- **Running and Music Playback:** The top-left of the Figure 16 shows the interface the user sees when the user is running. The top two graphics show the distance and calorie count of the current run. These along with the time statistic updates continuously. The time icon can be clicked to pause the run and the clock. The *Stop Run* button can be pressed to stop the current run. The music playback occupies the bottom of the UI. The playback progression is shown on a line. The name of the song, the artist and the album name are shown. At the bottom, the music playback buttons consist of *Previous, Play/Pause,* and *Next.* The buttons perform the standard music playback functions.

- **Run Summary:** The top-right of the Figure 16 is what the user sees once the user presses the *Stop Run* button. The summary of the distance, calories burnt, duration for the current run is shown. The *Challenge A Friend* button can be clicked to challenge a friend to run the same distance in lesser time. Once the user presses the button, the user can enter the name of the friend in a popup UI and challenge the friend. The *Run Summary* UI also shows whether the challenge was successfully completed, if the run was initiated as a part of an incoming challenge.

- **History:** The bottom-left of the Figure 16 shows the historical summary of the user's running statistics. The time period of the statistics can be selected by clicking on the

Past Week button, to change it to *Past Month* or *All Time*. Graphs can show information about the user's steps, distance, time and calories burnt.

- **Challenges:** The bottom-right of the Figure 16 shows the interface for accepting challenges and friend requests. The top graphics shows the statistics of the user's challenges as the success rate and rank among friends. The *Pending Requests* pane shows a list of challenges from friends. Every row shows the picture of the friend, challenge distance and challenge time. Incoming friend requests also come up here. This pane is scrollable by swiping up and down. Every row is clickable to go to the *Starting a Run* interface. The bottom of the UI has a textbox to add a friend by using their username.

- **Training:** The Figure **??** shows the interface for *Training*. The user can select from a set of three predefined training regimens according to intended intensity of the workout. The three categories are *Beginner, Intermediate* and *Advanced*. Every training routine consists of a preset sequence of jogging and sprinting of different durations. The UI shows the level details. Every level row is clickable and takes the user to the Starting a Run interface.

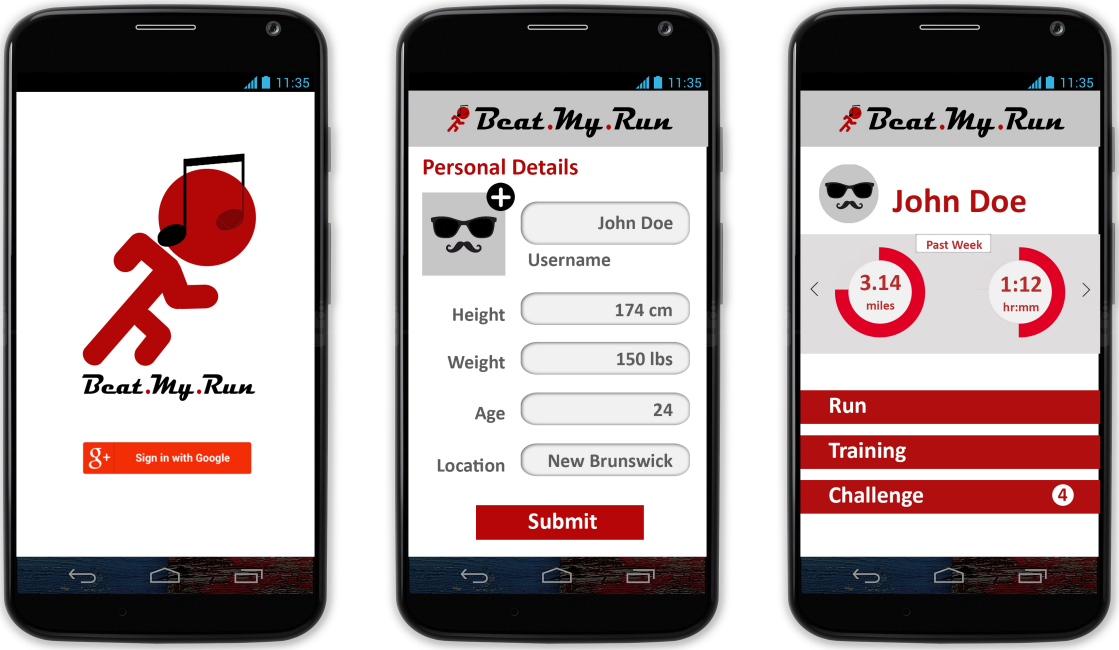


Figure 15: Mock-ups: Login, Personal Details and Main Menu

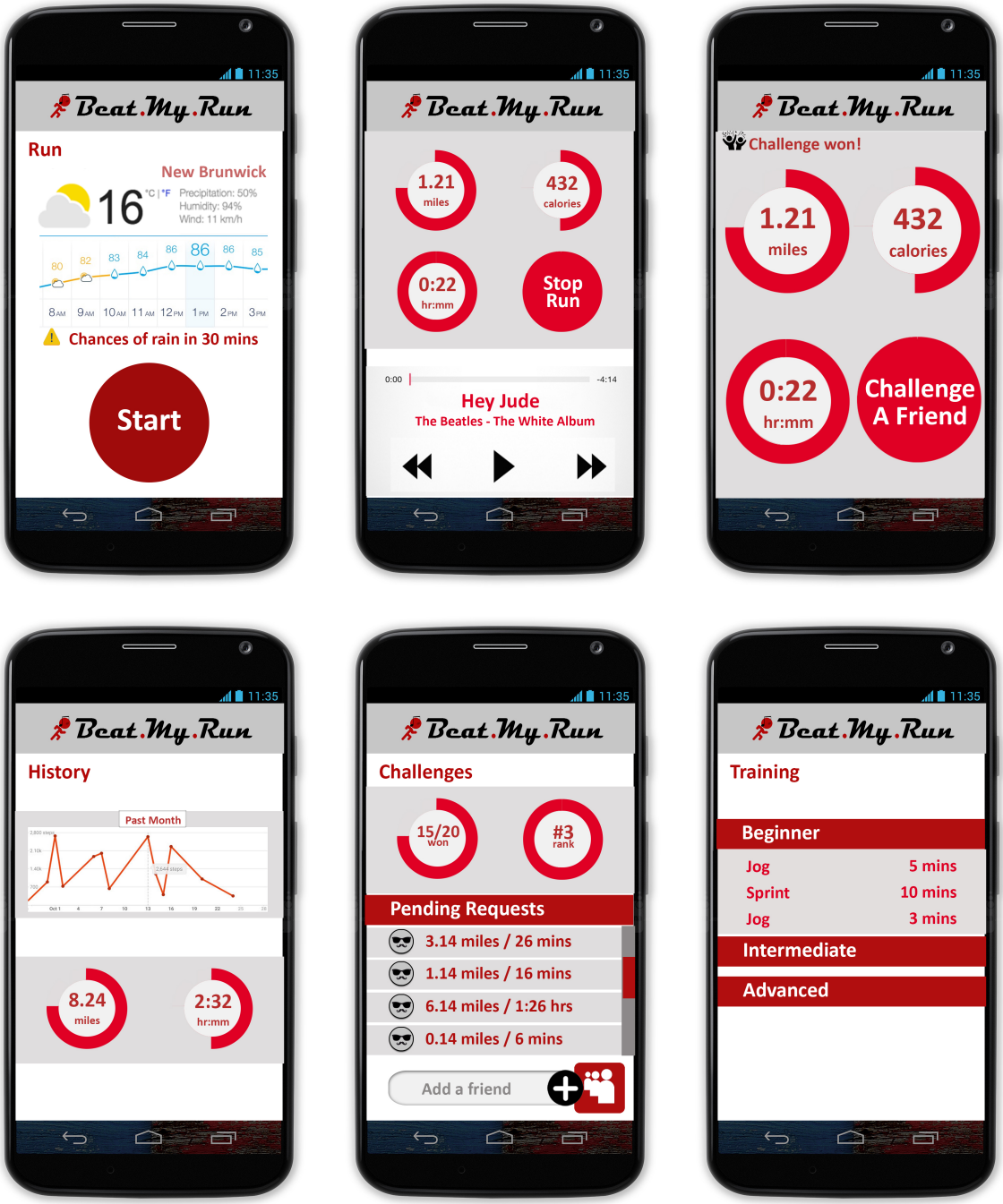


Figure 16: Mock-ups: Starting a Run, Running and Music Playback, Run Summary, History, Challenges and Training

5.2 User Effort Estimation

User-interface estimation refers to the mouse clicks or keystrokes needed to navigate through different windows of the user interface until you reach the appropriate context where you can enter the data. ("Context" roughly corresponds to the window in which the data entry will take place.)

1. **Login**
   (a) *NAVIGATION*: total 2 mouse clicks, as follows
      - Click button "Sign in with google"
      - Click button "Allow access to Beat.My.Run"
2. **Change my personal photo**
   (a) *NAVIGATION*: total 4 mouse clicks, as follows
      - Click on the name link in the main page
      - Click on the "plus" icon next on the image
      - Select a photo from your phone's library
      - Press the "select" button
3. **Fill or change my personal details**
   (a) *NAVIGATION*: total 1 mouse clicks, as follows
      - Click on the name link in the main page
        — after completing data entry as shown below —
      - Press the "submit" button to submit the changes
   (b) *DATA ENTRY*: total 1 mouse click and 11 keys (minimum)
      - Click cursor to Username field
      - Write your username (keys>1)
      - Press the "Tab" key to move to the next text field (âĂIJHeightâĂİ)
      - Write your height (4 keys)
      - Press the "Tab" key to move to the next text field (âĂIJWeightâĂİ)
      - Write your weight (3 keys)
      - Press the "Tab" key to move to the next text field (âĂIJAgeâĂİ)
      - Write your age (2 keys)

- Press the "Tab" key to move to the next text field (âĂIJLocationâĂİ)
- Write your location (keys>1)
- Press the âĂIJselectâĂİ button

4. **Run**

   (a) *NAVIGATION*: total 2 mouse clicks, as follows

   - Click "Run" in the main screen
   - Click button "Start" to start the run

5. **Stop/Resume run**

   (a) *NAVIGATION*: total 1 mouse clicks, as follows

   - Click "Stop Run"/"Resume Run" while in the running screen

6. **Start/resume song**

   (a) *NAVIGATION*: total 1 mouse clicks, as follows

   - Click the play/pause button to play/pause the song while in the running screen

7. **Go to the previous/next song**

   (a) *NAVIGATION*: total 1 mouse clicks, as follows

   - Click the previous/next button to go to the previous/next song while in the running screen

8. **Challenge a friend after finishing the run**

   (a) *NAVIGATION*: total 3 mouse clicks and 3 keystrokes as follows

   - Click the "challenge a friend" button
     — after completing data entry as shown below —
   - Select the name of your friend from the list

   (b) *DATA ENTRY*:

   - Click cursor to search field
   - Write the first three letters of the name of your friends

9. **Challenge a friend**

   (a) *NAVIGATION*: total 3 mouse clicks and 3 keystrokes as follows

   - Click the âĂIJchallengesâĂİ button
     — after completing data entry as shown below —
   - Select the name of your friend from the list retrieved

   (b) *DATA ENTRY*:

   - Click cursor to search field
   - Write the first three letters of the name of your friends

10. **Accept a challenge invitation**

    (a) *NAVIGATION*: total 2 mouse clicks

    - Click the "Challenges" button
    - Select the challenge you want to accept from the list

11. **Training**

    (a) *NAVIGATION*: total 3 mouse clicks

    - Click the "Training" button
    - Select the level of your training
    - Select the "Start" button to start your run

12. **Show my history**

    (a) *NAVIGATION*: total 1 mouse click

    - Click the "history" link in the main screen

5.3 Effort Estimation using Use Case Points

Use Case Points (UCP) [12] is a software estimation technique used to forecast the software size for software development projects.The Use Case Points (UCP) is calculated based on the following formula:

$$UCP = (UUCW + UAW) \times TCF \times ECF$$

where UUCW is the unadjusted use case weight, UAW the unadjusted actor weight, TCF the technical factor and ECF the environmental factor.

Unadjusted Use Case Weight (UUCW)

The UUCW is calculated based on the number and complexity of the use cases for the system. To find the UUCW for our system, each of our use cases must be identified and classified as Simple, Average or Complex based on the number of transactions the use case contains. Each classification has a predefined weight assigned. Once all use cases have been classified as simple, average or complex, the total weight (UUCW) is determine by summing the corresponding weights for each use case. The following chart shows the different classifications of use cases based on the number of transactions and the weight value assigned for each use case within the classification.

Use Case Classification	**No. of Transactions**	**Weight**
Simple	1 to 3 transactions	5
Average	4 to 7 transactions	10
Complex	8 or more transactions	15

According to the uses cases presented before we have the following use cases represented as Complex: Login, Run, Show Challenges, Show Statistics, the following as Average: Getting song information details, Challenge a friend and finally the following as Simple: Personal Details, Logout. Therefore:

$$UUCW = (\text{Num of Simple UCs} \times 5) + (\text{Num of Average UCs} \times 10) + (\text{Num of Complex UCs} \times 15)$$
$$= 2 \times 5 + 2 \times 10 + 4 \times 15 = 90$$

Unadjusted Actor Weight (UAW)

The calculation for UAW is as follows:

$$UAW = (\text{Num of Simple Actors} \times 1) + (\text{Num of Average Actors} \times 2) + (\text{Num of Complex Actors} \times 3)$$

The following chart shows the different classifications of actors and the weight value assigned. For our application we have 3 simple actors, 3 average actors and 1 complex actor. So the UAW calculation becomes:

$$UAW = 3 + 3 \times 2 + 1 \times 3 = 12$$

Technical Complexity Factor (TCF)

Actor Classification	Type of Actor	Weight
Simple	External system that must interact with the system using a well-defined API	1
Average	External system that must interact with the system using standard communication protocols (e.g. TCP/IP, FTP, HTTP, database)	2
Complex	Human actor using a GUI application interface	3

The TCF is calculated by:

$$TCF = 0.6 + TF/100$$

To calculate the ECF, each of the environmental factors is assigned a value based on the team experience level. The diagram below shows the assigned values for our application. The values are multiplied by the weighted values and the total EF is determined.
So for our application:

$$TCF = 0.6 + (40/100) = 1.00$$

Environmental Complexity Factor (ECF)

To calculate the ECF, each of the environmental factors is assigned a value based on the team experience level. The diagram below shows the assigned values for our application.
So the ECF for our application is:

$$ECF = 1.4 + (-0.03xEF) = 0.92$$

Thus, if we put everything together the use case points are:

$$UCP = (UUCW + UAW) \times TCF \times ECF = (90 + 12) \times 1 \times 0.92 = 93.84$$

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
T1	Distributed system	2	5	10
T2	Response time/performance objectives	1	4	4
T3	End-user efficiency	1	5	5
T4	Internal processing complexity	1	4	4
T5	Code reusability	1	5	5
T6	Easy to install	0.5	5	2.5
T7	Easy to use	0.5	5	2.5
T8	Portability to other platforms	2	0	0
T9	System maintenance	1	2	2
T10	Concurrent/parallel processing	1	4	4
T11	Access for third parties	1	0	0
T12	End user training	1	1	1
			Total	40

Factor	Description	Weight	Assigned Value	Weight x Assigned Value
E1	Familiarity with development process	1.5	2	3
E2	Object-oriented experience of team	1	4	4
E3	Lead analyst capability	0.5	1	0.5
E4	Motivation of the team	1	5	5
E5	Stability of requirements	1	5	5
E6	Part-time staff	-1.0	0	0
E7	Difficult programming language	-1.0	3	-3
E8	Application experience	0.5	3	1.5
			Total	16

6 DOMAIN ANALYSIS

6.1 Domain Model

A Domain Model explain meaningful conceptual classes in a problem domain.It represents the real world concepts and not the software components. Domain model relates objects in problem domain to each other. The objects in the domain model are the candidates for programming. The domain analysis of the problem has been done with the help of

- **Concepts:** Objects in the domain
- **Attributes:** Properties of the concept objects
- **Associations:** Relationships between concept objects

The following figure illustrates the complete process of domain analysis[13] in the context of Software Engineering. The information about the domain is captured in the model and can be used and reused effectively.

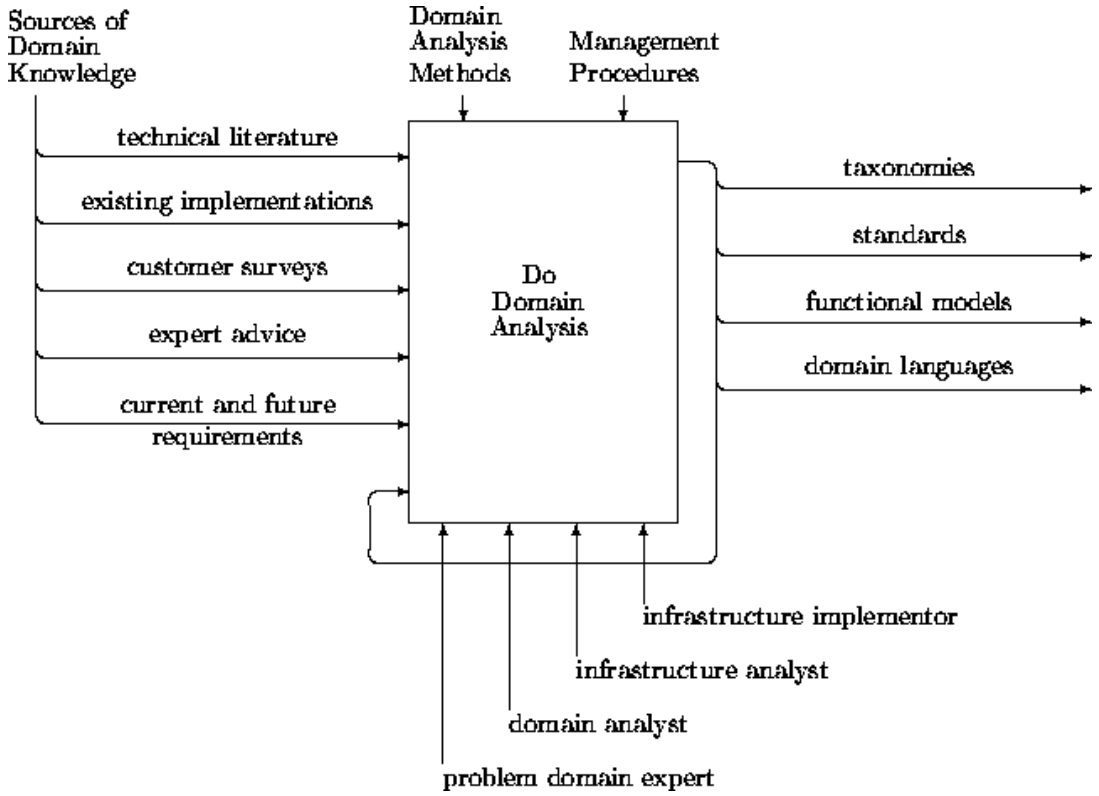


Figure 17: A model of the domain analysis process

6.1.1 Concept Definitions

As we list the Requirements, we figure out the components that work towards fulfilling the requirement. These, components are called concepts.[14]Concepts can be ideas, thing or object in the domain.A class describes a set of objects with the same semantics, properties and behavior.When used for domain modeling, it is a visualization of a real world concept.Concepts can be physical objects,roles and responsibilities of people.We derive domain model concepts and corresponding responsibilities from the formerly defined system use cases.In use case login GUIGenerator is a concept name that is responsible for generating the initial login GUI. They are in the form of noun phrases. The Tables below show the various Concepts Mapped to their Responsibilities. The Type D and K refers to 'Does and 'Knows' which means the D type Concepts are the one which perform tasks and K-type Concepts are the ones which just knows or are notified of the various tasks that happen.

Table 13: Concept Definition of Use Case:Login

Responsibility Description	**Type**	**Concept Name**
Responsible for generating the initial-login GUI	D	GUIGenerator
Detects the button selection	D	SelectionDetector
It connects to the Google API and authenticates the user	D	GoogleAuthenticator
Downloads all useful google account information (user profile details and youtube song preferences)	D	GoogleDataDownloader
It keeps all the google related data of the user (user profile details and youtube song preferences)	K	GoogleAccountKeeper
Responsible for signing out from the app	D	Disconnector

Table 14: Concept Definition of Use Case: Personal Details

Responsibility Description	**Type**	**Concept Name**
Responsible for generating the personal information GUI	D	GUIGenerator
Keeps all the personal details of the user	K	PersonalDetailsKeeper
It detects the submit button selection	D	SelectionDetector
Upload new profile picture	D	ProfilePictureUploader
It passes the user entered information to the PersonalDetailsKeeper and retrieves all the account information from the database	D	AccountController
Stores data from the PersonalDetailsKeeper to the Database	D	DatabaseManager

Table 15: Concept Definition of Use Case: Get Song Information

Responsibility Description	**Type**	**Concept Name**
Establishes the initial API connection	D	EchonestAPIConnector
It will query the EchonestAPI by using the GoogleAccountKeeper to get the song details and write the response in the SongInfoKeeper	D	Controller
Stores the Echonest responses containing the details of all the songs in the user's Youtube preferences	K	SongInfoKeeper
Stores the song information in the SongInfoKeeper in to the database.	D	DatabaseManager

Table 16: Concept Definition of Use Case: Respond to Challenge

Responsibility Description	**Type**	**Concept Name**
Displays the GUI with pending challenges	D	GUIGenerator
Detects accept or reject of the selected challenge	D	SelectionDetector
Container for the collection of open challenges for the user	K	PendingChallengeInfoKeeper
Populates the PendingChallengeInfoKeeper from the database	D	DatabaseManager
Notifies the opponent about a rejected challenge	D	ChallengeNotifier
To scroll through the list of challenges in the UI	D	ScrollDetector

Table 17: Concept Definition of Use Case: View Statistics

Responsibility Description	**Type**	**Concept Name**
Displays the GUI with statistics visualization	D	GUIGenerator
Scrolls through past week, past month and all time statistics visualizations	D	ScrollDetector
Contains the statistics of the user	K	StatisticsKeeper
Retrieve the statistics from the database and populates the StatisticsKeeper	D	DatabaseManager
Depending upon view of past week, past month and all time, it calculates the statistics to display and forwards it to the GUIGenerator to form the visualization.	D	StatisticsCalculator

Table 18: Concept Definition of Use Case: Run

Responsibility Description	**Type**	**Concept Name**
Responsible for generating GUI	D	GUIGenerator
Detects the selection of:The option Run from the menu StopPlay/pause, previous, next in the media player	D	SelectionDetector
Responds to the Selection of Start/Pause/Resume RunStop	D	RunController
Accesses sensors through the AndroidSDK to calculate speed of the user during the run and release access to the sensors on completion of the run.	D	AccelerometerController
Stores the distance run, time and calories burnt for the current run.	K	RunInfoKeeper
Calculates the values in the RunInfoKeeper	D	RunInfoCalculator
Store data gathered after the run by the RunInfoKeeper in the Database and retrieve beats per minutes of songs from Database	D	DatabaseManager
Stores the information about songs in the user preferences from the database. It contains the Song name, artist and other details, as well as BPM of the song.	K	SongInfoKeeper
It selects the proper song by comparing the bpm data of all the songs kept by the SongInfoKeeper and the bpm data of the user held by the RunInfoKeeper. It forwards the selected song to the MusicController.	D	SongSelector
Plays/pauses or goes to next/ previous song during music playback	D	MusicController

Table 19: Concept Definition of Use Case: Train

Responsibility Description	**Type**	ConceptName
Responsible for generating menu GUI	D	GUIGenerator
Detect the selection of the option Train from the menu	D	SelectionDetector
SelectionDetector invokes the TrainingManager of the training level and initializes SongSelector from the run use case and starts the run use case.	D	TrainingManager

Table 20: Concept Definition of Use Case: Send a Challenge

Responsibility Description	**Type**	**Concept Name**
Detects the selection of Challenge option	D	SelectionDetector
Displays a GUI with option to enter friend username	D	GUIGenerator
Searches for the opponent with the username within the Database	D	FriendFinder
It keeps the list of all the user information data to store results from the FriendFinder	K	UserInfoKeeper
Aggregates the RunInfoKeeper, the current user's username and the opponent's username.	K	SendChallengeInfoKeeper
Sends a request to the DatabaseManager to store the SendChallengeInfoKeeper	D	SendChallengeController
Stores the ChallengeInfoKeeper into the Database	D	DatabaseManager
If the run was initiated as a challenge, check outcome from the RunInfoKeeper and invoke ChallengeNotifier	D	ChallengeOutcomeDecider
Notify friend about challenge thrown or the challenge result	D	ChallengeNotifier

6.1.2 Attribute Definitions

An attribute can be defined as description of a named slot of a specified type in a domain class.Each instance of the class separately holds a value.Once the concept definitions are made, each concept performs its Responsibility using various attributes.Its through these various attributes each concept is implemented.Attributes are the required quantities for the concepts to perform their functions.Attributes can be a button or an entry field depending on the concept. Following tables show the Attributes of all the Concepts created previously along with the Attribute Definitions for each use case.

Table 21: Attribute Definition for Use Case : Login

Concept	**Attributes**	**Attribute Description**
GUIGenerator	button	To go to the login credentials
GoogleAuthenticator	username	Google account username
	password	Google account password
GoogleAccountKeeper	email	Email id associated with google account
	list of songs	List of songs from Youtube history

Table 22: Attribute Definition for Use Case : Personal Details

Concept	**Attributes**	**Attribute Description**
GUIGenerator	entry fields	To obtain user information
	profile picture	To obtain profile picture
PersonalDetailsKeeper	user's identity	The user's name, age, height, weight, location and picture
ProfilePictureUploader	profile picture	Needed to upload profile picture

Table 23: Attribute Definition for Use Case : Song Information

Concept	**Attributes**	**Attribute Description**
SongInfoKeeper	listOfSongs	User's Youtube song names
	songAttributes	bpm, duration, tempo, genre

Table 24: Attribute Definition for Use Case: Run

Concept	**Attribute Name**	**Attribute Description**
GUIGenerator	start run button	Starts the run
	pause run button	Pause the run
	music player	Displays music information
	statistics	Running metrics
RunInfoKeeper	run-related data	The user's distance run, time and calories burnt for the current run.
SongInfoKeeper	listOfSongs	User's Youtube song names
SongSelector	selectedSong	The selected song based on the user's tempo
	training level	If in training mode, select bpm sequence

Table 25: Attribute Definition for Use Case : Train

Concept	**Attribute Name**	**Attribute Description**
GUIGenerator	training level buttons	UI for three training level selection
TrainingManager	selected training level	user selected training level

Table 26: Attribute Definition for Use Case : Send a Challenge

Concept	**Attribute Name**	**Attribute Description**
GUIGenerator	Challenge button	Displays challenge button
FriendFinder	search text	Search text used for database query
UserInfoKeeper	list of usernames	List of usernames matching search text
SendChallengeInfoKeeper	current user name	Username of the challenge sender
	opponent username	Username of selected opponent
	challenge details	Distance and time of current run

Table 27: Attribute Definition for Use Case : Respond to Challenges

Concept	**Attribute Name**	**Attribute Description**
GUIGenerator	List of open challenges UI	Scrolling view UI of open challenges
	Accept/Reject icons	Accept and reject options for every open challenge
PendingChallengeInfoKeeper	List of SendChallenge-InfoKeepers	List of open challenges

Table 28: Attribute Definition for Use Case : View Statistics

Concept	**Attribute Name**	**Attribute Description**
GUIGenerator	graphs	Scrollable visualizations for the statistics data
StatisticsKeeper	exercise history information	distance, time and calories burnt per day
StatisticsCalculator	period of time	The period for calculating the statistics, ie. past week, past month or all time.

6.1.3 Association Definitions

Associations describe the relationship between classes.It specify the meaningful and interesting connection between them.There can be more than one association between concepts and it can be unidirectional and bidirectional.It is a link between concept classes which are the fundamental building block for describing relationships in domain model.Associations are created when a concept need to know about other.It can be a input from other concept or even an invoke from a concept.So first all the concept pairs are identified,then the relation between them are noted down.In usecase login SelectionDectector should send request to GoogleAuthenticator and it should authenticate. So we group them together and the Association here is send request.Associations are shown in domain model as a line that connects two concepts with an arrow to specify the direction and the association name.The association definitions ,name and the corresponding concept pairs for each use case is listed below.

Table 29: Association definitions of Use Case: Respond to Challenge

Concept pair	**Association Description**	**Association Name**
GUIGenerator ↔ PendingChallengeInfoKeeper	The GUIGenerator displays all the open challenges taken from the ChallengeInfoKeeper	displays
DatabaseManager ↔ PendingChallengeInfoKeeper	The DatabaseManager populates the PendingChallengeInfoKeeper from the database	populates
SelectionDetector ↔ ChallengeNotifier	The SelectionDetector informs the ChallengeNotifier about what the user has selected (accept/reject)	informs
ScrollDetector ↔ GUIGenerator	The ScrollDetector informs the GUIGenerator to display the rest of the data	informs

Table 30: Association definitions of Use Case: Login

Concept pair	**Association Description**	**Association Name**
SelectionDetector↔GoogleAuthenticator	Send Authentication request and Authenticate	Send request
GoogleDataDownloader ↔ GoogleAccountKeeper	The GoogleDataDownloader populates the data in the GoogleAccountKeeper	Populate the data
GUIGenerator↔Disconnector	The Disconnector has to inform the GUIGenerator that the user signed out so it changes the UI	Inform

Table 31: Association definitions of Use Case: Get Song Information

Concept pair	**Association Description**	**Association Name**
Controller ↔ EchonestAPIConnector	Controller passes the song details from the GoogleAccountKeeper to the EchonestAPI and stores the responses in the SongInfoKeeper	query song info
Controller ↔ DatabaseManager	Store the SongInfoKeeper in the Database	store data

Table 32: Association definitions of Use Case: Train

Concept pair	**Association Description**	**Association Name**
SelectionDetector ↔ TrainingManager	Communicates the training level	invokes
TrainingManager ↔ SongSelector	Calculates bpm sequences for the training and affects song selection logic	provides bpm sequence
TrainingManager ↔ GUIGenerator	Changes to Run GUI	updates

Table 33: Association definitions of Use Case: Run

Concept pair	**Association Description**	**Association Name**
SelectionDetector ↔ MusicController	Starts or stops the Music on selection	plays/stops
SelectionDetector ↔ RunController	Starts or stops run according to selection	starts/stops
RunController ↔ RunInfoCalculator	Calculates the Bpm once run starts	sends run info
RunInfoCalculator ↔ AccelerometerController	Gets the number of steps	gets user steps
RunInfoCalculator ↔ RunInfoKeeper	Calculates the statistics and stores it	calculates
SongInfoKeeper ↔ SongSelector	SongInfoKeeper informs SongSelector about the current user song characteristics	informs
SongSelector ↔ RunInfoKeeper	SongSelector reads the running information	reads
RunController ↔ DatabaseManager	Asks the DatabaseManager to store RunInfoKeeper	ask
SongSelector ↔ MusicController	Selects the next song to play	selects song
RunController ↔ GUIController	Changes the GUI at end of run	changes UI
MusicController ↔ GUIController	Changes the GUI to update music information and control buttons	changes UI
RunInfoKeeper ↔ GUIController	Changes the GUI to update instantaneous statistics	changes UI

Table 34: Association definitions of Use Case: Send Challenge

Concept pair	**Association Description**	**Association Name**
SelectionDetector ↔ GUIGenerator	Presents the challenge option and textbox to search for friends	accept challenge and search friend
SelectionDetector ↔ FriendFinder	Senses the search text for the name of a friend to challenge	find friend
SelectionDetector ↔ SendChallengeController	Sends request to sendChallengeInfoKeeper to store the username of user and opponent along with RunInfoKeeper information	initiate new challenge
FriendFinder ↔ DatabaseManager	FriendFinder requests DataManager to find the username entered into the textbox, within the Database	requests
FriendFinder ↔ UserInfoKeeper	Updates the results of the database query of list of usernames in the UserInfoKeeper	updates
FriendFinder ↔ GUIGenerator	Show the list of usernames matching search text	changes
SendChallengeController ↔ DatabaseManager	Stores the new challenge entry with SendChallengeInfoKeeper	store
ChallengeOutcomeDecider ↔ DatabaseManager	Stores the result of the challenge in database	stores result
SendChallengeController ↔ ChallengeNotifier	Sends notification about new challenge	notify
ChallengeOutcomeDecider ↔ ChallengeNotifier	Sends notification about challenge outcome	notify

Table 35: Association definitions of Use Case: Show/Edit Personal Information

Concept pair	**Association Description**	**Association Name**
SelectionDetector ↔ AccountController	SelectionDetector invokes accountController	invokes
AccountController ↔ DatabaseManager	Passes the PersonalDetailsKeeper to the database	passes data
ProfilePictureUploader ↔ PersonalDetailsKeeper	Passes the profile picture to the PersonalDetailsKeeper	passes picture
PersonalDetailsKeeper ↔ GUIGenerator	Passes the profile picture from PersonalDetailsKeeper to the GUIGenerator	passes picture
SelectionDetector ↔ GUIGenerator	changes the GUI	change UI

Table 36: Association definitions of Use Case: View Statistics

Concept pair	**Association Description**	**Association Name**
ScrollDetector ↔ GUIGenerator	The ScrollDetector informs the GUIGenerator to display the rest of the statistics	scroll up/down
StatisticsCalculator ↔ StatisticKeeper	Calculates the statistics for the given period	calculates
DatabaseManager ↔ StatisticsKeeper	The DatabaseManager populates the StatisticsKeeper	populates
StatisticsCalculator ↔ GUIGenerator	Forwards the data for the generation of visualizations	forwards data

6.1.4 Traceability Matrix

Traceability Matrix is a table that shows the relationship between two elements.Here the usecases are mapped to domain concepts.Priority weight is given to each use case and we identify the concepts that are related to that usecase.The matrix helps to identify wheather all the concepts are covered for a usecase.The priority weight assigned can be referenced during implementation for how work should be planned. The below table show the traceability matrix for concepts and usecase.

Use cases	PW	GUIGenerator	SelectionDetector	GoogleAuthenticator	GoogleDataDownloader	GoogleAccountKeeper	Disconnector	PersonalDetailsKeeper	ProfilePictureUploader	AccountController	DatabaseManager	EchonestAPIConnector	Controller	SongInfoKeeper	RunController	AccelerometerController	RunInfoKeeper
Login	5	✗	✗	✗	✗	✗	✗										
Personal Details	4	✗	✗					✗	✗	✗	✗						
Song detail info	5										✗	✗	✗	✗			
Run	5	✗	✗								✗				✗	✗	✗
Train	3	✗	✗														
Send Challenge	2	✗	✗								✗						
Challenge Respond	2	✗	✗								✗						
View Statistics	1	✗									✗						

Use cases	PW	RunInfoCalculator	SongInfoKeeper	SongSelector	MusicController	TrainingManager	FriendFinder	UserInfoKeeper	SendChallengeInfoKeeper	SendChallengeController	ChallengeOutcomeDecider	ChallengeNotifier	PendingChallengeInfoKeeper	ScrollDetector	StatisticsKeeper	StatisticsCalculator
Login	5															
Personal Details	4															
Song detail info	5															
Run	5	✗	✗	✗	✗											
Train	3					✗										
Send Challenge	2						✗	✗	✗	✗	✗	✗				
Challenge Respond	2											✗	✗	✗		
View Statistics	1													✗	✗	✗

Table 37: Traceability Matrix for Concepts versus Usecases

6.1.5 Domain Model Diagrams

A Domain model is constructed from a set of abstractions pertaining to the general context and knowledge base of the application. It is a representation of the solution in terms of the units defined in the the application domain. Domain model is created from the defined Concepts, Associations and Attributes.

First we identify the concepts in the domain. Concepts names are usually nouns and are written in the top compartment of the class box. Then associations to define relationship between concepts class is identified. Associations are shown as lines with arrows specifying the direction of flow between the box. Attributes necessary for particular concepts is preserved.Attributes are shown in second compartment of the class box. The domain model will not include any software.

The graphical representation of a domain model allows the visualization of the relationships between the different concepts and actors in the domain. The domain model for each usecase is shown below.

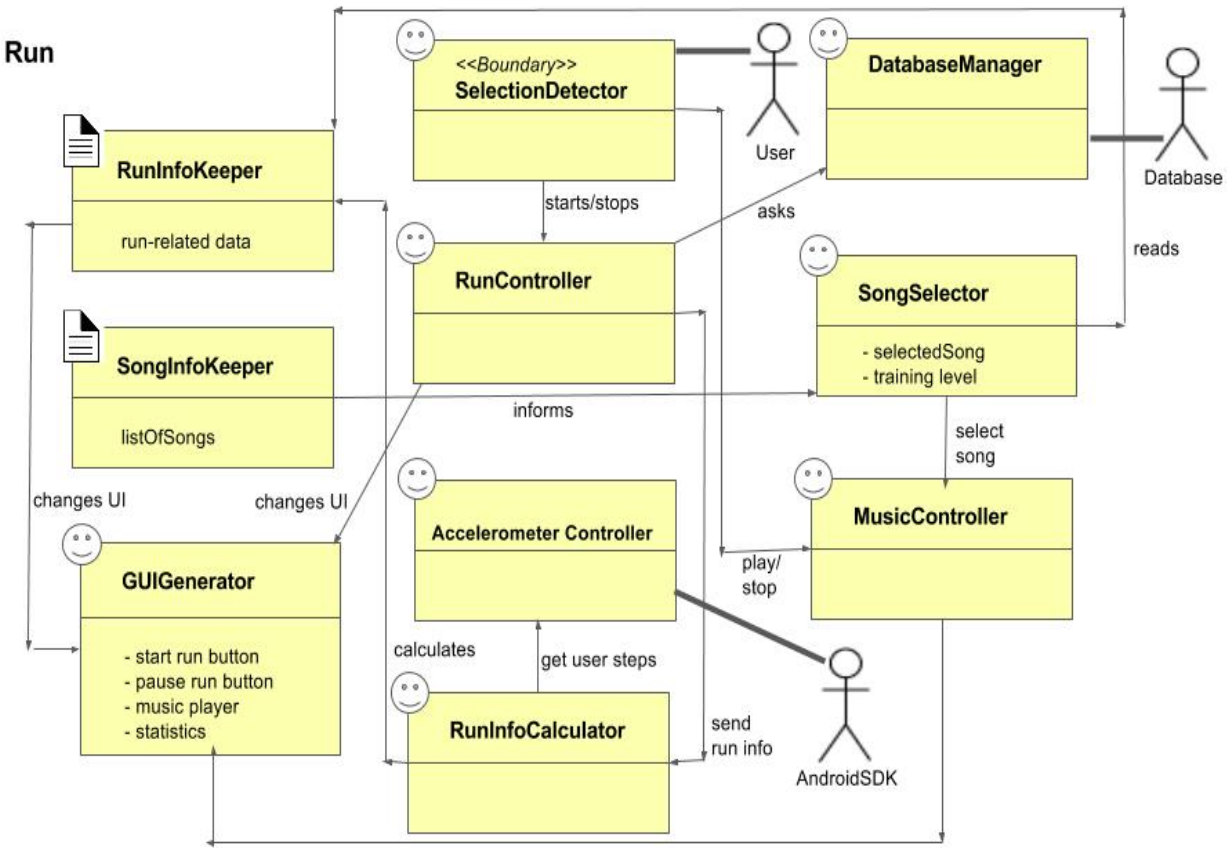


Figure 18: Domain Model:Run

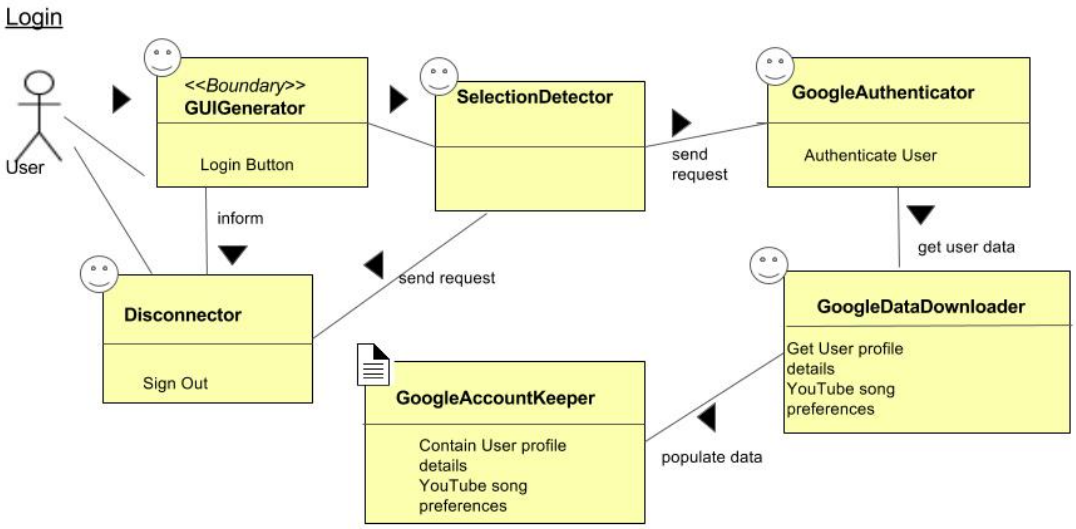


Figure 19: Domain Model:Login

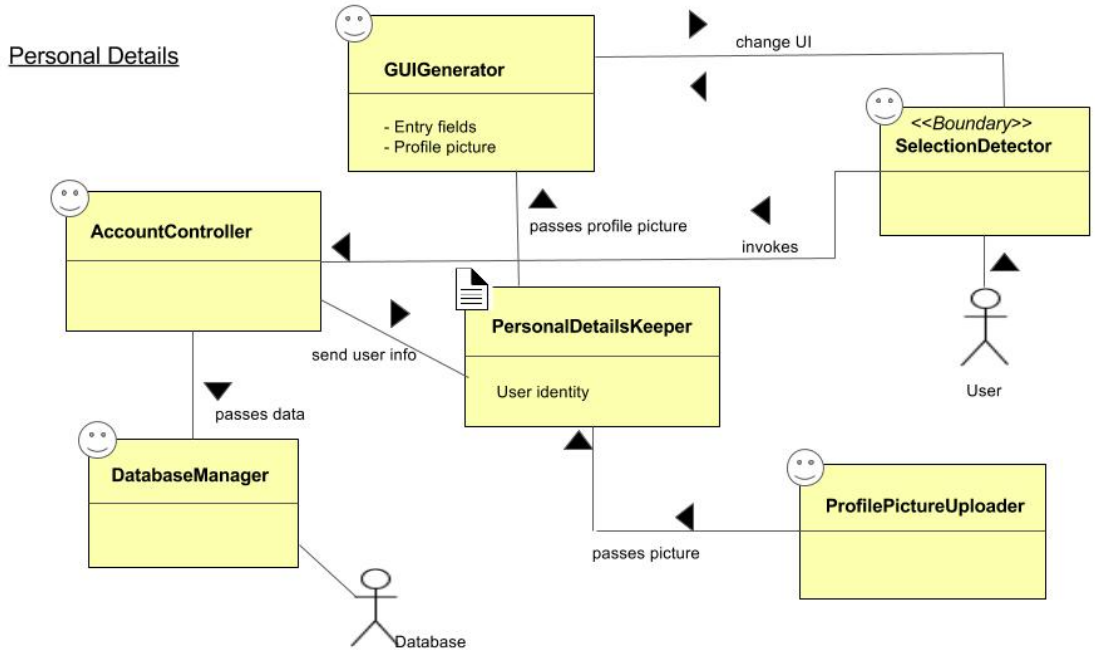


Figure 20: Domain Model:Personal Details

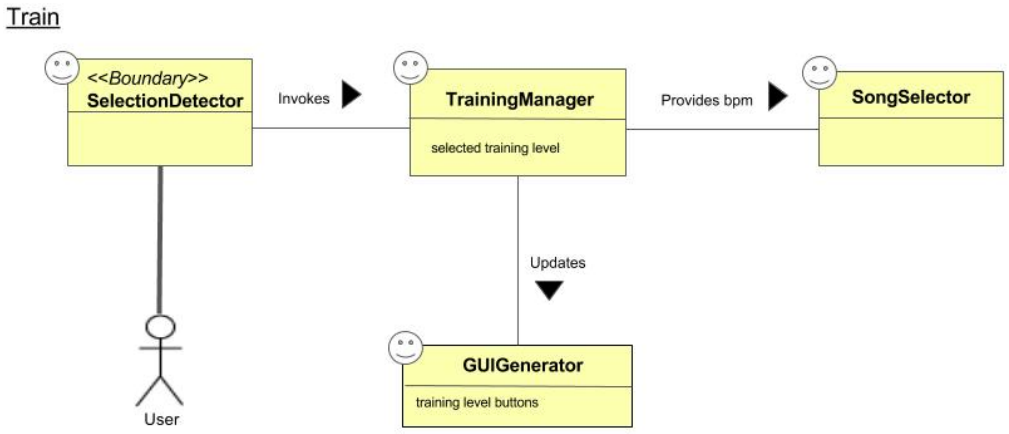


Figure 21: Domain Model:Train

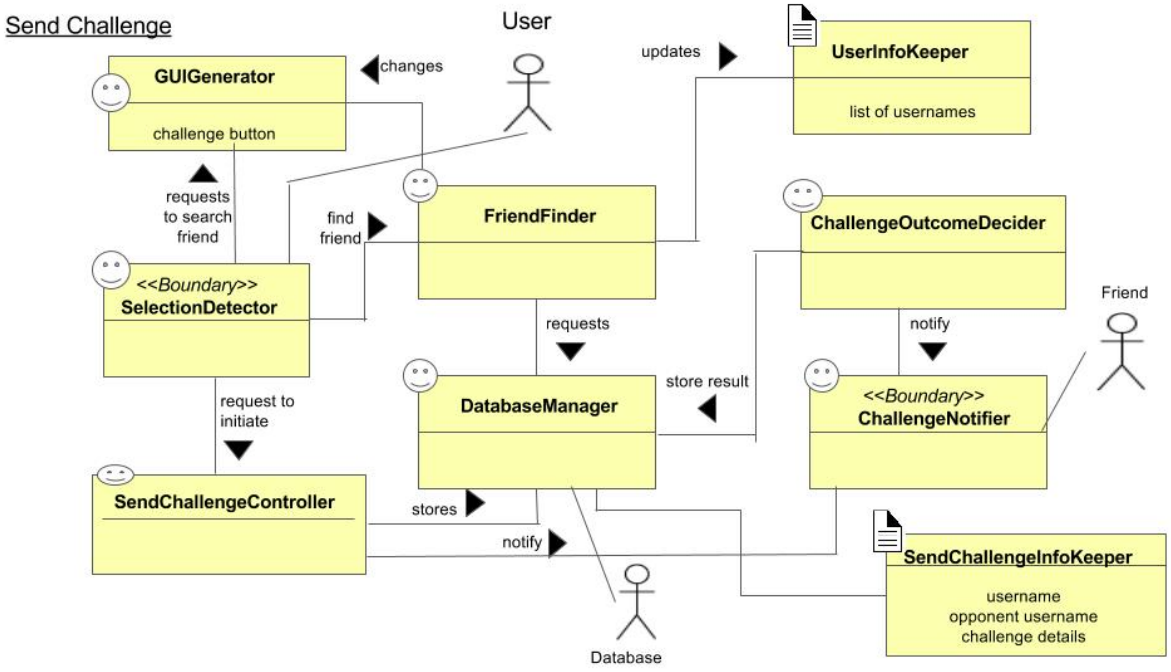


Figure 22: Domain Model:Send Challenge

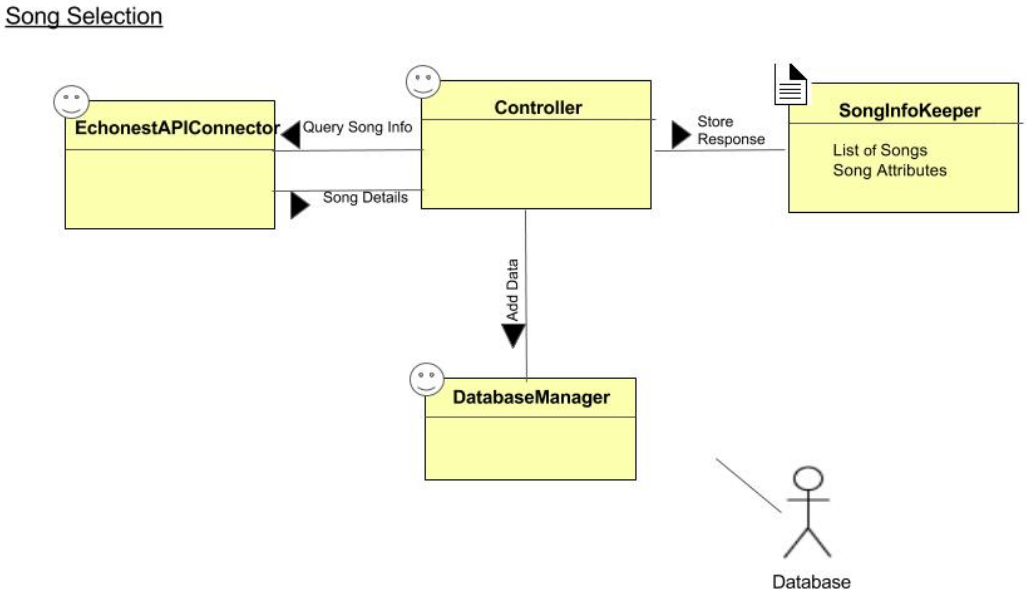


Figure 23: Domain Models:Song Selection

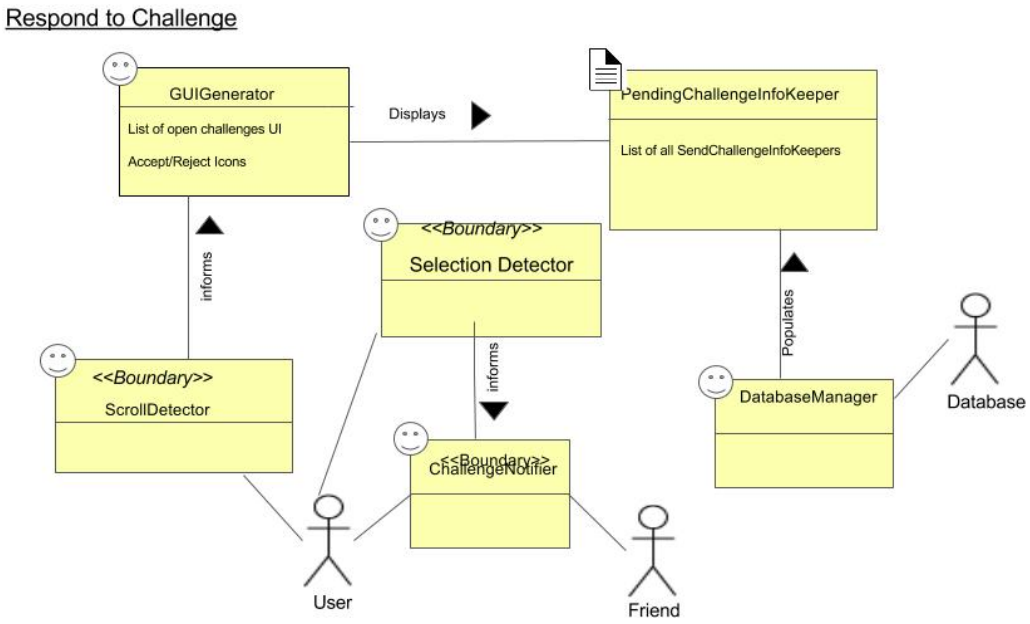


Figure 24: Domain Model:Respond to Challenge

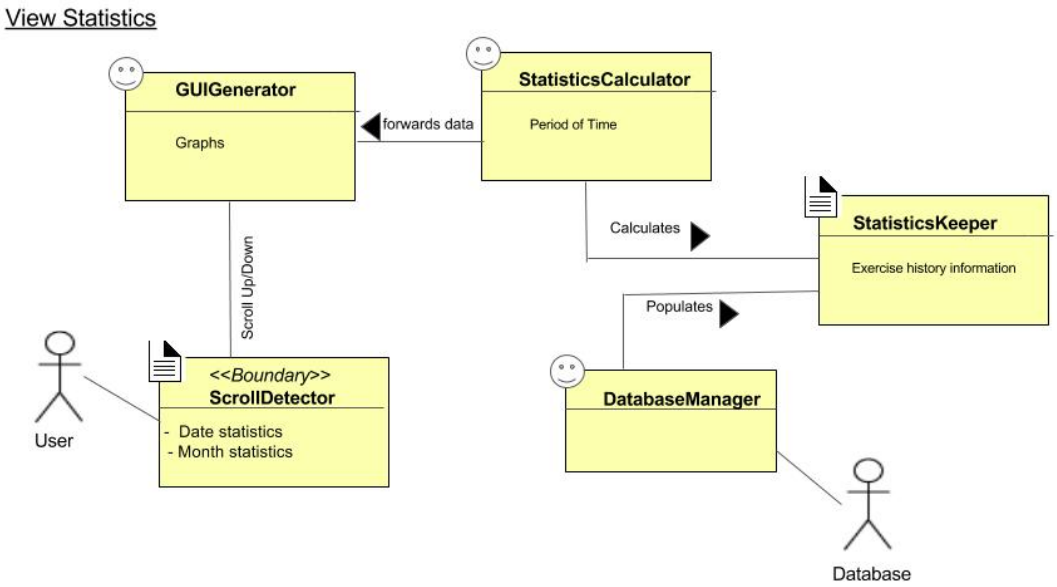


Figure 25: Domain Model:View Statistics

6.2 System Operation Contracts

Operations contracts specify any important conditions about the attributes in the domain model.It specify preconditions and post-conditions for each attributes.While making a operation contract we need to think about the state before the action and after the action.Preconditions can be enter values to a field or an association is formed or closed.Post-condition can be change in state or database updated.Thus post-condition specifies what the system will do for that attribute when all the preconditions are performed. Thus contracts helps to model the behavior of a system.The tables below specify the system operation contract of attributes for each use case.

Table 38: System Operation Contracts for Use Case : Login

Operation	**Login**
Preconditions	button='pressed', username not empty, password not empty
Postconditions	email = 'email of the user' and listOfSongs not empty

Table 39: System Operation Contracts for Use Case : Personal Details

Operation	**Show/Edit personal information**
Preconditions	User's name, age, height, weight and location not empty
Postconditions	Database updated with user details

Table 40: System Operation Contracts for Use Case : Song Information Details

Operation	**Get all the song detail information**
Preconditions	listOfSongs is not empty
Postconditions	songAttributes is not empty

Table 41: System Operation Contracts for Use Case : Run

Operation	**Run**
Preconditions	run button='pressed' and listOfSongs is not empty
Postconditions	run-related data is not empty, selectedSong is not empty and it is playing

Table 42: System Operation Contracts for Use Case : Train

Operation	**Train**
Preconditions	one of the training buttons='pressed'
Postconditions	selected training level is not empty and SongSelector training level = selected training level

Table 43: System Operation Contracts for Use Case : Send Challenge

Operation	**Send Challenge**
Preconditions	SendChallengeInfoKeeper is not empty
Postconditions	Database is updated with new SendChallengeInfoKeeper entry and opponent is notified.

Table 44: System Operation Contracts for Use Case : Respond to a Challenge

Operation	**Respond to a Challenge**
Preconditions	PendingChallengeInfoKeeper is not empty
Postconditions	Run Operation executed or the selected challenge entry is deleted from open challenges database

Table 45: System Operation Contracts for Use Case : View Statistics

Operation	**View Statistics**
Preconditions	exercise history is not empty and period of time is specified
Postconditions	visualization updated with calculated subset of exercise history

6.3 Mathematical Model

The project has many different components that require devoted mathematical models to be implemented. From the characteristics that can be used to analyze running or walking, we choose acceleration as the relevant parameter.The first step in order to find the acceleration of the user is to compute the number of steps he is doing while walking or running. We don't intend to acquire accelerometer data from the user's phone in crude form but rather use an already implemented android software that calculates the number of steps of the user. Thus, after computing the steps parameter, we will use the following algorithm [15] in order to get to get the distance parameter, the speed parameter and the calories parameter.

Distance parameter
The distance parameter which is actually the distance travelled by the user is calculated by the following formula.

$$\text{Distance} = \text{number of steps} \times \text{distance per step} \tag{1}$$

The Distance per step depends on the speed and the height of 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 two seconds. We use the steps counted in every two seconds to judge the current stride length. The following table shows the experimental data used to judge the current stride.

Table 46: Stride as a Function of Speed (steps per 2 s) and Height

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	1.2xHeight

Speed parameter

As we know the speed can be calculated by:

$$\text{Speed} = \text{distance}/\text{time} \tag{2}$$

so in order to get the speed parameter, as steps per 2 s and stride we will use the following:

$$\text{Speed} = \text{steps per 2 s} \times \text{stride}/2\ \text{s} \tag{3}$$

Calories parameter

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 estimate it using a conventional approximation, however. Table 3 shows a typical relationship between calorie expenditure and running speed.

Table 47: Calories Expended vs. Running Speed

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

From this table, we get:

$$\text{Calories (C/kg/h)} = 1.25 \times \text{running speed (km/h)} \tag{4}$$

However because we want m\s the equation becomes:

$$\text{Calories (C/kg/h)} = 1.25 \times \text{speed (m/s)} \times 3600/1000 = 4.5 \times \text{speed (m/s)} \tag{5}$$

The calories parameter would be updated every 2 s with the distance and speed parameters. So, to account for a given athlete's weight, we can convert our last equation as following: Weight (kg) is a user input, and one hour is equal to 1800 2-second intervals.

$$\text{Calories (C/2 s) } 4.5 \times \text{speed} \times \text{weight}/1800 = \text{speed} \times \text{weight}/400 \tag{6}$$

Now 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 the following equation since the caloric expenditure is around 1 C/kg/hour while resting.

$$\text{Calories (C/2 s)} = 1 \times \text{weight}/1800 \tag{7}$$

Finally, we get the total calories by adding the calories for all 2-second intervals.

The selection of which track to play requires a mathematical model as well. This consists of selecting the track with the closest BPM, that is to say minimizing the difference in BPM:

$$\min\left(|target_{BPM} - track_{BPM}|\right) \tag{8}$$

If time permits, this simple model can be replaced with a more complex model incorporating Machine Learning to learn which tracks are more effective than others at changing pulse.

7 INTERACTION DIAGRAMS

Interaction Diagram: Login

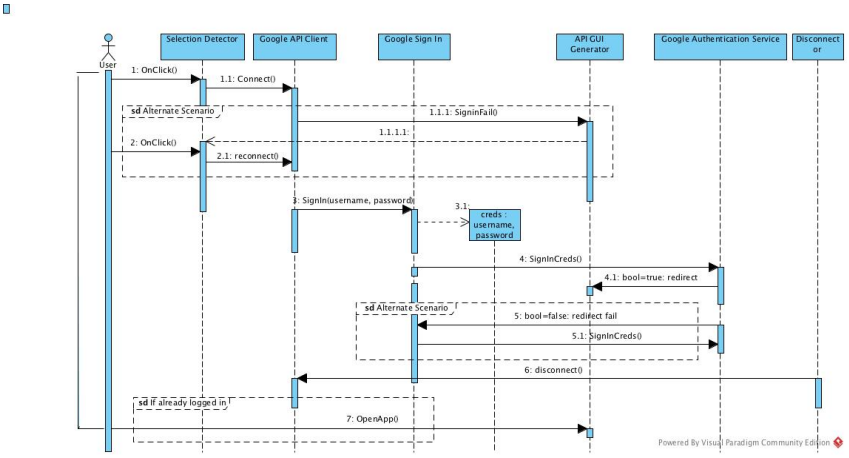


Figure 26: Interaction Diagram: Login

Interaction diagram for use case 1 Login is shown in figure 27. This use case allows the user to login into the system.Both sign in and already signed in case are considered here.First, User clicks the sign in button "onClick()" and the "Selection Detector" identifies this and tries to connect to the "Google API Client" ie, "Connect()" from "Selection Detector" to "Google

API Client".Now alternate scenarios are considered.First,sign in fails and "SignInFail()" is send from "Google API Client" to "APP GUI Generator".This scenario takes flow back to "Selection Detector".Now User again press the sign in button and "Selection Detector" tries to reconnect by "reconnect()" to "Google API Client". Now the regular flow from "Google API Client" continue by providing user name and password "SignIn(uname,pwd)" to "Google Sign In".Now "signIn(creds)" are send to "Google Auth Service" and is stored.when "CredsStoreUserdata()" successfully stores the user details a boolean value true is redirected.If boolean value returned is false alternate scenario is considered."Google Sign In" again send "signIn(creds)" to "Google Auth Service".After sign in details are successfully stored "disconnectcnx()" is send back to "Google API Client" thus closing the session.Now if user has already signed in,while user clicks sign in button "openApp()" command is send to "APP GUI Generator" to continue with the application.This whole flow of events cover the Login Use case.

Interaction Diagram: Personal Details

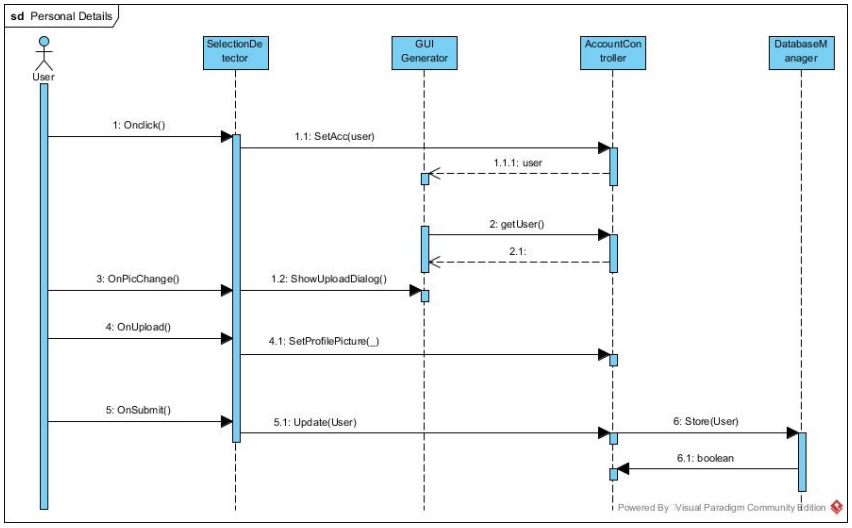


Figure 27: Interaction Diagram: Personal Details

Interaction diagram for Personal Details is shown in figure 28.This use case allows the user to store the personal details into the database.When the "User" enters the personal details "Selection Detector" detects this and send these data "SetAcc(user)" to "Account Controller"."GUI Generator" is updated with the new data.Whenever "getUser()" is requested to "Account Controller" it returns the personal data of user and is displayed in GUI. When "User" what to change picture "OnPicChange()", "Selection Detector" detects this request and send "ShowUploadDialog()" to "GUI Generator". Now upload option is displayed in GUI. When "User" clicks upload button "OnUpload()","Selection Detector" detects this request and send SetProfilePicture() to "Account Controller" . "OnSubmit()","Selection Detector" detects this request and send Update(User) to "Account Controller".The picture is Stored "Store(User)" in "DataBase" and "DataBase Manager" returns a boolean confirmation response

Interaction Diagram: Song Information Details

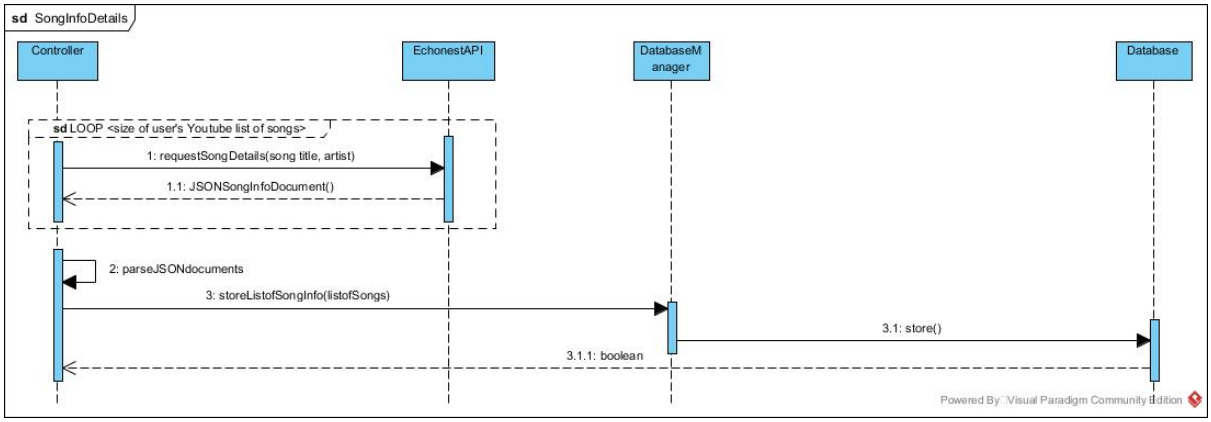


Figure 28: Interaction Diagram: Song Information Details

The Interaction diagram for Song Information Details use case is shown in figure 29 .Song select use case is for obtains the beats per minutes of songs in the userâĂŹs playlist.The selection of songs occur in a loop of events.The list of songs is user's youtube songs."Controller" request "RequestsongDetails(songtitle,artist)" to "Echonest API" which returns back a JSON script "JSONSongInfoDocument()" that contains the song details."Controller" then parse the JSON script "parse JSON document". This information "StoreListofSongs(listofsongs)âĂİ is then send to "Data Manager" which is then stored to "Database" which is retrieved later to play the appropriate song."Controller" is send a Boolean confirmation of successful storage of data in database.

Interaction Diagram: Run

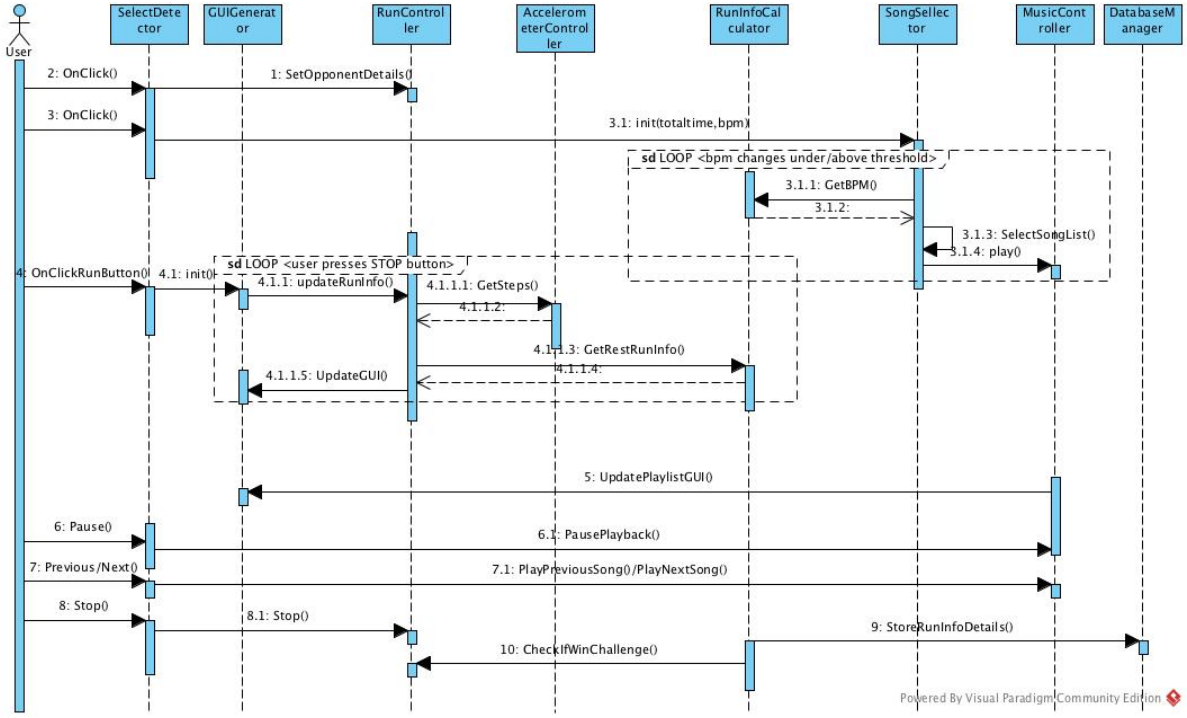


Figure 29: Interaction Diagram: Run

Figure 30 shows the Interaction diagram for Use case Run. Here, the app finds the speed of the user and plays a song with an equivalent beats per minute.Run use case can be selected separately or it can be selected from challenge.While accepting challenge "SetOpponentDetails()" is send to "Run Controller" to set the running limits. First the time and bpm is initialized. Songs are selected and played according to the bpm and this process occur in a loop of events.The loop exit only when bpm value change under or above the threshold value."Song Selector" request "GetBPM()" to "Run Info Calculator" and it returns the bpm value. "Song Selector will get songs according to the bpm value and request "Music Controller" to "play()" the song. When "User" clicks run "OnClickRunButton()" ,"Selection Detector" Send "Init()" to "GUI generator" to show the GUI for run condition. Loop of event occurs till User press stop button.In this loop, "Run Controller" request "GetSteps()" from "Accelerometer Controller" which returns the value back to run controller. "Run Controller" will also request "Run Info Calculator" for "GetRestRunInfo()" and these information are displayed in run GUI. If "User" select "Pause()" or "PlayPreviousSong()/PlayNextSong()" the "Selection Detector" pass the corresponding data to "Music Controller". In case of "User" selecting "Stop()", "Selection Detector" pass "Stop()" to "Run Controller". Now "RunInfoCalculator" checks for challenge result and send the "StoreRunInfoDetails()" to "DatabaseManager" save it in Database.

Interaction Diagram: Train

Figure 31 shows the Interaction diagram of Use Case Train.This Allows the user to select

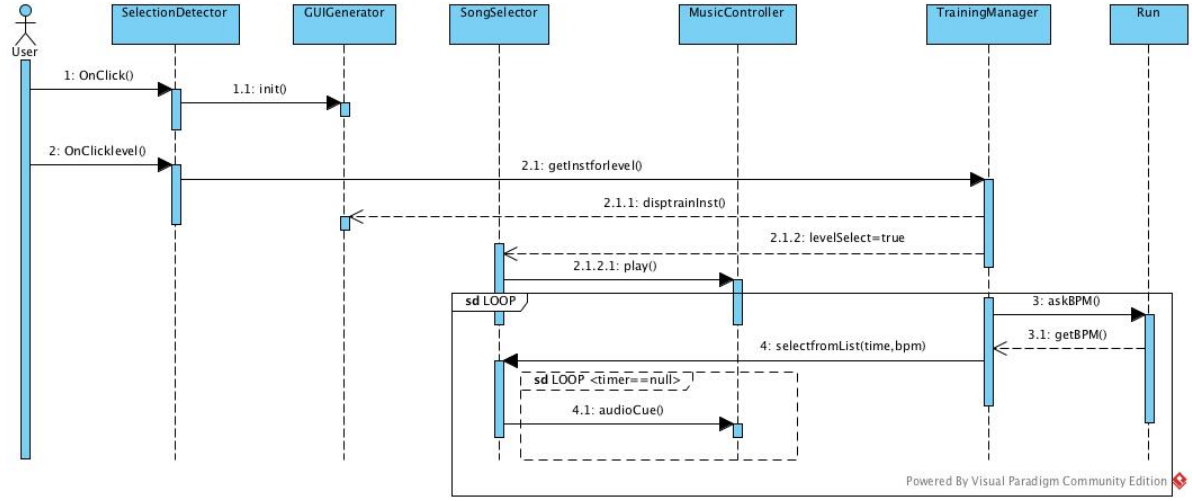


Figure 30: Interaction Diagram: Train

one of the three training levels of the system.First , "OnClick" User selects the Train button and the "Selection Detector" initialize the GUI of train So the flow is "init()" from "Selection Detector" to "GUI Generator".Next,the user selects the training levels(Beginner,Intermediate and Expert).So on "OnClickLevel" by user , selection detector sends "getInstforlevel()" request to "Training Manager" to select the appropriate level of training.Then, "dispTrainInst()" request is send to âĂIJGUI Generator" for displaying corresponding GUI and "Song Selector" is updated with information "levelSelect=True". After that "Music Controller" receives the command to play music from song selector.Flow is "play()" from "Song Selector" to "Music Controller".After this loop of events occur."Training Manager" request "askBpm()" from "Run" which in turn sends the "getBpm()" back to "Training Manager".The details for "getBpm()" are already handled in Run Use case and the "Training Manager" gets the details of BPM in order to select the music.After receiving the bpm details, "Training Manager" passes bpm and time details to "Song Selector" ie, "selectfromlist(time,bpm)"."Song Selector" now based on the information "play()" the song in "Music Player".Each Training level has sub levels like jog or run.To keep track of this a timer is introduced and after each subsection timer will reset and an audio cue is played to notify user about the change.So inside loop "timer" and "audiocue" is included. "timer" is set to null at the start. This loop of activities continue till the training level is completed or until user stop the system.This whole flow of events cover the Train Usecase.

Interaction Diagram: Send Challenge

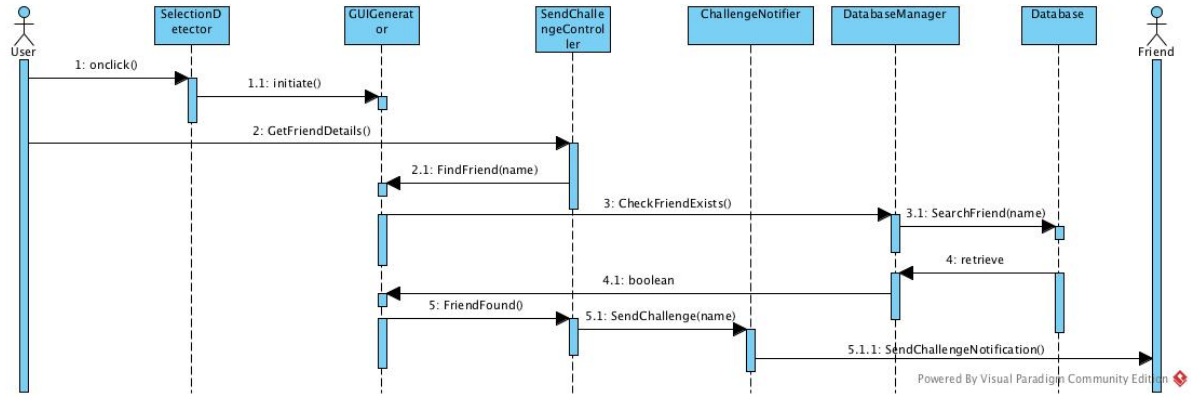


Figure 31: Interaction Diagram: Send Challenge

Figure 32 shows the interaction diagram for Send Challenge Use case.This allows the user to challenge one of his friends or one of the users of the application. Here there are two actors "User" and "Friend"."GUI Generator" is initialized when "User" selects Send a challenge. "User" searches for the friend and "GUI Generator" sends "CheckFriendExists()" to "Database Manager"."SearchFriend(name)" function checks for the name in "Database" and if found retrives it to back to "Database Manager" and "GUI Generator" receives positive response.Now "Send Challenge Controller" notifies "Challenge Notifier" with "SendChallenge(name)".At last "SendChallengeNotification()" is passed to "Friend".

Interaction Diagram: Respond to a Challenge

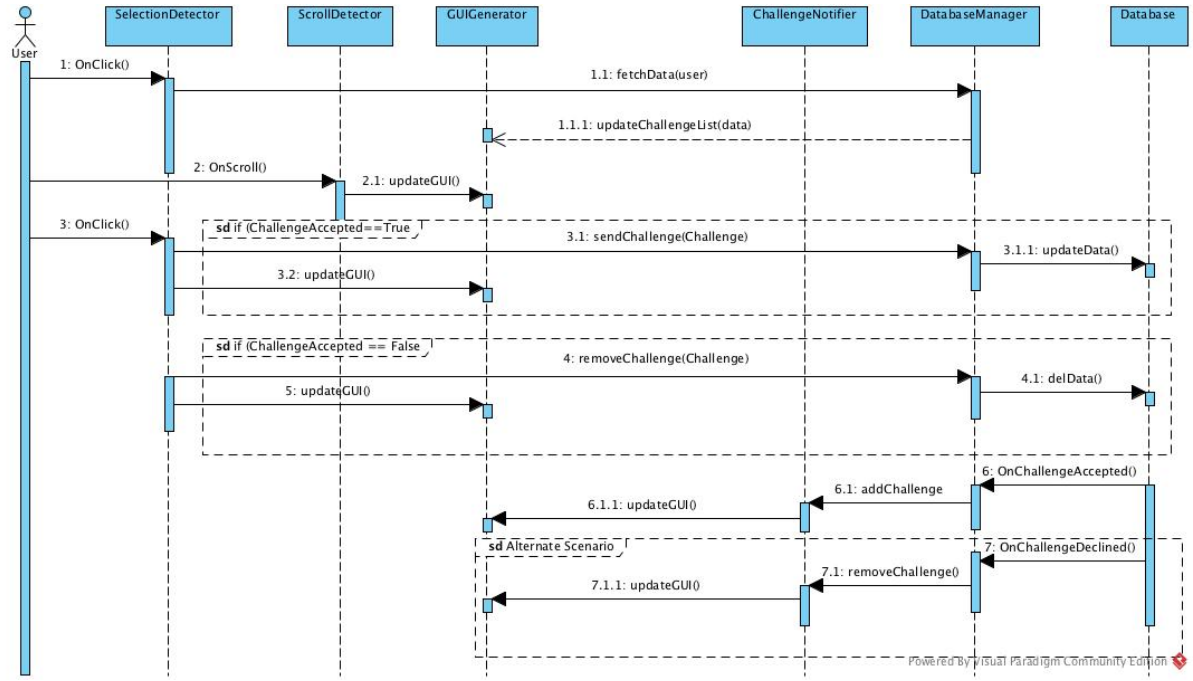


Figure 32: Interaction Diagram: Respond to Challenge

This use case describes how user responds to a challenge."User" clicks "onclick()" the challenge button and "Selection Detector" identifies this and sends "fetchData(user)" to "Database Manager" to fetch data from "Database".Now "Database Manager" request "getData()" to "Database".The data from "Database" is used to "updateChallengeList(data)" in the "GUI Generator".Also if the "User" is Scrolling "onScroll" for statistics for different time period, "Scroll Detector" detects this and ask "updateGUI" the "GUI Generator" to update accordingly.Now there are two scenarios Challege accepted or Rejected.According to the "User" selection "ChallengeAccepted" is updated. If "ChallengeAccepted=TRUE" ,flow of events are "Database Manager" receives "sendchallenge(challenge)" and it updates "updtData()" all the data related to that challenge in "Database".Also "Selection Detector" sends "updateGUI()" to "GUI Generator" to display the corresponding GUI. If "ChallengeAccepted==FALSE" ,flow of events are "Database Manager" receives "removechallenge(challenge)" and it removes "delData()" all the data related to that challenge in "Database".Also "Selection Detector" sends "updateGUI()" to "GUI Generator" to display the corresponding GUI.Now if challenge is added "Database Manager" notifies "addchallenge()" "Challenge Notifier" about the new challenge added to the database."GUI Generator" also receives "updateGUI()" to update the

new challenge information in GUI. In case of "onChallengeDeclined()" "Database Manager" sends "removechallenge()" and notifies "Challenge Notifier" about the challenge removed ."GUI Generator" also receives "updateGUI()" to update the new challenge information in GUI.Figure 33 show the Interaction diagram for Respond to a challenge Use case.

Interaction Diagram: View Statistics

View Statistics is for displaying all the information about the user.First,"User" request the

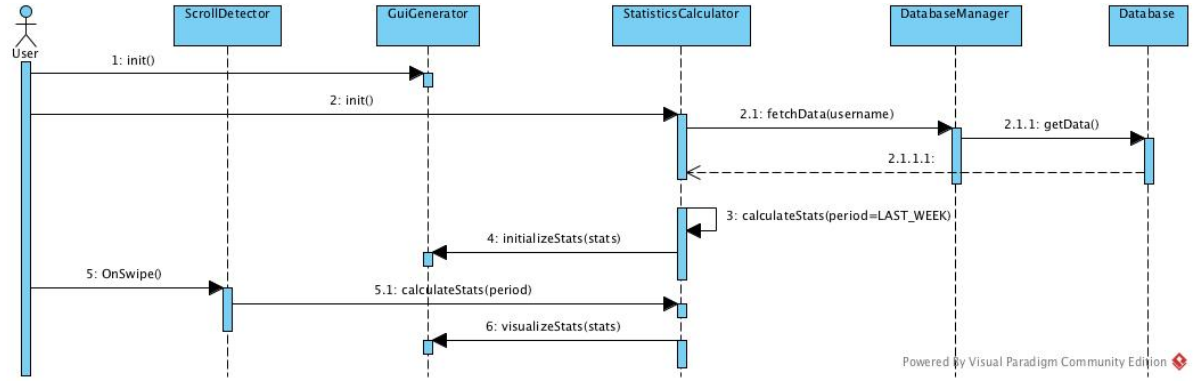


Figure 33: Interaction Diagram: View Statistics

GUI for statistics ie, init() from "User" to "GUI generator"."User" is also initializing "Statistics Calculator" at the same time.An "init()" send to "Statistics Calculator" invokes the "Database Manager" through "fetchData(uname)".Now "Database Manager" sends "getData()" to "Database".Now data is fetched from database and is send to""Statistics Calculator".Here statistics are calculated for the specified period "calculateStats(period=lastweek)" and is then send to "GUI Generator"."VisualizeStats(stats)" from "Statistics Calculator" to GUI Generator" is then displayed in corresponding graphs GUI."User" can also select to see the statistics for a period of time.This is done by "Onswipe()" in "Scroll Detector".Now a particular time period is selected and this is now send to "Statistics Calculator" i.e, "CalculateStats(period)". Now the calculated data is send from "Statistics Calculator" as "VisualizeStats(stats)" to GUI Generator" and is then displayed in corresponding graph GUIs.Figure 34 show the Interaction diagram for Respond to a challenge Use case.

8 CLASS DIAGRAM AND INTERFACE SPECIFICATION

8.1 Class Diagram

Class Diagram 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.The class diagram in Figure 34 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.
    - The SDK also allows access to functional components like *Media Players*.
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.
3. The API calls to external API's would create Java based HTTP requests to the corresponding APIs.The API sends back response for the request which will contain the information requested.

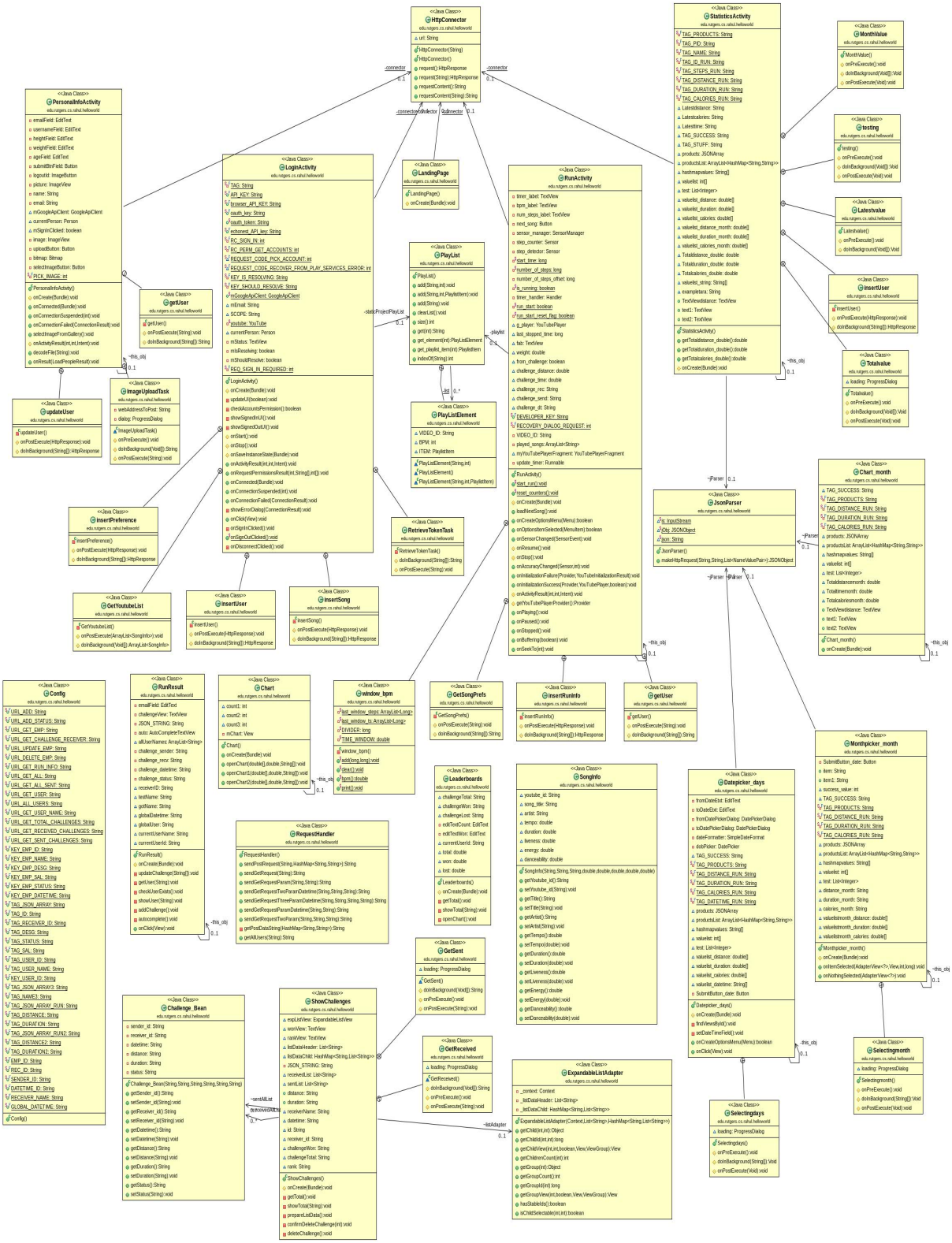


Figure 34: Class Diagram

8.2 Datatype and Operation signatures

The datatypes, operation signatures and the class description has been specified for every class. The information relates the class to the underlying concepts.

Table 48: LoginActivity

Variable/Method	**Desciption**
isSignedIn:bool	To check if user is signed in.
mIsResolving:bool	To check if connection resolution is in progress.
mShouldResolve:bool	To check if connection should resolve automatically when possible.
onCreate():void	This function is used to start an activity and set xml.
updateUI(boolean isSignedin): void	Update UI after sign in
checkAccountsPermission: bool	Authenticates the login credentials
showSignedinUI: void	Show signed in UI
showSignoutUI: void	Show signed in UI
onClick(View v): void	To perform method on button clicked.
onActivityResult(int, int, Intent): void	Manages sign in ctivity.

Table 49: PersonalInfoActivity

Variable/Method	**Description**
emailField: EditText	Gets the email id of the user
usernameField: EditText	Gets the username
submitButton: Image Button	Gets the details on click of button
logoutID: Image Button	Logout the user on click of button
getUser(): User	Gets the details of user
updateUser(): boolean	Checks if updates need to be made to user information
onStart(): void	Establishes connection with the Google API client
onStop(): void	Disconnects the connection from Google API client
onSubmit(): void	Stores all user data on click of button
onChangePicture(): void	Adds or changes profile picture uploaded by user
onResult(LoadPeopleResult): void	Shows result of run activity for the particular user

Table 50: User

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

Table 51: GetYoutubeList

Variable/Method	**Desciption**
doInBackground(Object[]):<PlaylistItem>	To retrieve token
onPostExecute(List<PlaylistItem>): void	Callback function for doInBackground

Table 52: RetrieveTokenTask

Variable/Method	**Desciption**
doInBackground(String[]):String	To retrieve token
onPostExecute(String): void	Callback function for doInBackground

Table 53: SongInfo

Variable/Method	**Description**
youtube_id:String	Contains the YouTubeID needed for the song
title: String	Contains song title
artist: String	Contains song artist
tempo:float	Contains the song tempo
liveliness: float	Contains sond liveliness parameter
duration: float	Contains song duration
energy: float	Contains the song energy parameter
danceability: float	Contains the song danceability parameter
getYoutube_id: String	Gets the YouTube song id
getTitle(): String	Obtains the song title
getArtist(String):void	Obtains the song artist
getTempo(float):void	Obtains the song tempo
setTitle(): String	Sets the song title
setArtist(String):void	Sets the song artist value
setTempo(float):void	Sets the tempo of the song from value

Table 54: PlayList

Variable/Method	**Desciption**
add(String,int): void	Add song to playlist
clearList(): void	Delete all songs from playlist
size(): int	Get size of the playlist
get_element(int): PlayListElement	Obtain VIDEO_ID,BPM from PlayListElement
get_playlist_item(int):PlaylistItem	Obtain PlayList
indexOf(String):int	Obtain index of the song

Table 55: RunActivity

Variable/Method	**Description**
timer_label: TextView	Contains the timer information
bpm_label: TextView	Holds the run information
sensor_manager: SensorManager	Gives the accelerometer readings
number_of_steps: long	Holds the number of steps taken
is_running : bool	To determine if run button is clicked
run_start : bool	Initialized when run is started
listOfSongs LinkedList<SongInfo>	Contains the list of songs
onCreateOptionsMenu(Menu):boolean	Creates the menu for the song list
onOptionsItemSelected(Menu item):boolean	The selected songs is returned
onSensorChanged(Sensor event):void	Detects change in run
onInitializationFailure(Provider, YouTubeInitializationResult):void	To handle YouTube initialization failure
onInitializationSuccess(Provider, YouTubePlayer,boolean):void	Handles succesful initialization of YouTube player
getYouTubePlayerProvider():Provider	To obtain the YouTube player interface
onPlaying():void	Plays the song while run is selected
onPaused():void	Pauses the run, temporarily stores the music and run information
onStopped():void	Stops the run, contains the run information
getSongs():LinkedList>	Get songs from the contained list of songs
chooseSong(LinkedList>):SongInfo	To choose a song from the list of songs

Table 56: DBInterface

Variable/Method	**Desciption**
insertUser(Context): int	Insert Challenged Opponent into database
deleteUser(User):int	Delete Challenged Opponent when challenged is declined
updateChallenge(ChallengeInfo): boolean	Save Challenge info into database
getChallenge(ChallengeInfo): boolean	Get the Challenge info from the database
getMonthlyStats(int):ArrayList<RunInfo>	Get the statistics of past month run activity from in database to display graph
getYearlyStats(int):ArrayList<RunInfo>	Get the statistics of past year run activity from in database to display graph
insertSongDetails(): Boolean	Insert song bpm obtained from Echonest
getSongDetails(int): SongInfo	Get songs based on bpm
getUserPlaylist(int): ArrayList<SongInfo>	Obtain user's playlist to find songs based on bpm
updateChallenge():void	Update Challenge Information
getuserList():<ArrayList>	Get list of users registered on the app
compareTime(SQL Database): User	Compare run information to declared winner of challenge

Table 57: DBHelper

Method	**Description**
DATABASEVERSION:int	The version of the database
DATABASENAME:String	The name of the database
onCreate(SQLite Database):void	To create the Database tables
onUpgrade(SQLite Database, int, int): void	To change the table information

Table 58: ChallengeActivity

Variable/Method	**Desciption**
btnAdd: Button	When clicked, insert name of the opponent
btnGetAll: Button	When clicked, displays names of all opponents
Result: Button	When clicked, displays result of challenges
getChallengeList():ArrayList<ChallengeInfo>	Get method to obtain list of challenges
setChallengeList(ArrayList<ChallengeInfo>: void	Set method to update list of challenges to database

Table 59: ChallengeInfo

Variable/Method	**Desciption**
senderId: String	Id of sender
receiverId:String	Id of receiver
distance: double	Distance covered in Run
duration: double	Duration of Run

Table 60: StatGraphActivity

Variable/Method	Description
runInfo:ArrayList<RunInfo>	Contains the run information
mChart:View	To view the charts
mMonth:String[]	Contains the days of the week
mMonth1: String[]	Contains the months
openChart():void	Creates the chart of run information for previous week
openChartmnth():void	Created chart of run information for the previous month

8.3 Mapping of Classes to Concepts

- **LoginActivity.java** ⇒ *GUIGenerator, SelectionDetector, GoogleAuthenticator, GoogleDataDownloader, GoogleAccountKeeper, Disconnector, EchonestAPIConnector.*
  Login Activity class includes all the specified domain concepts.Concept GUIGenerator is required in this class for displaying the login GUI.Google Authenticator, GoogleDataDownloader, GoogleAccountKeeper concepts are used for accessing the application with Google credentials and for retrieving the user details and preferences.Disconnector concept is required for closing the session and for logging off from application.EchonestAPI Connector is used for accessing all the valid songs from youtube account of user.

- **RunActivity.java** ⇒ *GUIGenerator, SelectionDetector, RunInfoCalculator, Song Selector*
  GUIGenerator and SelectionDetector domain concepts are for displaying the run GUI and for selecting the appropriate options in the GUI. RunInfoCalculator concept calculates the values in RunInfo- Keeper concept.SongSelector concept selects the proper song by comparing the bpm data of all the songs kept by the SongInfoKeeper and the bpm data of the user held by the RunInfoKeeper.

- **SongInfoActivity.java** ⇒ *SongInfoKeeper*
  SongInfoKeeper concept is used in this class for Storing the information about songs in the user preferences from the database. It contains the Song name, artist and other details, as well as BPM of the song.

- **PlaylistActivity.java** ⇒*SongSelector*
  SongSelector concept is required for creating a user preferred playlist.

- **ChallengeActivity.java** ⇒*SelectionDetector, GUIGenerator, FriendFinder, ChallengeOutcomeDecider, ChallengeNotifier , ScrollDetector*
  GUIGenerator and SelectionDetector domain concepts are for displaying the challenge GUI and for selecting the appropriate options in the GUI.Friendfinder concept Searches for the opponent with the user- name within the Database.If the run was initiated as a challenge, ChallengeOutcomeDecider check outcome from the RunInfoKeeper and invoke ChallengeNotifier.ChallengeNotifier then notify friend about challenge thrown or the challenge result.

- **DBInterfaceActivity.java** ⇒*DatabaseManager*
  Database Manager concept does the interfacing activity from different class to Database.This is an important concept for storing the data.

- **StatActivity.java** ⇒ *StatisticsCalculator ,ScrollDetector, GUIGenerator*
  GUIGenerator and ScrollDetector domain concepts are for displaying the statistics GUI. Depending upon view of past week, past month and all time, StatisticsCalculator concept calculates the statistics to display and forwards it to the GUIGenerator to form the visualization.

- **PersonalInfoActivity.java** ⇒ *GUIGenerator, PersonalDetailsKeeper, SelectionDetector, ProfilePictureUploader*
  PersonalDetailsKeeper concept is required in this class for storing the user details.This concept stores all the personal details of the user.GUIGenerator displays the personal information GUI.SelectionDetector is required for detecting all the selections made by user.Uploading new profile picture is done by ProfilePictureUploader and this concept is used in this class for changing the profile pictures.

- **RunInfoActivity.java** ⇒*RunInfoKeeper*
  RunInfoActivity stores the distance run, time and calories burnt for the current run.This concept is the data keeper for run activity.

- **ChallengeInfoActivity.java** ⇒ *SendChallengeInfoKeeper, PendingChallengeInfoKeeper*
  ChallengeInfoActivity class has different keeper concepts for storing data related to challenge.SendChallengeInfoKeeper aggregates the RunInfoKeeper data, the current user's username and the opponent's username.Pending ChallengeInfoKeeper is a container for the collection of open challenges for the user.

- **UserActivity.java** ⇒*UserInfoKeeper*
  It keeps the list of all the user information data to store results from the FriendFinder.

- **PlaylistElementActivity.java** ⇒*SongSelector*
  SongSelector gets the songs and creates the playlist.

- **RetrieveTokenTaskActivity.java** ⇒ *GoogleDataDownloader, GoogleAuthenticator*
  In this class the connection to user Google account is established.This happens during login activity. GoogleDataDownloader Downloads all useful Google account information (user profile details and Youtube song preferences).GoogleAuthenticator initiate Google API connection and authenticates the user.

- **GetYoutubeListActivity.java** ⇒ *GoogleDataDownloader*
  GoogleDataDownloader concept is used in this class to get the songs user has liked to

create the playlist.This concept downloads the user details from google account during login activity.

- **StatGraphActivity.java** ⇒*StatisticsCalculator*
  StatisticsCalculator will calculate the data from StatisticsKeeper and plots the graph.

8.4 Design Patterns

In general, Android SDK does not target any particular design pattern. In the beginning of our development process, we had a to make a choice between Model-View-Controller (MVC) and Model-View-Presenter(MVP) pattern.

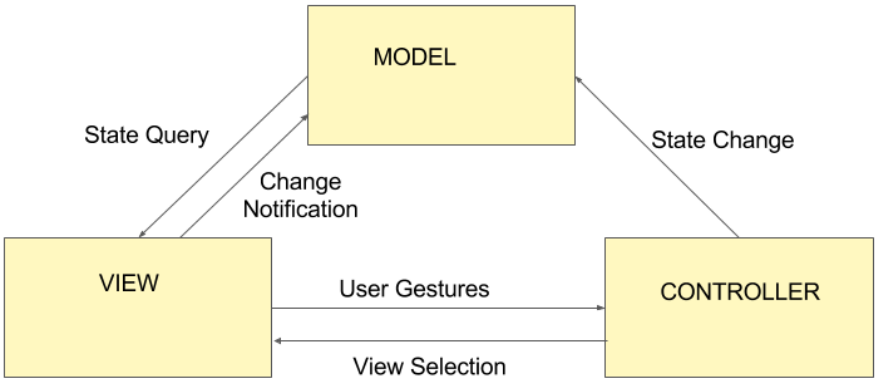


Figure 35: Model-View-Controller Architectural Pattern

In the MVC pattern, the Model represents all the underlying data, the View manipulates this data for from the Model and renders it for display on the screen and the Controller manages the interaction of the user with the View to handle events that cause the data in the model to change. But, in android, the responsibilities of the Controller and View cannot be completely distinguished. Also, the MVC pattern is not preferred in scenarios where the user interactions leading to multiple changes in views with increasing complexity are significant and hence, should be given more control. Accepting these factors, we decided begin our development with the Model-View- Presenter architectural pattern.

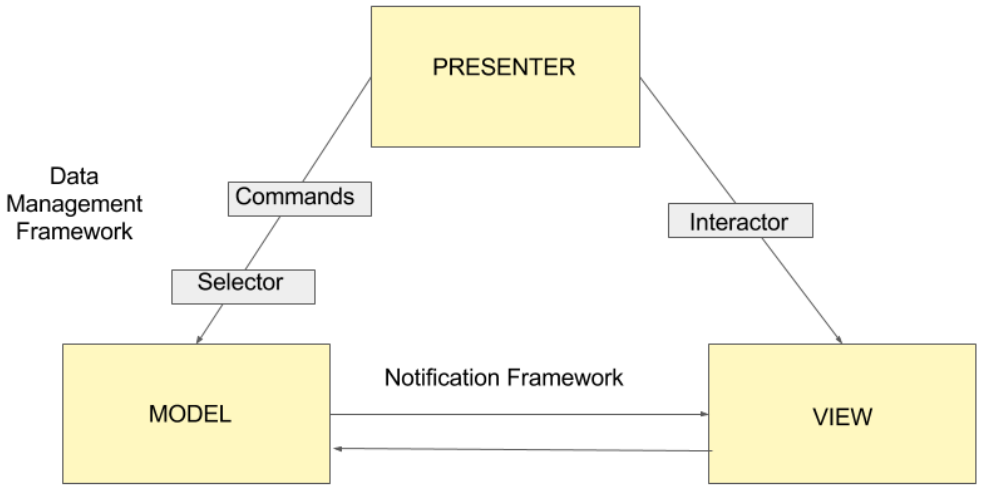


Figure 36: Model-View-Presenter Architectural Pattern

Unlike in MVC, here the View is not directly triggered by the Controller. The user provides input gestures to the view. The View translates the input into an event that is sent to the Presenter. The Presenter coordinates the calls to change the model. The Model changes and returns its values to the Presenter. The Presenter returns the values to the View and the View decides how to display data. There maybe multiple views and all views do not need to be visual. The Interactor resolves how different events map to changes in data. These events consist of the user-initiated actions like mouse movements and clicks, keyboard keystrokes or some touch gestures. In most interactive applications, there is a particularly extensive set of touch interactors like menu selections, drag and drop, button and check box clicks, and drawable gesture operations. The Command decides how the data is changed and the selector resolves how the data should be specified. In Model-View-Presenter, the role of the Presenter is to interpret the events and gestures initiated by the user and provide the business logic that maps them onto the appropriate commands. This interpretation is then used to manipulate the data in the Model in the expected way.

8.5 Object Constraint Language (OCL) Contracts

context PersonalInfoActivity
inv: self. weight >= 0 and self. height >= 0 and self. age >= 0

context RunActivity
inv: self. number_of_steps >= 0

context RunActivity
inv: LoginActivity. mGoogleApiClient.isConnected()==true

context RunActivity
inv: if (self.from_challenge==true) then
self.challenge_distance <> null and self.challenge_time <> null and
self.challenge_rec <> null and self.challenge_send <> null and
self.challenge_dt <> null and
self.challenge_rec = LoginActivity.currentPerson.getId()
endif

context RunActivity.window_bpm::add(in timestamp:DateTime, in steps:Long): void
pre: self.last_window_ts->forAll(ts | ts < timestamp)
self.last_window_steps->forAll(st | st < steps)
post: self.last_window_ts->forAll(ts |
if timestampts>30seconds then
last_window_ts.remove(ts) and
last_window_st.remove(st)
endif)

context RunActivity.window_bpm::bpm(): void
pre: self.last_window_ts.size()>0
self.last_window_steps.size()>0
post: bpm = (self.last_window_steps.max() - self.last_window_steps.min())/
(self.last_window_ts.max() - self.last_window_ts.min())

context StatisticsActivity
inv: LoginActivity. mGoogleApiClient.isConnected()==true

context ShowChallenges
inv: LoginActivity. mGoogleApiClient.isConnected()==true

context ShowChallenges::acceptChallenge(in challenge:ChallengeBean): void
pre: self.receiver_id=LoginActivity.currentPerson.getId()
post: intent = RunActivity

context RunResult::sendChallenge(in challenge:ChallengeBean): void
inv: self.sender_id=LoginActivity.currentPerson.getId()

context RunResult::winChallenge(in challenge:ChallengeBean): void
pre: ChallengeBean.receiver_id=LoginActivity.currentPerson.getId()
inv: (ChallengeBean.distance/ChallengeBean.duration)<(self.distance/self.duration)

context RunResult::loseChallenge(in challenge:ChallengeBean): void
pre: ChallengeBean.receiver_id=LoginActivity.currentPerson.getId()
inv: (ChallengeBean.distance/ChallengeBean.duration)>=(self.distance/self.duration)

context LoginActivity
inv: if self.mGoogleApiClient.isConnected()==true then
self.currentPerson<>null
endif

context LoginActivity
inv: if self.currentPerson<>null then
self.currentPerson.getAccountName()<>null and
self.currentPerson.getDisplayName()<>null and
self.currentPerson.getId()<>null endif

context StatisticsActivity inv: fromDate<=toDate

9 SYSTEM ARCHITECTURE AND SYSTEM DESIGN

9.1 Architectural Style

The App uses a client server architectural model wherein one or more client devices requests information from a server device. The server typically responds with the requested information. The initial step is logging into the application which requires communication with the Google API followed by continuous interactions with the database server to get most of the information related to the run parameters and song selections. The challenges are also posted to friends using information stored in database. The client/server architectural style describes this kind of relationship between a client and one or more servers. The work flow is divided and since the client when connected to a server over a network sends repeated requests for challenging a friend and other database interactions this type of architecture is suitable. The user interface interacts with the user to authenticate, collect, store and retrieve information. The various subsystems are Accelerometer, Mobile Interface, Server and Database. The major advantage of this type of architecture would be centralization, proper management, scalability and accessibility.

9.2 Identifying Subsystems

The application system which works as a whole consists of many sub systems that work with each other to coordinate using well-defined protocols in order to help the application work. The diagram below shows all the components of the system. A brief explanation of the role of each of the component and how they take part in the application is as follows:

The client communicates with the server by HTTP Protocol to access the Database on the Host.

The Application has to communicate with the Google API for accessing the Google Account of the user with a specific Key. It also communicates with the Youtube API in a similar way to access the Youtube account of the User in order to retreive the song list of the user.

Also, to calculate the BPM of the songs it communicates with the Echonest API so the the BPM can be compared wit the steps of the User and the songs can be chosen accordingly from the list.

The Accelerometer is used here to calculate the steps of the user while the user exercises.

A Dynamic Interface is designed in Android studio which gives the User the facility to use the various features of the App and makes it easy.

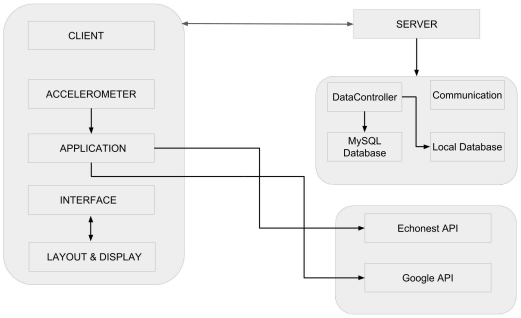


Figure 37: Subsystems of our application

9.3 Mapping subsystems to Hardware

Server : Personal computer is used as hardware for server subsystem .
Client : Smart phones are used as hardware.The application runs in user smartphone.
Accelerometer : Phone accelerometer is the hardware used for determining the steps of user.

9.4 Persistent Data Storage

Considering that our system is an Android application and can't get access to MySQL, our database system is built on SQLite, which is an extremely featured, convenient as well as simple database. Also, taking into account that our application will be used by many users, database is required to store all of their personal information (personal details, all their running history, their music preferences etc). Thus, the data is separated into five parts which are User, RunInfo, Challenges, SongInfo, MusicPreference. These data are stored and are persistent on the Android phone.Thus, the user gets logged in automatically, the next time he opens the App.Also he can view his activity history as teh database persists time and along. The User relation contains information about the user, such as name, email, age, weight, height, image etc. The SongInfo relation keeps detailed information about the songs that the user has liked on youtube or about his playlists. This information is taken from the Echonest API and it is the title, artist, tempo, energy, danceability, etc. The correlation of who user has liked which song is stored in the MusicPreference relation (this way, we won't have duplicates of songs and user ids in the SongInfo relation). In addition, the RunInfo relation aggregates all the information of the completed runs of the user, such as distance, duration, calories etc. The full database schema of our application is depicted in Fig. 38.

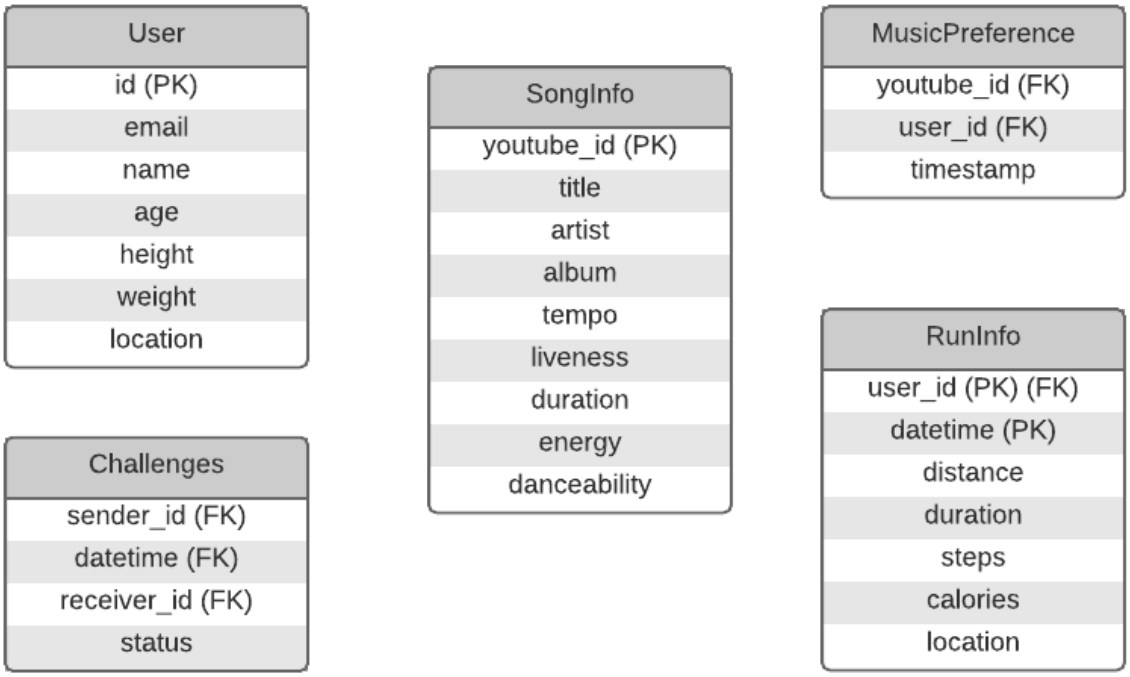


Figure 38: Database Schema

9.5 Network Protocol

Protocols are a set of rules in which computers communicate with each other. The protocol says what part of the conversation comes at which time. It also says how to end the communication. Thus, when we have an application distributed in multi computers, there should be a way for them to connect and coordinate by communicating. Here we have different components which use the different protocols: The Controller talks to the APIs with the HTTP Protocol like: The System makes a request to the Google and the Youtube API with the HTTP protocol. Also, to communicate with the Echonest,HTTP protocol is used. To store and retrieve the data from the database, the system communicates with the SQLite database using queries to create, update and retrieve the data. Also, the system communicates with the Android SDK to use the Sensors and the Timer. Thus a protocol is required to connect with sensors through the SDK.

The protocols keep the different systems of the application well-synchronised.

9.6 Global Control Flow

- Execution order: The system has major parts which are event-driven system which means it waits in a loop for events, and every user can generate the timers in our system.
- The system of event-response type forms part of the current application.

9.7 Hardware Requirements

The application requires a smartphone running Android 4 platform (Kitkat) or higher. The smartphone should have a built-in Accelerometer, so that the steps of the user can be retrieved through the Anddroid SDK.The screen resolution should be high that 480x854 pixels. The hardrive 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.

10 Algorithm and Data Structures

10.1 Algorithm

The project has many different components that require devoted mathematical models to be implemented. From the characteristics that can be used to analyze running or walking, we choose acceleration as the relevant parameter.The first step in order to find the acceleration of the user is to compute the number of steps he is doing while walking or running. We don't intend to acquire accelerometer data from the user's phone in crude form but rather use an already implemented android software that calculates the number of steps of the user. Thus, after computing the steps parameter, we will use the following algorithm [15] in order to get to get the distance parameter, the speed parameter and the calories parameter.

Distance parameter
The distance parameter which is actually the distance travelled by the user is calculated by the following formula.

$$\text{Distance} = \text{number of steps} \times \text{distance per step} \tag{9}$$

The Distance per step depends on the speed and the height of 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 two seconds. We use the steps counted in every two seconds

to judge the current stride length. The following table shows the experimental data used to judge the current stride.

Table 61: Stride as a Function of Speed (steps per 2 s) and Height

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

Speed parameter

As we know the speed can be calculated by:

$$\text{Speed} = \text{distance}/\text{time} \tag{10}$$

so in order to get the speed parameter, as steps per 2 s and stride we will use the following:

$$\text{Speed} = \text{steps per 2 s} \times \text{stride}/2\ \text{s} \tag{11}$$

Calories parameter

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 estimate it using a conventional approximation, however. Table 3 shows a typical relationship between calorie expenditure and running speed.

From this table, we get:

$$\text{Calories (C/kg/h)} = 1.25 \times \text{running speed (km/h)} \tag{12}$$

However because we want m\s the equation becomes:

$$\text{Calories (C/kg/h)} = 1.25 \times \text{speed (m/s)} \times 3600/1000 = 4.5 \times \text{speed (m/s)} \tag{13}$$

Table 62: Calories Expended vs. Running Speed

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

The calories parameter would be updated every 2 s with the distance and speed parameters. So, to account for a given athlete's weight, we can convert our last equation as following: Weight (kg) is a user input, and one hour is equal to 1800 2-second intervals.

$$\text{Calories (C/2 s)}\ 4.5 \times \text{speed} \times \text{weight}/1800 = \text{speed} \times \text{weight}/400 \tag{14}$$

Now 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 the following equation since the caloric expenditure is around 1 C/kg/hour while resting.

$$\text{Calories (C/2 s)} = 1 \times \text{weight}/1800 \tag{15}$$

Finally, we get the total calories by adding the calories for all 2-second intervals.

The selection of which track to play requires a mathematical model as well. This consists of selecting the track with the closest BPM, that is to say minimizing the difference in BPM:

$$\min\left(|target_{BPM} - track_{BPM}|\right) \tag{16}$$

If time permits, this simple model can be replaced with a more complex model incorporating Machine Learning to learn which tracks are more effective than others at changing pulse.

10.2 Data Structures

We are using the SQLite database management system to store the information pertinent to our system. Our database will use the following data types and its use is explained below.

1. Identification Number(ID)-Int- These are auto numbered, used for distinguishing each challenges.

2. Name - String - This field stores the name of friends to which challenges are send.

3. Time -Int - This field stores time details for completing the challenge .This values corresponds with the name of the friend in the name field.

4. Distance -Int- This field stores distance needed to covered for each person who performs the challenge.

11 USER INTERFACE DESIGN AND IMPLEMENTATION

User Interface for this application includes different pages according to the user selection.Each interface page is easily accessible and is explained below

Figure 39: The UI flow shows, in sequence, the Login page, the main landing page, the top navigation menu, the start run page, changing a song during a run and the result of a run.

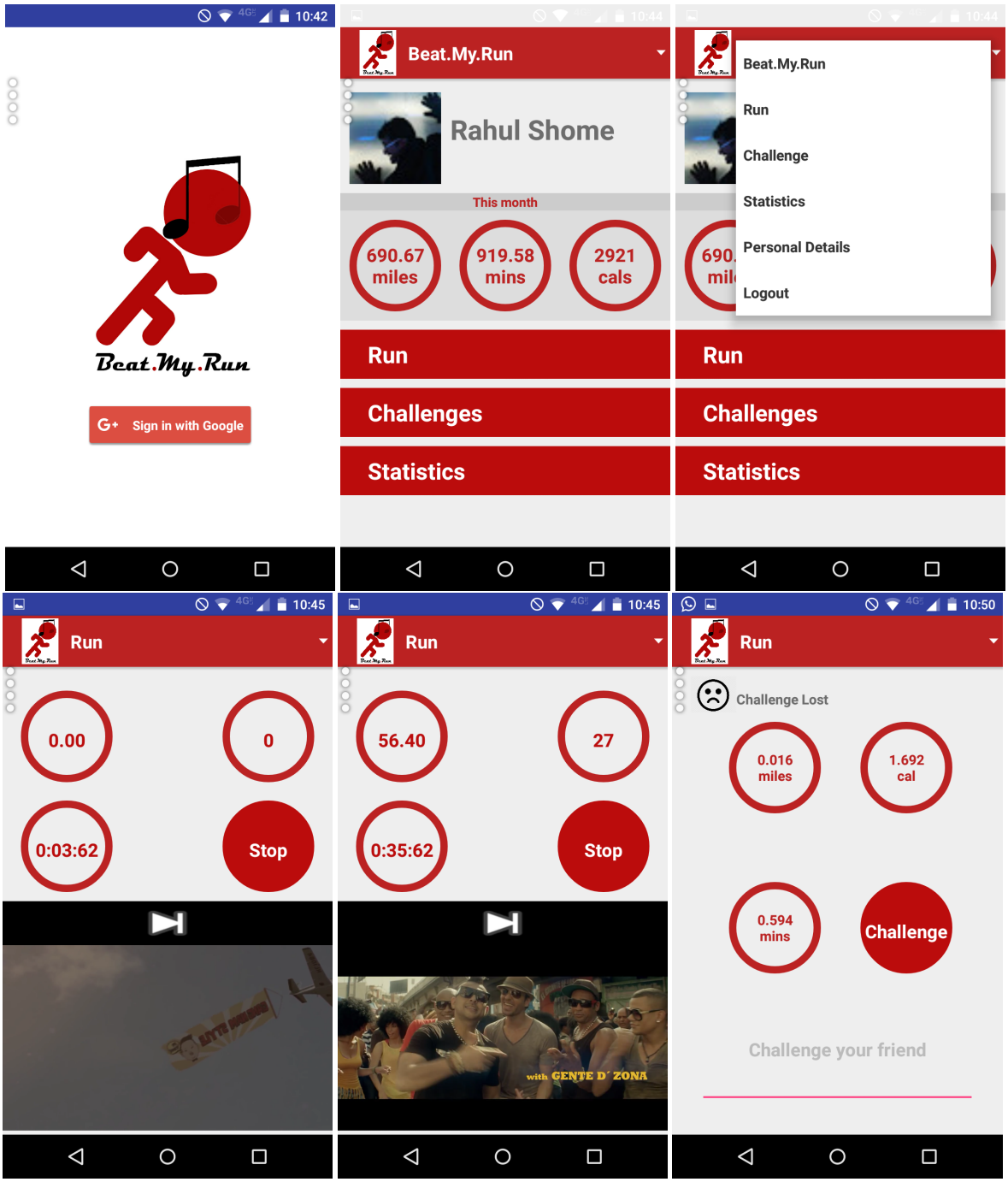


Figure 40: The UI flow shows the autocomplete field for sending a challenge, feedback on the challenge being sent, the statistics main page, the date picker fields for period statistics, the date picker interface and the period statistics graphs.

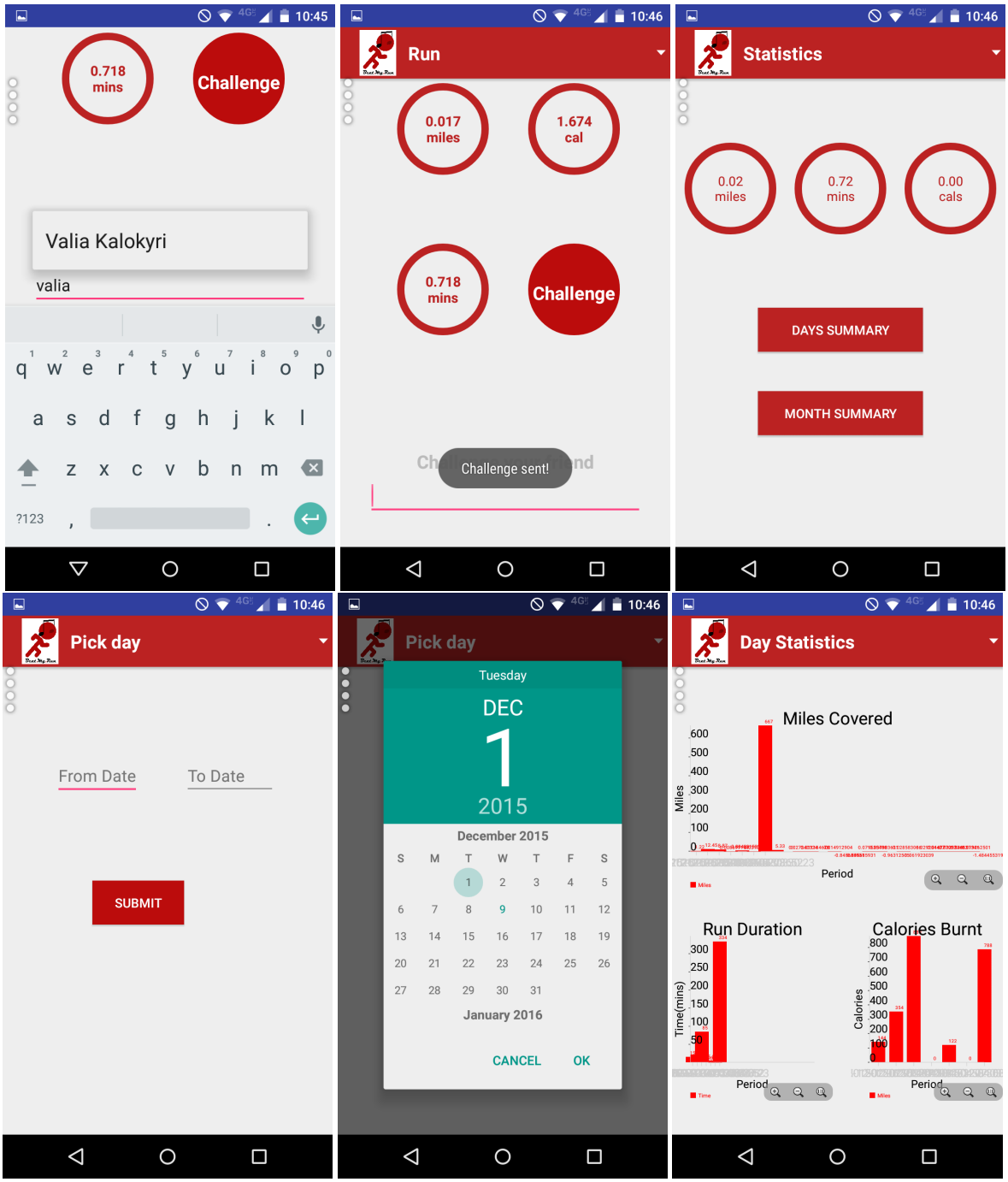


Figure 41: The UI flow shows the selectable list of month and year, the selected month and year, the monthly summary page, the personal details page, the challenge list, and the feedback on a lost challenge.

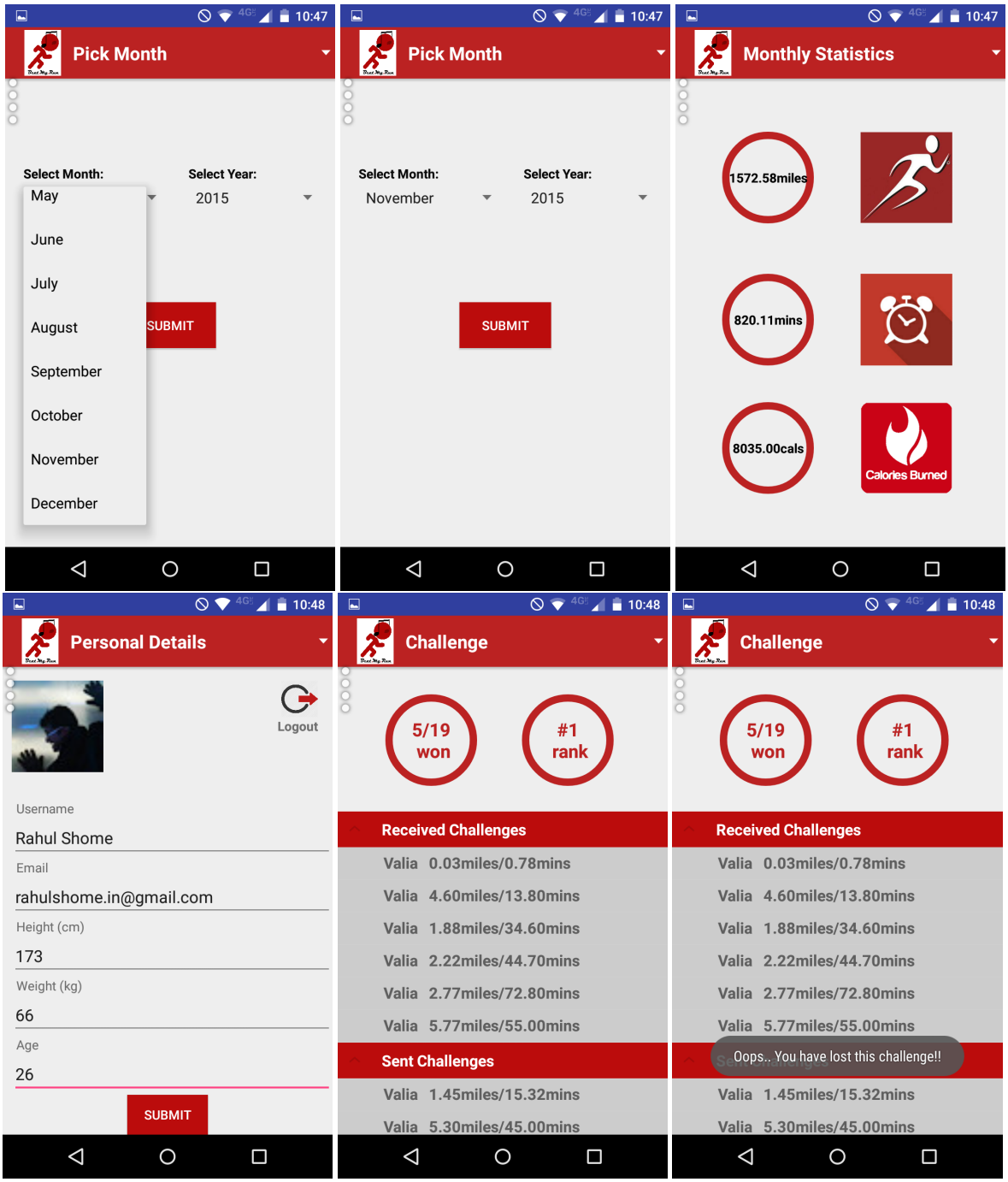
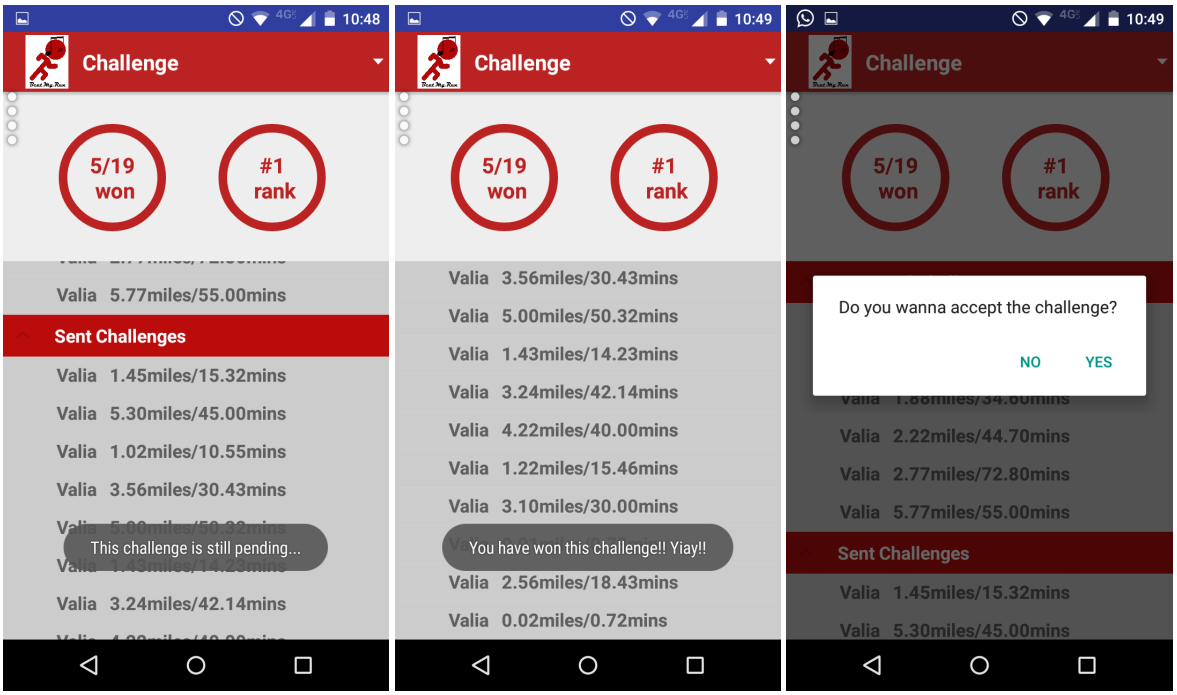


Figure 42: The UI flow shows the feedback on a pending challenge, the feedback on a won challenge and the dialog to accept a challenge and start a new run in response to the challenge.



12 DESIGN OF TESTS

12.0.1 Class Tests

To test the functionality of the project, tests should be performed on the various functions and modules of the project like:

1. Establishing a database successfully.
2. Retrieving desired results from the database
3. Implementing the Login activity of the user by proper authentication.
4. Testing the functionality of playing the songs from the Youtube account of the user.
5. Testing the Sending Challenge module.
6. Testing the results of the challenges.
7. Testing the Graphs and Charts that are formed from the activity history of the user.
8. Testing the Accelerometer of the reading based on the speed.
9. Testing whether the UIs of various xml pages appear as they should be.

The results:

1. The database could be created successfully.
2. The data could be stored and retrieved as required.
3. The User could Login successfully into the App.
4. Songs were played from the very user's Youtube account who was logged in.
5. Challenge could be sent to a friend and also viewed.
6. The distance and time of the participants of the challenge were compared and a winner was declared.
7. Graphs and Charts were produced successfully based on the Activity of the user.
8. The Accelerometer worked well, receiving the steps of the user and calculating them.
9. All the xml pages appeared as they should look.

Test-case ID	TC1_User authentication
Assumption	-MySQL server is running. -The webpage displays the sign-in input screen and the user clicks on the sign-in button.
Pass/Fail criteria	-The test passes if the user creates a new google account and s/he logs in or if the user uses an existing account and s/he is able to log in. The user must also grant the application the necessary permissions for his/her google and youtube account. -The test fails if the user doesn't provide valid credentials for a google account or doesn't grant permissions.
Input Data	-New account details in case the user doesn't have a google account. -Username and password of the user's existing google account if the account is not synced with the phone. -Selection of the synced google account appearing on the screen.
Test Procedure	**Expected Result**
Step 1. User creates a new google account and grants all privileges to the application.	User's google information is stored in the database. The personal information page is shown to the user.
Step 2. User fills in his/her google account credentials and grants privileges to the application.	If the user logs in for the first time the user's google information is stored in the database and the personal information page is shown to the user. Otherwise, the landing/main page of the application is shown.
Step 3. User selects his/her synced google account from the screen.	The landing/main page of the application is shown.
Step 4. User fills in incorrect google account credentials.	An error message is shown to the user and prompts him/her to type his/her credential again.
Step 5. User doesn't grant privileges to the application.	The application closes.

Test-case ID	TC2_Getting user's youtube liked videos
Assumption	-MySQL server is running. -The user has already authenticated and granted access to his youtube account.
Pass/Fail criteria	-The test passes if the systems gets all the user's youtube liked videos and their information. -The test fails if the network connection is down or the youtube server is down.
Input Data	The user's youtube access token
Test Procedure	**Expected Result**
Step 1. The system successfully initiates a connection with the youtube server by passing the user's youtube access token.	User's youtube video information from all his channels is stored in the database. If the user has already used the application before, only the new liked videos are stored. If not, his whole collection of songs is stored.
Step 2. The system fails to initiate a connection with the youtube server.	An error message is shown to the user and prompts him/her to try again later.
Step 3. The youtube access token is no longer valid.	An error message is shown to the user and prompts him/her to reauthenticate his/her youtube account.

Test-case ID	TC3_Getting song information for user's liked youtube videos
Assumption	-MySQL server is running. -The user is already logged in the system
Pass/Fail criteria	-The test passes if the systems gets all the user's song information by initiating a connection to the Echonest server. -The test fails if the network connection is down or the Echonest server is down.
Input Data	Title of user's youtube liked videos
Test Procedure	**Expected Result**
Step 1. The system successfully initiates a connection with the echonest server and every song title is searched in the echonest database.	Get the highest ranked result and stored all its information in the database.
Step 2. The system successfully initiates a connection with the echonest server and a non-song title (youtube video which is not a song) is searched in the echonest database.	The search won't yield any result and nothing is stored in the database
Step 3. The system fails to initiate a connection with the echonest server.	If the search doesn't yield any result the song is not stored in the database (it means it's not a song video).

Test-case ID	TC4_Updating personal information
Assumption	-MySQL server is running. -The user is already logged in the system -The user's information is retrieved from the database and populated in the UI fields.
Pass/Fail criteria	-The test passes if the user has entered valid inputs in all the fields and presses the submit button. -The test fails if the user presses the submit button and s/he has given wrong inputs in at least one of the fields.
Input Data	All the user information data existing in the various fields of the personal information page.
Test Procedure	**Expected Result**
Step 1. The user has correctly filled all the personal information fields and presses the submit button.	The user's entered data are stored in the database and the user is informed about the successfull update.
Step 2. The user has wrongly typed alphabetical data inside the weight or height or age text fields which should be numerical.	The system informs the user about the errors and prompts him/her to correct the mentioned fields and try resubmitting again.

Test-case ID	TC5_Fetching user's music preferences
Assumption	-MySQL server is running. -The application can access the phone's internet connection. -The user is on the Run Activity UI.
Pass/Fail criteria	-The test passes if the songs in the user's music preference and the song information are retrieved from the database. -The test passes if the songs in the user's music preference and the song information are not retrieved from the database.
Input Data	The logged in user's id
Test Procedure	**Expected Result**
Step 1. The youtube ids and the song information are correctly retrieved from the database for the user id.	The streaming links and information is stored in an internal playlist data structure.
Step 2. The music information or the youtube stream is not loaded properly.	The youtube player is not loaded with playable videos and remains blank.

Test-case ID	TC6_Listen to step sensor
Assumption	-The app has privileges to access the steps sensor -The user is logged in and has opened the Run interface.
Pass/Fail criteria	-The test passes if the sensor can be registered to and reports steps if the user takes steps with the phone on his body. -The test fails if the sensor cannot be registered to or the sensor does not report steps for when the user takes steps.
Input Data	-Trigger to listen to sensing, from the Run Activity. -Steps taken by the user with the phone on his/her body.
Test Procedure	**Expected Result**
Step 1. The User clicks on the interface to start the run, the sensor is subscribed to and the user commences taking steps with the phone on his/her body.	The steps sensor is subscribed to and the callback updates the UI with the latest number of steps. The number of steps is taken into consideration for calculating the distance run and BPM as well.
Step 2. The User clicks on the interface to start the run, but the sensor does not allow for subscribers.	The application reports an error in getting the sensor data, for the absence of the sensor or malfunction.

Test-case ID	TC7_Music playback
Assumption	-MySQL server is running. -The application can access the phone's internet connection. -The user's music preferences have been retrieved into the database. -The user is on the Run Activity UI.
Pass/Fail criteria	-The test passes if the youtube links from the user's music preferences are loaded onto the player and streamed at the user inputs. -The test fails if songs cannot be loaded from youtube.
Input Data	User inputs on the play toggle buttons.
Test Procedure	**Expected Result**
Step 2. The streaming links from the youtube ids are loaded into the youtube player.	The videos are cued on the player and play over the phone's internet connection. The UI gets updated with the video and the sound plays over the speakers.
Step 3. The user skips the current song or the current songs ends.	The next song is selected from the playlist data structure by choosing the one with the tempo that matches the BPM over the past 30 seconds. This video is loaded onto the player.
Step 4. The user pauses or plays the current song.	The video on the youtube player is paused or resumed.
Step 5. The youtube stream is not loaded properly.	The youtube player is not loaded with playable videos and remains blank.

Test-case ID	TC8_View run results
Assumption	-MySQL server is running. -The user has just completed the run by clicking the Stop Run button.
Pass/Fail criteria	-The test passes if the information from the current run is displayed on the screen and if the run was initiated as a response to a challenge, the result of the challenge is also shown. -The test fails if the information from the current run is not retrieved and the result of the challenge, if the run was initiated as a response to a challenge, is not calculated.
Input Data	-The summary information from the current run. -The user id of the logged in user.
Test Procedure	**Expected Result**
Step 1. The user clicks on the Stop Run button after a run in response to a challenge.	The run information is entered into the database for the logged in user. The run information is compared to the challenge details and the result is decided. The result of the challenge is updated in the database. The result of the challenege is shown in the UI along with the run information.
Step 1. The user clicks on the Stop Run button after a run not in response to a challenge.	The run information is entered into the database for the logged in user. The run information is shown in the UI.
Step 1. The user clicks on the Stop Run button after a run with incorrect run or challenge information.	The run information is not entered into the database for the logged in user. The challenge information is also not updated. The UI is returned to the landing page/main page after showing an error message.

Test-case ID	TC9_Challenge A Friend
Assumption	-MySQL server is running. -The App displays the Time of the current Run and display a button for challenging a friend using the values of the current Run. -Also the App will allow the user to add a friendâĂŹs name as an Autocomplete-Text Box.
Pass/Fail criteria	-The test passes if the user enters a friendâĂŹs name which is a user of the App as well. The test fails if the user doesnâĂŹt provide a friend name who is the user of the App too, to send a challenge. -The test fails if the information from the current run is not retrieved and the result of the challenge, if the run was initiated as a response to a challenge, is not calculated.
Input Data	- Friend name to whom challenge has to be sent.
Test Procedure	**Expected Result**
Step 1. User enters a valid friend name and then clicks on the challenge button.	The name of the friend is dynamically fetched from the database from all the Users of the App. When the user clicks on Challenge, the challenge is sent to that friend whose name he had entered in the Text Box.
Step 2. User enters a wrong name and then clicks on the Challenge Button	The database cannot fetch any name from the database. However , it allows user to enter a new name. But, when he clicks on the challenge button, it displays âĂŸno such friend foundâĂŹ, as no such user exists.

Test-case ID	TC10_View List of Challenges
Assumption	-Mysql database is running. -MySQL server is running. -MySQL server is running. -The user has entered the Challenges option from the Home page or the top Menu bar. -The app displays the Button options to view the Received and the Send Challenges. -The app also displays the Number of Challenges Won VS Number of Challenges participated. -Also, the Rank of the current user is displayed.
Pass/Fail criteria	-The test passes if the user can view his Sent and Received challenges by clicking on the respective buttons and also the Rank of the user is displayed along with the count of the number of challenges he Won from the number he participated in. -The test fails if the user is unable to click any of the buttons to view the past Sent and Received Challenges. -The test fails if the user cannot view the count of the number of challenges he Won from the number he participated in. -The test fails if the user cannot view his Rank.
Input Data	- Click any button from Sent Challenges and Received Challenges.
Test Procedure	**Expected Result**
Step 1. User clicks the Sent Challenges button	The database dynamically fetches the challenge details of the challenges that the user had sent. If there are no Sent Challenges, the user is still allowed to click the button but the list is Empty.
Step 2. User clicks the Received Challenges button	The database dynamically fetches the challenge details of the challenges that the user had received. If there are no Received Challenges, the user is still allowed to click the button but the list is Empty.

Test-case ID	TC11_View Status of a specific Challenge
Assumption	-Mysql database is running. -MySQL server is running. -MySQL server is running. -The user has clicked on either the Sent Challenges button or the Received Challenges Button and list of the challenges appear. -The App allows the User to click on a specific challenge to view its Status.
Pass/Fail criteria	-The test passes if the user can click a specific challenge from the Sent Challenges list and view the Status of the Challenge whether he has won it, lost it or the challenge is pending. Also, if the user can click a specific challenge from the Received Challenges list and view the Status of the Challenge whether he has won it, lost it or the user gets the option to accept or decline a challenge if it is pending from his side. -The test fails if the user is unable to click any of the challenges. -The test fails if when the user clicks on the challenge, the Status is not displayed.
Input Data	-Click any challenge from the list of Received Challenges or Sent Challenges.
Test Procedure	**Expected Result**
Step 1. User clicks a specific challenge from Sent Challenges list.	The database dynamically fetches the status of the challenge and displays a Toast saying whether the Challenge was won or lost by the current user or is pending on the end of the friend he has sent the challenge to.
Step 2. User clicks a specific challenge from Received Challenges list.	The database dynamically fetches the status of the challenge and displays a Toast saying whether the Challenge was won or lost by the current user or is pending on his end to accept it in which case he receives an option to Accept it or Decline it.

Test-case ID	TC12_Respond to a Challenge
Assumption	-Mysql database is running. -MySQL server is running. -MySQL server is running. -The user has clicked on a specific challenge from the Received Challenges list which is pending to accept from his side and thus gives an option to Accept or Decline it -The App allows the user to Accept or Decline a challenge.
Pass/Fail criteria	-The test passes if the user can click on the challenge which is pending on his side and gets to choose from Accepting it or Declining it. -The test fails if the user is unable to click on any of the challenges which is pending on his side. -The test fails if when the user clicks on the challenge pending on his side, it doesnâĂŹt get an option to Accept it or Decline it. -The test fails if the user Accepts or Declines a challenge but no action is performed. -The test fails when according to the option chosen by the user the status of the challenge does not change in the database.
Input Data	-Clicks on a challenge which is pending on his side and either Accepts or Declines it.
Test Procedure	**Expected Result**
Step 1. User Accepts the challenge	The challenge should get Accepted and the app allows the user to start his Run to complete the challenge.
Step 2. User Declines the challenge	The challenge should get Declined and the challenge is deleted from the database.

Test-case ID	TC13_View statistics options
Assumption	-Mysql database is running. -MySQL server is running. -The user can view the button for statistics page from app main page and after completing run and challenge .
Pass/Fail criteria	-The test passes if the user can click on the Statistics option and the statistics main page and the latest run details are displayed. -The test fails if the user is unable to see the statistics page when the corresponding button is pressed. -The test fails if the latest run details are not displayed. -The test fails if the user cannot find the buttons to view day statistics and month statistics.
Input Data	-User clicks the Statistics option from the main menu. -User clicks the statistics option after completing run and challenges.
Test Procedure	**Expected Result**
Step 1. User click Statistics option.	-The Statistics main page and the latest run details should be displayed. -The buttons for viewing daily and monthly statistics should be displayed.

Test-case ID	TC14_View monthly statistics
Assumption	-Mysql database is running. -MySQL server is running. -The button to view monthly statistics is present.
Pass/Fail criteria	-The test passes if the user can click on the monthly statistics button and the page to select the month and year is displayed . -The test passes when the data for distance,time and calories for the selected month is displayed in the bar graph. -The test fails if the user is unable to see the option for selecting month and year. -The test fails if the month and year option does not show all the months and previous years for the user to select . -The test fails if the bar graph is not displayed with data for the corresponding selected month.
Input Data	-User clicks the month statistics option from the statistics main page. .
Test Procedure	**Expected Result**
Step 1. User click Monthly statistics option.	-The page for selecting month and year is displayed. -User is able to select a month and year from the drop down.
Step 2. User selects a month and year for displaying the data.	-The data for distance,time and calories are displayed in the bar graph. -User is able to zoom and pan the graph.Also the data should be centered and aligned according to the number of data for the corresponding month.

Test-case ID	TC15_View user-defined period statistics
Assumption	-Mysql database is running. -MySQL server is running. -The button to view day statistics is present.
Pass/Fail criteria	-The test passes if the user can click on the day statistics button and the page to select the from date and to date is displayed . -The test passes when the data for distance,time and calories for the selected period of days are displayed in the bar graph. -The test fails if the user is unable to see the option for selecting start date and end date. -The test fails if the calender is not shown when user press start date and end date options. -The test fails if the bar graph is not displayed with data for the corresponding days selected.
Input Data	-User clicks the days statistics option from the statistics main page. .
Test Procedure	**Expected Result**
Step 1. User click day statistics option.	-The page for selecting start date and end date is displayed. -Calender is shown and the user is able to select dates.
Step 2. User selects two dates for displaying the data.	-The data for distance,time and calories are displayed in the bar graph. -User is able to zoom and pan the graph.Also the data should be centered and aligned according to the data for the corresponding dates selected.

12.0.2 Test Coverage

Test coverage measures the degree to which source code of the program is tested.We categorize our testing based on use cases so our approach of test coverage is state based testing.So we define the states our software can take and test by comparing the actual result and expected result.Test cases are written for each use case so that each module is tested completely, thus ensuring total test coverage for our code.

12.0.3 Integration Testing

In Integration testing modules developed by each sub group is combined and tested.Integration testing is an important part of our project as each modules need to communicate with other modules and share date between them.So each modules are combined and tested as groups in multiple ways.Both valid and invalid transitions between modules, data sharing and different exceptions are rigorously tested while integrating.

Also each module retrieve and store data to database.So the php scripts for storing and retrieving data from each module need to be tested.Also when a module need to access data from another module it can be taken from database directly.So after integrating all modules these data storage and retrieval need to be tested.

13 PROJECT MANAGEMENT

This project has three major subsystems. Some of them would be developed in parallel by dividing the whole team for this project to three development teams. For each subsystem (team) the plan is to use an agile development method. The current section introduces the most important and obvious parts of those subsystems. The group will start to develop it in parallel with the fundamental tasks such as finalizing specification phase or introducing system architecture phase.

13.1 Merging the Contributions from Individual Team Members

As the work of the project was equally divided among the members, all of the members contributed in keeping the Nomenclature and the Architecture of the program to be consistent.The modules were tested individually by the members to follow an appropriate pattern and also while combining all the modules the same was taken care of. While combining the modules there were few of the basic challenges faced. Some of which were:

Table 63: Task Breakdown based on subsystems

Subsystems	Code	Task
Mobile Sensing	T01	User input specification in the application
	T02	Mobile sensors readings (accelerometer, GPS)
	T03	Server communication to send logs
User Interface	T04	Graphical user interface design
	T05	User account profiles and information management
	T06	Data visualization from user information.
Infrastructure	T07	Application/server communications protocol specification
	T09	Specifying the information the system needs to maintain
	T10	Database creation to maintain all the logs
	T11	External API integration(*google, accuweather, echonest*)

- The files had to be re-factored in order to avoid conflict and create ambiguity for the compiler
- Also,the merging was done on GitHub, which caused a few merging errors which were solved intelligently.
- Pushing and Syncing in GitHub was a task which failed often, but was done successfully

Thus, the contributors made a great and successful effort to make the project merge and work well.

13.2 Project Coordination and Progress Report

The Demo of the project covers almost half of the implementation of the Final Project. Ofcourse, specific portions are left to be implemented and are worked upon.

Use Cases implemented

- The Login Use Case is implemented as in the user can Login with his Google account and the account it succesfully authenticated.

- The Google API and Youtube API were implemented such that when the user starts his run he is able to listen to the songs that he has Liked on his Youtube account.
- Also In the RunActivity Usecase the steps of the user are received and are calculated by the Accelerometer.
- Thus the communication with the APIs and the Accelerometer has been made to take place by HTTP protocol.
- The SQLIte Database have been successfully implemented which allows a database to be persistently stored on the phone device. This allows the user to view his History of Activities and also saves his Login account for the next time he Logs In.
- The Database is created, updated and retrieved successfully using the Android API 22, thus communication with the database has been set up.
- Also the Challenge Usend challenge to his friends and also view his past sent Challenges.
- A comparison of the parameters like Time and Distance have been implemented by quering the Database and a Winner is declared.
- In addition, the User is able to have a Graphical view of his activity by the Statistics Usecase. Here the History of the User has been graphically represented as graphs and charts along a period of time. Example, the user can view his activity along the past week, month and year.
- In the RunActivity Usecase the system has to communicate with the Echonest API in order to calculate the BPM of the speed of the user. Then the BPM of the song and the BPM of the speed is compared and a song from the Youtube account of the user is played according to the Exercise tempo.
- In the Send Challenge Usecase, when the challenge is sent, the Data containing the Distance and Time of the User is sent along with the request of challenge in-order for the friend to know his target.
- The feature of Sending a Notification of a challenge has to be implemented.
- In Statistics Usecase, the User should be able to view his History of Activities in a Graphical Mode automatically by the System retrieving the data history and forming the graphs Dynamically.

The following subsection introduces all the deadlines we met for this project. These deadlines are for project deliveries such as reports, presentations, etc. But, as mentioned before, the development phase will be held in parallel with all these due to the lack of sufficient time.

Table 64: Project's deadlines

Milestones	**Description**	**Planned Date**
M0	Project Proposal	Sep. 25, 2015
	Finalize the project scope after getting feedback	Oct. 3, 2015
M1	First Report	Oct. 16, 2015
	Statement of work and requirements	Oct. 06, 2015
	Functional requirements Spec and user interface	Oct. 11, 2015
	Full report submission	Oct. 16, 2015
M2	Second Report	Nov. 9, 2015
	Interaction Diagrams	Oct. 23, 2015
	Class Diagram and System Architecture	Nov. 5, 2015
	Full report submission	Nov. 9, 2015
M3	Third report	Dec. 10, 2015
M4	First Demo	Oct. 31, 2015
M5	Second Demo	Dec. 7, 2015
M6	Electronic Project Archive	Dec. 12, 2015

13.3 Plan of Work

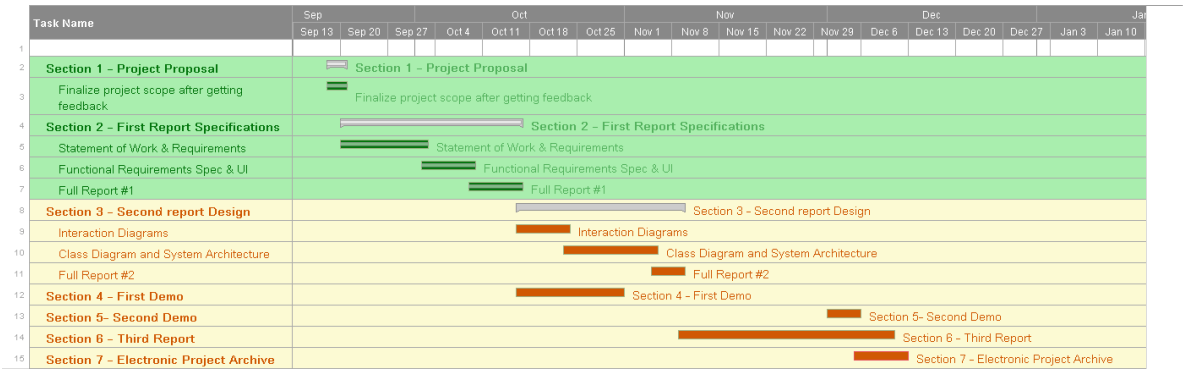


Figure 43: Gantt Chart

13.4 Break down of Responsibilities

1. **L**ogin, Personal Information, Run Use case

    - Program Code Writing by: Rahul Shome, Valia Kalokyri
    - Google + login GUI, Personal Info GUI, Run page GUI (plus Youtube Player)
    - Authenticate user with Google + API
    - Retrieve user's personal google information
    - Authenticate user with his Youtube channels
    - Retrieve his/her liked songs and/or history watch
    - Connection to android SDK sensor manager (for getting user steps)
    - Implementation of Youtube player API
    - Implementation of a Runnable timer
    - User flow of connecting timer events, sensor event and youtube player events
    - Calculation of BPM based on the user's steps
    - Created API developer account for Echonest API and tested via http requests on the browser (not on the app right now)
    - Merging all the components together
    - Debugging:
    - by using computerâĂŹs android emulator(NEXUS 5, API 23)

- and android phone (MOTO X 2013)
- Project management:
- opening accounts on public websites for the project-> Google +API, YouTube API, Echonest API
- organizing meetings
- coordinating activities
- installing and maintaining the developerâĂŹs resources (github version control, Android Studio IDE)
- combining all of the contributions into the correct files and formats

2. **C**reate, Edit, View, Resolve Challenge Use case

- Program Coding Writing by: Careena Braganza and Nirali Vinit Shah
- Activities : DBHandler, ChallengeRepo, ChallengeMain, ChallengeDetail, ChallengeActivity Central database schema
- Implementation of SQLite Database to be used for Challenge Computations using SQLiteOpenHelper package (Challenge Relation)
- CRUD operations for challenges
- Dynamic GUI with all the list of challenge elements populated from the database
- Mathematical computation for finding the winner of a challenge
- Debugging:
- by using computer's android emulator(NEXUS 5, API 22)
- merging the module with the others and solving the errors
- Project management:
- Program Documentation
- organizing meetings
- coordinating activities
- installing and maintaining the developer's resources (github version control, Android Studio IDE)
- combining all of the contributions into the correct files and formats

3. **S**tatistics Use case

- Program Coding Writing by: Nivetha Balasamy and Thara Philipson
- Implementation of GUI for monthly and weekly statistics for: distance, calories, duration of runs
  StatActivity, StatGraphActivity GUI
- Defining data series, view layout spacing and combining view and renderer for Chart visualization
- Dynamic runtime data visualization library aChartEngine was integrated and interfaced with the android application
- Debugging
- using Android emulator , Nexus 5, API 23
- Integrating with android phone and adjusting the view.
- Presentation slides
- Project management:
- organizing meetings
- coordinating activities
- installing and maintaining the developer's resources (github version control, Android Studio IDE)

13.5 History of Work

13.5.1 Deadlines

With regards to the overarching milestones set in the course we stuck to all the deadlines and submitted everything on time. The work for all these checkpoints in the milestone M0-6 were completed on the planned deadline dates.

13.5.2 Key Accomplishments

The key accomplishments in the project can be enumerated as follows:

1. The application was build from the ground up. It was not based on any previous project.
2. The application was designed as a completely pure android application.
3. The end product is an application that can possibly be published publicly with minor improvements.

4. The UI of the application has been standardized to be uniform over all the pages.
5. The persistant storage was deployed on a publicly hosted server as a MySQL database.
6. All of the UseCases discussed Run, Challenge and Statistics were completed fully, along with additions.
7. All of the teammembers were not experts in Android programming but picked it up during the course.
8. The android package has upwards of 20000 lines of code.

13.5.3 Future Work

We plan to iron out few of the bugs and publish the app on the Google Play Store. Getting more statistics from a real population of app users will also let us gather interesting analytics from the user's exercise data.

14 PROJECT MANAGEMENT

This project has three major subsystems. Some of them would be developed in parallel by dividing the whole team for this project to three development teams. For each subsystem (team) the plan is to use an agile development method. The current section introduces the most important and obvious parts of those subsystems. The group will start to develop it in parallel with the fundamental tasks such as finalizing specification phase or introducing system architecture phase.

14.1 Product Ownership

- Nivetha and Thara will be responsible for the View Statistics,Personal Details and Login Use Cases
- Valia and Rahul will work on Run and Train Use Cases
- Careena and Nirali will work on Send Challenge,Song Details and Respond to Challenge Use cases
- These three teams will work on their assigned use cases in parallel. We use agile developing method in each team's plan of work separately. There are 3-4 iterations per subsystems in the Table 1 that need to be done for each team. The first ones are the

Table 65: Task Breakdown based on subsystems

Subsystems	Code	Task
Mobile Sensing	T01	User input specification in the application
	T02	Mobile sensors readings (accelerometer, GPS)
	T03	Server communication to send logs
User Interface	T04	Graphical user interface design
	T05	User account profiles and information management
	T06	Data visualization from user information.
Infrastructure	T07	Application/server communications protocol specification
	T09	Specifying the information the system needs to maintain
	T10	Database creation to maintain all the logs
	T11	External API integration(*google, accuweather, echonest*)

ones that all teams need to start to develop from beginning. All the specification tasks will be confirmed with all the members to finalize.

The following subsection introduces all the deadlines we will meet for this project. These deadlines are for project deliveries such as reports, presentations, etc. But, as mentioned before, the development phase will be held in parallel with all these due to the lack of sufficient time.

Table 66: Project's deadlines

Milestones	**Description**	**Planned Date**
M0	Project Proposal	Sep. 25, 2015
	Finalize the project scope after getting feedback	Oct. 3, 2015
M1	First Report	Oct. 16, 2015
	Statement of work and requirements	Oct. 06, 2015
	Functional requirements Spec and user interface	Oct. 11, 2015
	Full report submission	Oct. 16, 2015
M2	Second Report	Nov. 9, 2015
	Interaction Diagrams	Oct. 23, 2015
	Class Diagram and System Architecture	Nov. 5, 2015
	Full report submission	Nov. 9, 2015
M3	Third report	Dec. 10, 2015
M4	First Demo	Oct. 31, 2015
M5	Second Demo	Dec. 7, 2015
M6	Electronic Project Archive	Dec. 12, 2015

14.2 Gantt Chart

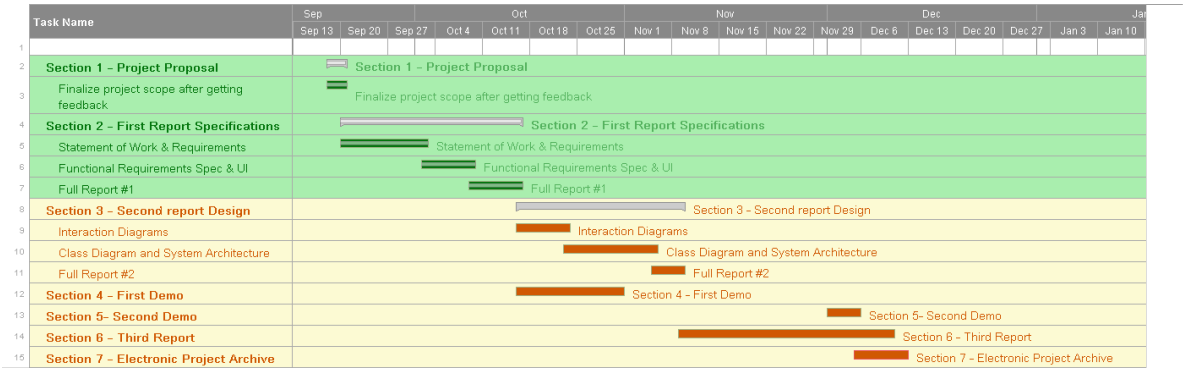


Figure 44: Gantt Chart

REFERENCES

[1] I. Holme and S. Anderssen, "Increases in physical activity is as important as smoking cessation for reduction in total mortality in elderly men: 12 years of follow-up of the oslo ii study," *British journal of sports medicine*, vol. 49, no. 11, pp. 743–748, 2015.

[2] S. H. Boutcher and M. Trenske, "The effects of sensory deprivation and music on perceived exertion and affect during exercise," *Journal of sport and exercise psychology*, vol. 12, no. 2, pp. 167–176, 1990.

[3] J. A. Potteiger, J. M. Schroeder, and K. L. Goff, "Influence of music on ratings of perceived exertion during 20 minutes of moderate intensity exercise," *Perceptual and Motor Skills*, vol. 91, no. 3, pp. 848–854, 2000.

[4] K. A. Brownley, R. G. McMurray, and A. C. Hackney, "Effects of music on physiological and affective responses to graded treadmill exercise in trained and untrained runners," *International Journal of Psychophysiology*, vol. 19, no. 3, pp. 193–201, 1995.

[5] C. I. Karageorghis and P. C. Terry, "The psychophysical effects of music in sport and exercise: A review," *Journal of Sport Behavior*, vol. 20, no. 1, p. 54, 1997.

[6] J. Edworthy and H. Waring, "The effects of music tempo and loudness level on treadmill exercise," *Ergonomics*, vol. 49, no. 15, pp. 1597–1610, 2006.

[7] C. I. Karageorghis, P. C. Terry, and A. M. Lane, "Development and initial validation of an instrument to assess the motivational qualities of music in exercise and sport: The brunel music rating inventory," *Journal of sports sciences*, vol. 17, no. 9, pp. 713–724, 1999.

[8] C. Bacon, T. Myers, and C. Karageorghis, "Effect of music-movement synchrony on exercise oxygen consumption.," *The Journal of sports medicine and physical fitness*, vol. 52, no. 4, pp. 359–365, 2012.

[9] AboutHealth, "http://running.about.com/od/getstartedwithrunning/ht/runwalk.html,"

[10] M. Viru, A. Hackney, K. Karelson, T. Janson, M. Kuus, and A. Viru, "Competition effects on physiological responses to exercise: performance, cardiorespiratory and hormonal factors," *Acta Physiologica Hungarica*, vol. 97, no. 1, pp. 22–30, 2010.

[11] I.-M. Lee, E. J. Shiroma, F. Lobelo, P. Puska, S. N. Blair, P. T. Katzmarzyk, L. P. A. S. W. Group, *et al.*, "Effect of physical inactivity on major non-communicable diseases worldwide: an analysis of burden of disease and life expectancy," *The lancet*, vol. 380, no. 9838, pp. 219–229, 2012.

[12] Wikipedia, "Use case points, https://en.wikipedia.org/wiki/use_case_points,"

[13] R. Prieto-Diaz and G. Arango, "Domain analysis: Acquisition of reusable information for software construction," 1989.

[14] D. M. Eichberg, "Domain model and domain modelling,"

[15] N. Zhao, "Full-featured pedometer design realized with 3-axis digital accelerometer," *Analog Dialogue*, vol. 44, no. 06, 2010.