SOFTWARE ENGINEERING I

REPORT 2

Beat.My.Run

November 9, 2015

Nivetha Balasamy

Careena Braganza

Valia Kalokyri

Thara Philipson

Nirali Shah

Rahul Shome

Rutgers University

Contents

1 Interaction Diagrams **3**

2 Class Diagram and Interface Specification **11**
2.1 Class Diagram 11
2.2 Datatype and Operation signatures 13
2.3 Mapping of Classes to Concepts 19

3 System Architecture and System Design **21**
3.1 Architectural Style 21
3.2 Identifying Subsystems 21
3.3 Mapping subsystems to Hardware 22
3.4 Persistent Data Storage 22
3.5 Network Protocol 22
3.6 Global Control Flow 23
3.7 Hardware Requirements 23

4 Algorithm and Data Structures **24**
4.1 Algorithm 24
4.2 Data Structures 26

5 User Interface Design and Implementation **26**

6 Design of Tests **34**
6.0.1 Class Tests 34

7 Project Management **35**
7.1 Merging the Contributions from Individual Team Members 35
7.2 Project Coordination and Progress Report 36
7.3 Plan of Work 39
7.4 Break down of Responsibilities 39

References **41**

1 INTERACTION DIAGRAMS

Interaction Diagram: Login

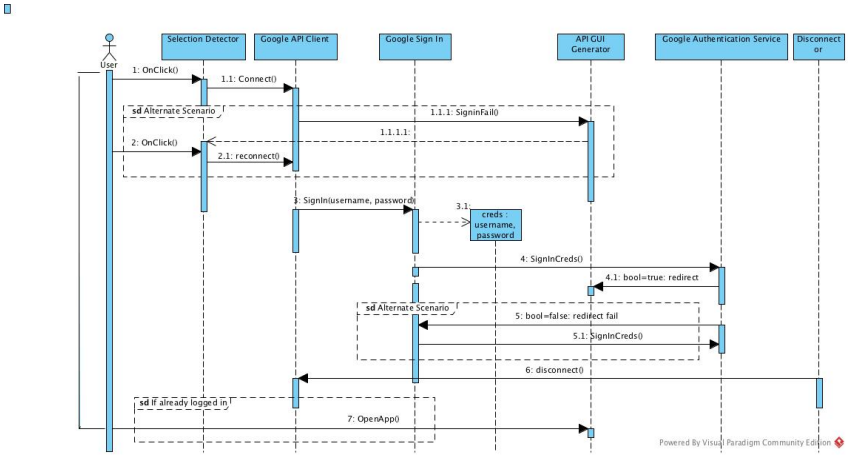


Figure 1: Interaction Diagram: Login

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

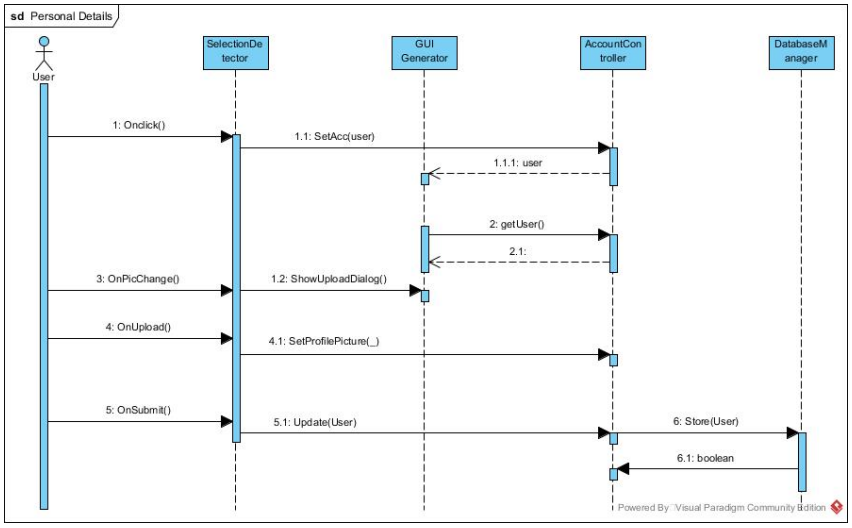


Figure 2: Interaction Diagram: Personal Details

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

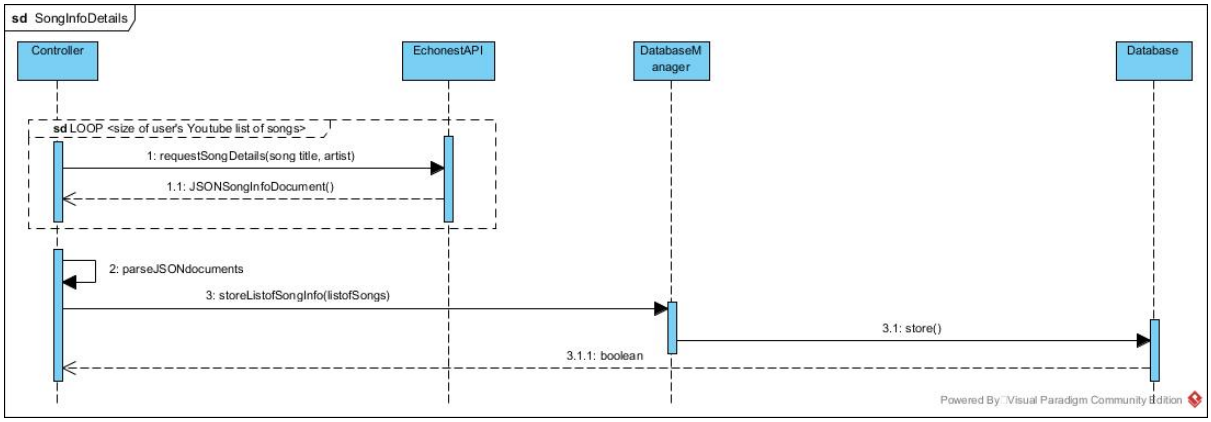


Figure 3: Interaction Diagram: Song Information Details

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

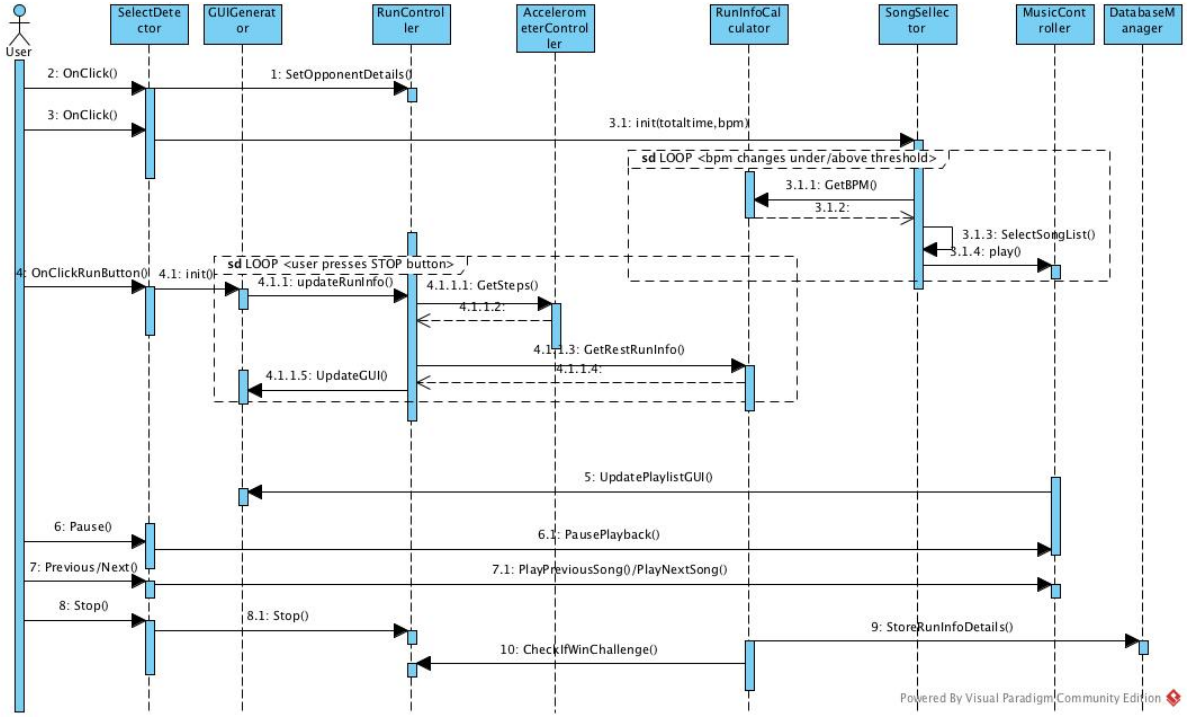


Figure 4: Interaction Diagram: Run

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 one

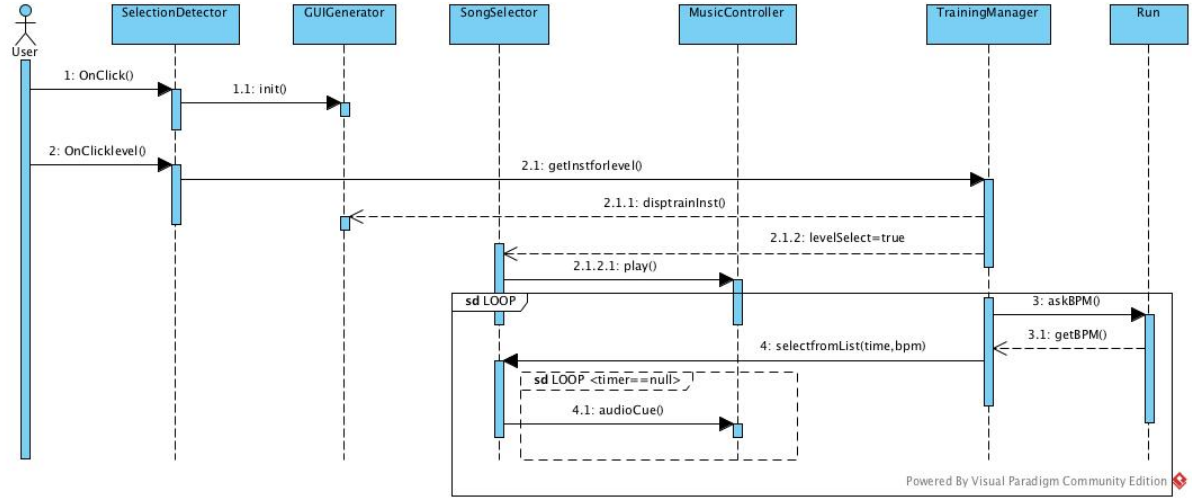


Figure 5: Interaction Diagram: Train

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 âĂIJSelection DetectorâĂİ 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

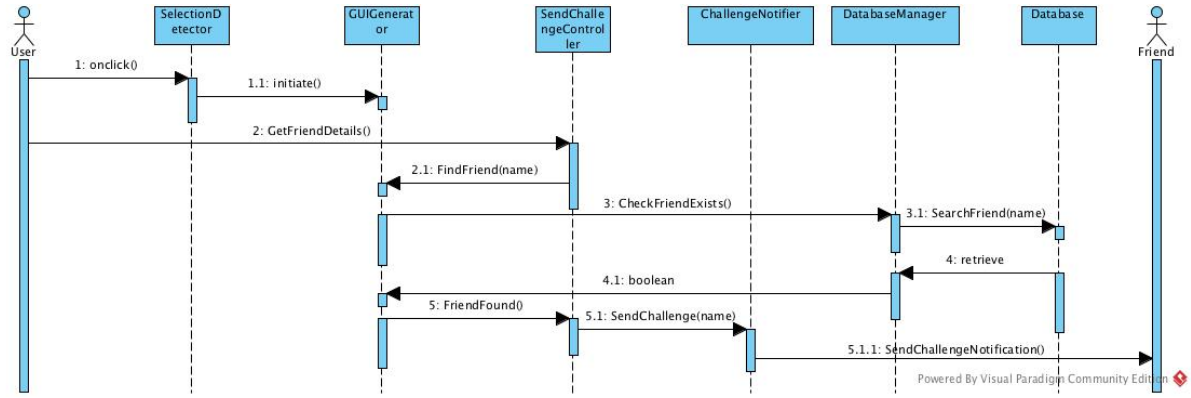


Figure 6: Interaction Diagram: Send Challenge

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

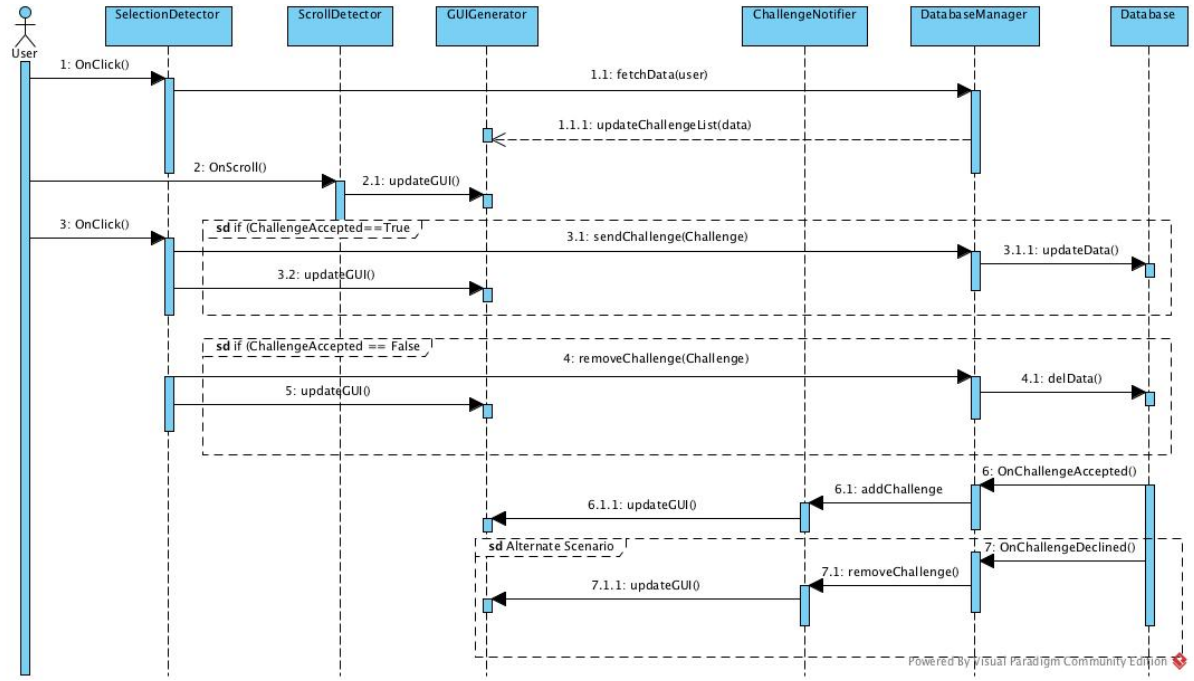


Figure 7: Interaction Diagram: Respond to Challenge

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

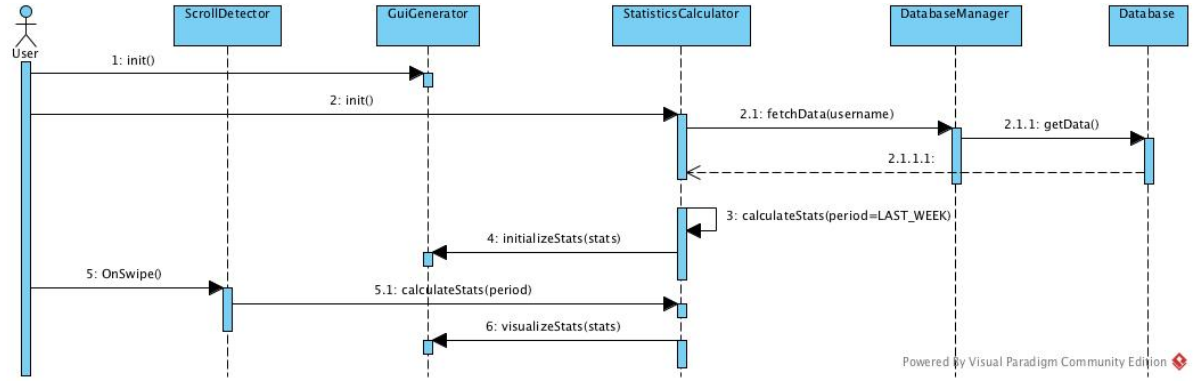


Figure 8: Interaction Diagram: View Statistics

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.

2 CLASS DIAGRAM AND INTERFACE SPECIFICATION

2.1 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 9 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.

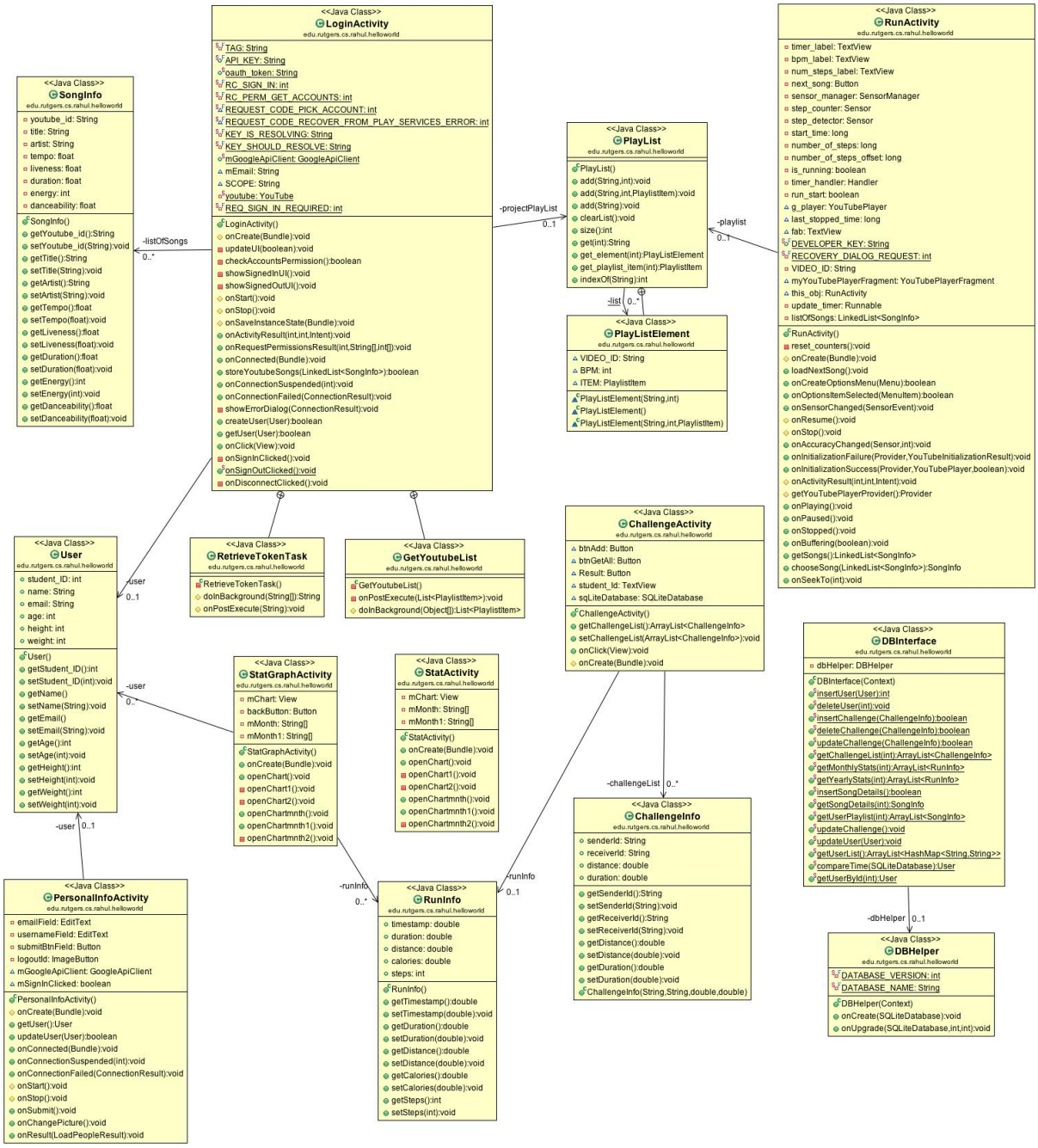


Figure 9: Class Diagram

2.2 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 1: 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 2: 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 3: 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 4: GetYoutubeList

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

Table 5: RetrieveTokenTask

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

Table 6: 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 7: 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 8: 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 9: 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 10: 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 11: 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 12: 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 13: 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

2.3 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 userâĂŹs username and the opponentâĂŹs 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.

3 System Architecture and System Design

3.1 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.

3.2 Identifying Subsystems

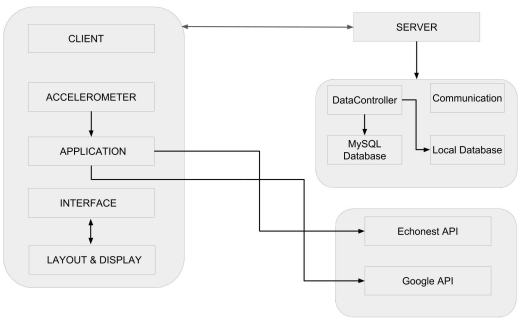


Figure 10: Subsystems of our application

3.3 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.

3.4 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. 11.

3.5 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

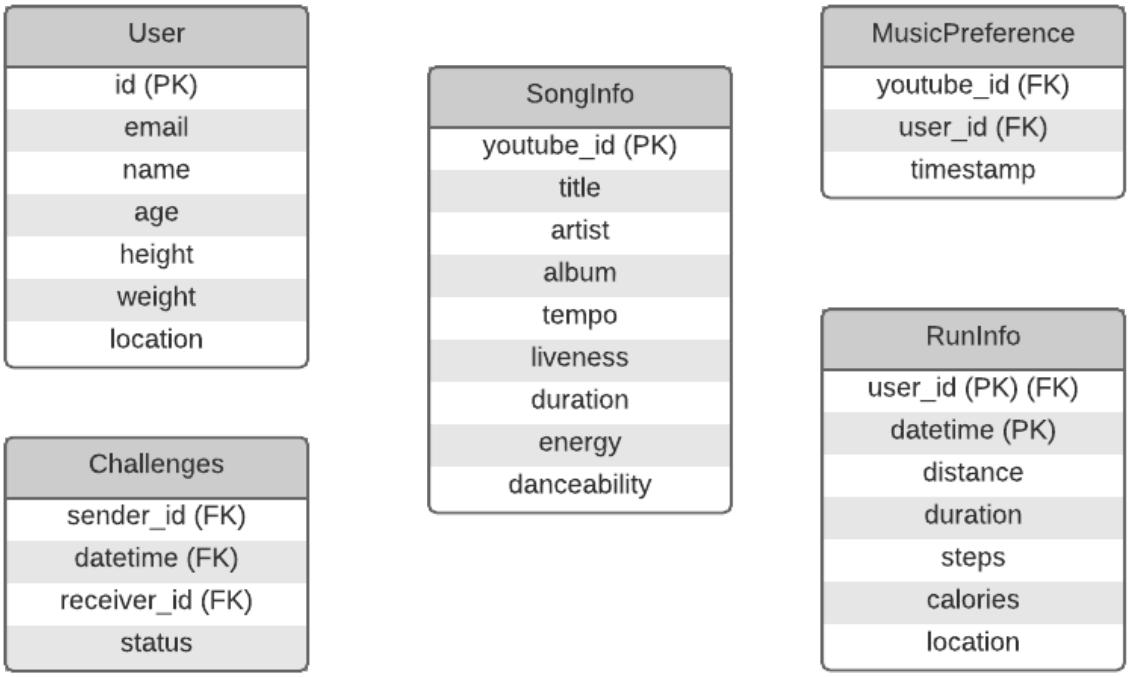


Figure 11: Database Schema

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.

3.6 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.

3.7 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.

4 ALGORITHM AND DATA STRUCTURES

4.1 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 [1] 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 14: 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:

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

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 15: 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.

4.2 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.

5 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 12: Login Page : When user open the application following page appears. This is for inputting user Google credential for logging into the application

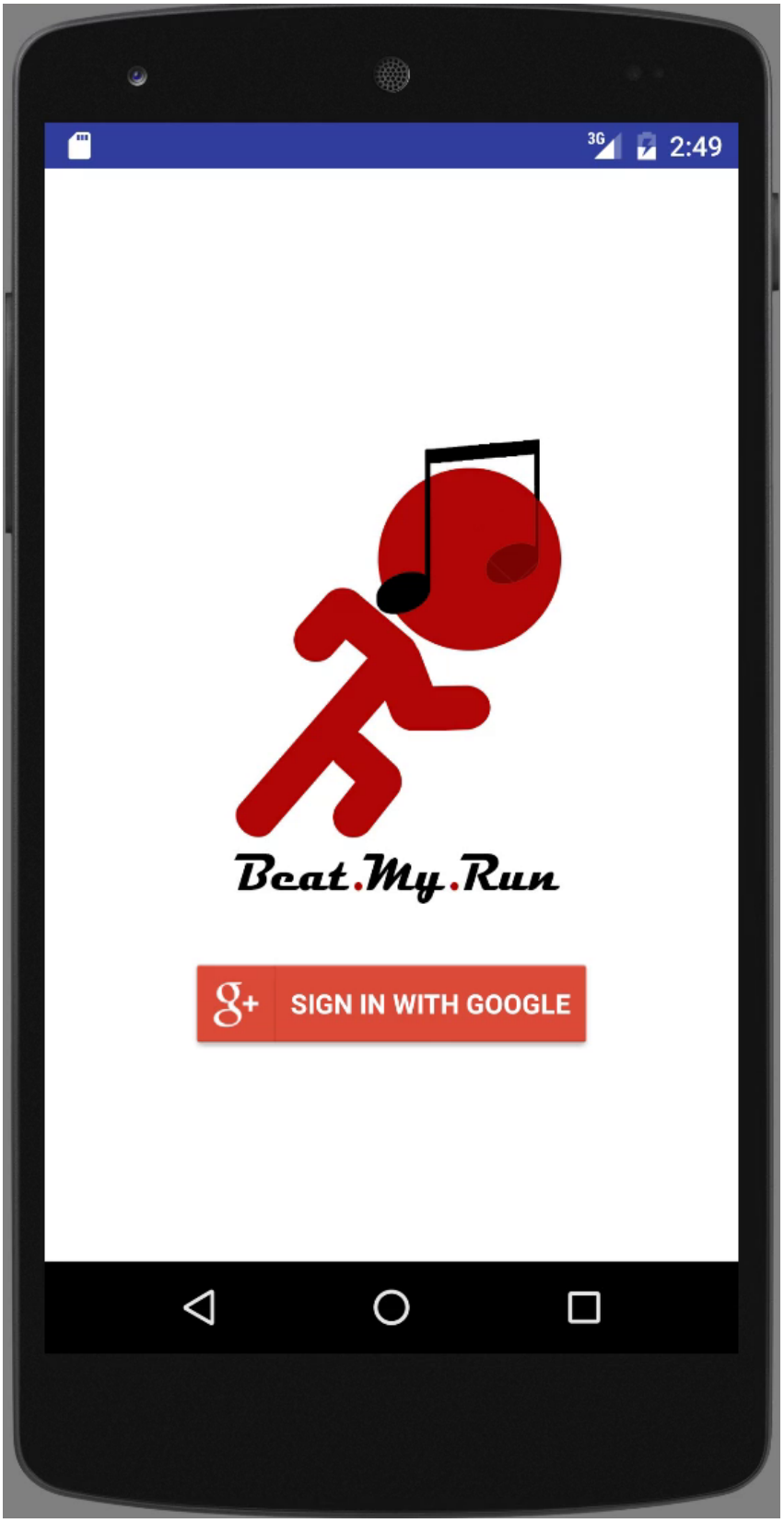


Figure 13:
Choosing Accounts: If the application was accessed before all those account details are shown and you can press and directly login to application
Personal Details Page : After login is successful personal details page is displayed with user name and email ID already updated from your account. User can update the height, weight and age fields and press the submit button to update it.

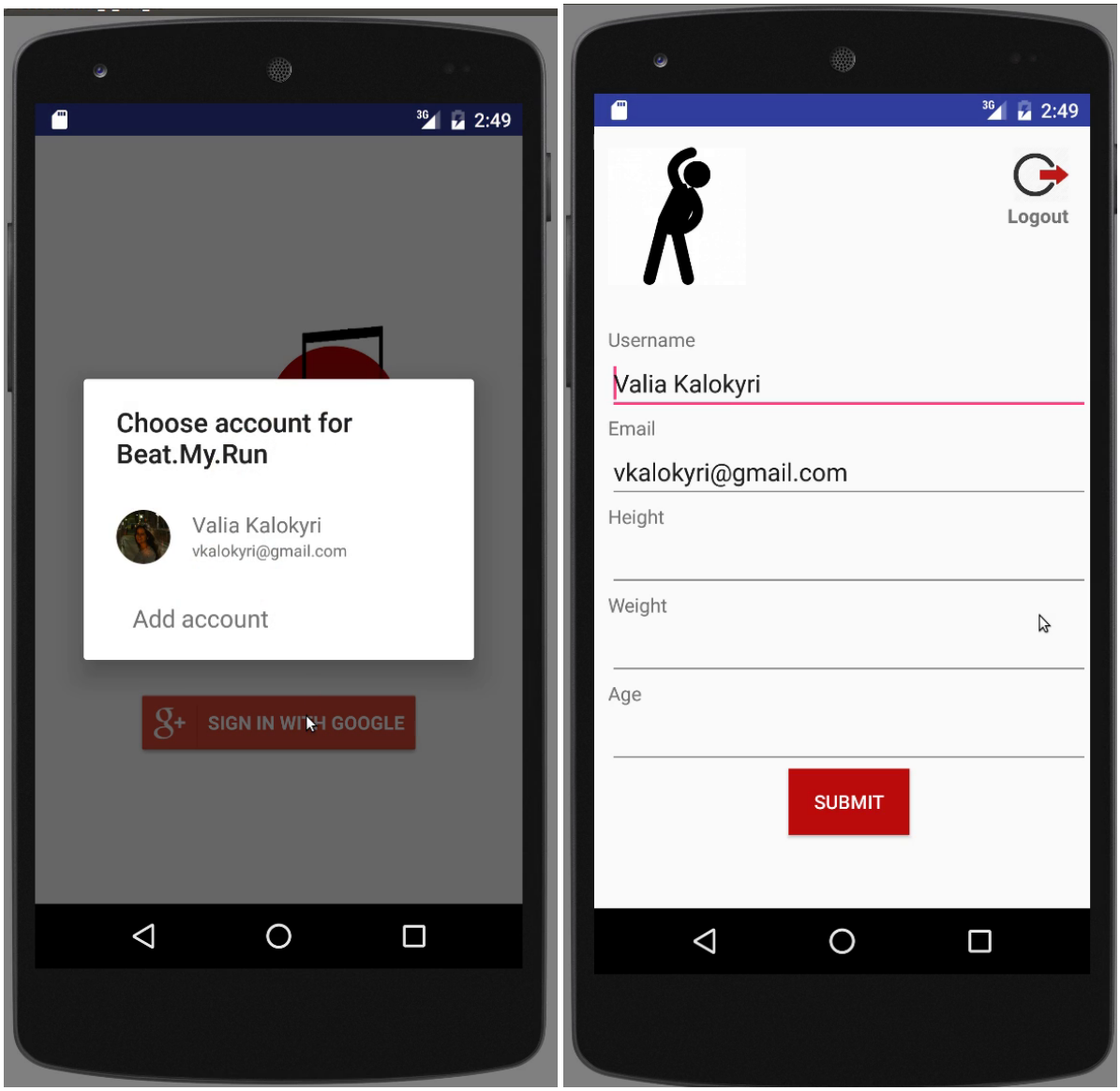


Figure 14:
Run Page : Now the music player is shown with play/pause/next options. Also the beats per minute, steps, timer, stop buttons are displayed in the same page which updates according to the user activity.
Challenge Page : After user completes the workout challenge option can be selected. In this page send a challenge, view a challenge and view results options are available.

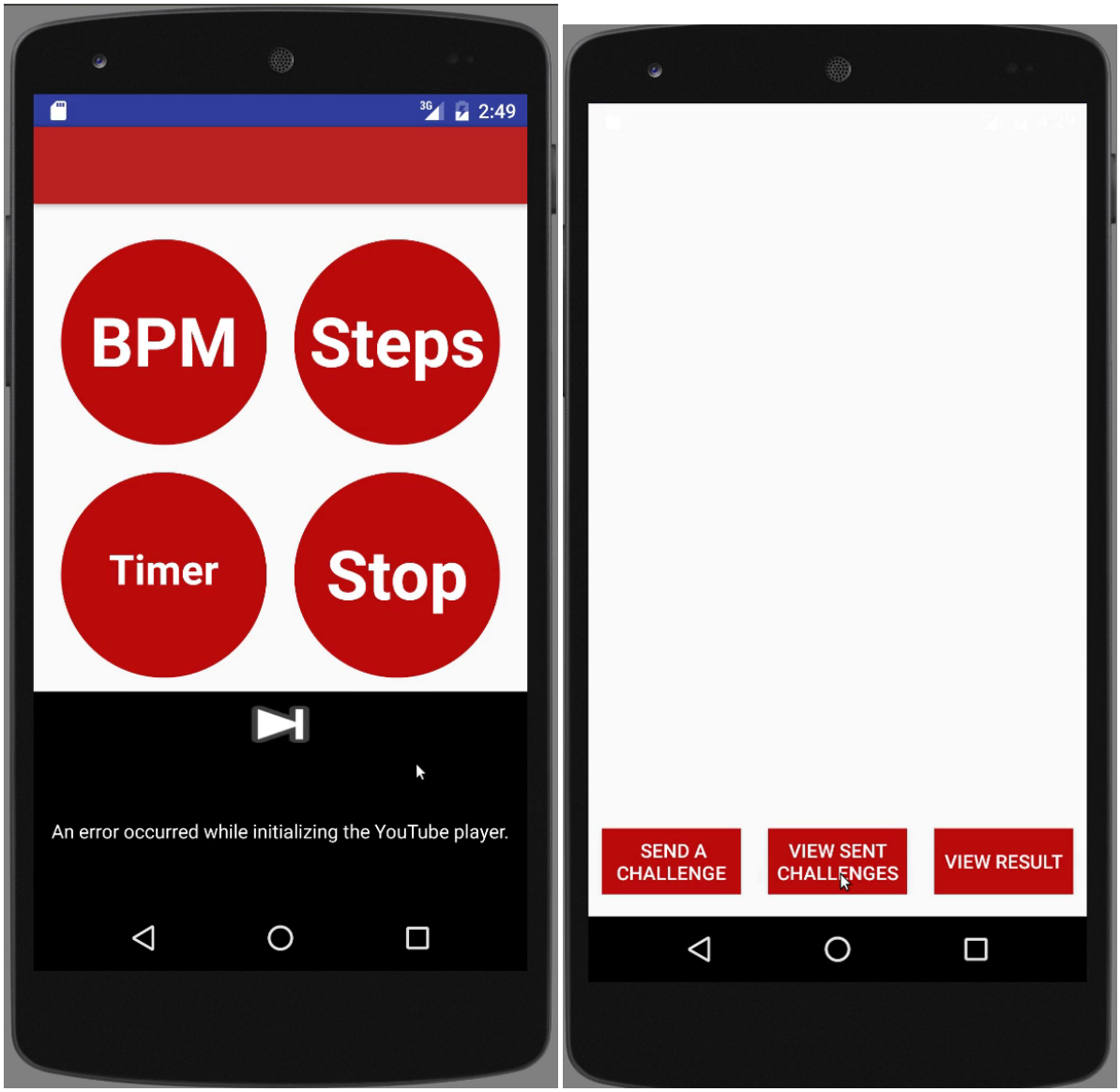


Figure 15:
Challenges : If no challenges are currently available, no challenge available message is displayed while view send challenge button is pressed.
Opponent Details Page : If the user wants to challenge a friend, send a challenge button is pressed and page for entering opponent details are displayed. In the name field friends can be searched.

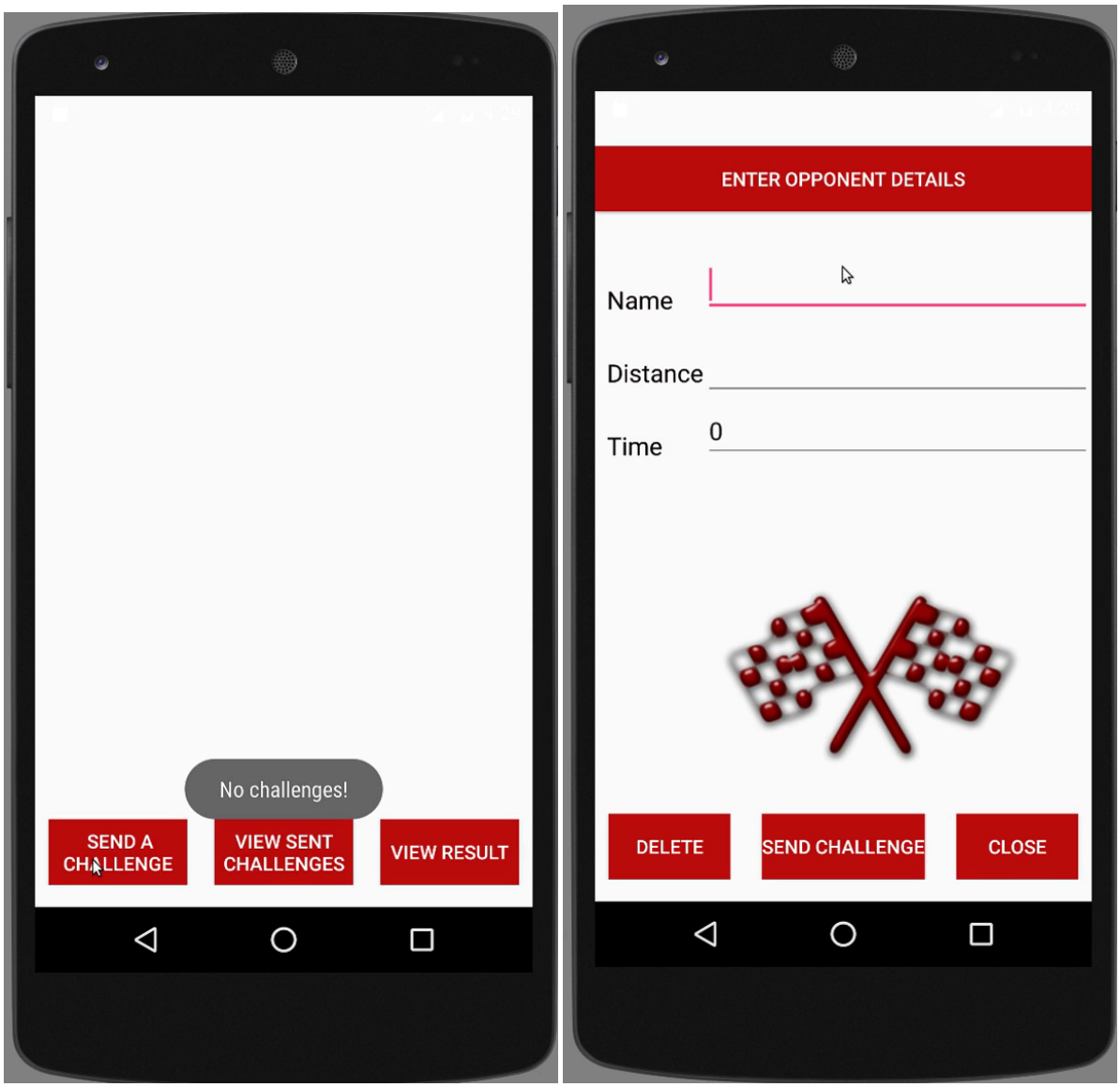


Figure 16:
Send Challenge : When user presses send challenge button, challenge send message will pop up and challenge request is send to corresponding friend.
View Send Challenge Page : Challenges send by user can be viewed by pressing view send challenges. The list of all challenge send by user is displayed in this page.

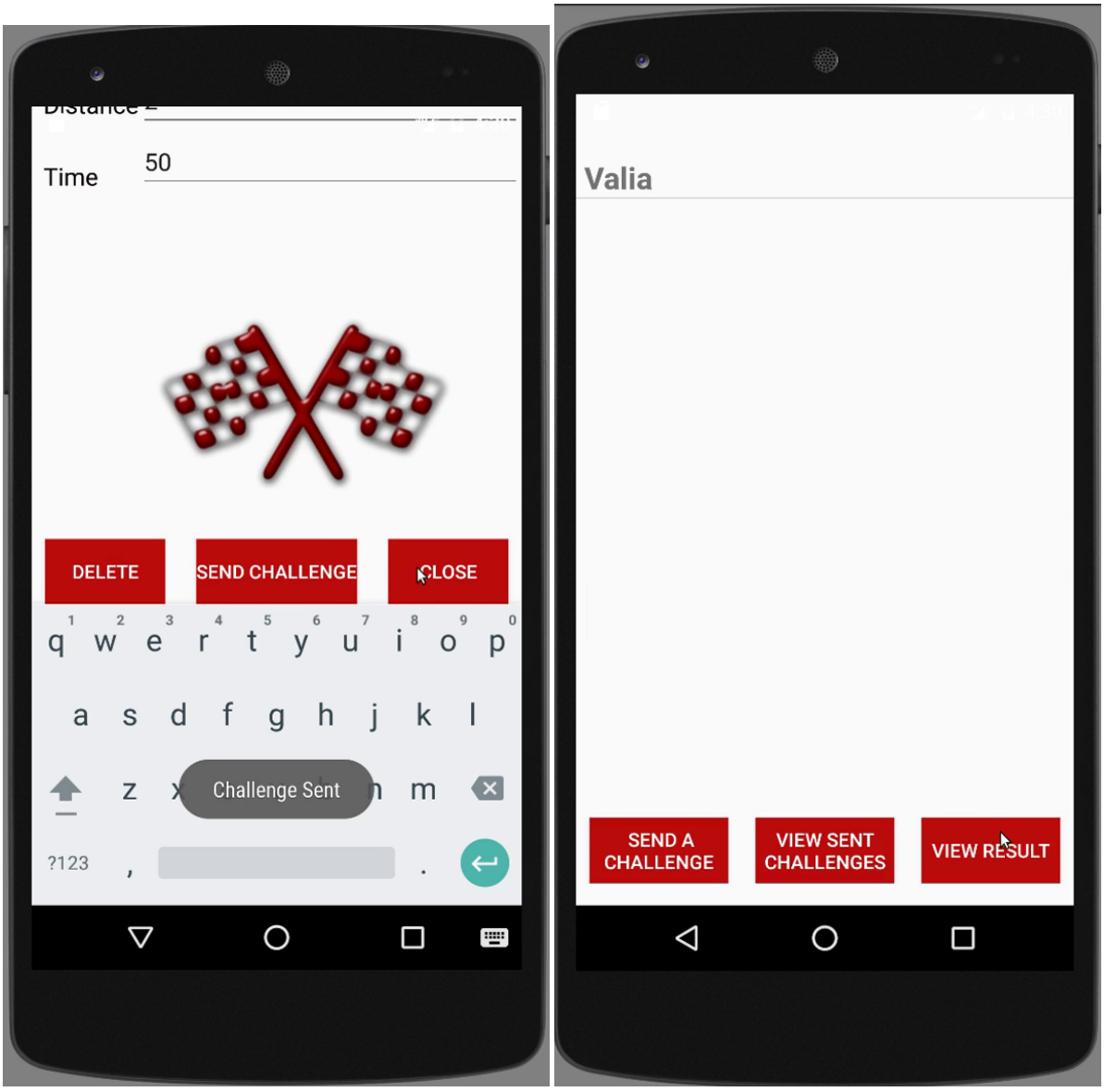


Figure 17:
Winner Details Page : View result page shows the result of all challenges completed. It displays the winner.
Statistics Page : User past exercise details can be found in statistics page. User can select either month or weeks option to view the data for corresponding selected time period.

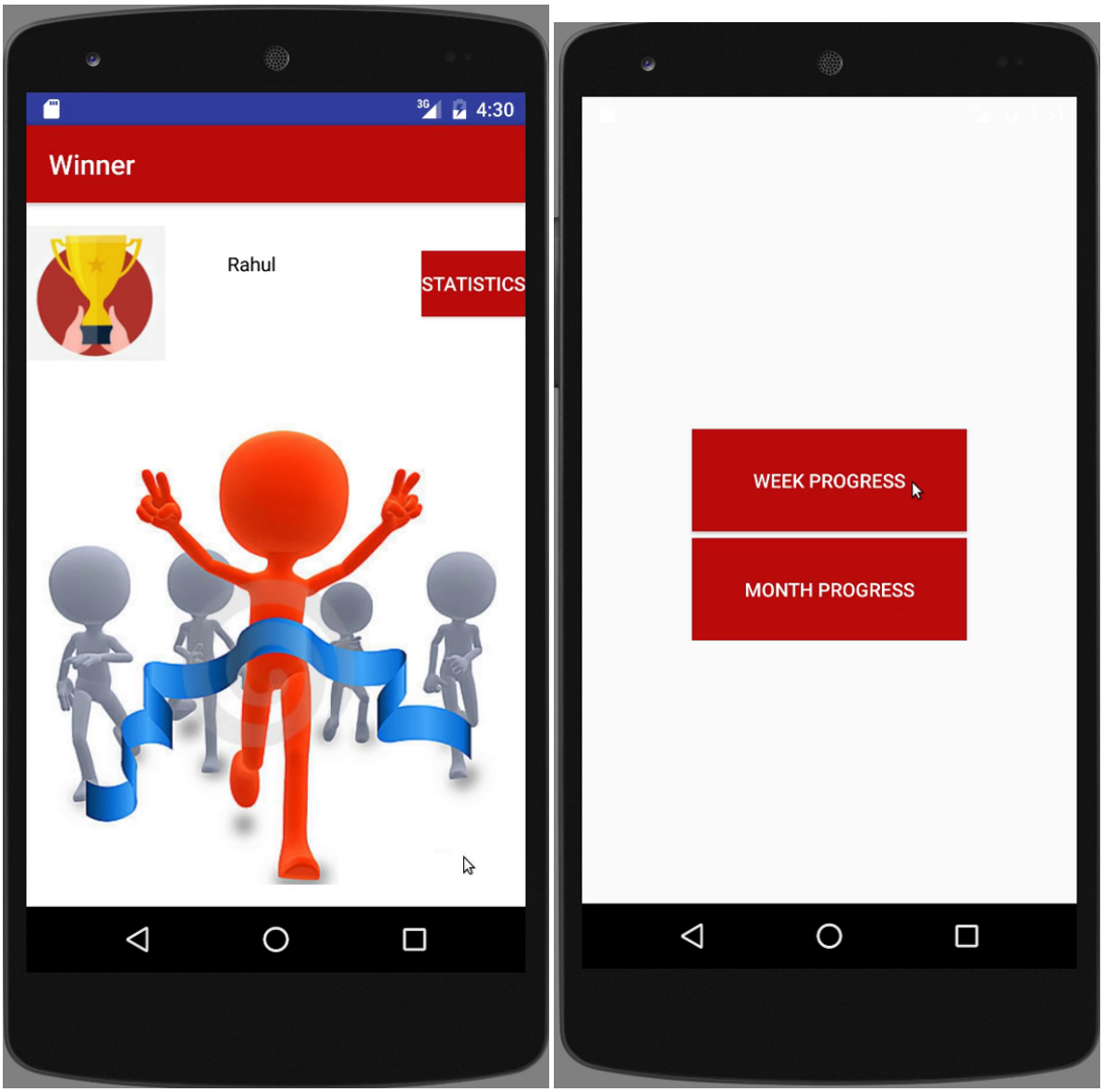
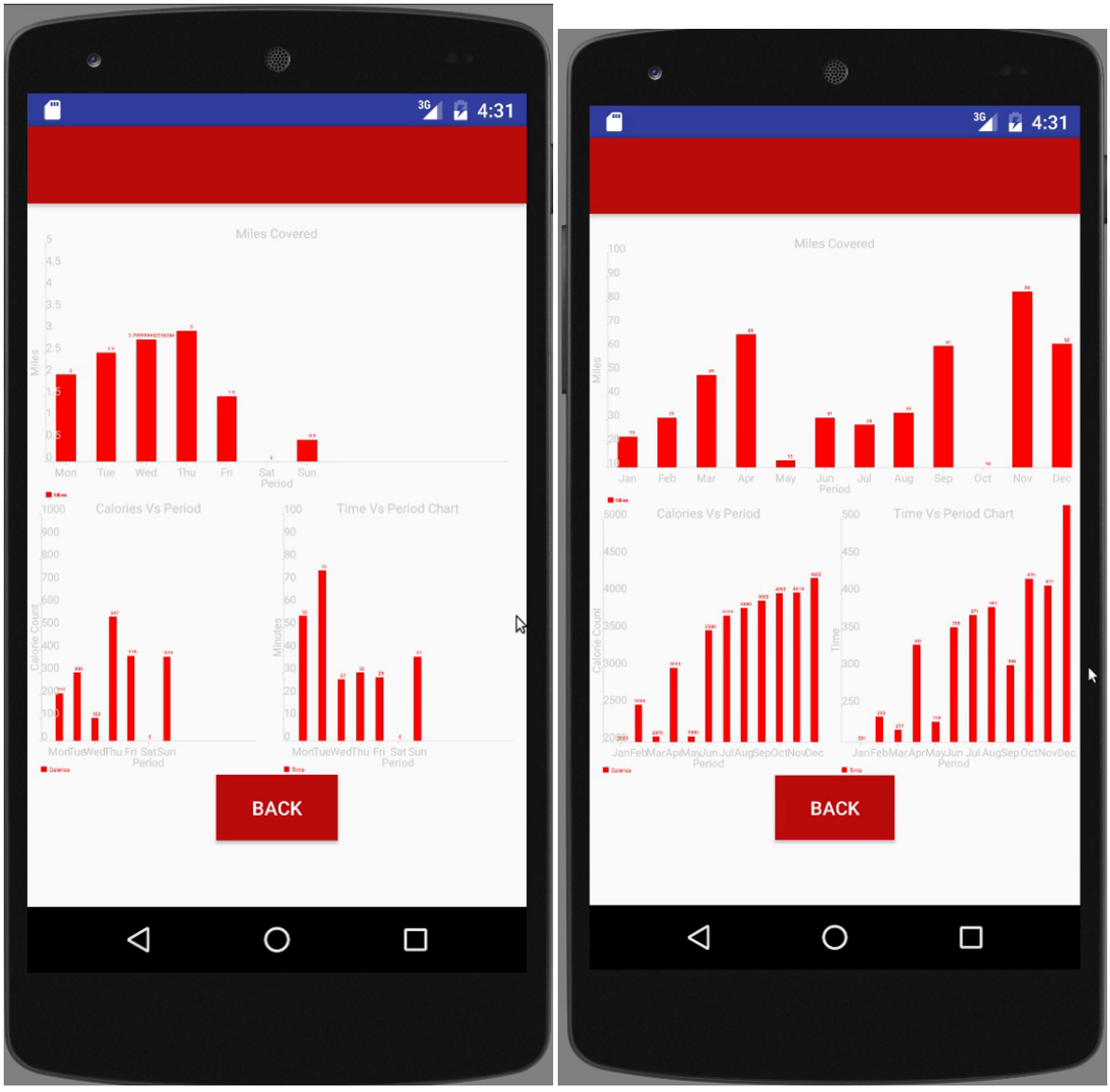


Figure 18:
Statistics Weekly : The graph for miles covered, time spent and calories burned can be viewed as graph in this page.
Statistics Monthly: The graph for miles covered, time spent and calories burned can be viewed as graph in this page.



6 DESIGN OF TESTS

6.0.1 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.

7 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.

Table 16: 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*)

7.1 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:

- 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.

7.2 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.

Parts Currently being Tackled

- 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 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 17: 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

7.3 Plan of Work

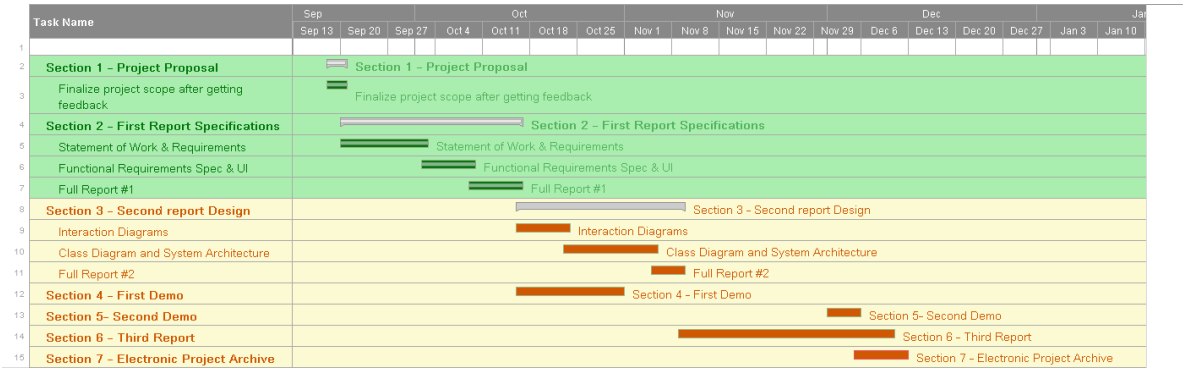


Figure 19: Gantt Chart

7.4 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 userâĂŹs 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 userâĂŹs 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 computerâĂŹs 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 developerâĂŹs 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 developerâĂŹs resources (github version control, Android Studio IDE)

REFERENCES

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