

# **A Smart-Phone App for Runners**

**Roston .S. Pereira**

A thesis submitted for the degree of Master of Science in Advanced Computer Science

Supervisor: Dr John Gan

School of Computer Science and Electronic Engineering

University of Essex

August 2014

Surname: Pereira

First Name: Roston

Qualification Sought: Master of Science in Advanced Computer Science

Title of Project: A Smartphone App for Runners

Supervisor: Dr John Gan

Date: 21<sup>st</sup> August, 2014

---

## **Abstract**

This thesis has been submitted as part of the module for CE901-MSc Dissertation. This document is a report on a smartphone app that will be used by runners. The report gives detailed information about the app and the different methodologies that were used to develop it. The report also gives details of other similar apps and the market size for such types of apps. The second half of this report gives detailed information of all the requirements for developing this app. And the succeeding part of the report gives a brief description of different features of this app, their design and how they were implemented. The report also makes use of images and code snippets for better understanding of the app. The latter part of the report emphasizes on the testing of the app; the report also provides details of the test cases that were used in order to evaluate the app. The last part of this report concentrates on how the project was evaluated; the challenges that were faced while developing this app and the future work or enhancements that can be done in this app. And lastly the report has a Gantt chart that gives a graphical view of the schedule that was planned to accomplish the task of developing this app.

## **Acknowledgement**

I would like to thank Dr John Gan my project supervisor, who guided me throughout my dissertation and ensured that I was always on the correct path. He was always helpful and available whenever required.

I would also like to thank Professor Simon M. Lucas for his lecture and lab notes of CE881-Mobile and Social Application Programming. These notes have been of great help in completing this dissertation.

Thanks to Bagilevi for his tutorials and code on creating a pedometer for my app; Sue Smith for providing wonderful tutorials on how to create a music player. And last but not the least a special thanks to profgustin for providing wonderful tutorials on working with Google Maps.

# Table of Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>1</b>
1.1	Project Goals .....	1
1.2	Project Methodology .....	2
1.3	App Description .....	3
<b>2</b>	<b>LITERATURE REVIEW AND BACKGROUND RESEARCH.....</b>	<b>4</b>
2.1	Literature Review .....	4
2.2	Background Research.....	10
<b>3</b>	<b>PROJECT REQUIREMENTS .....</b>	<b>12</b>
3.1	Functional Requirements .....	12
3.2	Non-Functional Requirements.....	13
<b>4</b>	<b>DESIGN AND IMPLEMENTATION.....</b>	<b>14</b>
4.1	Development Tools .....	14
4.2	Software Design .....	14
4.3	UML diagram.....	19
4.4	GPS Navigation/ Google Maps.....	20
4.5	Calories Calculator.....	23
4.6	Pedometer.....	23
4.7	Timer .....	26
4.8	Music Player.....	27
4.9	Distress Button .....	30
4.10	Database .....	34
<b>5</b>	<b>TESTING .....</b>	<b>41</b>
5.1	Black Box Testing .....	41
5.2	Compatibility Testing.....	44
<b>6</b>	<b>CONCLUSION .....</b>	<b>46</b>
6.1	Project Evaluation .....	46
6.2	Project Management.....	46
6.3	Future Work .....	47
<b>7</b>	<b>APPENDIX .....</b>	<b>49</b>
7.1	Gantt Chart.....	49
7.2	Code .....	50
<b>8</b>	<b>REFERENCES.....</b>	<b>57</b>

## Table of Figures

Figure 1 Total number of Apps in the Google play store. ....	4 -
Figure 2 Top 10 Google play categories [4] .....	6 -
Figure 3 Average daily usage of Health and Fitness Apps [13] .....	8 -
Figure 4 .....	9 -
Figure 5 Starting screen of the app .....	15 -
Figure 6 Main screen of the app .....	16 -
Figure 7 Screen that will display the list of songs .....	17 -
Figure 8 Contact details screen .....	18 -
Figure 9 UML Diagram .....	19 -
Figure 10 XML layout file that displays the Google maps in this app .....	21 -
Figure 11 Method that is used to display the current location of the user .....	22 -
Figure 12 Method to calculate the step .....	25 -
Figure 13 Method that is used to calculate time .....	27 -
Figure 14 Method to retrieve audio files from user's device .....	28 -
Figure 15 Code that is used to sort the audio file in alphabetical order.....	29 -
Figure 16 Code that is used to display the songs list in a list view.....	29 -
Figure 17 Code to start playing an audio file.....	29 -
Figure 18 Code to keep the song playing when user has navigated to some other activity -	30 -
Figure 19 Method that is used to send SMS .....	31 -
Figure 20 Method that is used to take a screenshot .....	32 -
Figure 21 Method that is used to convert and save the screenshot into bitmap image.....	33 -
Figure 22 Code that is used to send email during mishap .....	33 -
Figure 23 Declaring Content Provider in the manifest files [21].....	35 -

Figure 24 Method to save contact details in SQLite database .....	- 36 -
Figure 25 Method that is used to retrieve data from the SQLite database.....	- 37 -
Figure 26 Method that is used to update contact details in the database .....	- 37 -
Figure 27 Method that is used to delete data from the database.....	- 38 -
Figure 28 Retrieving preferences from application level shared preference [23] .....	- 38 -
Figure 29 Retrieving preferences from Private Preferences [23] .....	- 39 -
Figure 30 Methods in Editor Class that can be used to insert data [23] .....	- 39 -
Figure 31 Updating Shared Preference [23] .....	- 40 -
Figure 32 Retrieving Shared Preference [23] .....	- 40 -
Figure 33 Saving data to a Shared Preference .....	- 40 -

# 1 Introduction

The aim of this documentation is to provide all the stake holders with a detailed report on the design and implementation of the smart phone app. The report has been broken down into several sections for easy understanding. The first section gives details about the main aim of this project; and the methodology that was used to complete the project. The second section provides the reader with the market scope for such types of app and details of some other apps from the play store that are similar to this app.

The third section defines the requirements for developing this app and the features that were to be included in the app. The fourth section gives a detailed report of how the different features of this app were designed and implemented.

The fifth and sixth sections of this report give details of how the app was tested using different testing methodologies and what possible improvements can be made in the app. The last two sections contain an updated Gantt chart and references.

## 1.1 Project Goals

The main aim of this project was to create a smartphone app that could measure the distance travelled by a person, the number of calories burned while travelling that distance and the time taken to cover that distance. The project also aimed at adding some more features like music player and GPS navigation with Google maps.

One of the challenges that were faced while developing this app was the integration of the Google Maps API with the application. The integration of Google maps was accomplished successfully with the help of some tutorial.

The other challenge faced while developing this app were, having a simple and user friendly GUI that could be operated by any naïve user. Efforts have been made to keep the app simple and user friendly by aligning the components (i.e. the buttons) properly and giving them a reasonable size. While developing this app efforts have been made to keep the app uniform on all devices i.e. the GUI of the app looks similar on all devices.

## **1.2 Project Methodology**

In order to complete the project on time and without any bugs, it was decided to use agile software development approach with some practices of extreme programming. I decided to use agile development approach because the app was big and had many different features that were dependent on each other. While developing the app, some of the extreme programming approaches like test driven development and continuous design improvement were used.

The project was divided into two iterations; the first iteration comprised of designing the pedometer, GPS navigation system and the music player. The second iteration consisted of enhancing the components that were developed in the first iteration and designing the database. The second iteration also included implementing all the features of the app.

As the project makes use of some extreme programming practices like simple design and continuous design improvement; the design of the app was changed for a few times in order to make the GUI simple and easy to use [\[1\]](#).

As the project used test driven development approach; unit testing was used to test each piece of code in the app. The unit testing included testing different classes that are present in this app [\[1\]](#).

The app was developed using tools like IntelliJ Idea and Google play services. It also makes use of Google maps API to integrate Google maps with this application.



### **1.3 App Description**

With advancement in technology and invention of sophisticated gadgets, the life style of people is changing; this changing life style is making them more and more lazy. Most of us spend our time sitting in front of a TV or a laptop; as a result our bodies don't get the exercise they need. Here is an app that will encourage you to get on your feet and live a healthy life.

This fitness app can be used to keep a track of the distance a person walks or runs. Not just tracking the distance covered, the app will also show you the different route you can take, with the help of maps. If you get bored by walking alone the app can entertain you with music; the app has a music player that can play your favourite songs. Apart from entertaining you, the app also takes care of your health; it calculates the number of calories you burn while you are running. The app keeps a tab of the time required to cover a particular distance and if the user achieves a unique target like covering a distance within limited time or if a user burns more calories in a particular workout session; the app will save all these targets you have achieved in its database. And the next time when you open your app it will display these targets on a leader board; the leader board will display all the achievements of a user and encourage them to do more exercise.

This app can be your life long companion; it will also help you in your bad times. If a user meets with an accident the app can help you in alerting your friend and/or family members. The app consists of a distress button that can be used in case of a mishap; the distress button will alert your friends by sending them SMS and emails about an accident and giving them your location information via email.

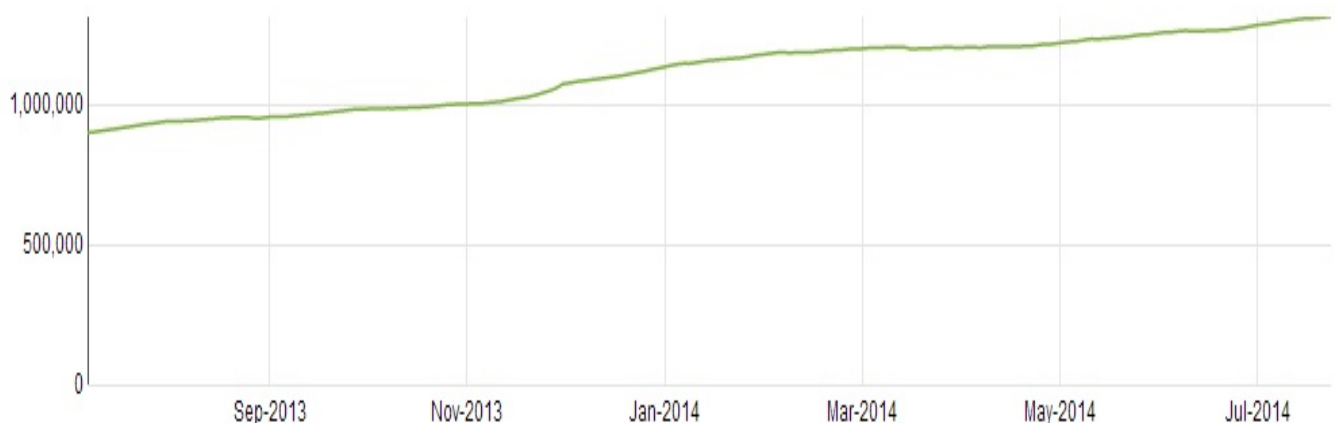
## 2 Literature Review AND Background Research

### 2.1 Literature Review

In the recent years due to advancement in technology the mobile phone market has been through a huge transformation. The transformations of mobile phones from Symbian Operating System to android begin in the year 2008. Android, Inc. a California based company first started building the Android Operating System in the year 2003 and later on in July 2005 Google Inc. acquired Android, Inc. and released the Android source code under the open source license. The first smartphone with Android operating system was released in the market by HTC on October 22<sup>nd</sup> 2008.

Android is an open-source operating system based on the Linux kernel, normally used in Smartphones, tablets and PC's. Apart from smartphones and tablets, android Operating Systems also support many other devices like Android TV's, wrist watches and much more. There are thousands of apps in the app market. Android user can download apps from the android play store or the Amazon Appstore; whereas Apple users can download apps from the Apple's app store. As of August 3, 2014 the Google play store consisted of 1,317,661 apps. All these apps are either free apps or paid apps [2].

Number of apps on Google Play



**Figure 1 Total number of Apps in the Google play store [2]**

The Google play store consists of various types of apps, like gaming apps, medical apps and so on. Below is a list of apps by category that are currently active in the App store [\[3\]](#).

Current Active Application Count By Category		
Category	Total	% of Total
Games	224,019	19.66%
Education	120,614	10.58%
Business	101,143	8.88%
Lifestyle	90,296	7.92%
Entertainment	83,866	7.36%
Utilities	61,187	5.37%
Travel	55,720	4.89%
Book	52,515	4.61%
Music	38,149	3.35%
Productivity	33,279	2.92%
Health & Fitness	31,215	2.74%
Sports	31,164	2.73%
Reference	30,520	2.68%
Photo & Video	29,502	2.59%
News	28,232	2.48%
Finance	27,861	2.44%
Medical	24,641	2.16%
Food & Drink	24,035	2.11%
Social Networking	20,964	1.84%
Navigation	14,996	1.32%
Catalogs	7,737	0.68%
Weather	4,422	0.39%
Graphics & Design	949	0.08%
Developer Tools	899	0.08%
Video	855	0.08%
Photography	808	0.07%
Tech News	1	0.00%
Healthcare & Fitness	1	0.00%
<b>Total</b>	<b>1,139,590</b>	

Table (1) Current Active App by Category, in the app store [\[3\]](#)

According to a survey conducted by AppBrain, the statistics show that Entertainment apps like games are most popular apps in the Google play store, followed by lifestyle apps and education apps [4].

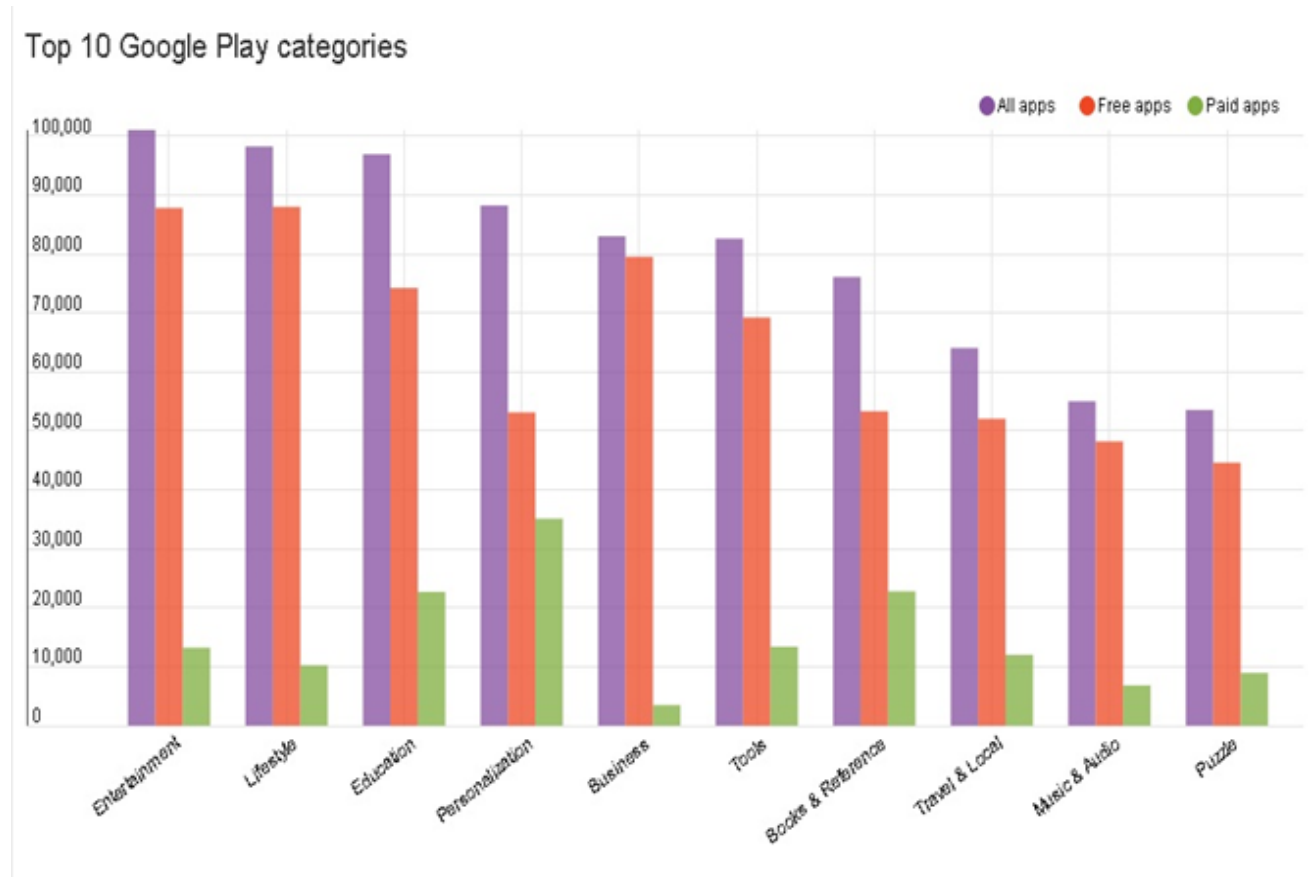


Figure 2 Top 10 Google play categories [4]

In this modern era people have become more health conscious, due to the widespread awareness of health issues that are caused by overweight and obesity; more and more people are making efforts to work-out in the gym or do some daily exercise at home. As more people are getting health conscious, the demand for health care apps is increasing. The app market consists of thousands of healthcare apps that are used for different purposes; like some of the apps are used for personal health monitoring while some others are used for fitness training. The app market also boasts of health care apps that are used by Asthma patients, Diabetes patients, patients with Heart conditions and so on. The market for health and fitness apps is ever growing, but there arises a big question “*Can Mobile Virtual Fitness apps replace Human fitness trainers?*”

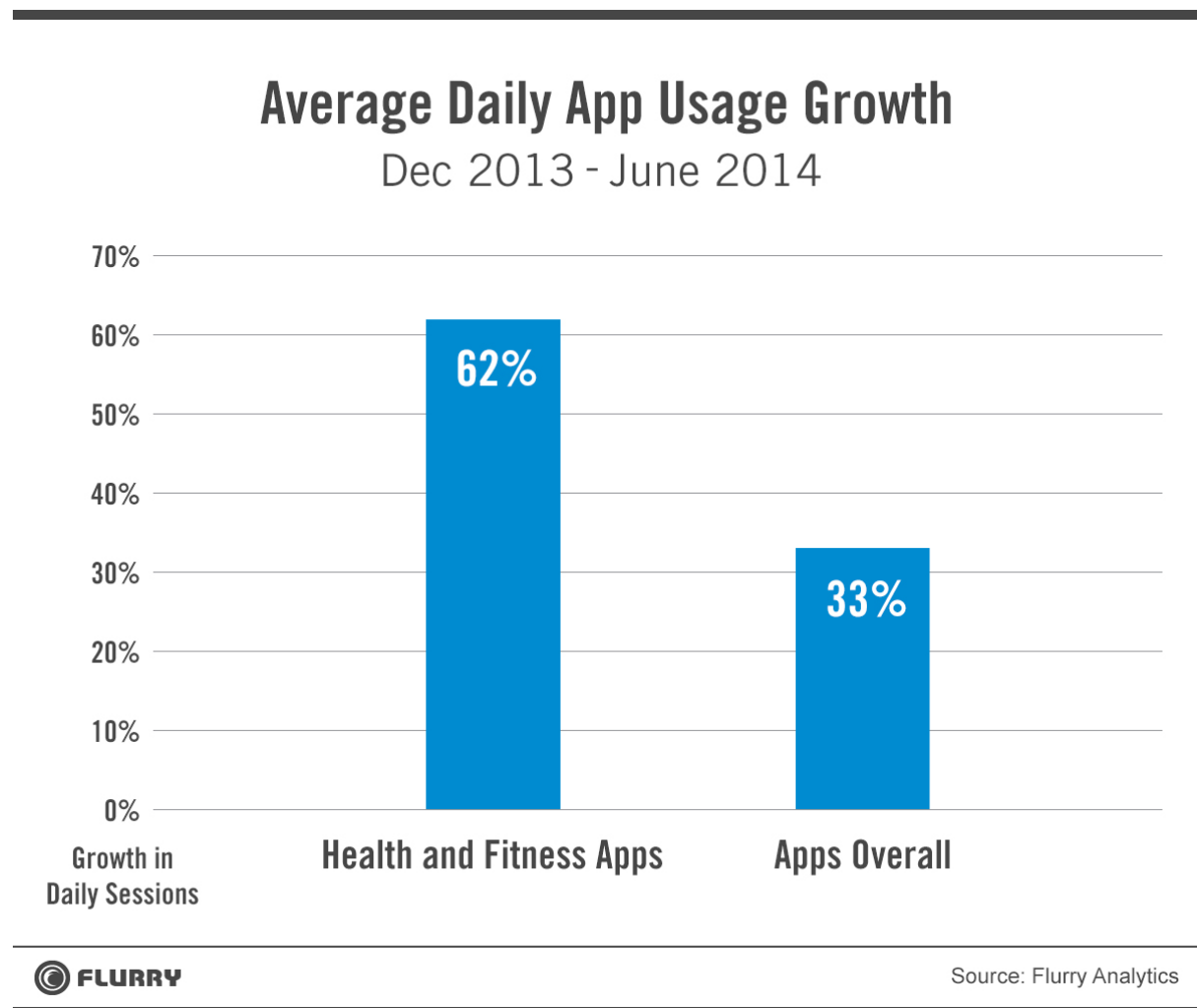
The answer to this is **No**; fitness apps cannot replace human trainers, but they can serve as an assistant to the trainers and the trainees [5]. Using fitness apps cannot guarantee 100% results, but a recent study by [6] says that “*Exercisers who use tech to support workouts consider themselves to be more successful at achieving their weight and fitness goals*”. The sole reason for fitness apps to be successful is their ability to quantify and monitor efforts made by a user. The study also points out that, any visual element like graphs or interactive charts have a direct impact on the way people follow and stick to their daily routine of exercise.

In the recent years the app market has been flooded with thousands of health care and fitness app, but the demand for them is not as high as other lifestyle app. A recent study by [7] reveals that, the app market contains thousands of health care apps that provide sophisticated features of monitoring heart rate and blood pressure, but these apps do not consider specific health constraints. The app-market boasts of thousands of health care apps that have sophisticated features, but none of them consider motivating the users or consider the personal interest of a user. The study suggests that if app developers develop apps that consider the personal interest of the user and motivates them to do exercise, the usage of health care apps will increase rapidly [7] [8].

A lot of health care apps depend on sensors like accelerometer, GPS, microphone etc. for their working; but there always arises a question “**How accurate are these sensors**”. A study by [9] reveals that sensors like GPS are not 100% reliable; consider if the information of the electronic maps are out of date, the GPS can lead you in wrong directions; plus if there are any traffic incidents or road maintenance event in real time, the GPS system will plan a misguided route [9] [10].

With the mobile health care industry booming a lot of people have been using health care apps to monitor their blood pressure, heart rate and several other chronic diseases. Smartphones are playing an important role in people’s lives; today’s smartphones are equipped with hundreds of sensors that can be used to detect diseases before they become life threatening. Smartphones and healthcare apps have made it possible to manage and control chronic diseases by monitoring the health of a person on a regular basis. A recent study has shown that chronic diseases like Asthma and Diabetes insipidus can be prevented if a person’s health is continuously monitored [11] [12].

A recent survey has shown that the usage of Health and Fitness apps has grown by 62% in the year 2014 [\[13\]](#).

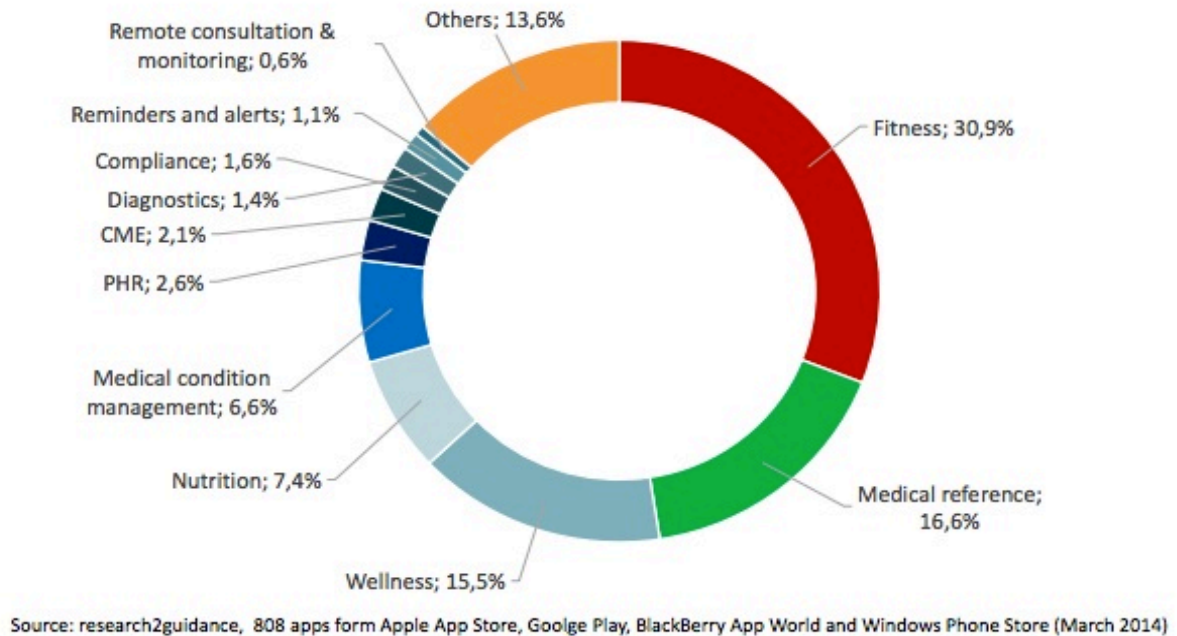


**Figure 3 Average daily usage of Health and Fitness Apps [\[13\]](#)**

A report by Research2Guidance reveals that, in the category of Health and Fitness apps; Fitness apps are the largest mobile health apps and most widely used apps.

## *research2guidance 2: Fitness and Medical reference apps are the largest mHealth app categories*

**mHealth app category share**



**Figure 4**

The health care app market is rapidly growing and it is estimated that by end of 2016 the market for sports and fitness app will cross \$ 400 million. The market for fitness apps has been growing rapidly from \$ 120 million in the year 2010 to \$ 400 million by the end of year 2016 [14]. According to a report by IHS the market for sports and fitness apps will expand by more than 60% in five years. And the installation of sports and fitness apps which was 156 million in 2012 will grow up to 248 million by the end of year 2017 [15].

Fitness apps have been in huge demand and consumers are relying on apps for day-to-day exercises routines and to manage their health and fitness. The market scope for fitness apps is huge with some top health apps generating as many as up to four million free and three hundred thousand paid downloads per day [16].

## 2.2 Background Research

In this modern world, smartphones are playing an important role in people's lives. Today every other person has a smartphone; that can perform some simple task like calling a friend to some complex tasks of Health monitoring.

Day-by-day as more people are becoming health conscious, the demand for health and fitness apps is growing. Sports and fitness apps have become an integral part of our daily lives and millions of people are using them to monitoring their health, to keep track of their daily exercise, to measure the distance covered by a person while walking or running and much more.

The Google play store boasts of thousands of health and fitness apps, of which Cardio Trainer is the best and top rated runner's app in the play store. For an application to be top rated it needs to have something unique which other apps of the same category do not have. And cardio trainer exactly does this; it has a clean, clear, easy to use interface and can accurately measure the speed and distance covered by the user. Cardio trainer is also accurate with its maps and has an internal music player. Despite of having thousands of fitness apps in the play store, most of the best and top rated apps are of paid version, limiting its access to few people. The play store also has some more apps like.

- **Run Keeper:** Run Keeper is the second best fitness app in the play store. It is the master of social networking; if a person joins some runners group on Facebook or twitter they can share and compete with the other members of the group. Despite being in the second place Run Keeper does not have personalization options, cannot calculate the number of calories burned by a person and no built-in music player.
- **Run-Tastic:** Run-Tastic is the third best fitness app in the play store. It is best suitable for cardio exercises like running, biking and hiking. Runtastic has a good and powerful mapping feature. But it also has some drawbacks like no internal music player, limited personalization and you can only view the map of your route after the workout is completed.



- **Calorie Counter:** Calorie Counter is a fitness app that can be used to keep a tab of calories burned by a person. This app has a user-friendly GUI and a large database of food, this food database tells how much calories a particular food item can have. The app can give advice on how much calorie intake a person should have per day.
- **Map My Ride:** Map my Ride is a fitness app that enables you to keep a track of your day-to-day exercise like running and cycling. It has a user-friendly GUI and can keep records of your daily workout including calories burned, distance covered and pace. It also can record the route you had taken while doing your workout. The app also has an in-built functionality that enables it to pause a workout session when a user is held up by a traffic light. This app is highly rich in its features; it can also detect other people near you who are using this same app.
- **Endomondo Sports Tracker PRO:** Endomondo is a paid fitness app that can be used for running, cycling, walking, and kayaking. It has some unique features like interval programs that give the user audio feedback on regular intervals of time. Endomondo is one such app that can even track your heart rate; not just the heart rate it can also display graphical information of the distance you have covered, heart rate, speed and altitude throughout the workout. In Endomondo you can also set workout and calorie goals and the audio coach will guide you through in achieving those goals.

This project aims at developing a smartphone app for runners, as the app was to be developed for a smart phone, it was decided to use Android programming language because android is one of the most popular operating systems that is widely used. The app was developed using IntelliJ Idea and Java programming language. The app was developed using Java because code written in Java is machine independent and is compatible with all the machines. During the development process of this app, the environment made use of API level 17 (i.e. Android version 4.2 Jelly Bean). The app was developed on Android version 4.2, but it also supports all the versions from Android Version 4.1 and above.

## 3 Project Requirements

Requirements are objectives that a project must meet. Requirements can be categorized into functional requirement and non-functional requirements. Functional requirements define a function of a system or its component, whereas non-functional requirement are requirement that specify criteria that can be used to judge the operation of a system, rather than a specific behaviour [\[17\]](#).

### 3.1 Functional Requirements

- **Music Player:** The app has an internal music player, which can play all the audio files that are present on the user's device. The music player also has the functionality of progressively playing music when the app is performing some other activity like displaying maps.
- **GPS Navigation/ Maps:** This app provides the user with maps of the route he is travelling. The app will make use of Google maps and the in-built GPS navigation of the user's device.
- **Leader board:** The leader board will display the history of the user and the details of every session, so that the user can compare each and every session and get motivated to do some more workout. The leader board will display the session time, distance covered and the number of calories burned.
- **Settings:** The app will make use of some user details like weight. The settings button will allow the user to configure all these parameters.
- **Contact Settings:** The app contain a list of email addresses and phone numbers which will be used by the app in case of any mishap with the user (i.e. details of email ids and phone numbers will be used for the distress button).

- **Fitness Monitoring:** The app will display the number of calories burned and the distance covered by a user during a session. The distance covered will be measured using a pedometer.
- **Distress Button:** The Distress button is designed to alert some friends or family members in case of emergency situations. In case if a user meets with a mishap, this button will be of immense help. When a user presses this button the app will automatically send SMS and emails to a predefined list of friends and/or family members, alerting them that the user has met with an accident. The list of friends and family members will be decided by the user of this app.
- **Audio Alerts:** The app has audio alerts which will notify the user when the workout goal has been achieved. The app will also give the user timely (say after every one minute) updates of the distance covered and the calories burned during a session.

### 3.2 Non-Functional Requirements

- **User Friendly GUI:** The app has a simple and easy to use GUI that can be easily understood and operated by any person with ease.
- **Data Format:** All the data of this app like the contact details, email id's is stored in a SQLite database and some other details like weight of the user is stored using shared Preferences.
- **Tools:** The app was developed using IntelliJ Idea; it makes use of Google play services and some support libraries, android support V4 libraries are used so that the app can be used on older versions of android devices as well.
- **Version:** This app supports all android devices with API level 16 and above.

## **4 Design and Implementation**

### **4.1 Development Tools**

The app was developed using IntelliJ Idea 13 and makes use of Google play services to integrate Google maps with this app. The development environment uses latest SDK version for simple and easy development of the app. In order for app to function properly and support most of the versions of android, it makes use of support libraries like “android support v4”.

### **4.2 Software Design**

The runner app has a used friendly GUI that can be operated by any naïve user. The design of this app is done in a simple way and all the components like the buttons are made of reasonable size for easy use. The app consists of a main screen from where the user can either start the app or quit the app.

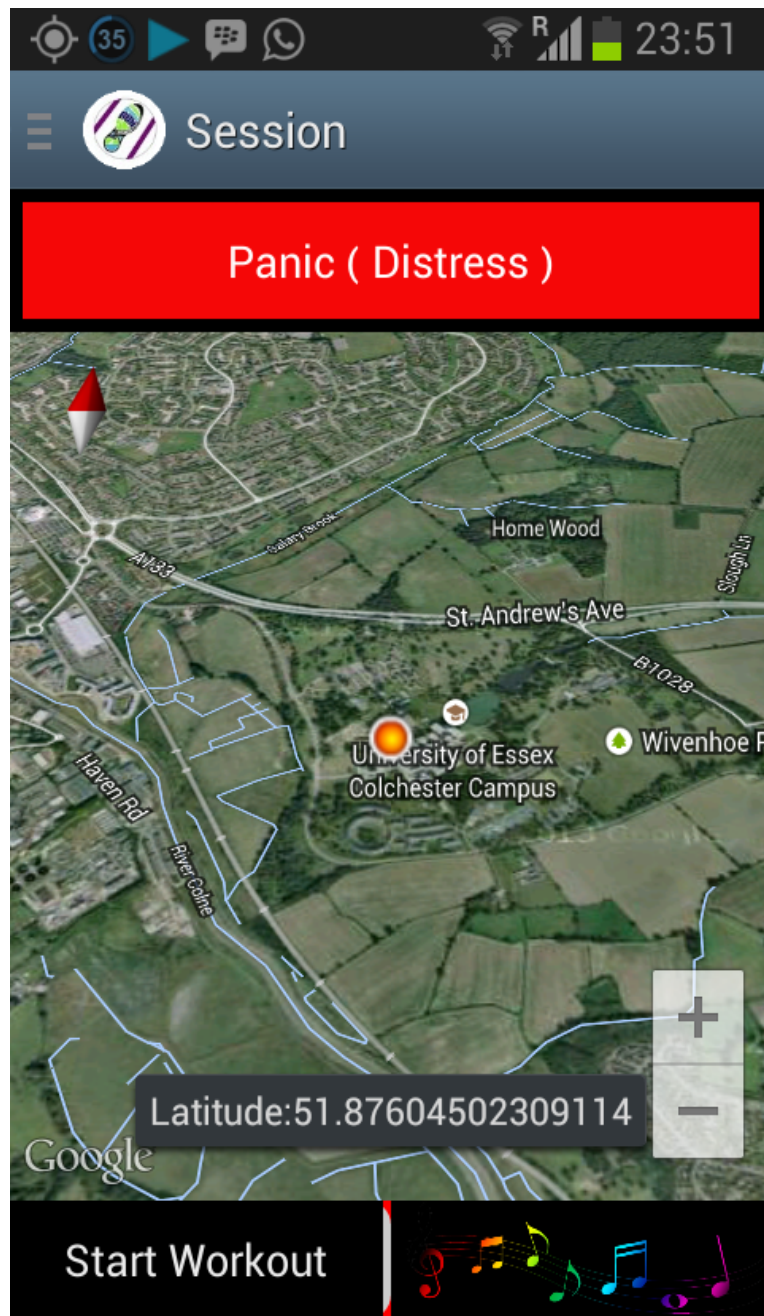
When the user presses the start button he will be redirected to a new page where the user will be able to start a work out session, play music and view history. The quit button will just close the app without making any changes or configurations to the app.

The settings button will allow the user to configure some user details like the height and weight of the user. The settings button allows the user to configure all those parameters that would be required during a workout session.



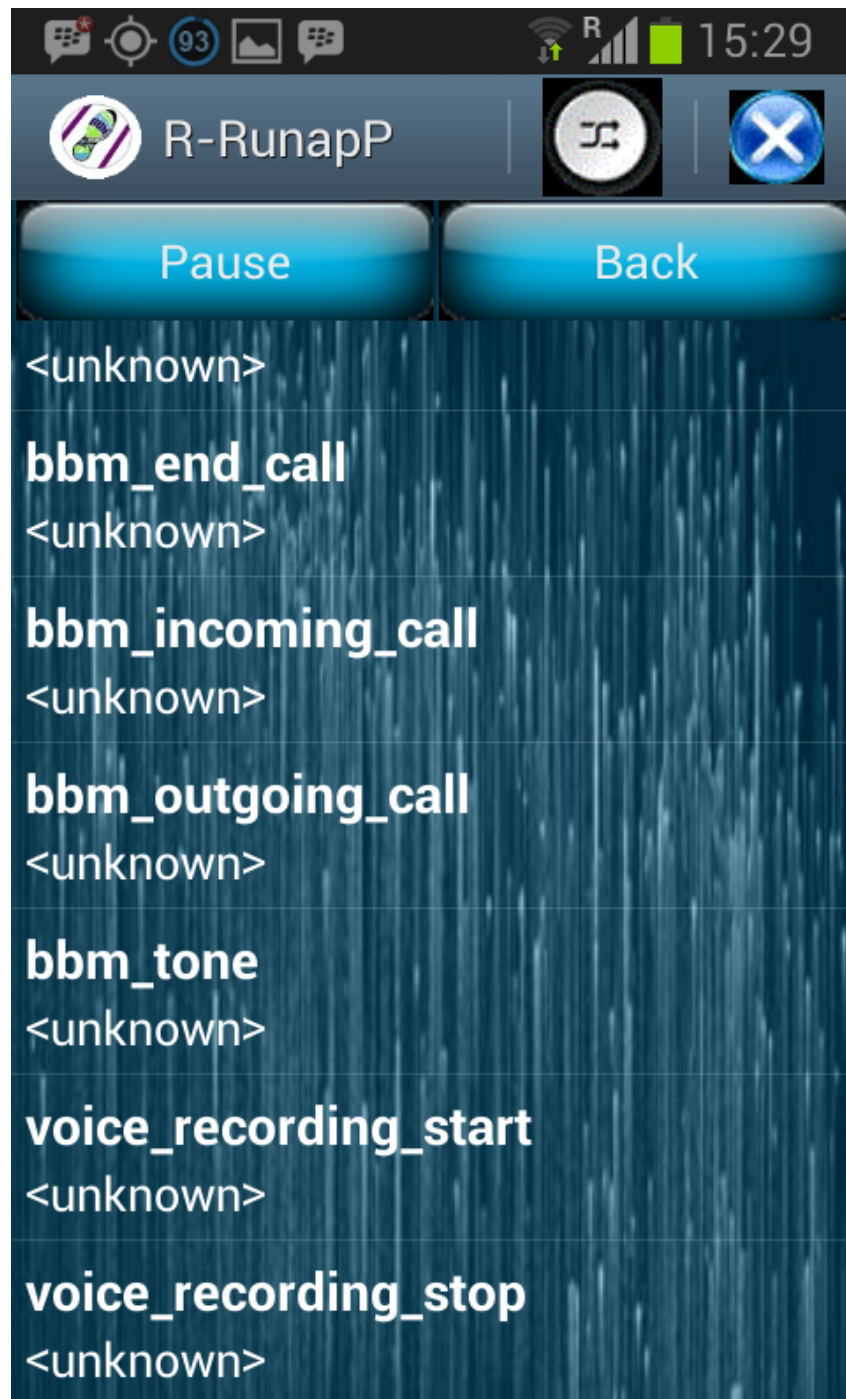
**Figure 5 Starting screen of the app**

Once when the user presses the start button the app will redirect the user to the next page where the user will be displayed with Google maps that display his current location, access to music player, access to the distress button and some more buttons that will enable the user to start a workout session.



**Figure 6 Main screen of the app**

Once when the user is on this page, the GPS will keep on updating the maps and displaying the user's current location. This screen gives access to the distress button and the music player. If the user presses on the music player button he will be redirected to a new page where he will see a list of all the songs that are on the user's device.

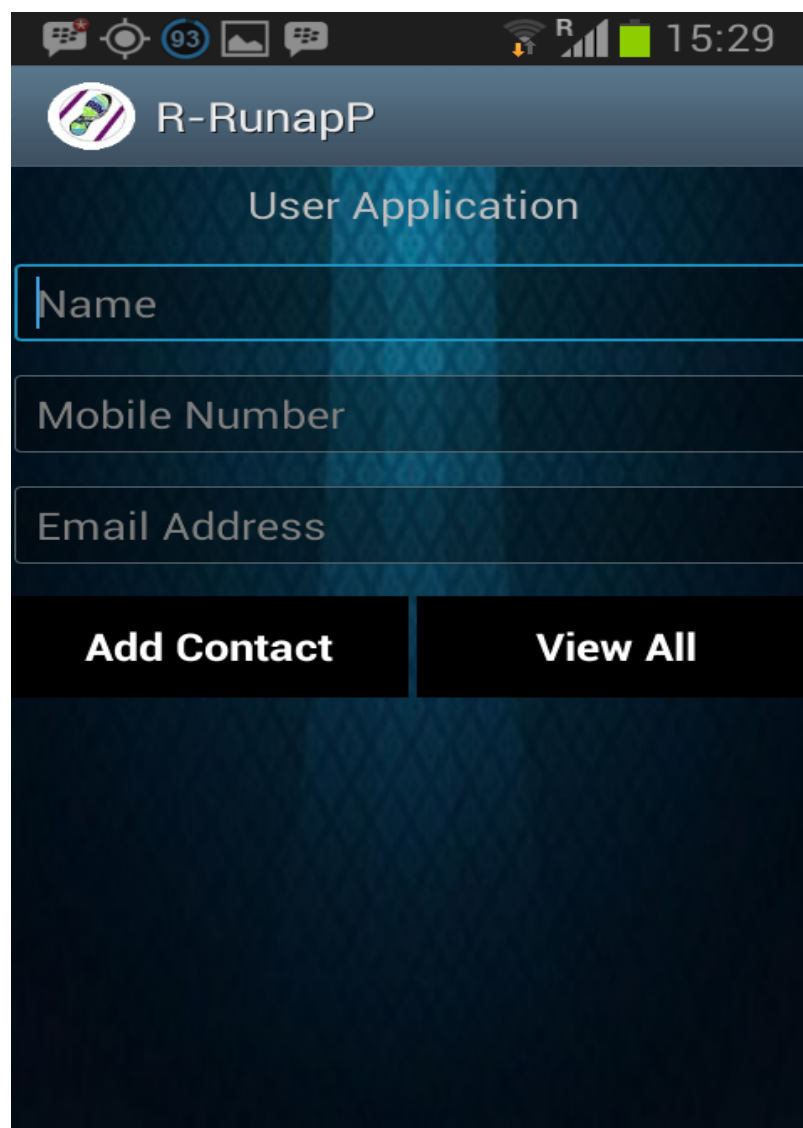


**Figure 7 Screen that will display the list of songs**

Once the user is on this screen, the app will display a list of songs with the title of the song and the name of the artist. Whenever the user wishes to play the song he can just click on that song and it will start playing. When the song is playing the user will be displayed with a software keyboard from where he can fast-forward, skip, pause and stop a song. The screen all provides the user with the pause and back button. The app has a pause button; the app

provides this button because on some devices the software keyboard disappears after a few seconds; so to make it easy for the user the app provides these buttons. The app also has the back button; this button is specially designed to keep the music playing even if the user navigates to some other page. The back button allows the user to navigate to other pages and the music is not paused or stopped.

If the user wants to change some parameters like changing the contact details, he can do this by choosing the settings tab from the main screen. When the user clicks on settings tab the below screen will appear.



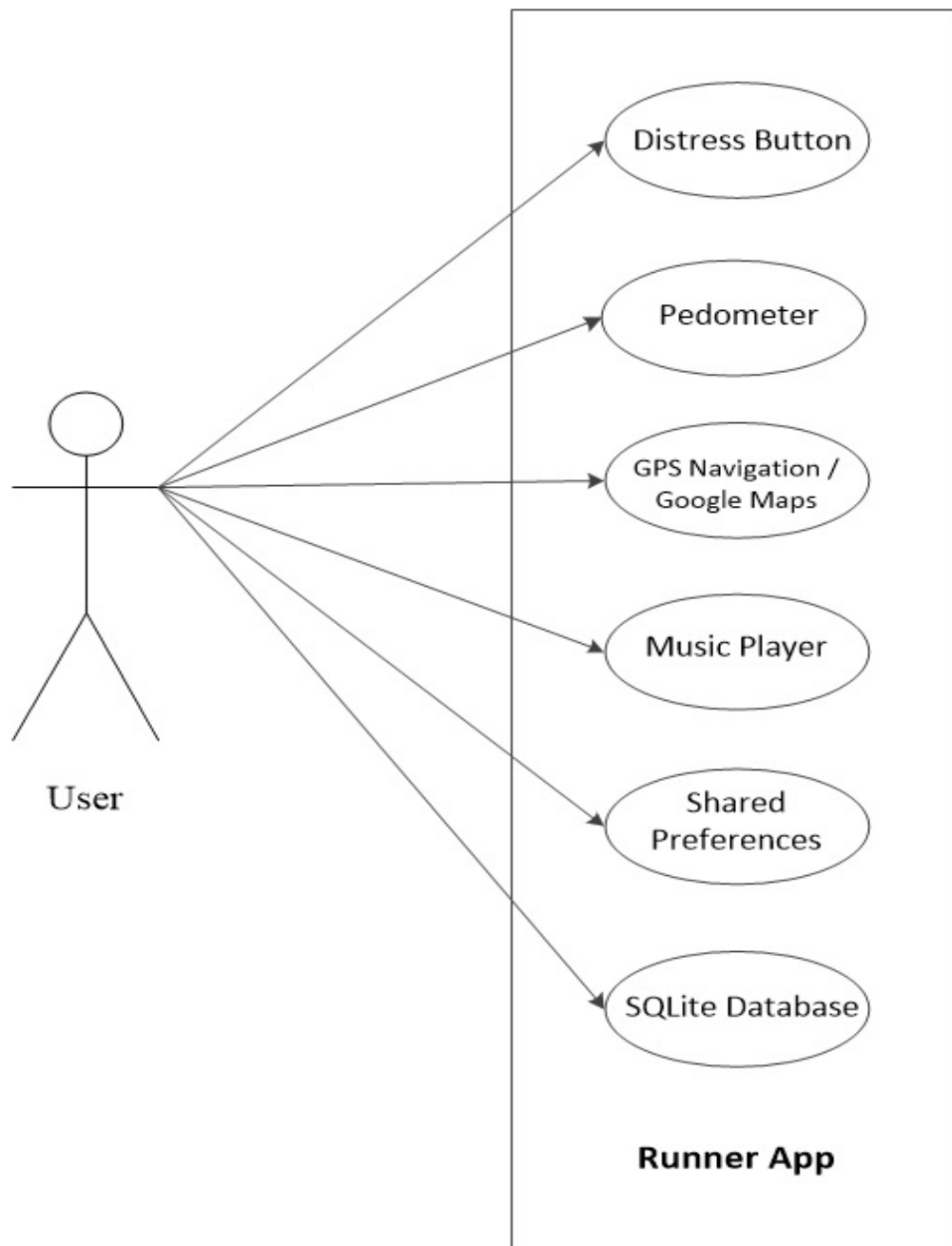
The screenshot shows a mobile application interface for 'R-RunapP'. At the top, there is a status bar with various icons and the time 15:29. Below the status bar is a header with the app's logo and name. The main content area is titled 'User Application' and contains three input fields for 'Name', 'Mobile Number', and 'Email Address'. At the bottom, there are two buttons: 'Add Contact' and 'View All'.

**Figure 8 Contact details screen**



### 4.3 UML Diagram

UML stands for Unified Modelling Language. UML diagrams are used to visualize the design of a system. A use case diagram is used to identify and organise the system requirements. Normally a use case diagram is the interaction between the user and the system [18].



**Figure 9 UML Diagram**

## 4.4 GPS Navigation / Google Maps

The app provides the user with navigation functionality; this is accomplished with the help of the in-built GPS sensor on the user's device and the Google maps. The app makes use of Google maps for navigation; and Google maps can be integrated with the app by using the Google Maps Android API. This runner app makes use of the Google Maps Android API version 2.

In order to get Google maps working with an android app we have to follow some steps. At first we have to install and configure the Google Play Services SDK and then add Google play services version to your app and write some permissions in the android manifest file. After writing the permissions we have to get an android certificate and a Google maps API key. The Google maps API key can be obtained by visiting this link "<http://code.google.com/apis/console>" and making some changes to the services provided by Google. To obtain the API key we first have to click on the "Create Project" button on the webpage that the above link will display. After clicking this button the webpage displays a list of all the services that are provided by Google. From the list, scroll down to the tab where it says "Google Maps Android API v2" and turn on the services of Google maps for android. The next step is to go to the "API Access" tab and click on the "Create new Android key" button, once when you have clicked this button the webpage displays a window where it asks you to enter the SHA certificate fingerprint.

Obtaining a SHA certificate finger print is a simple task. First open your project in the IntelliJ Idea environment and click on the "Build" tab; then from the list select "Generate Signed APK". A new window called "Generate Signed APK Wizard" appears; from this window copy the location of the key store i.e. copy the key store path. Now open the command prompt and copy paste the key store path with the following command "`-list -v -alias androiddebugkey -keystore "C:\Users\xyz\.android\debug.keystore" -storepass android -keypass android`" and press enter. The command prompt will display all the details including the SHA1 key.

Now copy and paste the SHA1 key in the window that was opened to generate the Google Maps Android API key; followed by a semicolon and the package name of your application.

Then click on “create”. And the window will display the API key that we use for our application.

Now that we have obtained the Google maps API key, the next step is to integrate Google maps with our application. For this we first have to open IntelliJ idea and install the Google play services package in our SDK, this can be done via the SDK manager. Once Google play services are installed successfully, we have to import the library files in our application. There are some simple steps that need to be followed in order to accomplish this. At first open your application in IntelliJ Idea environment and then click on the “Project Structure” tab. A new window pops up, from the left of this window select “Modules” and then press on the plus sign (+) on the top part of the window and select “Import Module”. Then navigate to the Google play services folder in your SDK and select “google-play-services\_lib” from the libproject directory of Google play services. Then without closing the window select your application name and then click on the “Dependencies” tab to see your project dependencies. After opening the dependencies tab, click on the plus sign (+) that is on the right of the window and select “Module Dependency”. Then navigate to the Google play services library file and select ok. And from the window press apply button and Google play services are added to the application.

After all the above procedures are followed Google maps are ready to use. In this application the google maps are displayed with the help of a map fragment. Below is a code that is used to display the google maps in this app.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:background="@drawable/map"
    android:layout_height="0dp"
    android:layout_width="match_parent"
    android:layout_weight="5" android:gravity="center">

    <fragment android:layout_width="match_parent" android:layout_height="match_parent"
        android:id="@+id/the_map"
        android:name="com.google.android.gms.maps.MapFragment"
        map:cameraTilt="45"
        map:cameraZoom="14"/>

</LinearLayout>
```

**Figure 10 XML layout file that displays the Google maps in this app**

In this app the user is provided with the current location, wherever he is; this is done by continuously updating the maps that display his location. The maps also display some nearby landmarks like restaurants and hotels that are near to the user's location. Below is a piece of code that is used to get the longitude and latitude information and display the current location on the maps.

```
private void updatePlaces(){

    //update location
    locMan = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    Location lastLoc = locMan.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

    double lat = lastLoc.getLatitude();
    double lng = lastLoc.getLongitude();

    LatLng lastLatLng = new LatLng(lat, lng);

    if(userMarker!=null) {

        userMarker.remove();
    }

    userMarker = theMap.addMarker(new MarkerOptions()
        .position(lastLatLng)
        .title("You are here")
        .icon(BitmapDescriptorFactory.fromResource(userIcon))
        .snippet("Your last recorded location"));

    theMap.animateCamera(CameraUpdateFactory.newLatLng(lastLatLng), 3000, null);

    String placesSearchStr = "https://maps.googleapis.com/maps/api/place/nearbysearch/" +
        "json?location="+lat+", "+lng+
        "&radius=1000&sensor=true" +
        "&types=food|bar|store|museum|art_gallery"+
        "&key=AIzaSyA9gBN_LgaXoRmbL543Pq_rBd8QPgmTUr4";

    new GetPlaces().execute(placesSearchStr);
    locMan.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 30000, 100, this);
}
```

**Figure 11 Method that is used to display the current location of the user**

The above method takes the longitude and latitude information and places a red marker on the current location of the user.

## 4.5 Calorie Calculator

The main aim of this app is to calculate the number of calories burned and distance travelled by a person. As the latest smartphones have got smarter, they have a lot of sensors like proximity sensors, Accelerometer etc. that can be used to calculate the distance travelled by a person. But so far there are no sensors that can calculate how many calories a person has burned during a workout or running.

[19] Provides a way to accurately calculate the number of calories burned by a person during jogging. The number of calories burned can be calculated as *“Take 0.75 x (your weight in pounds). Multiply this total time the number of miles ran, and it accurately indicates total loss. For example, a jogger who weighs 150 pounds and runs four miles would burn roughly 450 calories. We get this number by taking 150 x 0.75, which equals 112.5, then multiplying by four, since the jogger ran four miles”* [19].

## 4.6 Pedometer

The aim of this project is to create a smart phone app for calculating the number of calories a person burns during a particular workout. As we have seen in section 4.5 that due to advancement in technology smartphones have become smarter and have different sensors that can be used to calculate the distance travelled by a person. But unfortunately all these sensors cannot calculate the number of calories a person burns. As [19] provides a way of calculating the number of calories burned, it also requires us to do some calculation, like measuring the steps taken by a user.

In order to count the number of steps taken by a user, modern smartphones use something called Pedometer. A Pedometer makes use of the Accelerometer sensor to count the number of steps a person takes. A Pedometer can be used to count steps while a person is walking,

jogging or running. It can also be used to keep a track of your daily exercise routine and to measure your performance. A pedometer can be used to calculate the number of steps taken by a user, but there arises a question “How accurately can it count the number of steps taken?” [\[20\]](#).

A pedometer cannot accurately count the number of steps taken by a user; a pedometer is not 100% accurate but it provides a rough idea of the number of steps taken by a person. A pedometer has its own limitations; sometimes it may also mistake the action of picking up a phone or putting it down, as a step. So how does a pedometer work? [\[20\]](#)

A Pedometer works in conjunction with the Accelerometer sensor. Whenever we start the app, it keeps on monitoring the changes in the accelerometer sensor and based on these changes it determines whether a step was taken or not. An accelerometer sensor is designed to measure the angle at which a device is held, the direction in which the device is moved, at what speed it is moved and the gravity. A pedometer makes use of all these parameters to determine if a step is taken or not. An accelerometer sensor provides information of three different angles; it measures the changes in the X-axis, Y-axis and Z-axis [\[20\]](#).

Whenever a person takes a step, depending on the force with which he has taken the step the value on the X-axis changes and at the same time the value on the Y-axis also changes. The Y-axis is the height at which the device is moved; say in case the device is kept in a pocket, so whenever a person takes a step his leg goes up, this movement of the device is measured by the Y-axis. Based on the combinations of the X-axis and the Y-axis pattern, a step is identified [\[20\]](#).

With the change in technology, latest versions of android (i.e. Android Kit Kat 4.4) have the feature of step detection built in the platform itself. Android has introduced the feature of step detection in its platform. This introduction of the step counter has made it easy for the programmers, as they don't have to write extra code to detect steps taken by a user [\[20\]](#).

This app also makes use of the accelerometer sensor to detect if the user has taken a step or not. In order to accomplish this task, the class implements a “SensorEventListener” interface. The “SensorEventListener” interface provides the app with methods that give details about the X-axis, Y-axis and the Z-axis. All these combinations and patterns of X-axis, Y-axis and

Z-axis can be used to determine if a step is taken or not. Below is a method that is used to check if a step is taken or not.

```
public void onSensorChanged(SensorEvent event) {

    Sensor sensor = event.sensor;
    synchronized (this) {
        if (sensor.getType() == Sensor.TYPE_ORIENTATION) {
        } else {
            int j = (sensor.getType() == Sensor.TYPE_ACCELEROMETER) ? 1 : 0;
            if (j == 1) {
                float vSum = 0;
                for (int i = 0; i < 3; i++) {
                    final float v = mYOffset + event.values[i] * mScale[j];
                    vSum += v;
                }
                int k = 0;
                float v = vSum / 3;

                float direction = (v > mLastValues[k] ? 1 : (v < mLastValues[k] ? -1 : 0));
                if (direction == -mLastDirections[k]) {
                    // Direction changed
                    int extType = (direction > 0 ? 0 : 1); // minumum or maximum?
                    mLastExtremes[extType][k] = mLastValues[k];
                    float diff = Math.abs(mLastExtremes[extType][k] - mLastExtremes[1 - extType][k]);

                    if (diff > mLimit) {

                        boolean isAlmostAsLargeAsPrevious = diff > (mLastDiff[k] * 2 / 3);
                        boolean isPreviousLargeEnough = mLastDiff[k] > (diff / 3);
                        boolean isNotContra = (mLastMatch != 1 - extType);

                        if (isAlmostAsLargeAsPrevious && isPreviousLargeEnough && isNotContra) {
                            Log.i(TAG, "step");
                            for (StepListener stepListener : mStepListeners) {
                                stepListener.onStep();
                            }
                            mLastMatch = extType;
                        } else {
                            mLastMatch = -1;
                        }
                    }
                    mLastDiff[k] = diff;
                }
                mLastDirections[k] = direction;
                mLastValues[k] = v;
            }
        }
    }
}
```

**Figure 12 Method to calculate the step**

This app also allows the user to adjust the gravity and sensitivity of the accelerometer, so that the app can be used at different location like high terrains and mountains. By adjusting the sensitivity of the sensor, users can put in more efforts to run a distance, which will lead to increase in the stamina and more calories burned.

## **4.7 Timer**

The app consists of a timer that keeps tab of the total time a user spends during a particular workout session. Whenever a user starts a session, the app automatically starts recording the time for that particular session; and the timer stops whenever the user stops a session.

This app makes use of the in-built timer class that is present in “java.util”. The “java.util” contains two different classes that can be used for a timer. Both these classes “java.util.Timer” and “java.util.TimerTask” contains methods that can be used to build a timer.

This app makes use of the TimerTask to calculate or keep a tab of the time taken during a particular session. The TimerTask implements the runnable interface, so if we want to use TimerTask we have to override the “run ()” method. Below is a code that is used in this app to calculate the time taken by a user during a session.



```

public void startTimer(View view) {

    final Handler handler = new Handler();
    Timer ourtimer = new Timer();
    timerTask = new TimerTask() {
        public void run() {
            handler.post(new Runnable() {
                public void run() {
                    TextView timer = (TextView) findViewById(R.id.androidtimer);

                    timer.setText(n + " Seconds");
                    n++;
                }
            });
        }
    };

    ourtimer.schedule(timerTask, 0, 1000);
}

public void stopTimer(View view) {
    timerTask.cancel();
    timerTask = null;
    n = 0;
}

```

**Figure 13 Method that is used to calculate time**

## 4.8 Music Player

The app has an internal music player that allows a user to play music during a workout. The music player in this app has been designed in such a way that if a user starts playing a song and then wants to do some other activity like check the map or change some settings, he can go to those pages and the music will still be playing in the background.

The music player in this app plays all the songs that are on the user's device. The music player uses the "ContentResolver" class to retrieve all the songs that are on user's device and displays them on the screen. The app also makes use of the "MediaPlayer" class to play the music and "MediaController" class to control the playback.

The app uses “ContentResolver” class and “Cursor” class to retrieve audio file from the user’s device and show them to the user in a list view. Below is a method that retrieves all the audio files that are on the user’s device.

```
public void getSongList(){  
  
    //query external audio  
    ContentResolver musicResolver = getContentResolver();  
    Uri musicUri = android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;  
    Cursor musicCursor = musicResolver.query(musicUri, null, null, null, null);  
    //iterate over results if valid  
  
    if(musicCursor!=null && musicCursor.moveToFirst()){  
        //get columns  
        int titleColumn = musicCursor.getColumnIndex  
            (android.provider.MediaStore.Audio.Media.TITLE);  
        int idColumn = musicCursor.getColumnIndex  
            (android.provider.MediaStore.Audio.Media._ID);  
        int artistColumn = musicCursor.getColumnIndex  
            (android.provider.MediaStore.Audio.Media.ARTIST);  
        //add songs to list  
        do {  
            long thisId = musicCursor.getLong(idColumn);  
            String thisTitle = musicCursor.getString(titleColumn);  
            String thisArtist = musicCursor.getString(artistColumn);  
            songList.add(new Song(thisId, thisTitle, thisArtist));  
        }  
        while (musicCursor.moveToNext());  
    }  
}
```

**Figure 14 Method to retrieve audio files from user’s device**

After retrieving all the audio files from the user’s device we use the “BaseAdapter” class in order to display the audio files that were just retrieved. But before displaying the songs to the user, this app sorts all the songs in an alphabetical order and this is done by comparing the titles of these songs. Below is a code snippet that is used to sort all the songs in an alphabetical order.

```
//sort alphabetically by title
Collections.sort(songList, new Comparator<Song>() {
    public int compare(Song a, Song b){
        return a.getTitle().compareTo(b.getTitle());
    }
});
```

**Figure 15 Code that is used to sort the audio file in alphabetical order**

After retrieving the audio files and sorting them alphabetically, all these songs are presented to the user in a list view. And this is done by using the “BaseAdapter” class.

```
//create and set adapter
SongAdapter songAdt = new SongAdapter(this, songList);
songView.setAdapter(songAdt);
```

**Figure 16 Code that is used to display the songs list in a list view**

Now that we have retrieved the songs and displayed them to the user, we now work on how to play the songs that the user desires to play. This is accomplished using the “MediaPlayer” class. Below is a code snippet that is used to play the audio file.

```
@Override
public void onPrepared(MediaPlayer mp) {
    //start playback
    mp.start();
}
```

**Figure 17 Code to start playing an audio file**

Now the app can retrieve audio file from a user’s device, display list of songs to the user and play the audio file. But an actual music player has lot more functionalities than this, like fast-forwarding a song, playing any song at random and much more. All these functionalities of a music player are accomplished using the “MediaController” class. The media controller class consists of a standard widget for a music player with buttons like play/pause, fast-forward,

rewind and skip. After implementing the “MediaController” method, we use the “PendingIntent” class in order for the app to continue playback when the user has navigated to some other activity. Below is the code that is used to continue playback in case user navigates to some other activity.

```
@Override
public void onPrepared(MediaPlayer mp) {
    //start playback
    mp.start();
    //notification
    Intent notIntent = new Intent(this, MainActivity.class);
    notIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    PendingIntent pendInt = PendingIntent.getActivity(this, 0,
        notIntent, PendingIntent.FLAG_UPDATE_CURRENT);

    Notification.Builder builder = new Notification.Builder(this);

    builder.setContentIntent(pendInt)
        .setSmallIcon(R.drawable.play)
        .setTicker(songTitle)
        .setOngoing(true)
        .setContentTitle("Playing")
        .setContentText(songTitle);
    Notification not = builder.build();
    startForeground(NOTIFY_ID, not);
}
```

**Figure 18** Code to keep the song playing when user has navigated to some other activity

## 4.9 Distress Button

The distress button is a unique concept for a fitness app and of utmost significance. The distress button is used in case of mishap, whenever a user meets with any mishap and presses this button, the app will automatically send emails and SMS to a list of friends and/or family members alerting them that the user has met with a mishap and they should come and help. The app allows the user to store contact details and email addresses of any five people, who

should be contacted in case of mishap. All these contact details and email addresses are stored in a SQLite database. When the distress button is pressed the app fetches all the contact numbers and email addresses from the database and forwards SMS and Emails to those contacts. Below is a code that is used to send SMS to a friend and/or family member.

```
public void sendSMS() {
    String SENT = "Message Sent.";
    String DELIVERED = "Message Delivered.";

    try {

        PendingIntent sentPI = PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
        PendingIntent deliveredPI = PendingIntent.getBroadcast(this, 0, new Intent(DELIVERED), 0);

        registerReceiver((context, intent) -> {

            switch (getResultCode()) {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS Sent.", Toast.LENGTH_LONG).show();
                    break;

                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                    Toast.makeText(getBaseContext(), "Generic Failure", Toast.LENGTH_LONG).show();
                    break;

                case SmsManager.RESULT_ERROR_NO_SERVICE:
                    Toast.makeText(getBaseContext(), "No Service", Toast.LENGTH_LONG).show();
                    break;
            }

        }, new IntentFilter(SENT));

        registerReceiver((context, intent) -> {

            switch (getResultCode()) {
                case Activity.RESULT_OK:
                    Toast.makeText(getBaseContext(), "SMS Delivered.", Toast.LENGTH_LONG).show();
                    break;

                case Activity.RESULT_CANCELED:
                    Toast.makeText(getBaseContext(), "SMS not Delivered.", Toast.LENGTH_LONG).show();
                    break;
            }

        }, new IntentFilter(DELIVERED));

        SmsManager sms = SmsManager.getDefault();

        sms.sendTextMessage(theNumber, null, theMessage, sentPI, deliveredPI);
    }
}
```

**Figure 19 Method that is used to send SMS**



In order to accomplish the task of sending SMS, we have to make some configurations and write some permission in the android manifest file. These permissions will allow the device to send text message to other device. In the manifest file we have to write the following permission “<uses-permission android: name=”android.permission.SEND\_SMS”/>”. After writing this permission, we now use the “SmsManager” class in order to send SMS from our application. The “SmsManager” class is used to send text messages from our application to other devices.

When the user presses the distress button, the app automatically sends SMS and emails to a list of people. As we have seen that the app uses “SmsManager” to send text message; the procedure for sending email from this app is slightly different. Whenever the user presses the distress button the app call the “Screenshot” method, this method is used to take a screenshot of the layout file on which the Google maps are displayed. The app takes a screenshot of these maps so that it can attach them to an email and send it to those people who should be contacted in case of any accident. With the help of this screenshot the friends and/or family members will know the exact location of the user and thus it will be easy for them to find the user and medical support could be provided as quickly as possible. Below is a code snippet that is used to take a screenshot of the Google maps.

```
public void Screenshot(View view) {  
  
    View v1 = getWindow().getDecorView().getRootView();  
    v1.setDrawingCacheEnabled(true);  
    myBitmap = v1.getDrawingCache();  
    saveBitmap(myBitmap);  
  
}
```

**Figure 20 Method that is used to take a screenshot**

After taking the screenshot, the app converts the screenshot into a Bitmap image and saves it to the external storage of the device. Once when the image is converted and saved to an external storage the “Screenshot” method calls the “sendEmail” method in order to forward

the image to the concerned people. Below is the code that is used to convert and save the screenshot as a bitmap image.

```
private void saveBitmap(Bitmap myBitmap) {

    String filePath = Environment.getExternalStorageDirectory() + File.separator + "Pictures/screenshot.png";
    File imagePath = new File(filePath);
    FileOutputStream fos;

    try {
        fos = new FileOutputStream(imagePath);
        myBitmap.compress(Bitmap.CompressFormat.PNG, 100, fos);
        fos.flush();
        fos.close();

        sendEmail(filePath, message);
    } catch (FileNotFoundException e) {
        Log.e("GREC", e.getMessage(), e);
    } catch (IOException e) {
        Log.e("GREC", e.getMessage(), e);
    }
}
```

**Figure 21 Method that is used to convert and save the screenshot into bitmap image**

When the “sendEmail ()” method is called, at first it retrieves all the email addresses and then finds the location where the bitmap image is saved. Once the location is known it then begins the task of composing email to the respective people. Below is the code that is used to send email.

```
public void sendEmail(String path, String message) {
    try {
        String[] to = new String[]{"roston.pereira9@gmail.com"};
        String subject = ("A message from your App.");
        Intent emailIntent = new Intent(Intent.ACTION_SEND);
        emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
        emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
        emailIntent.putExtra(Intent.EXTRA_TEXT, message);
        emailIntent.setType("image/png");
        Uri myUri = Uri.parse("file://" + path);
        emailIntent.putExtra(Intent.EXTRA_STREAM, myUri);

        startActivity(Intent.createChooser(emailIntent, "Email"));
    } catch (Exception e) {
    }
}
```

**Figure 22 Code that is used to send email during mishap**

## 4.10 Database

A database is a collection of information that is organised in orderly format, particularly in tabular form. The app makes use of SQLite database to store the contact details and email addresses of the friends and/or family members of the user. The app stores name of the person to be contacted, his/her phone number and their email address; which will be used in case of any accident.

SQLite is an open source relational database that supports standard SQL queries. For using SQLite in our application we need to import two main packages i.e. the “android.database” package and “android.database.sqlite” package. The “android.database” package contains all the classes that are necessary to work with SQLite and the “android.database.sqlite” package contains all the SQLite specific classes [\[21\]](#).

Android applications make use of the “SQLiteOpenHelper” class in order to create or update a database. In case an application needs to create a database this is done by overriding the “onCreate” method in a class. And if the application requires upgrading a database, this is done by overriding the “onUpgrade” method. The SQLiteOpenHelper” class contains two methods called “getReadableDatabase” and “getWritableDatabase” that gives access to a SQLite Database object, which in-turn could be used to read or write [\[21\]](#).

The SQLite database supports methods like insert (), update () and delete (), by which a user can store or delete any data from the database. In addition the SQLite database also supports methods like “execSQL ()”, which allow us to execute a SQL statement directly. And in order to insert or update any database entries, we use the “ContentValues” object. The “ContentValues” object represents a key value pair, where key represents the column value identifier and value represents the content that is to be inserted in the database. To insert data into a database we use queries and SQLite provides three different ways in which we can create queries. In SQLite we can use the “rawQuery ()” or “query ()” method to create a query. SQLite also provides a third option of using the “SQLiteQueryBuilder” class to create queries. The “SQLiteQueryBuilder” class is a convenience class that can be used to build SQL queries, whereas the “rawQuery ()” method accepts SQL statements as input and the “query ()” method provides a structured interface for writing the SQL queries [\[21\]](#).



In SQLite database whenever a query is executed the results are retrieved via a Cursor object. A cursor object is similar to a “ResultSet” object in SQL. SQLite provides almost the same methods and objects as in SQL to retrieve or insert data. In case if a user needs to know the number of entries in a table, that is possible by using the “getCount ()” method. SQLite also provides some other methods that can be used to navigate between different rows of a database table. “moveToFirst” and “moveToNext” methods can be used to navigate between rows of a table in SQLite. A cursor also allows us to get the column index for a column in a SQLite database [21].

In SQLite a data access object is responsible for accessing and modifying data in a table. A data access object can also be used to open and close the connections to a database. Normally in SQLite, a database is private to an application i.e. only a particular application can access its own database, no other application can access that database. But SQLite also provides a way by which one application can access data of another application and this can be done using the content provider. A content provider is usually used to share data with other applications and this is accomplished using structured interface. A content provider can be accessed via a URI, but if an application needs to create its own content provider this can be done by extending a class with “android.content.ContentProvider”. After extending a class with “android.content.ContentProvider”, it is also necessary to declare the content provider in manifest file. Below is a code that shows how to declare a content provider in the manifest file [21].

```
<provider
    android:authorities="de.vogella.android.todos.contentprovider"
    android:name=".contentprovider.MyTodoContentProvider" >
</provider>
```

**Figure 23 Declaring Content Provider in the manifest files [21]**

SQLite provides a Loader class that allows you to load data in a fragment or activity asynchronously. It can also be used to monitor the source of data. To implement a loader class we can make use of the abstract class “AsyncTaskLoader”; the “AsyncTaskLoader” acts as a basis for implementing your own loader class [21].

SQLite provides a “CursorLoader” class to handle the database connections. And whenever a Cursor loader goes out of scope, the “onLoaderReset” method is called. One of the important aspects of a cursor is its opening and closing; this has to be done considering the lifecycle of a component otherwise it may lead to errors and app failure. This failure can be avoided by using the “managedQuery” method [21].

This app makes use of SQLite database to store and access contact details of the people who should be contacted in case of the user meeting with any accidents. Below is a code that is used to save contact details in the SQLite database.

```
// Adding new contact
public void Add_Contact(Contact contact) {

    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName()); // Contact Name
    values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone
    values.put(KEY_EMAIL, contact.getEmail()); // Contact Email
    // Inserting Row
    db.insert(TABLE_CONTACTS, null, values);
    db.close(); // Closing database connection
}
```

**Figure 24 Method to save contact details in SQLite database**

Whenever the user presses the distress button the app will automatically retrieve data from the database and send SMS and emails to those people. The retrieving of data in this app is done in the following way.

```

// Getting single contact
Contact Get_Contact(int id) {

    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_CONTACTS, new String[]{KEY_ID,
        KEY_NAME, KEY_PH_NO, KEY_EMAIL}, KEY_ID + "=?",
        new String[]{String.valueOf(id)}, null, null, null, null);
    if (cursor != null)
        cursor.moveToFirst();

    Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
        cursor.getString(1), cursor.getString(2), cursor.getString(3));
    // return contact
    cursor.close();
    db.close();

    return contact;
}

```

**Figure 25 Method that is used to retrieve data from the SQLite database**

In order to make the app user friendly it has been given a simple GUI. The app allows the user to update details in the contact list; in case if a particular person on that list got a new phone number; the user can edit the contact details for that person and store the new phone number. Updating of contact details is done in the following way.

```

// Updating single contact
public int Update_Contact(Contact contact) {

    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());
    values.put(KEY_EMAIL, contact.getEmail());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
        new String[]{String.valueOf(contact.getID())});
}

```

**Figure 26 Method that is used to update contact details in the database**

The app also allows the user to delete a contact permanently from the list. And this task is accomplished by the following method.

```
// Deleting single contact
public void Delete_Contact(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_CONTACTS, KEY_ID + " = ?",
        new String[]{String.valueOf(id)});
    db.close();
}
```

**Figure 27 Method that is used to delete data from the database**

The app uses shared preferences to store the user's details like the weight. Shared Preferences are another way of storing data in android and they can store preferences at activity level or application level. Data stored in shared preferences remain for a long time i.e. the data does not get deleted even if the user closes the application or the device is switched off [22]. A single shared preference is just a key-value pair, where the key is a string that identifies the preference and the value is what has to be stored in it [23]. The value or data that is saved in shared preference can have different data types like Boolean, float, int, long and string [22]. In order to get Shared Preferences working properly we have to import the package "android.Content". The "android.Content" package contains an interface called "SharedPreferences" that has all the necessary classes that are required for Shared preference.

Shared Preferences can store preferences at activity level or application level. Preferences that are stored at application level can be accessed by all the activities, whereas preferences that are stored at activity level can only be accessed by that particular activity. Application level preferences can be retrieved by using the "getSharedPreferences" method that is present in the "android.content.SharedPreferences" package. Below is a code that demonstrates how to retrieve preference from an application level shared preference [23].

```
import android.content.SharedPreferences;
...
SharedPreferences settings =
    getSharedPreferences("MyGamePreferences", MODE_PRIVATE);
```

**Figure 28 Retrieving preferences from application level shared preference [23]**

Shared Preferences can be stored at activity level and application level. An activity can also have a private preference, which is only accessible to that particular activity. An activity can have only one set of private preference. Below is a code that demonstrates how to retrieve a preference from a private activity [\[23\]](#).

```
import android.content.SharedPreferences;
...
SharedPreferences settingsActivity = getPreferences(MODE_PRIVATE);
```

**Figure 29 Retrieving preferences from Private Preferences [\[23\]](#)**

Once data is retrieved from a shared preference it can be edited by using edit method. The “Editor” class that is available in “SharedPreferences.Editor” contains methods that can be used to add, modify or delete preference content. The Editor class has following methods to store preferences of different data types [\[23\]](#).

- Store boolean values with the `putBoolean()` method
- Store float values with the `putFloat()` method
- Store int values with the `putInt()` method
- Store long values with the `putLong()` method
- Store String values with the `putString()` method

**Figure 30 Methods in Editor Class that can be used to insert data [\[23\]](#)**

The Editor class also contains methods to remove preferences. The “remove ()” method is used to remove a specific preference by name and “clear ()” method is used to remove all the preferences. Once all the editing is done the changes can be saved using the “commit ()” method that is available in the Editor class [\[23\]](#).

Updating a preference is a simple task. Below is a code that shows how to modify a “PaidUser” preference.



```

SharedPreferences gameSettings = getSharedPreferences("MyGam
SharedPreferences.Editor prefEditor = gameSettings.edit();
prefEditor.putBoolean("PaidUser", true);
prefEditor.commit();

```

**Figure 31 Updating Shared Preference [\[23\]](#)**

Retrieving a preference is slightly different from saving or updating a preference. A shared preference can be retrieved by retrieving a Shared Preference object and then using that object to retrieve the preference by name. Below is an image that shows how preferences of different data types can be retrieved [\[23\]](#).

```

SharedPreferences gameSettings = getSharedPreferences("MyGam
SharedPreferences.Editor prefEditor = gameSettings.edit();
prefEditor.putBoolean("PaidUser", true);
prefEditor.commit();

```

**Figure 32 Retrieving Shared Preference [\[23\]](#)**

Below is a code snippet that is used in the app to set a desired pace.

```

private void setDesiredPaceOrSpeed(float desiredPaceOrSpeed) {
    if (mService != null) {
        if (mMaintain == PedometerSettings.M_PACE) {
            mService.setDesiredPace((int)desiredPaceOrSpeed);
        }
        else
        if (mMaintain == PedometerSettings.M_SPEED) {
            mService.setDesiredSpeed(desiredPaceOrSpeed);
        }
    }
}

```

**Figure 33 Saving data to a Shared Preference**

# 5 Testing

Testing is an important phase of the software development lifecycle; because this phase evaluates whether the app was a success or failure. Testing the app thoroughly using different testing techniques will expose it for any bugs or errors. A project can only be considered as a success if the app is error free. Testing an app under different conditions will ensure that the app is compatible and error free. This project makes use of different test methodologies; all these methodologies have their own limitations, so an app cannot be tested to be 100% bug free.

Testing for this app was done in two different phases. In the first phase the app was unit tested before integrating all the components of the app. And in the second phase the app was integrated with all the components and tested using black box testing. The first phase of testing was carried out to ensure that each and every method of a class was giving the desired results. And the second phase tested the full app for any errors or bug and most importantly to check if all the components were working in co-ordination with each other. So after integrating the app it was tested using black box testing and it was also deployed on different android devices for compatibility testing. Due to time constraints the app could not be tested using white-box testing.

## 5.1 Black Box Testing

Black-Box testing is a testing technique that is used to test the functionality of software without getting into the internal structure of the software. A black-box tester need not have extensive programming knowledge. Black-box testing can be used in any phase of the software development lifecycle. It can be used from the unit testing level till the acceptance testing level. Black-box testing can be done by any person if he knows what exactly the software has to do, without getting into the internal structure of the software.

<b>Test</b>	<b>Inputs</b>	<b>Expected Output</b>	<b>Actual Output</b>
Start Button and Google maps (With GPS and WIFI enabled)	Start Button Pressed	Takes user to the main screen and shows the current location of the user on maps.	User is directed to main screen and the app shows current location of the user on maps.
Start Button and Google maps (With GPS enabled and WIFI disabled)	Start Button Pressed	Takes user to the main screen and displays a toast message telling to switch on the WIFI or data connection.	User is directed to main screen and the app shows a toast message alerting the user to Switch on Internet.
Start Button and Google maps (With GPS disabled and WIFI enabled)	Start Button Pressed	Takes user to the main screen and displays a toast message telling to switch on the GPS.	The app shows an error.
Start Button and Google maps (With GPS and WIFI disabled)	Start Button Pressed	Takes user to the main screen and displays a toast message telling to switch on the WIFI or data connection and GPS.	The app shows an error.
Distress Button (With WIFI enabled)	Distress Button Pressed	Sends SMS and emails to friends.	App sends SMS and Email.
Distress Button (With WIFI disabled)	Distress Button Pressed	Sends SMS and displays a toast message to switch on the WIFI or data connection.	App sends SMS and displays a toast to switch on WIFI.
Music Button (With Audio files present on device)	Music Button Pressed	Takes the user to a next page that displays a list of songs.	Redirects user to the Music Players page and displays a list of Songs.
Music Button (With no Audio files on device)	Music Button Pressed	Takes the user to the page that displays a list of songs, but without any songs in the list.	Redirects user to the Music Players page but does not show a list of songs.



Play Song	Click on the Desired Song	The app plays the song.	Plays the Audio file.
Pause Song	Pause Button Pressed	Song stops playing.	Pause's the Audio file.
Shuffle Button	Shuffle Button Pressed	Plays any song at random.	Plays songs are random.
Skip Song Button	Skip Song Button Pressed	Skips a song and plays the next song that is on the list.	Skips a song.
Fast-Forward Song Button	Fast-Forward Song Button Pressed	Song is fast-forwarded.	Fast-forwards a song and continues to play it.
Back Button (With Song Playing)	Back Button Pressed	Takes user to the main screen, while playing music in the background.	Takes user to main screen, with song playing in the background.
Back Button (Without Song Playing)	Back Button Pressed	Takes user to the main screen.	Takes user to main screen.
Start Session Button	Start Session Button Pressed	Takes user to a new page that displays the number of calories burned, steps taken and distance covered.	Starts a session, and calculates the number of calories burned and distance covered.
Pause Session Button	Pause Session Button Pressed	Stops counting the steps and distance	Freezes all the calculations.
Reset Session Data Button	Reset Session Data Button Pressed	Reset all the parameters like calories burned and distance. Reset button changes all these parameters to 0.	Makes all the parameters as 0.
Session Settings Button	Session Settings Button Pressed	Displays a list of all possible settings that can be changed.	Displays a settings page.
User Input	Body Weight (Invalid Value)	Throw an error.	Accepts all the values.
User Input	Body Weight (Valid Value)	Calculates the number of calories burned.	Accepts the values and calculates the number

			of calories burned during workout session.
Back Button	Back Button Pressed	Takes user to the main screen.	Redirects user to the main screen.
Connections Tab	Connections Tab Pressed	Takes user to a screen where he can see a list of contacts.	Redirects user to a new screen allowing edit or delete contact details.
User Input	Name of Contact Person (Valid Input)	Accepts input and takes the cursor to the next field.	Accepts input and saves in database.
User Input	Name of Contact Person (Invalid Input)	Displays a message saying that only alphabets are allowed.	Shows Invalid input message.
User Input	Mobile number of Contact Person (Valid Input)	Accepts input and takes the cursor to the next field.	Accepts input and saves in database
User Input	Name of Contact Person (Invalid Input)	Displays a message saying that only numbers are allowed.	Shows Invalid input message.
User Input	Email address of Contact Person (Valid Input)	Accepts input and takes the cursor to the Add contact button.	Accepts input and saves in database
User Input	Email address of Contact Person (Invalid Input)	Displays a message saying invalid email address.	Shows Invalid input message.

Table (2)

## 5.2 Compatibility Testing

This phase of testing is important and necessary to ensure that the app is stable and supports most of the android devices. Compatibility testing was carried out to ensure that the app is supported by different android devices. The app was installed and checked on some android devices; the results showed that this app is compatible with all the android devices that have

an API level of 16 and above. The app contains some code that are only supported by android version 3.0 and above (i.e. API level 16 and above).

After testing the app on different devices it was noted that the app could maintain the same appearance or GUI on all the devices. During testing on different devices I came across a problem where the software keyboard disappears after a few second. On some devices it was noted that the software keyboard that is used to control the music player, disappears after few seconds. In order to tackle this problem a few more buttons were added to the music player's page, by which the user could pause the song without having to wait for the software keyboard to display buttons.

So with compatibility testing it was concluded that the GUI of the app appears same on all the devices and the app supports all the devices that have an API level of 16 and above.

# 6 Conclusion

## 6.1 Project Evaluation

Project evaluation is a technique that gives us accountability and knowledge of a project. Accountability of a project tells us whether a project was relevant or not and if it has achieved its target. Accountability of a project gives scope for further enhancement of the project. And knowledge is what we gain from that project.

The success of this project can only be evaluated if the app is fully functional and gives the desired output. This project was aimed at creating an app that can calculate the number of calories a user burns and the distance covered. So far the app was success; I was able to get Google maps, music player and the distress button working successfully. The app can also save contact details in its SQLite database.

The app has a pedometer that counts the number of steps taken and the calories burned; but the results for step counting are not 100% accurate and the design of the app could not be maintained as decided because of some integration issues. But overall the app was success and during the development process of this app, I got to learn a lot of new things; especially the Google maps. This project taught me how to manage my time effectively and prioritize different task; I also gained an in-depth knowledge of android programming.

## 6.2 Project Management

The aim of this project was to develop an android app that can be used to calculate the distance travelled by a person and the number of calories burned while covering that distance. The app in its nature was big and challenging as it required implementing many additional features like music player and GPS Navigation to make this app the best one.

In order to successfully complete this app I decided to use the agile software development approach with some extensive programming practices. The project makes use of agile

methodology because many of the features in this app depend on each other to be fully functional.

The project was big and had to be completed within limited time. So it was necessary to manage my time effectively, as I had limited knowledge of developing android applications. In order to complete the project on time it was necessary to draft a schedule, this was accomplished by using a Gantt chart. Before starting the project I drafted a Gantt chart which had detailed information of when to start the project, which task should be started by what date and by what date should be the full project get completed.

During the implementation of this project, agile software development methodology was used and efforts were made to stick to the plans that were drafted on the Gantt chart. But during the development process there were some features that took more time to complete compared to the time that was scheduled on the Gantt chart.

During the process of building this app I faced many challenges, but all this challenges were worth it, as it gave me an opportunity to acquire some significant knowledge of android programming. It also thought me how to manage my time effectively and gave me an in-depth knowledge of Google maps.

### **6.3 Future Work**

The aim of this project was to develop a smart phone that could calculate the number of calories burned by a user and an app that had a user friendly GUI. These tasks were accomplished successfully and the app can work on all android devices that have an API level of 16 and above. The future work for this app would be to add some more features and enhance the existing features to make this app the best one. Below is a list of possible features that can be added in the future.

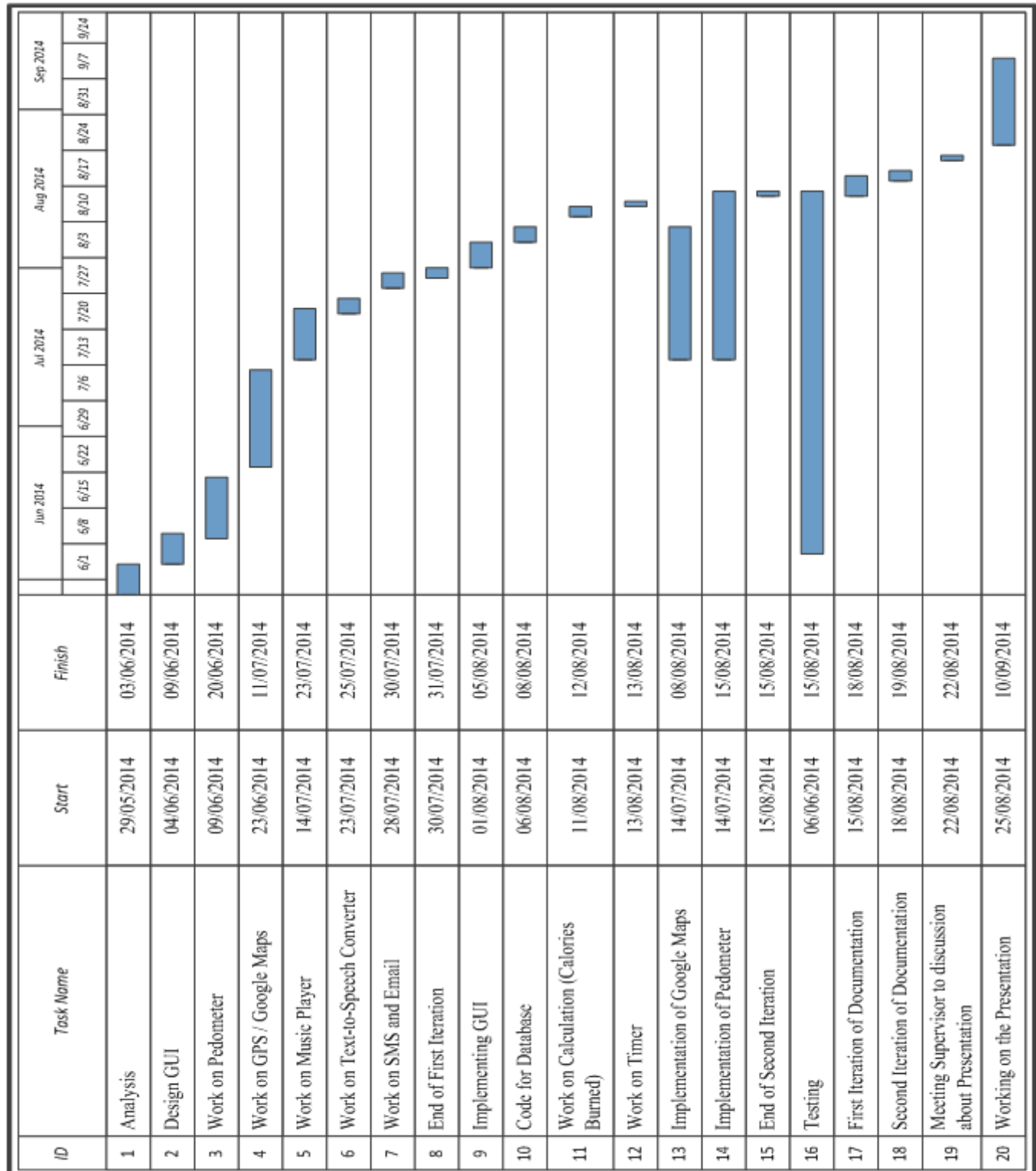
- **Maps API:** It is proposed that in the future the app could make use of some other Map API, because the Google maps API has some limitations. Google Inc. does not allow a device to download the Google maps on a memory card or internal storage; as

a result it is necessary for the user of this app to have a 3G or WIFI connection that connects the device continuously to the internet. If we make use of Non-Google Maps API which can be downloaded on the users device, it will make it convenient for those users who do not have a Wi-Fi or 3G connection; plus it would be beneficial for user who like to go for jogging on high terrains.

- **Voice Recognition:** One of the features that can be added to this app would be voice recognition. The app could be built to follow the commands of a user; all the features like starting a session or playing some song could be controlled by voice recognition.
- **Heart Rate:** In future the app can be modified to measure the heart rate of a person; by measuring the heart rate of a person it enables us to detect health problems which go unnoticed otherwise. Heart rate monitoring can also help us in adjusting the intensity of exercise according to our fitness level.
- **Fitness Tips:** The app could also provide the user with some fitness tips on regular intervals that would encourage them to do more exercise. Apart from fitness tips it can also suggest the user on how many calories a person should intake (via food) per day.
- **Social Networking:** The app could be connected to a social networking site, and the social networking site should have a page that enables all the users to post their data like how many kilometres a person has run or if a user has burned more calories in a limited time. All these posts will encourage the other user to go an extra mile and maintain a good health. Different users could even compete with each other and encourage other to do so.

# 7 Appendix

## 7.1 Gantt chart



## 7.2 Code

```
public class Pedometer extends Activity {

    private static final String TAG = "Pedometer";
    private SharedPreferences mStgs;
    private PedometerSettings mPdStgs;
    private Utils mUls;
    private TextView mStpVal;
    private TextView mPaceVal;
    private TextView mDistVal;
    private TextView mSpdVal;
    private TextView mCalVal;
    TextView mDesPace;
    private int mStepVal;
    private int mPcVal;
    private float mDstVal;
    private float mSpedVal;
    private int mCaloVal;
    private float mDsrdPcOrSpd;
    private int mMtn;
    private boolean mMetr;
    private float mMatn;
    private boolean mQuit = false;
    private boolean mIsRun;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "[ACTIVITY] onCreate");
        super.onCreate(savedInstanceState);

        mStepVal = 0;
        mPcVal = 0;

        setContentView(R.layout.main);

        mUls = Utils.getInstance();
    }

    @Override
    protected void onStart() {
        Log.i(TAG, "[ACTIVITY] onStart");
        super.onStart();
    }

    @Override
    protected void onResume() {
        Log.i(TAG, "[ACTIVITY] onResume");
        super.onResume();
    }
}
```



```

mStgs = PreferenceManager.getDefaultSharedPreferences(this);
mPdStgs = new PedometerSettings(mStgs);

mUls.setSpeak(mStgs.getBoolean("speak", false));

mIsRun = mPdStgs.isServiceRunning();

if (!mIsRun && mPdStgs.isNewStart()) {
    startStepService();
    bindStepService();
}
else if (mIsRun) {
    bindStepService();
}

mPdStgs.clearServiceRunning();

mStpVal = (TextView) findViewById(R.id.stp_val);
mPaceVal = (TextView) findViewById(R.id.pc_val);
mDistVal = (TextView) findViewById(R.id.dist_val);
mSpdVal = (TextView) findViewById(R.id.spd_val);
mCalVal = (TextView) findViewById(R.id.cal_val);
mDesPace = (TextView) findViewById(R.id.des_pc_val);

mMetr = mPdStgs.isMetric();
((TextView) findViewById(R.id.dist_unit)).setText(getString(
    mMetr
    ? R.string.kms
    : R.string.miles
));
((TextView) findViewById(R.id.spd_unit)).setText(getString(
    mMetr
    ? R.string.kms_per_hr
    : R.string.miles_per_hr
));

mMtn = mPdStgs.getMaintainOption();
((LinearLayout) this.findViewById(R.id.des_pace_ctrl)).setVisibility(
    mMtn != PedometerSettings.M_NONE
    ? View.VISIBLE
    : View.GONE
);
if (mMtn == PedometerSettings.M_PACE) {
    mMatn = 5f;
    mDsrdPcOrSpd = (float)mPdStgs.getDesiredPace();
}
else
if (mMtn == PedometerSettings.M_SPEED) {
    mDsrdPcOrSpd = mPdStgs.getDesiredSpeed();
    mMatn = 0.1f;
}
Button button1 = (Button) findViewById(R.id.button_desired_pace_lower);
button1.setOnClickListener(new View.OnClickListener() {

```

```

        public void onClick(View v) {
            mDsrdPcOrSpd -= mMatn;
            mDsrdPcOrSpd = Math.round(mDsrdPcOrSpd * 10) / 10f;
            displayDesiredPaceOrSpeed();
            setDesiredPaceOrSpeed(mDsrdPcOrSpd);
        }
    });
    Button button2 = (Button) findViewById(R.id.btn_des_pace_raise);
    button2.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            mDsrdPcOrSpd += mMatn;
            mDsrdPcOrSpd = Math.round(mDsrdPcOrSpd * 10) / 10f;
            displayDesiredPaceOrSpeed();
            setDesiredPaceOrSpeed(mDsrdPcOrSpd);
        }
    });
    if (mMtn != PedometerSettings.M_NONE) {
        ((TextView) findViewById(R.id.des_pace_lbl)).setText(
            mMtn == PedometerSettings.M_PACE
            ? R.string.des_pace
            : R.string.des_speed
        );
    }

    displayDesiredPaceOrSpeed();
}

private void displayDesiredPaceOrSpeed() {
    if (mMtn == PedometerSettings.M_PACE) {
        mDesPace.setText("" + (int)mDsrdPcOrSpd);
    }
    else {
        mDesPace.setText("" + mDsrdPcOrSpd);
    }
}

@Override
protected void onPause() {
    Log.i(TAG, "[ACTIVITY] onPause");
    if (mIsRun) {
        unbindStepService();
    }
    if (mQuit) {
        mPdStgs.saveServiceRunningWithNullTimestamp(mIsRun);
    }
    else {
        mPdStgs.saveServiceRunningWithTimestamp(mIsRun);
    }
}

super.onPause();
savePaceSetting();
}

@Override

```

```

protected void onStop() {
    Log.i(TAG, "[ACTIVITY] onStop");
    super.onStop();
}

protected void onDestroy() {
    Log.i(TAG, "[ACTIVITY] onDestroy");
    super.onDestroy();
}

protected void onRestart() {
    Log.i(TAG, "[ACTIVITY] onRestart");
    super.onDestroy();
}

private void setDesiredPaceOrSpeed(float desiredPaceOrSpeed) {
    if (mSer != null) {
        if (mMtn == PedometerSettings.M_PACE) {
            mSer.setDesiredPace((int)desiredPaceOrSpeed);
        }
        else
            if (mMtn == PedometerSettings.M_SPEED) {
                mSer.setDesiredSpeed(desiredPaceOrSpeed);
            }
    }
}

private void savePaceSetting() {
    mPdStgs.savePaceOrSpeedSetting(mMtn, mDsrdPcOrSpd);
}

private StepService mSer;

private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder service) {
        mSer = ((StepService.StepBinder)service).getService();

        mSer.registerCallback(mCallback);
        mSer.reloadSettings();
    }

    public void onServiceDisconnected(ComponentName className) {
        mSer = null;
    }
};

private void startStepService() {
    if (! mIsRun) {
        Log.i(TAG, "[SERVICE] Start");
        mIsRun = true;
        startService(new Intent(Pedometer.this,
            StepService.class));
    }
}

```

```

    }

    private void bindStepService() {
        Log.i(TAG, "[SERVICE] Bind");
        bindService(new Intent(Pedometer.this,
            StepService.class), mConnection, Context.BIND_AUTO_CREATE +
Context.BIND_DEBUG_UNBIND);
    }

    private void unbindStepService() {
        Log.i(TAG, "[SERVICE] Unbind");
        unbindService(mConnection);
    }

    private void stopStepService() {
        Log.i(TAG, "[SERVICE] Stop");
        if (mService != null) {
            Log.i(TAG, "[SERVICE] stopService");
            stopService(new Intent(Pedometer.this,
                StepService.class));
        }
        mIsRun = false;
    }

    private void resetValues(boolean updateDisplay) {
        if (mService != null && mIsRun) {
            mService.resetValues();
        }
        else {
            mStpVal.setText("0");
            mPaceVal.setText("0");
            mDistVal.setText("0");
            mSpdVal.setText("0");
            mCalVal.setText("0");
            SharedPreferences state = getSharedPreferences("state", 0);
            SharedPreferences.Editor stateEditor = state.edit();
            if (updateDisplay) {
                stateEditor.putInt("steps", 0);
                stateEditor.putInt("pace", 0);
                stateEditor.putFloat("distance", 0);
                stateEditor.putFloat("speed", 0);
                stateEditor.putFloat("calories", 0);
                stateEditor.commit();
            }
        }
    }

    private static final int MENU_SETTINGS = 8;
    private static final int MENU_QUIT    = 9;

    private static final int MENU_PAUSE = 1;
    private static final int MENU_RESUME = 2;
    private static final int MENU_RESET = 3;

```

```

public boolean onPrepareOptionsMenu(Menu me) {
    me.clear();
    if (mIsRun) {
        me.add(0, MENU_PAUSE, 0, R.string.pause)
            .setIcon(android.R.drawable.ic_media_pause)
            .setShortcut('1', 'p');
    }
    else {
        me.add(0, MENU_RESUME, 0, R.string.resume)
            .setIcon(android.R.drawable.ic_media_play)
            .setShortcut('1', 'p');
    }
    me.add(0, MENU_RESET, 0, R.string.reset)
        .setIcon(android.R.drawable.ic_me_cls_clr_cncl)
        .setShortcut('2', 'r');
    me.add(0, MENU_SETTINGS, 0, R.string.settings)
        .setIcon(android.R.drawable.me_pref)
        .setShortcut('8', 's')
        .setIntent(new Intent(this, Settings.class));
    me.add(0, MENU_QUIT, 0, R.string.quit)
        .setIcon(android.R.drawable.ic_lck_pwr)
        .setShortcut('9', 'q');
    return true;
}

```

```

/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem it) {
    switch (it.getItemId()) {
        case MENU_PAUSE:
            unbindStepService();
            stopStepService();
            return true;
        case MENU_RESUME:
            startStepService();
            bindStepService();
            return true;
        case MENU_RESET:
            resetValues(true);
            return true;
        case MENU_QUIT:
            resetValues(false);
            unbindStepService();
            stopStepService();
            mQuit = true;
            finish();
            return true;
    }
    return false;
}

```

```

@Override
public void onBackPressed()
{
    setContentView(R.layout.chw);
}

```

```
Intent intent=new Intent(Pedometer.this,activa.class);
startActivity(intent);
Pedometer.this.finish();

}
}
```

Code for Pedometer [\[24\]](#)

## 8 References

- [1] Wikipedia, “Extreme programming practices,” *Wikipedia*. [Online]. Available: [http://en.wikipedia.org/wiki/Extreme\\_programming\\_practices](http://en.wikipedia.org/wiki/Extreme_programming_practices).
- [2] AppBrain, “AppBrain Stats,” *AppBrain*, 2014. [Online]. Available: <http://www.appbrain.com/stats/stats-index>. [Accessed: 19-Aug-2014].
- [3] Steel Media Ltd, “No Title,” *PocketGamer.biz*, 2014. [Online]. Available: <http://www.pocketgamer.biz/metrics/app-store/categories/>. [Accessed: 19-Aug-2014].
- [4] AppBrain, “No TitleAppBrain Stats,” *AppBrain*, 2014. [Online]. Available: <http://www.appbrain.com/stats/android-market-app-categories>. [Accessed: 19-Aug-2014].
- [5] K. Chi-wai, M. So-ning, L. Wing-kuen, H. Sai-chuen, W. Ka-shun, and W. Choi-ki, “Can Mobile Virtual Fitness Apps Replace Human Fitness Trainer?,” in *Can mobile virtual fitness apps replace human fitness trainer?*, 2011, pp. 56–63.
- [6] NING|MODE SOCIAL, “Innovation Insights,” *Insights Site Creator*, 2014. [Online]. Available: <http://insights.wired.com/profiles/blogs/fitness-tech-and-quantified-health-a-new-era-of-staying-healthy-axzz39RfLshbi>. [Accessed: 10-Aug-2014].
- [7] A. Emrich, A. Theobalt, F. Leonhardt, S. Knoch, D. Werth, and P. Loos, “A Pervasive Mobile Assistance System for Health and Fitness Scenarios,” in *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 2898–2907.
- [8] A. Ramachandran, “Patient-Centered Mobile Apps for Chronic Disease Management,” 2014, pp. 948–952.
- [9] I.-C. Chang, H.-T. Tai, D.-L. Hsieh, F.-H. Yeh, and S.-H. Chang, “Design and Implementation of the Travelling Time- and Energy-Efficient Android GPS Navigation App with the VANET-Based A\* Route Planning Algorithm,” in *2013 International Symposium on Biometrics and Security Technologies*, 2013, pp. 85–92.

- [10] M. G. D. Elia and V. Paciello, "Sensors uncertainty on an Android smart phone," 2012, no. 1, pp. 698 – 702.
- [11] J. Oster, J. Behar, R. Colloca, Q. Li, Q. Li, G. D. Clifford, and U. Kingdom, "Open Source Java-based ECG analysis Software and Android app for Atrial Fibrillation Screening," 2013, pp. 731–734.
- [12] A. S. M. Mosa, I. Yoo, and L. Sheets, "A systematic review of healthcare applications for smartphones.," in *BMC medical informatics and decision making*, 2012, vol. 12, no. 1, p. 67.
- [13] Flurry, "Flurry," *Flurry*, 2014. [Online]. Available: <http://www.flurry.com/blog/flurry-insights/health-and-fitness-apps-finally-take-fueled-fitness-fanatics> - .U9-2V\_ldUUt. [Accessed: 12-Aug-2014].
- [14] Chris Gullo, "mobihealthnews," *Chester Street Publishing, Inc.* [Online]. Available: <http://mobihealthnews.com/14884/by-2016-400m-market-for-health-fitness-apps/>. [Accessed: 12-Aug-2014].
- [15] I. Pressroom, "Sports and Fitness App Market to Expand by More Than 60 Percent in Five Years," *IHS Inc.* [Online]. Available: <http://press.ihs.com/press-release/design-supply-chain/sports-and-fitness-app-market-expand-more-60-percent-five-years>. [Accessed: 14-Aug-2014].
- [16] C. Tode, "Mobile health app marketplace to take off, expected to reach \$26B by 2017," *Napean LLC*. [Online]. Available: <http://www.mobilemarketer.com/cms/news/research/15023.html>. [Accessed: 16-Aug-2014].
- [17] Wikipedia, "Non-functional Requirement." [Online]. Available: [http://en.wikipedia.org/wiki/Non-functional\\_requirement](http://en.wikipedia.org/wiki/Non-functional_requirement). [Accessed: 15-Aug-2014].
- [18] Wikipedia, "Unified Modeling Language," *Wikipedia*, 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Unified\\_Modeling\\_Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language). [Accessed: 17-Aug-2014].



- [19] FreeRiderInNOVA, “General Diet and Weight Loss Help,” *MyFitnessPal, Inc.* [Online]. Available: <http://www.myfitnesspal.com/topics/show/134478-accurate-formula-to-determine-calories-burned-jogging>. [Accessed: 14-Jun-2014].
- [20] Mohd Aslam, “Pedometer and Step Counter Mobile,” *MOHD ASLAM TECHNICAL AND NON-TECHNICAL BLOGS*. [Online]. Available: <http://mohdaslam.com/pedometer-and-step-counter-mobile-apps-how-does-it-work/>. [Accessed: 18-Aug-2014].
- [21] Lars and Vogel, “Android SQLite database and content provider,” *Vogella*. [Online]. Available: <http://www.vogella.com/tutorials/AndroidSQLite/article.html>. [Accessed: 11-Aug-2014].
- [22] Androidgreeve, “How to use Shared Preferences and manage User Sessions,” *ANDROIDGREEVE*, 2013. [Online]. Available: <http://androidgreeve.blogspot.co.uk/2014/01/How-to-use-Shared-Preferences-and-manage-User-Sessions.html>. [Accessed: 13-Aug-2014].
- [23] Shane Conder & Lauren Darcey, “Android Essentials: Application Preferences,” *Tuts+*. [Online]. Available: <http://code.tutsplus.com/tutorials/android-essentials-application-preferences--mobile-2914>. [Accessed: 18-Aug-2014].
- [24] Bagilevi, “Android Pedometer,” *GitHub*, 2014. [Online]. Available: <https://github.com/bagilevi/android-pedometer>. [Accessed: 08-Jul-2014].