10k Steps a Day – A Step Counter App for Android

BY

Md. Sadik Hasan Khan
ID: 121-15-1727

This Report Presented in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering

Supervised By
Nazmun Nessa Moon
Associate Professor
Department of CSE
Daffodil International University

Co-Supervised By
Gazi Zahirul Islam
Assistant Professor
Department of CSE
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

DECEMBER 2021

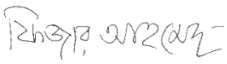
APPROVAL

This Project/internship titled **10k Steps a Day – A Step Counter App for Android**, submitted by **Md. Sadik Hasan Khan**, ID No: **121-15-1727** to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 4th January.

BOARD OF EXAMINERS

Chairman

Dr. Touhid Bhuiyan
Professor and Head
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

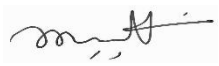


Internal Examiner

Dr. Fizar Ahmed
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Internal Examiner

Nusrat Jahan
Senior Lecturer
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



External Examiner

Dr. Mohammad Shorif Uddin
Professor
Department of Computer Science and Engineering
Jahangirnagar University

DECLARATION

We hereby declare that, this project has been done by us under the supervision of **Nazmun Nessa Moon, Associate Professor, Department of CSE** Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

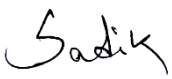
Supervised by:

Nazmun Nessa Moon
Associate Professor
Department of CSE
Daffodil International University

Co-Supervised by:

Gazi Zahirul Islam
Assistant Professor
Department of CSE
Daffodil International University

Submitted by:



Md. Sadik Hasan Khan
ID: 121-15-1727
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

I want to express my utmost gratefulness to the almighty Allah for His divine grace for which I could complete my final year project/internship successfully.

I am really grateful and indebted to **Nazmun Nessa Moon**, **Associate Professor**, Department of CSE, Daffodil International University, Dhaka. Thorough Knowledge & keen interest of our supervisor in the field of "*android based*" led to pull this project of successfully. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to acknowledge my mistake and correct them along the way.

I would also like to express our heartiest gratitude to Dr. Syed Akhter Hossain, Professor and Head**,** Department of CSE, for his kind help to finish my project and also to other faculty member and the staff of CSE department of Daffodil International University.

I would like to thank our entire course mate in Daffodil International University, who took part in this discussion while completing the course work.

Finally, I must acknowledge my parents selfless efforts and constant support with due respect.

ABSTRACT

As the name suggests, **"10k Steps a Day – A Step Counter App for Android"** is a health focus android app for counting steps. The significance of the project lies in its ability to prompt people to make some time from their busy schedule daily so that they can lead a healthy life without sickness. But the core focus of this app is to help all the people suffering from "Diabetes", a chronic and long lasting health condition affecting your body's energy consumption through food. Which needless to say, in Bangladesh, has an abundance of. Currently there almost 10 million people has diabetes in Bangladesh. That is to say, 1 in every 16 people in Bangladesh has this long lasting health condition. And there is no cure for this condition as of now other than controlling it through proper diet and exercise. To that end, there is no substitute for taking a routine walk every day. But how much does one actually need to walk a day? Research has shown that, every day only 10,000 steps is required for an adult to sustain a healthy life. This is why I have developed a step counter android app to record the steps taken every day. This app will benefit everyone regardless of them being suffering from diabetes or not. I have added many useful features to this app to motivate them to go for a walk every day. I used android studio to develop this project. And for testing and debugging I have used Pixel 5 emulator and Oneplus 3t smartphone. The app is properly tested in multiple ways to ensure its credibility.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES **PAGE NO**

CHAPTER 1

INTRODUCTION

1.1 Introduction

The demand for useful app, especially, health related app has never been higher. With the advent of technology it is now as easy as tapping a button on our phone to get all the health related information we need to check our health condition. Keeping that in mind, I set out to develop an android app that counts the steps taken in a day with options to set a goal and show the step count in a clear and the most basic way possible without extra complexity.

This app is especially helpful for people who has diabetes or requires walking a certain amount of distance on a regular basis. Initially there are a handful of things a user can do with the app including step counting or distance covered, split step count, setting a goal, and step size.

1.2 Motivation

The way things are heading, the health condition of the people in a fast-paced area, especially, Dhaka is becoming very challenging to maintain properly. The majority of people living in a big city are officegoers and have very little time to take proper care of their health. As a result, developing different types of health conditions such as diabetes, obesity, and other cardio-related complications has become common. But unfortunately, there is no provision for motivating them to walk the walk when it comes to taking care of their health.

Besides, this project is especially close to my heart because of my mother. She has been suffering from diabetes for as long as I can remember and requires walking every day to keep her sugar level checked. But until now there was no definitive way of keeping track of her activity. Although there are plenty of health apps for android, the majority of them are either complicated to use or require extensive knowledge to use them. Unfortunately, my mother doesn't fit either of the templates.

So I decided to build one for her and for people like her who need simple yet useful tools that they can use with just a tap considering most people have smartphones nowadays.

1.3 Objectives

- To check how many steps users have taken on a daily basis.
- To cut the complexity of using a health app.
- To provide the most useful information without any extra hassle of having prior knowledge of using an app.
- To help people lead a healthy lifestyle.
- To save time and money in the long run.

1.4 Expected Outcome

1. People becoming more conscious about their health.
2. People become motivated to walk and exercise every day.
3. Saving people's time and money by cutting the cost of hospitalization and medicine.
4. People become aware of diabetes and other cardio related health complications.

1.5 Report Layout

Chapter 1: Introduction

The motivation behind my project and what prompt me to build the app in the first place, its objectives, and the outcome that is expected of the project have been discussed here in this chapter. Also, the layout of the report has been laid down in the bottom part of this chapter.

Chapter 2: Background

Necessary background for the project has been projected in this chapter with a brief overview of related work that has been done so far. A competitive analysis with other

similar application has also been provided in conjunction with the scope of the problem and challenges I faced during the implementation of the project.

Chapter 3: Design Specification

This chapter discusses the design methodologies of the project along with its interactive UX element and other related dependencies.

Chapter 4: Implementation and Testing

From implementing the design and setting up the sensor for the device to ultimately adding various useful function and necessary technology requirements, this chapter discussed all the aspects of android app development.

Chapter 5: Conclusion and Future Scope

In this chapter I discussed the future scope and functionality improvement of the app along with its potential.

CHAPTER 2

BACKGROUND

2.1 Introduction

There have been a lot projects done using the same ideology so it was relatively easy for me to research and study the limitations of those project. It played a vital role in developing the project. Although there are a lot of project out there doing the same thing I focus on the most important aspect: functionality and usability. And background research of the project accommodated my endeavor to focus on these aspects.

Without the background research of the related works done, it would have been difficult to lay out a plan to developing the application. That, in effect, turn the tide and improving upon the existing project a bit more challenging. However, it provided me with the narrow gap that would ultimately help me successfully build the project.

2.2 Related Work

In the following, I have provided some of the works that have been already done using the same ideology. But most of them are not easy to understand and requires multiple steps just to get started such as registering into the server, log in, and so on. Here are some of the related projects that already exist:

- Step Counter - Pedometer, MStep [1]
- Step Tracker - Pedometer, iStep [2]
- Walk Tracker - Step Counter Free & Calorie Burner [3]
- Pedometer - Step Counter App [4]

Features: All these applications are more or less function the same. They all have bunch of features to offer with lots of options.

- They have different types of reminder to get you throughout the day with lots of activity
- Performance report

- Start, Pause, and Reset options
- Backup and restore data

Problems: However, in an effort to provide all these functionality they made some important sacrifice.

- Most of them are too complicated to use
- User experience design is not that great
- Too many options
- Limited offline facility and can't function properly without an internet connection.

2.3 Example

For instance, we can take a look at figure 2.1 for "Pedometer - Step Counter App" [4].

Figure 2.1: Screenshot of Pedometer - Step Counter App

From daily, weekly, and monthly step counting chart to calorie counter and other related functionalities, this app has it all. However, if you were to ask to walk us through the interface, chances are you won't be able to do it on the first go. That brings us to the next section: Pros and Pons

Pros:

- Great features
- Start, pause, and reset functionality
- Calorie counter
- Daily record

Cons:

- Less interactive UX
- Navigating through the app is a challenge
- Lots and lots of functions
- Less user friendly
- Requires prior knowledge
- It has online features so no guarantee on data security

2.3 Comparative Analysis

In contrast, I have built my app very simple, and easy to understand way. From the get go, one can immediately tell what is going on in the interface after opening the app for the very first time. I have carefully placed other menus on the upper-right corner of the app for easy access so user can control most aspects of the app.

Pros:

- Users can get to walking just after opening the app without having to worry about time consuming setting changes.
- Users can set the goal in just two steps
- Users can change their step length manually in either centimeters or feet.
- Notification bar at top constantly reminds them with real time feed of steps taken.

- I understand how valuable a user's personal information is, so I built it so that it can work 100% without any internet connection with full feature set.

Cons:

- No user database connection
- No start, pause, and reset options
- No extra function other than counting the step
- Not so much engaging

2.4 Scope of the problem

There were many challenges along the way, which I was able to tackle thanks to the developer's community on stackoverflow [5] and many other websites. However, building an app that requires hardware step-counter from the ground up was a hassle and almost seemed like reinventing the wheel. So I resort to an open-source project [6] that had already implemented the step-sensor mechanism and improve upon it. I used several open -source resources to facilitate different requirements including color wheel animation [7], color picker [8], and sensor implementation.

I use Android Studio 2020.3.1 with Gradle v7.0.2 and latest build for the project. I also used java instead of Kotlin to sync with the open-source project. My projects targeted SDK version is 30 (Android 11) and minimum SDK requirement is 26 (Android 8.0 Oreo)

2.5 Challenges

Here is a list of challenges I faced during my project development:

- Handling an old codebase is challenging to say the least.
- Downgrading from the latest Gradle build was also a challenge.
- Designing the interface and incorporating the animation
- Incorporating the color picker
- Developing a user friendly interface.

CHAPTER 3

DESIGN SPECIFICATION

The entire application is built upon a single layer of interaction module with a user interface and a menu that can be accessed by tapping on a vertical ellipses located in the upper-right side of the interface to offer a simple experience. In the following section I will discuss every single aspect of my app design.

3.1 UI (user interface) Design

Designing the user interface was the most crucial part of the development process. I wanted to make it as simple as possible by just providing the most important information that one might need.

Figure 3.1, shows the actual user interface with all the design elements

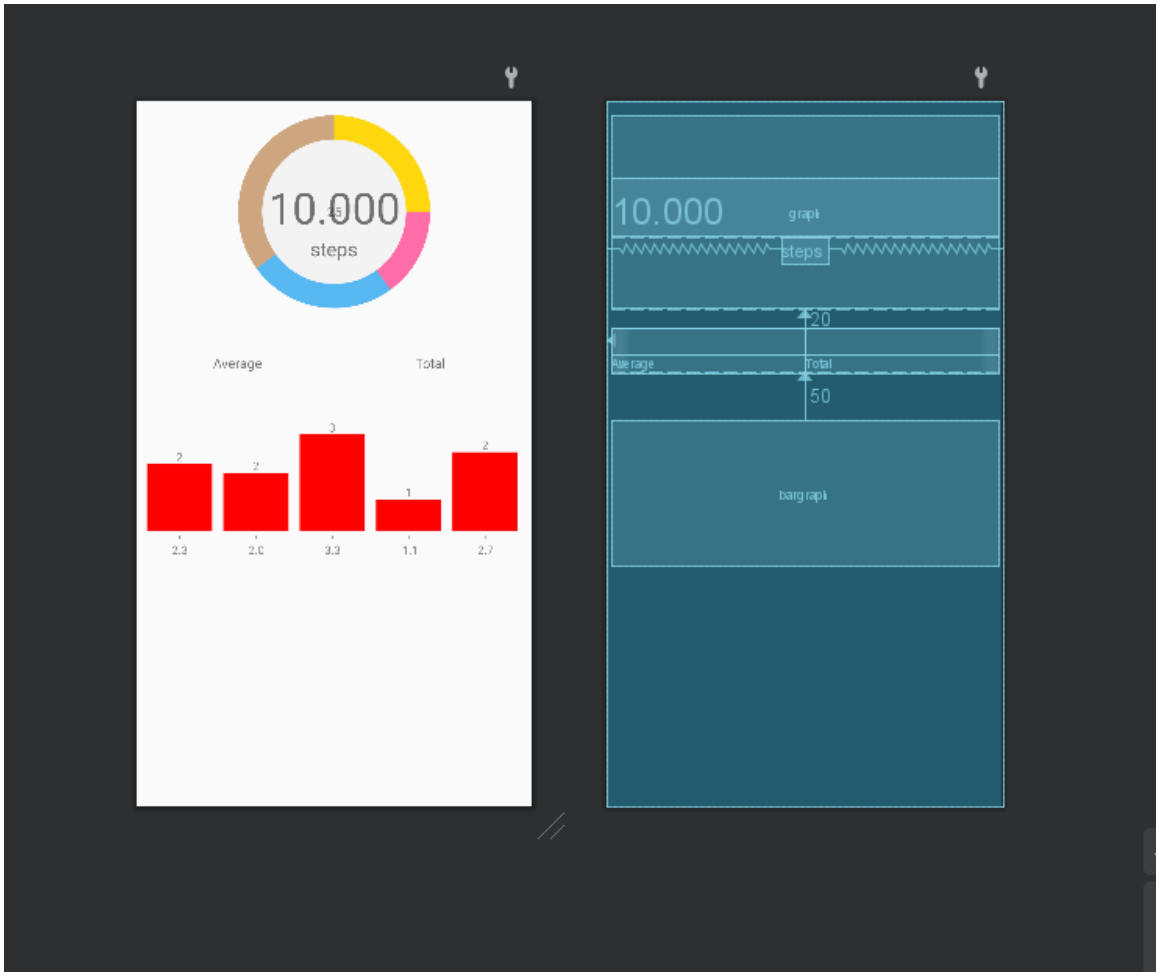


Figure 3.1: User interface design of the

In this design, there are 4 different components including a PieChart, two LinearLayouts, and a BarChart. PieChart shows the real time step count with a smooth animation showing you how close you are to your goal. In the LinearLayouts you will get to see the average and total step count that has been taken so far. And in the BarChart you will be able to see your daily step count with weekdays. Figure 3.2 shows fully developed UI in action.

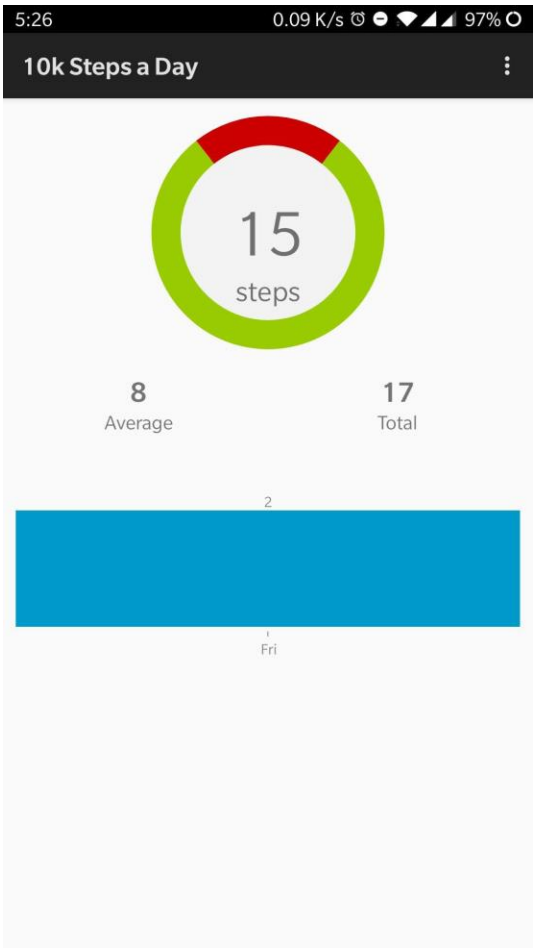


Figure 3.2: Final UI design

3.2 Menu Design

I have designed the menu to be as simple as possible. As you can see in the Figure 3.2, upon first click you will see a three drop-down options: Settings, Split count, and About. In the following sections I will be discussing the layout of every options and

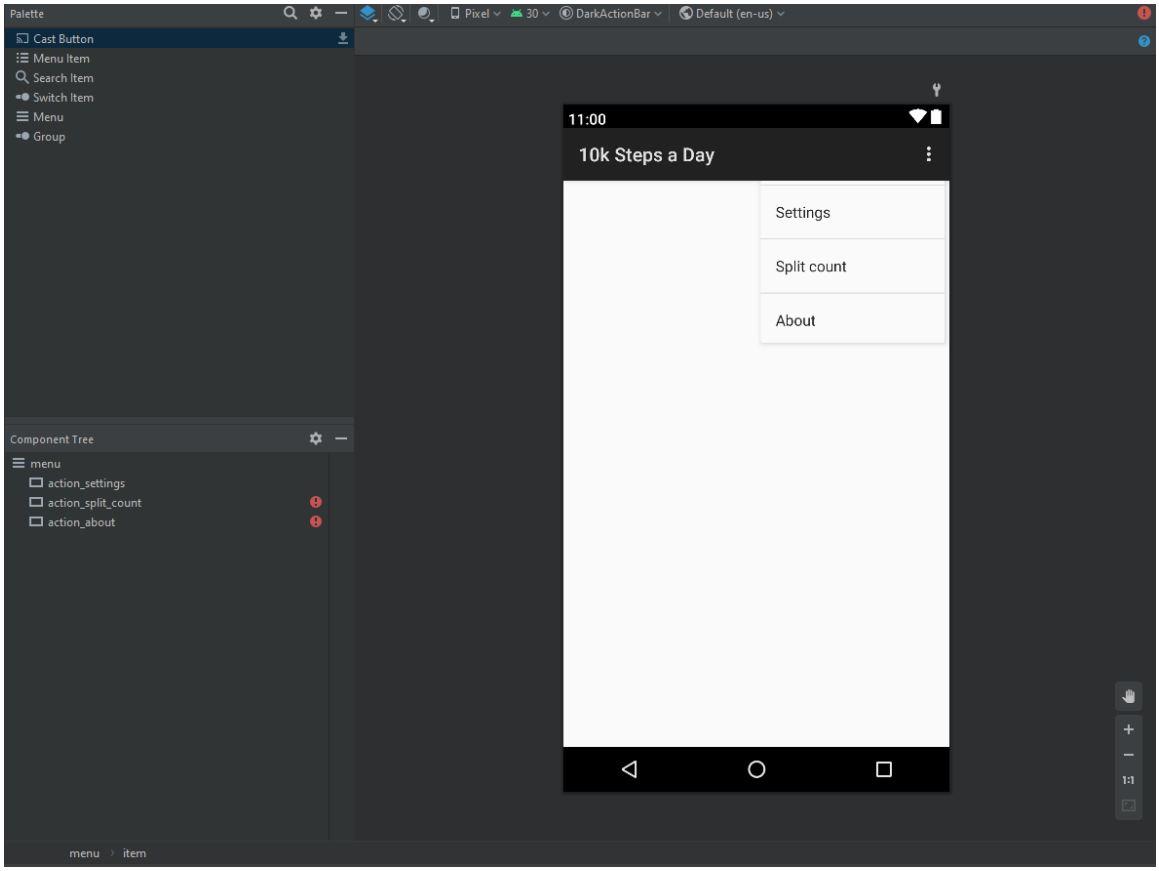


Figure 3.2: Menu design of the app

In the “Settings” option, you will see five different preferences with different functionality. Figure 3.3 will give you a comprehensive look at all the preferences.

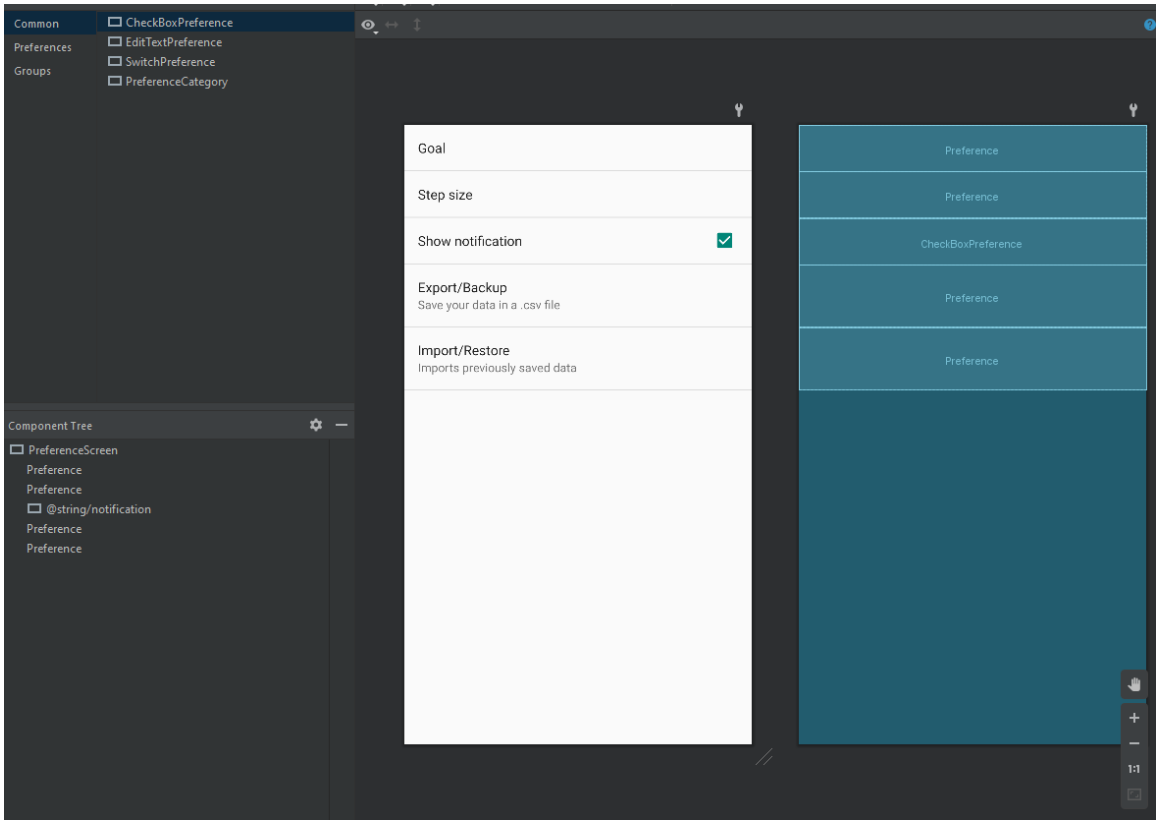


Figure 3.3: In the settings preference

The first preference is goal for setting up your daily step goal. The next preference is to set your step size considering everyone has different step size. So you can customize your preferred size (either in cm or ft). Figure 3.4 and 3.5 shows how it looks when you tap on “Goal” and “Step Size” preferences.

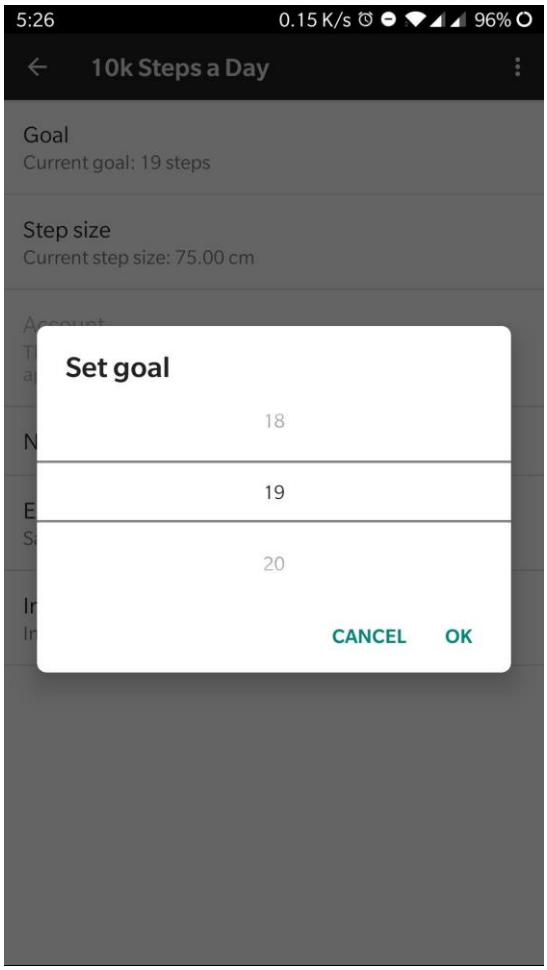


Figure 3.4: Set step size interface

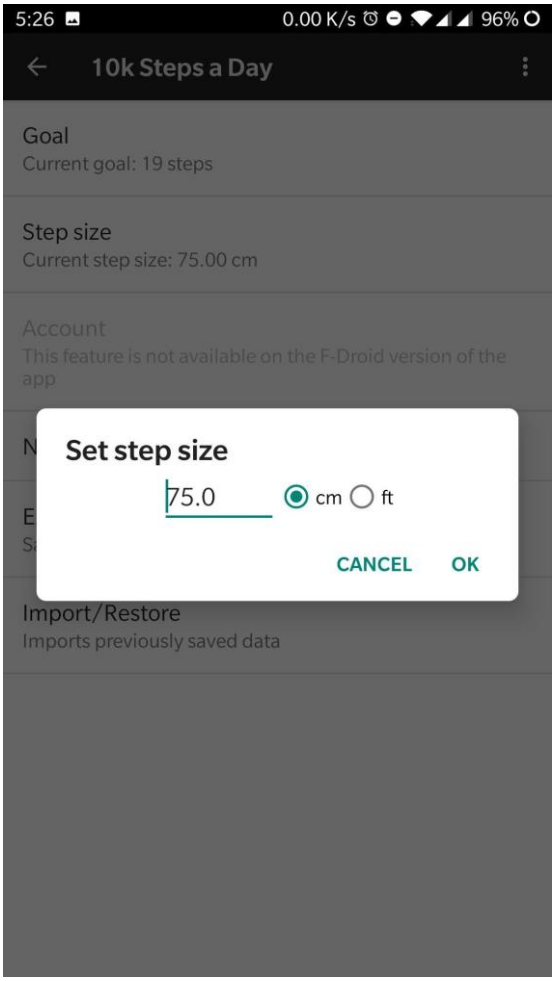


Figure 3.5: Set goal interface

Now if you want to back up all the data in your device for later use, you can by taping on the Export/Backup preference. Similarly you can also import data (.csv) to your app.

Next in the drop-down menu is the “Split count” option which when started shows the steps and distance covered since the function has been activated. In the “Split count” layout, you will see 3 (three) elements: steps, distance (in km), and start time and date. On the bottom there are two simple buttons: stop and close which will stop the Split count function and close the Split count window respectively. Figure 3.6 shows exactly how it looks when activated on screen.

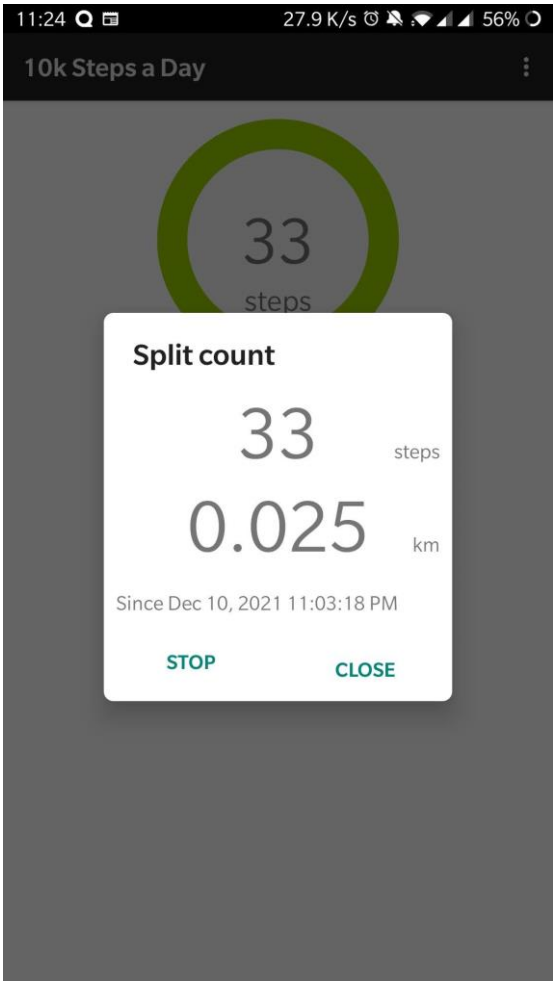


Figure 3.6: Split count window

Next in the menu is the “About” option which shows the creator and the version of the app. Figure 3.7 shows the “About” window.

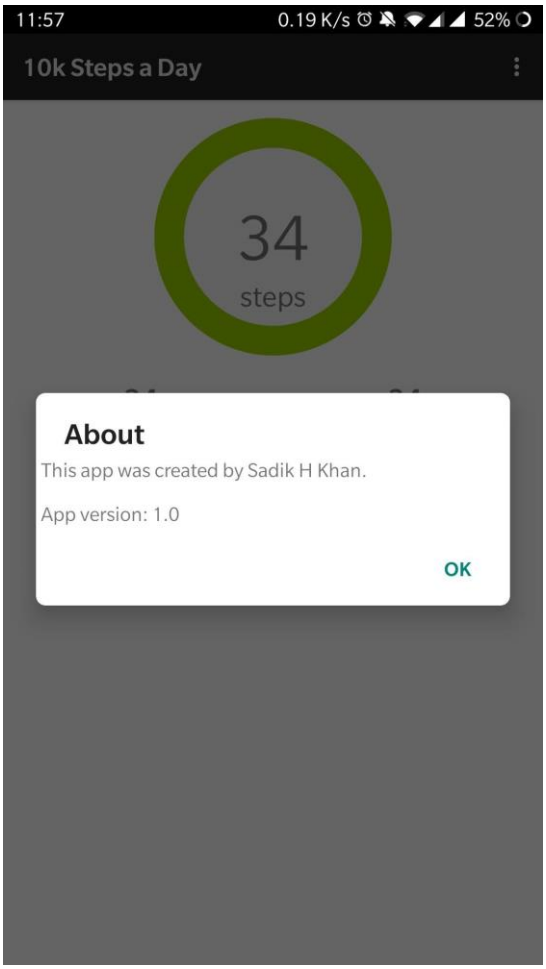


Figure 3.7: About window

3.3 Implementation Requirements

- Android Studio 2020.3.1
- Java 8
- Gradle v7.0.2
- Minimum SDK 26 (Android 8.0 Oreo) or higher

CHAPTER 4

IMPLEMENTATION

The implementation of this project required tackling many functional changes and handling many dependencies. From establishing the StepSensor to creating a database to store all the data and conducting all the statistical analysis and visualization, everything will be discussed in details in this chapter. However, for the sake of making the entire chapter understandable, implementation will be discussed in 3 parts: UI implementation, settings implementation, and StepSensor manipulation.

4.1 Implementation of the UI

The UI of the app is very simple yet a bit complicated to implement. The app uses a pie chart that changes according to the steps taken at any given point in time. That is to say, the pie chart is animated and can show the steps taken in green arch and steps needed to reach the goal in red. To implement this, the app uses *EazeGraph [7]*, shown in the figure 4.1, an open source android library for creating charts. Here is the code for adding the chart in the android app project folder in the 'build.gradle' under dependencies:

```
dependencies {
    compile 'com.github.blackfizz:eazegraph:1.2.2@aar'
    compile 'com.nineoldandroids:library:2.4.0'
}
```

Figure 4.1: Pie char library addition

Now there are two things that need to be implemented: one, pie char slice that will show how many steps have been taken so far and two, slice for the "missing" steps until reaching the goal. To do this, a new Fragment_overview class is created. The a basic building block of the UI needs to be implemented by *onCreateView()* method that uses a TextView to show the steps, average steps, and total steps. I registered the listener in the *onResume()* method and unregistered it in the *onPause()* method and update its latest value in the onSensorChanged() method. Figure 4.2, 4.3, 4.4, and 4.5 shows the implementation.

```java
@Override
public View onCreateView(final LayoutInflater inflater, final ViewGroup container, final Bundle savedInstanceState) {
    final View v = inflater.inflate(R.layout.fragment_overview,  root: null);
    stepsView = (TextView) v.findViewById(R.id.steps);
    totalView = (TextView) v.findViewById(R.id.total);
    averageView = (TextView) v.findViewById(R.id.average);

    pg = (PieChart) v.findViewById(R.id.graph);

    // slice for the steps taken today
    sliceCurrent = new PieModel( _legendLabel: "",  _value: 0, Color.parseColor( colorString: "#99CC00"));
    pg.addPieSlice(sliceCurrent);

    // slice for the "missing" steps until reaching the goal
    sliceGoal = new PieModel( _legendLabel: "", Fragment_Settings.DEFAULT_GOAL, Color.parseColor( colorString: "#CC0000"));
    pg.addPieSlice(sliceGoal);

    pg.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(final View view) {
            showSteps = !showSteps;
            stepsDistanceChanged();
        }
    });

    pg.setDrawValueInPie(false);
    pg.setUsePieRotation(true);
    pg.startAnimation();
    return v;
}
```

Figure 4.2: onCreateView method

```java
@Override
public void onResume() {
    super.onResume();
    getActivity().getActionBar().setDisplayHomeAsUpEnabled(false);

    Database db = Database.getInstance(getActivity());

    if (BuildConfig.DEBUG) db.logState();
    // read todays offset
    todayOffset = db.getSteps(Util.getToday());
    SharedPreferences prefs =
            getActivity().getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE);

    goal = prefs.getInt( s: "goal", Fragment_Settings.DEFAULT_GOAL);
    since_boot = db.getCurrentSteps();
    int pauseDifference = since_boot - prefs.getInt( s: "pauseCount", since_boot);
    // register a sensorlistener to live update the UI if a step is taken
    SensorManager sm = (SensorManager) getActivity().getSystemService(Context.SENSOR_SERVICE);
    Sensor sensor = sm.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    if (sensor == null) {
        new AlertDialog.Builder(getActivity()).setTitle(R.string.no_sensor)
                .setMessage(R.string.no_sensor_explain)
                .setOnDismissListener(new DialogInterface.OnDismissListener() {
                    @Override
                    public void onDismiss(final DialogInterface dialogInterface) {
                        getActivity().finish();
                    }
                }).setNeutralButton(android.R.string.ok, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(final DialogInterface dialogInterface, int i) {
                dialogInterface.dismiss();
            }
        }).create().show();
    } else {
        sm.registerListener( listener: this, sensor, SensorManager.SENSOR_DELAY_UI,  maxReportLatencyUs: 0);
    }
    since_boot -= pauseDifference;
    total_start = db.getTotalWithoutToday();
    total_days = db.getDays();
    db.close();
    stepsDistanceChanged();
}
```

Figure 4.3: onResume() method

```java
@Override
public void onPause() {
    super.onPause();
    try {
        SensorManager sm =
                (SensorManager) getActivity().getSystemService(Context.SENSOR_SERVICE);
        sm.unregisterListener(this);
    } catch (Exception e) {
        if (BuildConfig.DEBUG) Logger.log(e);
    }
    Database db = Database.getInstance(getActivity());
    db.saveCurrentSteps(since_boot);
    db.close();
}
```

Figure 4.4: onPause() method

```
@Override
public void onSensorChanged(final SensorEvent event) {
    if (BuildConfig.DEBUG) Logger.log(
            msg: "UI - sensorChanged | todayOffset: " + todayOffset + " since boot: " +
                    event.values[0]);
    if (event.values[0] > Integer.MAX_VALUE || event.values[0] == 0) {
        return;
    }
    if (todayOffset == Integer.MIN_VALUE) {
        // no values for today
        // we dont know when the reboot was, so set todays steps to 0 by
        // initializing them with -STEPS_SINCE_BOOT
        todayOffset = -(int) event.values[0];
        Database db = Database.getInstance(getActivity());
        db.insertNewDay(Util.getToday(), (int) event.values[0]);
        db.close();
    }
    since_boot = (int) event.values[0];
    updatePie();
}
```

Figure 4.5: onSensorChanged() method

However, under the updatePie() method the step count or distance is updated depending on the stepsize unit. Figure 4.6 shows the code that updates the pie graph to show todays steps/distance as well as the yesterday and total values.

```
private void updatePie() {
    if (BuildConfig.DEBUG) Logger.log( msg: "UI - update steps: " + since_boot);
    // todayOffset might still be Integer.MIN_VALUE on first start
    int steps_today = Math.max(todayOffset + since_boot, 0);
    sliceCurrent.setValue(steps_today);
    if (goal - steps_today > 0) {
        // goal not reached yet
        if (pg.getData().size() == 1) {
            // can happen if the goal value was changed: old goal value was
            // reached but now there are some steps missing for the new goal
            pg.addPieSlice(sliceGoal);
        }
        sliceGoal.setValue(goal - steps_today);
    } else {
        // goal reached
        pg.clearChart();
        pg.addPieSlice(sliceCurrent);
    }
    pg.update();
    if (showSteps) {
        stepsView.setText(formatter.format(steps_today));
        totalView.setText(formatter.format( number: total_start + steps_today));
        averageView.setText(formatter.format( number: (total_start + steps_today) / total_days));
    } else {
        // update only every 10 steps when displaying distance
        SharedPreferences prefs =
                getActivity().getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE);
        float stepsize = prefs.getFloat( s: "stepsize_value", Fragment_Settings.DEFAULT_STEP_SIZE);
        float distance_today = steps_today * stepsize;
        float distance_total = (total_start + steps_today) * stepsize;
        if (prefs.getString( s: "stepsize_unit", Fragment_Settings.DEFAULT_STEP_UNIT)
                .equals("cm")) {
            distance_today /= 100000;
            distance_total /= 100000;
        } else {
            distance_today /= 5280;
            distance_total /= 5280;
        }
        stepsView.setText(formatter.format(distance_today));
        totalView.setText(formatter.format(distance_total));
        averageView.setText(formatter.format( number: distance_total / total_days));
    }
}
```

Figure 4.6: updatePie() method

The app also shows weekly performance through a bar graph. And upon clicking the graph the app shows the record since the last boot. The code in figure 4.7 updates the bar graph to show the steps/distance of the last week.

```
private void updateBars() {
    SimpleDateFormat df = new SimpleDateFormat( pattern: "E", Locale.getDefault());
    BarChart barChart = (BarChart) getView().findViewById(R.id.bargraph);
    if (barChart.getData().size() > 0) barChart.clearChart();
    int steps;
    float distance, stepsize = Fragment_Settings.DEFAULT_STEP_SIZE;
    boolean stepsize_cm = true;
    if (!showSteps) {
        // load some more settings if distance is needed
        SharedPreferences prefs =
                getActivity().getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE);
        stepsize = prefs.getFloat( s: "stepsize_value", Fragment_Settings.DEFAULT_STEP_SIZE);
        stepsize_cm = prefs.getString( s: "stepsize_unit", Fragment_Settings.DEFAULT_STEP_UNIT)
                .equals("cm");
    }
    barChart.setShowDecimal(!showSteps); // show decimal in distance view only
    BarModel bm;
    Database db = Database.getInstance(getActivity());
    List<Pair<Long, Integer>> last = db.getLastEntries( num: 8);
    db.close();
    for (int i = last.size() - 1; i > 0; i--) {
        Pair<Long, Integer> current = last.get(i);
        steps = current.second;
        if (steps > 0) {
            bm = new BarModel(df.format(new Date(current.first)), _value: 0,
                    steps > goal ? Color.parseColor( colorString: "#99CC00") : Color.parseColor( colorString: "#0099cc"));
            if (showSteps) {
                bm.setValue(steps);
            } else {
                distance = steps * stepsize;
                if (stepsize_cm) {
                    distance /= 100000;
                } else {
                    distance /= 5280;
                }
                distance = Math.round(distance * 1000) / 1000f; // 3 decimals
                bm.setValue(distance);
            }
            barChart.addBar(bm);
        }
    }
    if (barChart.getData().size() > 0) {
        barChart.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(final View v) {
                Dialog_Statistics.getDialog(getActivity(), since_boot).show();
            }
        });
        barChart.startAnimation();
    } else {
        barChart.setVisibility(View.GONE);
    }
}

}
```

Figure 4.7: updateBars() method

4.2 Implementation of Settings menu

To implement the settings menu a *Fragment_settings* class is created which is the subclass of *PreferenceFragment* that also implements the *OnPreferenceClickListener* interface. Under the *onCreate()* method, every options in the settings is implemented by setting a

OnPreferenceClickListener for every preference. Figure 4.8 shows the code for the settings menu implementation.

```
public void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.settings);
    final SharedPreferences prefs =
            getActivity().getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE);
    findPreference( key: "import").setOnPreferenceClickListener(this);
    findPreference( key: "export").setOnPreferenceClickListener(this);
    if (Build.VERSION.SDK_INT >= 26) {
        findPreference( key: "notification").setOnPreferenceClickListener(this);
    } else {
        findPreference( key: "notification")
                .setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
                    @Override
                    public boolean onPreferenceChange(final Preference preference,
                                                      final Object newValue) {
                        prefs.edit().putBoolean( s: "notification", (Boolean) newValue).apply();

                        NotificationManager manager = (NotificationManager) getActivity()
                                .getSystemService(Context.NOTIFICATION_SERVICE);
                        if ((Boolean) newValue) {
                            manager.notify(SensorListener.NOTIFICATION_ID,
                                    SensorListener.getNotification(getActivity()));
                        } else {
                            manager.cancel(SensorListener.NOTIFICATION_ID);
                        }

                        return true;
                    }
                });
    }
    Preference account = findPreference( key: "account");
    PlaySettingsWrapper
            .setupAccountSetting(account, savedInstanceState, (Activity_Main) getActivity());

    Preference goal = findPreference( key: "goal");
    goal.setOnPreferenceClickListener(this);
    goal.setSummary("Current goal: {prefs.getInt("goal", DEFAULT_GOAL)} steps");

    Preference stepsize = findPreference( key: "stepsize");
    stepsize.setOnPreferenceClickListener(this);
    stepsize.setSummary("Current step size: {prefs.getFloat("stepsize_value", DE...");
    setHasOptionsMenu(true);
}
```

Figure 4.8: Implementation of the settings menu

4.2.1 Implementation of the Split Count function

Split Count is a standalone feature that is used to show the steps taken or distance covered in a specific time frame. If one starts the split count function it will keep recording the data from the moment it is started and show the information in a text view panel when evoked. That is because the hardware step sensor used to count the steps is always running in the background.

For this, a *Dialog_Split* abstract class is created in which a *getDialog()* method is created that shows the steps, distance covered, and the date. Moreover a *setOnClickListener()*

method is also implemented to listen the *onClick* event to start/stop the split count and close the dialog box. Figure 4.9 show the code.

```
abstract class Dialog_Split {

    private static boolean split_active;

    public static Dialog getDialog(final Context c, final int totalSteps) {
        final Dialog d = new Dialog(c);
        d.setTitle("Split count");
        d.setContentView(R.layout.dialog_split);

        final SharedPreferences prefs =
                c.getSharedPreferences( s: "pedometer", Context.MODE_MULTI_PROCESS);
        long split_date = prefs.getLong( s: "split_date",  l: -1);
        int split_steps = prefs.getInt( s: "split_steps", totalSteps);
        ((TextView) d.findViewById(R.id.steps))
                .setText(Fragment_Overview.formatter.format( number: totalSteps - split_steps));
        float stepsize = prefs.getFloat( s: "stepsize_value", Fragment_Settings.DEFAULT_STEP_SIZE);
        float distance = (totalSteps - split_steps) * stepsize;
        if (prefs.getString( s: "stepsize_unit", Fragment_Settings.DEFAULT_STEP_UNIT).equals("cm")) {
            distance /= 100000;
            ((TextView) d.findViewById(R.id.distanceunit)).setText("km");
        } else {
            distance /= 5280;
            ((TextView) d.findViewById(R.id.distanceunit)).setText("mi");
        }
        ((TextView) d.findViewById(R.id.distance))
                .setText(Fragment_Overview.formatter.format(distance));
        ((TextView) d.findViewById(R.id.date)).setText("Since {java.text.DateFormat.getDateTimeInstance().forma...");

        final View started = d.findViewById(R.id.started);
        final View stopped = d.findViewById(R.id.stopped);

        split_active = split_date > 0;

        started.setVisibility(split_active ? View.VISIBLE : View.GONE);
        stopped.setVisibility(split_active ? View.GONE : View.VISIBLE);

        final Button startstop = (Button) d.findViewById(R.id.start);
        startstop.setText(split_active ? "Stop" : "Start");
        startstop.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(final View v) {
                if (!split_active) {
                    prefs.edit().putLong( s: "split_date", System.currentTimeMillis())
                            .putInt( s: "split_steps", totalSteps).apply();
                    split_active = true;
                    d.dismiss();
                } else {
                    started.setVisibility(View.GONE);
                    stopped.setVisibility(View.VISIBLE);
                    prefs.edit().remove("split_date").remove("split_steps").apply();
                    split_active = false;
                }
                startstop.setText(split_active ? "Stop" : "Start");
            }
        });

        d.findViewById(R.id.close).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(final View v) { d.dismiss(); }
        });

        return d;
    }
}
```

Figure 4.9: implementation of dialog_split

4.3 StepSensor Manipulation

To manipulate the StepSensor and keep the step-sensor listener alive in the background to always get the number of steps since boot, a StepListener class which is the sub class of Service is implemented that also implements the *SensorEventListener* interface. In this class *onSensorChanged()* method is implemented to get the number of steps that is given by event.value[0]; although it's a float value, the fractional part is always zero. See figure 4.10 for the code.

```
@Override
public void onSensorChanged(final SensorEvent event) {
    if (event.values[0] > Integer.MAX_VALUE) {
        if (BuildConfig.DEBUG) Logger.log( msg: "probably not a real value: " + event.values[0]);
        return;
    } else {
        steps = (int) event.values[0];
        updateIfNecessary();
    }
}
```

Figure 4.10: onSensorChanged

Then a *getNotification(),updateIfNecessary(), and showNotification()* methods are implemented to show and handle the notification update. Figure 4.11, 4.12, and 4.13 show the code implementation of these function respectively.

```
public static Notification getNotification(final Context context) {
    if (BuildConfig.DEBUG) Logger.log( msg: "getNotification");
    SharedPreferences prefs = context.getSharedPreferences( s: "pedometer", Context.MODE_PRIVATE);
    int goal = prefs.getInt( s: "goal",  i: 10000);
    Database db = Database.getInstance(context);
    int today_offset = db.getSteps(Util.getToday());
    if (steps == 0)
        steps = db.getCurrentSteps(); // use saved value if we haven't anything better
    db.close();
    Notification.Builder notificationBuilder =
            Build.VERSION.SDK_INT >= 26 ? API26Wrapper.getNotificationBuilder(context) :
                    new Notification.Builder(context);
    if (steps > 0) {
        if (today_offset == Integer.MIN_VALUE) today_offset = -steps;
        NumberFormat format = NumberFormat.getInstance(Locale.getDefault());
        notificationBuilder.setProgress(goal,  progress: today_offset + steps,  indeterminate: false).setContentText(
                today_offset + steps >= goal ?
                        "Goal reached! {format.format((today_offset + steps))} s..." :
                        "{format.format((goal - today_offset - steps))} steps to go").setContentTitle(
                format.format( number: today_offset + steps) + " " + "steps");
    } else { // still no step value?
        notificationBuilder.setContentText(
                "Your progress will be shown here soon")
                .setContentTitle("Pedometer is counting");
                .setContentTitle(context.getString(R.string.notification_title));
    notificationBuilder.setPriority(Notification.PRIORITY_MIN).setShowWhen(false)
            .setContentIntent(PendingIntent
                    .getActivity(context,  requestCode: 0, new Intent(context, Activity_Main.class),
                            PendingIntent.FLAG_UPDATE_CURRENT))
            .setSmallIcon(R.drawable.ic_notification).setOngoing(true);
    return notificationBuilder.build();
}
```

Figure 4.11: getNotification() method

```
private boolean updateIfNecessary() {
    if (steps > lastSaveSteps + SAVE_OFFSET_STEPS ||
            (steps > 0 && System.currentTimeMillis() > lastSaveTime + SAVE_OFFSET_TIME)) {
        if (BuildConfig.DEBUG) Logger.log(
                msg: "saving steps: steps=" + steps + " lastSave=" + lastSaveSteps +
                        " lastSaveTime=" + new Date(lastSaveTime));
        Database db = Database.getInstance(this);
        if (db.getSteps(Util.getToday()) == Integer.MIN_VALUE) {
            int pauseDifference = steps -
                    getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE)
                            .getInt( s: "pauseCount", steps);
            db.insertNewDay(Util.getToday(), steps: steps - pauseDifference);
            if (pauseDifference > 0) {
                // update pauseCount for the new day
                getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE).edit()
                        .putInt( s: "pauseCount", steps).commit();
            }
        }
        db.saveCurrentSteps(steps);
        db.close();
        lastSaveSteps = steps;
        lastSaveTime = System.currentTimeMillis();
        showNotification(); // update notification
        WidgetUpdateService.enqueueUpdate( context: this);
        return true;
    } else {
        return false;
    }
}
```

Figure 4.12: updateIfNecessary() method

```
private void showNotification() {
    if (Build.VERSION.SDK_INT >= 26) {
        startForeground(NOTIFICATION_ID, getNotification( context: this));
    } else if (getSharedPreferences( name: "pedometer", Context.MODE_PRIVATE)
            .getBoolean( s: "notification",  b: true)) {
        ((NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE))
                .notify(NOTIFICATION_ID, getNotification( context: this));
    }
}
```

Figure 4.13: showNotification() method

Also, to keep it running and responds to events the step-sensor event needs to be broadcasted and step sensor needs to be registered. That is why *registerBroadcastReceiver()*and *reRegisterSensor()* methods are created. See figure 4.14 for the implementations.

```
private void registerBroadcastReceiver() {
    if (BuildConfig.DEBUG) Logger.log( msg: "register broadcastreceiver");
    IntentFilter filter = new IntentFilter();
    filter.addAction(Intent.ACTION_SHUTDOWN);
    registerReceiver(shutdownReceiver, filter);
}

private void reRegisterSensor() {
    if (BuildConfig.DEBUG) Logger.log( msg: "re-register sensor listener");
    SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    try {
        sm.unregisterListener(this);
    } catch (Exception e) {
        if (BuildConfig.DEBUG) Logger.log(e);
        e.printStackTrace();
    }

    if (BuildConfig.DEBUG) {
        Logger.log( msg: "step sensors: " + sm.getSensorList(Sensor.TYPE_STEP_COUNTER).size());
        if (sm.getSensorList(Sensor.TYPE_STEP_COUNTER).size() < 1) return; // emulator
        Logger.log( msg: "default: " + sm.getDefaultSensor(Sensor.TYPE_STEP_COUNTER).getName());
    }

    // enable batching with delay of max 5 min
    sm.registerListener( listener: this, sm.getDefaultSensor(Sensor.TYPE_STEP_COUNTER),
            SensorManager.SENSOR_DELAY_NORMAL, (int) (5 * MICROSECONDS_IN_ONE_MINUTE));
}
```

Figure 4.14: implementation of registerBroadcastReceiver() and reRegisterSensor() methods.

4.3.1 Implementing the Database

To store the data, a SQLite database is created and designed in a way so that it can be manipulated using simple queries. In the following sections I will describe how the database is created with sample code.

Figure 4.15 shows the basic methods (*getInstance(),close(), onCreate(), and onUpgrade()*) of a database and figure 4.16 shows how the query table is executed.

```java
public class Database extends SQLiteOpenHelper {

    private final static String DB_NAME = "steps";
    private final static int DB_VERSION = 2;

    private static Database instance;
    private static final AtomicInteger openCounter = new AtomicInteger();

    private Database(final Context context) {
        super(context, DB_NAME,  factory: null, DB_VERSION);
    }

    public static synchronized Database getInstance(final Context c) {
        if (instance == null) {
            instance = new Database(c.getApplicationContext());
        }
        openCounter.incrementAndGet();
        return instance;
    }

    @Override
    public void close() {
        if (openCounter.decrementAndGet() == 0) {
            super.close();
        }
    }

    @Override
    public void onCreate(final SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + DB_NAME + " (date INTEGER, steps INTEGER)");
    }

    @Override
    public void onUpgrade(final SQLiteDatabase db, int oldVersion, int newVersion) {
        if (oldVersion == 1) {
            // drop PRIMARY KEY constraint
            db.execSQL("CREATE TABLE " + DB_NAME + "2 (date INTEGER, steps INTEGER)");
            db.execSQL("INSERT INTO " + DB_NAME + "2 (date, steps) SELECT date, steps FROM " +
                    DB_NAME);
            db.execSQL("DROP TABLE " + DB_NAME);
            db.execSQL("ALTER TABLE " + DB_NAME + "2 RENAME TO " + DB_NAME + "");
        }
    }
```

Figure 4.15: implementation of the basic database methods

```java
/**
 * Query the 'steps' table.
 *
 * @param columns       the colums
 * @param selection     the selection
 * @param selectionArgs the selction arguments
 * @param groupBy       the group by statement
 * @param having        the having statement
 * @param orderBy       the order by statement
 * @return the cursor
 */
```

```
public Cursor query(final String[] columns, final String selection,
                    final String[] selectionArgs, final String groupBy, final String having,
                    final String orderBy, final String limit) {
    return getReadableDatabase()
            .query(DB_NAME, columns, selection, selectionArgs, groupBy, having, orderBy, limit);
}
```

Figure 4.16: Query table

The *insertNewDay()* method shown in figure 4.17 inserts a new entry in the database, if there is no entry for the given date yet. Steps should be the current number of steps and its negative value will be used as offset for the new date. Also adds 'steps' steps to the previous day, if there is an entry for that date.

```
public void insertNewDay(long date, int steps) {
    getWritableDatabase().beginTransaction();
    try {
        Cursor c = getReadableDatabase().query(DB_NAME, new String[]{"date"}, selection: "date = ?",
                new String[]{String.valueOf(date)}, groupBy: null, having: null, orderBy: null);
        if (c.getCount() == 0 && steps >= 0) {

            // add 'steps' to yesterdays count
            addToLastEntry(steps);

            // add today
            ContentValues values = new ContentValues();
            values.put("date", date);
            // use the negative steps as offset
            values.put("steps", -steps);
            getWritableDatabase().insert(DB_NAME, nullColumnHack: null, values);
        }
        c.close();
        if (BuildConfig.DEBUG) {
            Logger.log( msg: "insertDay " + date + " / " + steps);
            logState();
        }
        getWritableDatabase().setTransactionSuccessful();
    } finally {
        getWritableDatabase().endTransaction();
    }
}
```

Figure 4.17: insertNewDay() method

The *insertDayFromBackup()* method shown in figure 4.18 inserts a new entry in the database, overwriting any existing entry for the given date. This method is used for restoring data from a backup.

```
public boolean insertDayFromBackup(long date, int steps) {
    getWritableDatabase().beginTransaction();
    boolean newEntryCreated = false;
    try {
        ContentValues values = new ContentValues();
        values.put("steps", steps);
        int updatedRows = getWritableDatabase()
                .update(DB_NAME, values, whereClause: "date = ?", new String[]{String.valueOf(date)});
        if (updatedRows == 0) {
            values.put("date", date);
            getWritableDatabase().insert(DB_NAME, nullColumnHack: null, values);
            newEntryCreated = true;
        }
        getWritableDatabase().setTransactionSuccessful();
    } finally {
        getWritableDatabase().endTransaction();
    }
    return newEntryCreated;
}
```

Figure 4.18: The insertDayFromBackup() method

The *getSteps()* method shown in figure 4.19 gets the number of steps taken for a specific date. If date is Util.getToday(), this method returns the offset which needs to be added to the value returned by *getCurrentSteps()* to get todays steps.

```
public int getSteps(final long date) {
    Cursor c = getReadableDatabase().query(DB_NAME, new String[]{"steps"}, selection: "date = ?",
            new String[]{String.valueOf(date)}, groupBy: null, having: null, orderBy: null);
    c.moveToFirst();
    int re;
    if (c.getCount() == 0) re = Integer.MIN_VALUE;
    else re = c.getInt( i: 0);
    c.close();
    return re;
}
```

Figure 4.19: *getSteps()* method

The next two methods shown in figure 4.20 *saveCurrentSteps()* and *getCurrentSteps()* saves the current 'steps since boot' sensor value in the database and reads the latest saved value for the 'steps since boot' sensor value.

```
    public void saveCurrentSteps(int steps) {
        ContentValues values = new ContentValues();
        values.put("steps", steps);
        if (getWritableDatabase().update(DB_NAME, values,  whereClause: "date = -1",  whereArgs: null) == 0) {
            values.put("date", -1);
            getWritableDatabase().insert(DB_NAME,  nullColumnHack: null, values);
        }
        if (BuildConfig.DEBUG) {
            Logger.log( msg: "saving steps in db: " + steps);
        }
    }

    /** Reads the latest saved value for the 'steps since boot' sensor value. ...*/
    public int getCurrentSteps() {
        int re = getSteps( date: -1);
        return re == Integer.MIN_VALUE ? 0 : re;
    }
}
```

Figure 4.20: saveCurrentSteps() and getCurrentSteps() methods

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Discussion and Conclusion

The main goal of this app is to provide people with a simple and easy to use solution without the hassle of learning to use the app first. This app will also motivate people to get out of their homes and exercise. That way the entire community will get benefited and become conscious about their health, which ultimately lead them to lead a healthy and happy life. But this is just the beginning, I will add new features routinely to increase the functionality of the app.

5.2 Scope for Further Developments

When it comes to the scope for future development, there are umpteen ways the app can be improved. Here is a list of things that can be and will be added in the future.

- A safe and secure end-to-end encrypted user database to interact with other people who is using the app.
- An achievement section to reward people when they will achieve a goal.
- Addition of report graphs
- Start, pause, and reset option
- Themes

REFERENCES

[1] Google play website https://play.google.com/store/apps/details?id=pedometer.steptracker.calorieburner.stepcounter&hl=en&gl=US, last accessed on 04 December, 2021

[2] Google play store https://play.google.com/store/apps/details?id=steptracker.healthandfitness.walkingtracker.pedometer&hl=en&gl=US, last accessed on 04 December, 2021

[3] Google play store https://play.google.com/store/apps/details?id=health.step.walk, last accessed on 04 December, 2021

[4] Google play store https://play.google.com/store/apps/details?id=com.tayu.tau.pedometer, last accessed on 04 December, 2021

[5] Stackoverflow website, https://stackoverflow.com/questions/tagged/android?tab=Newest, last accessed on 04 December, 2021.

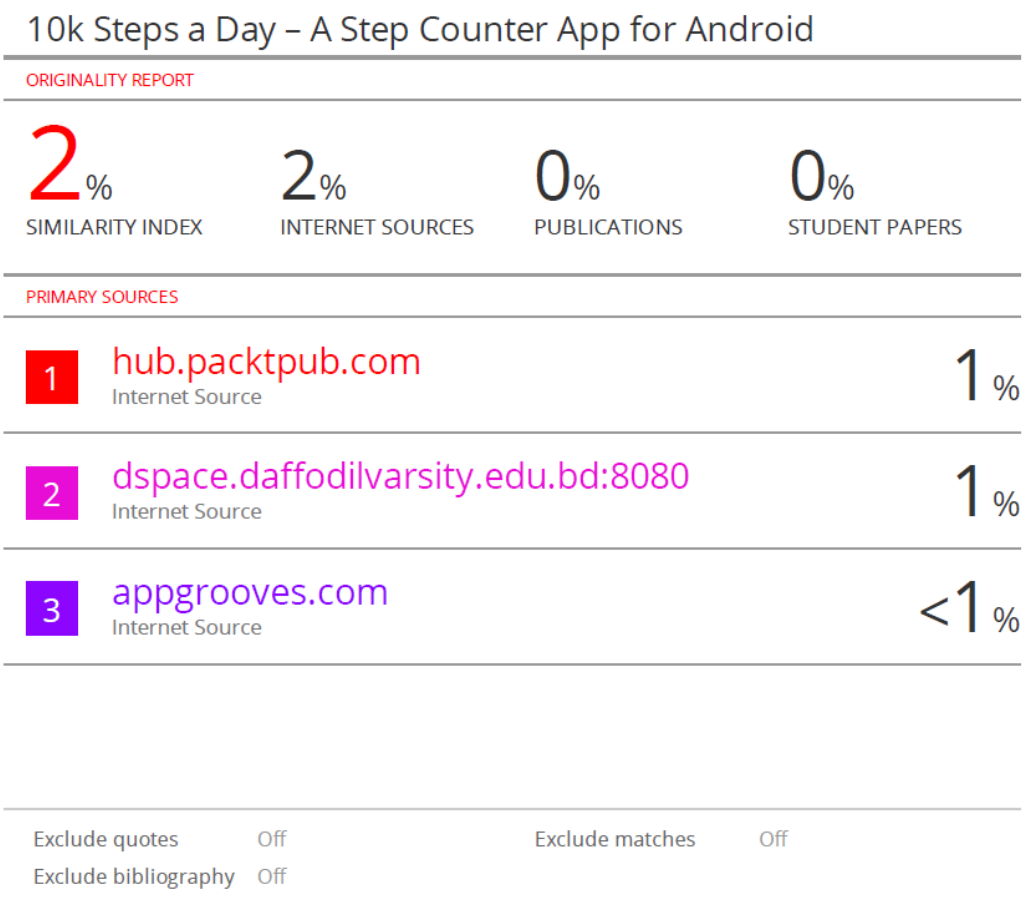
[6] Github website https://github.com/j4velin/Pedometer, last accessed on 04 December, 2021

[7] Github website https://github.com/blackfizz/EazeGraph, last accessed on 04 December, 2021

[8] Github website https://github.com/attenzione/android-ColorPickerPreference, last accessed on 04 December, 2021

PLAGIARISM

Plagiarism Report



10k Steps a Day – A Step Counter App for Android

ORIGINALITY REPORT

2%	2%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	hub.packtpub.com Internet Source	1%
2	dspace.daffodilvarsity.edu.bd:8080 Internet Source	1%
3	appgrooves.com Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		