DEVELOPMENT OF A MOBILE FINANCIAL PLANNER APPLICATION

FARIZ BIN IZHAM

OPEN UNIVERSITY MALAYSIA
2020

DEVELOPMENT OF A MOBILE FINANCIAL PLANNER APPLICATION

FARIZ BIN IZHAM

A Final Year Project submitted in fulfilment of the requirements
for the Diploma of Information Technology

Open University Malaysia
2020

DEVELOPMENT OF A MOBILE FINANCIAL PLANNER APPLICATION

ABSTRACT

Budgeting our monthly salary is essential to ensure financial peace of mind. This is true especially when it comes to family budget, standard of living, savings, and emergency funds. There are various ways of keeping track of our monthly spending but one device we use daily is a smartphone. This financial planner application is developed for smartphone running android operating system that calculate monthly utility bill prices by all electricity and water providers in Malaysia. The calculated values are then able to be saved into the application's database to be viewed and evaluated. Android studio is used to develop the application with the programming language being Java. The application made analysing and budgeting monthly spending easier by keeping track of the amount of spending each month.

Keywords: financial planner, application, smartphone, android studio, java

PEMBANGUNAN APLIKASI MUDAH ALIH PERANCANG KEWANGAN

ABSTRAK

Perancangan kewangan daripada gaji bulanan adalah penting bagi memastikan ketenangan fikiran. Perancangan kewangan memastikan keselamatan keluarga, taraf kehidupan, wang simpanan dan wang kecemasan terjaga. Terdapat pelbagai cara untuk memantau perbelanjaan bulanan tetapi satu peranti yang digunakan selalu adalah telefon pintar. Aplikasi perancang kewangan ini dibangunkan untuk sistem operasi android dan ianya mengira harga bil utiliti bulanan daripada kesemua penyedia elektrik dan air di Malaysia. Nilai yang dikira kemudian disimpan ke dalam pangkalan data aplikasi untuk dilihat dan dinilai. Android studio digunakan untuk membangunkan aplikasi ini, manakala, bahasa pengaturcaraan yang digunakan adalah Java. Aplikasi ini membantu membuat analisis dan merancang kewangan bulanan lebih mudah dengan pengawasan perbelanjaan bulanan.

Kata Kunci: perancang kewangan, aplikasi, telefon pintar, android studio, java

ACKNOWLEDGEMENT

I would like to take this opportunity to express my gratitude and appreciation to my supervisor, Puan Norlida Binti Nasaruddin for the guidance, patience, and invaluable advice throughout this project.

I also would like to express my appreciation to my family and friends for their endless support whenever I face problems. Without the mentioned parties, it is impossible for me to complete this project report successfully.

THANK YOU.

FARIZ BIN IZHAM
15 DECEMBER, 2020

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

LIST ABBREVIATIONS

IDE Integrated Development Environment

VM Virtual Machine

CPU Central Processing Unit

JIT Just-In-Time Compilation

AOT Ahead-Of-Time Compilation

NPE NullPointerException

SDLC Software Development Life Cycle

RAD Rapid Application Development

OS Operating System

ERD Entity Relationship Diagram

1NF First Normal Form

2NF Second Normal Form

3NF Third Normal Form

UML Unified Model Language

APK Android Package

CHAPTER 1

INTRODUCTION

1.1 Background

Utility bill calculator is an android platform application where the user can calculate the amount for electricity and water bills whenever they require without calculating each stage. The user will just input the final kWh amount, or the litres of water amount so that the application will show the cost in Ringgit Malaysia. The application was developed using Java language and Android Studio as the integrated development environment (IDE).

1.2 Problem Statement

In Malaysia, electricity and water cost are supposed to be calculated and billed every month but sometimes, there might be issues such as losing the bill paper during rain and storm. Sometimes the staff is unable to get into the proximity of the house due to a locked gate to check the electric meter and this requires them to add your previous bill to the next month which increases the amount it is supposed to be. The utility bill calculator application address this by letting the user calculate their own meter themselves by inputting the amount (kWh) or litres of water that they use at the end of every month.

1.3 Objectives

1. To provide an easy way for Malaysians to calculate their bills by calculating their kWh and litres of water amount into bill prices which is shown in Ringgit Malaysia and plan their monthly budget.

2. Provide flexibility for users to calculate their bills at any time of the month as opposed to once a month.

3. Easy to use application with user friendly user interface.

1.4 Project Scope & Limitation

1. Project Scope
    - The project is to build a simple financial planner application to calculate monthly utility bills.

2. Project Deliverables
    - Users will be able to calculate their utility bills by inputting the kWh and the litres of water amount into the application which the application will then calculate the final amount in Ringgit Malaysia.
    - Users will also be able to track their monthly bill price and other spending by looking at the calculation history.
    - The application can also be used as a financial planner to input other type of spending which can be saved into the database.
    - Tariff rates for all electricity and water provider can be checked using the application.

3. Project Justifications
    - Budgeting Malaysian's monthly allowance will be easier with the application as they can input the kWh and litres of water amount shown on their meter and the application will calculate the final amount in Ringgit Malaysia.

4. Project Timeline
    - Analysis, design & user interface design shall be completed within the first semester.
    - Application development and bug testing shall be completed within the second semester.

1.5 Software & hardware requirements

1.5.1 Software Requirements

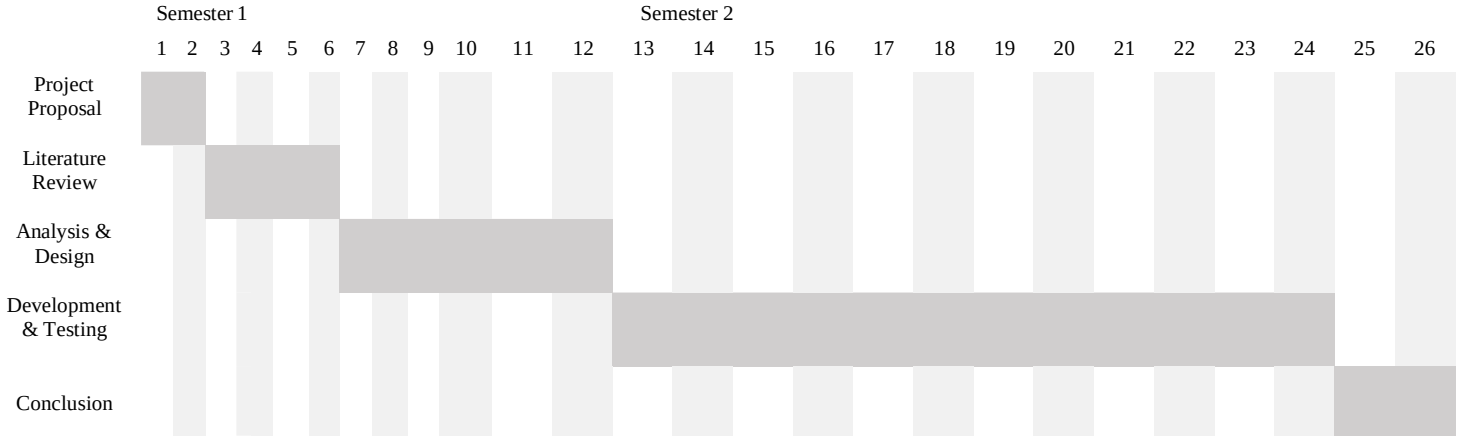
1. Android Studio to develop the project, the software can be run on any operating systems available.
2. Android Operating System, the operating system has to be used on an emulator or on a physical device.

1.5.2 Hardware Requirements

1. Laptop or Computer to run android studio and develop the project.
2. Android Smartphone running android operating system version 5.0 or above. The hardware does not have to be powerful as it is a lightweight application.

1.6 Gantt Chart

The Gantt charts is by weeks.



1.7 Summary

The objective of this application is to help Malaysians budget their monthly allowance by using the tip of their finger by utilizing their smartphone. The application will also show the final amount in Ringgit Malaysia after inputting the kWh and litres of water amount. The application can also be categorized as a simple financial planner with a simple and easy to use user interface to help users plan their finances correctly every month.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

2.1.1 Utility Bill Calculator Application

Utility bill calculator is an android application to be used in Malaysia as an alternative way to calculate monthly utility bills. This is great for monthly financing as users will be able to track their monthly bills consumption and budget accordingly. Furthermore, many aspects should be considered when developing an android application, for example, what is the design principles of the application? What language to use to develop the application? What options are there available when choosing between different software engineering methodologies? Various articles and journals are referred to in this literature review to answer these questions.

2.1.2 History of Android

According to an article by Callaham (2020), he stated that Android is originally developed to improve the operating systems of digital cameras. Furthermore, he says the company made pitches that showed Android installed on a camera are connected to a PC and the pictures can be stored to an "Android Datacentre". The current android is developed when the original Android developer was acquired by Google. When Apple launched the iPhone in 2007, Android was still in development by Google in secret. An article by Bhatt (2011) states that android was launched as an open-source mobile operating system that works on the Linux platform. He also added that it consists of the operating system, middleware and user interface, and application software. On the official Android about page, Google has also launched android for television, wristwatch, and in-car infotainment. Android has various release versions ranging from 1 to 11 as of 2020.

2.2 Java

A Java programming book by Herbert (2020) states that Java is a programming language that can be found on many types of devices ranging from smartphones to mainframe computers. He further adds that Java does not compile on native processor code but relies on a virtual machine (VM) which understands an intermediate format called Java bytecode. A journal made by Bose, Kundu, Mukherjee, & Banerjee (2018) adds that on android the original VM is called Dalvik and the VM interprets bytecode which is a set of instructions similar to machine code found in CPUs. Furthermore, they also state that the VM uses just-in-time compilation (JIT) and ahead-of-time compilation (AOT) to speed up the process. Android applications can be developed on Windows, Linux, or OS X thanks to these technologies. After developing the application, the Java compiler converts the source code into bytecode and executes them on the built-in Android VM.

2.3 Software Development Methodology

High-quality software is developed and tested with a process called software development life cycle (SDLC). Software development life cycle is a process that includes planning on how to develop, maintain, replace, and enhance or alter the software.

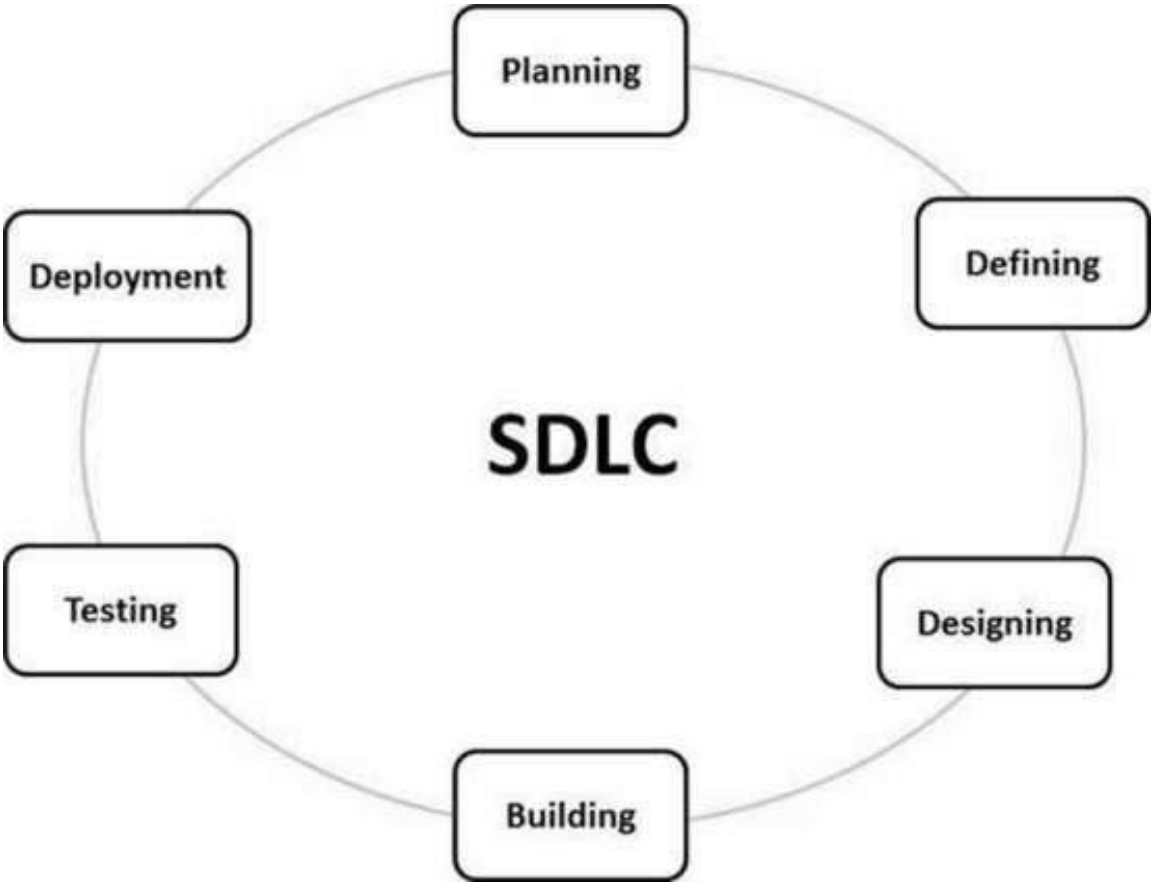


Figure 2.1: SDLC Overview [Digital image]. (n.d.). Retrieved from https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

An article on TutorialsPoint states that there are many software development life cycles, and these models are referred to as software development process model. They also wrote that each process has different processes or steps to succeed in the process of software development. In this project, the Rapid application development model is used.

2.3.1 Rapid Application Development

Rapid application development (RAD) is the opposite of the waterfall model whereas the waterfall focuses on planning and scheduling, RAD focuses on rapid prototype releases and iterations (Singh, 2019). The following are the summary of the steps required by RAD:

1. Define requirements
    - Requirements are gathered to define and finalize the project.

2. Prototyping
    - Begin development after gathering requirements, designers will be working with clients to develop and improve the product until it is ready.

3. User Feedback
    - Prototypes are released to the user and feedbacks is gathered to modify and improve the product.

4. Cutover
    - The finished product is launched; this is where the users are trained and the migration for the product takes place.

2.3.5 Summary

For the utility bill calculator application, the rapid application development model is used as it is the most time-efficient model which also produces a high-quality product as compared to the other SDLC model.

2.4 Financial Planner

A computerized financial planner is not a new idea, there has been a patented computer-implemented program for financial planning and advice system. William J. Moran patented a system that allows for an advisor to give client proactive and efficient service (U.S. Patent No. 6,430,542, 2002). The system includes features such as inter alia, demographic, and financial data files, a virtual executor , a financial simulator, an earmarker, a strategizer, and a proposal constructor. The utility bill calculator is a form of a computerized financial planner but for calculating utility bills.

2.4.1 Psychological foundations of financial planning for retirement

A journal by Hershey, Jacobs-Lawson, Mcardle, & Hamagami, (2007) shows the psychological mechanisms that come with financial planning for retirement. The authors found that if an individual has a higher income, they are more likely to have thought more about retirement which, in terms, have made a clear goal for retirement in their lives. Furthermore, in the journal, the authors have shown that those who are more knowledgeable about financial planning are more likely to contribute more to retirement savings to achieve financial independence. By using the utility bill calculator, users can gain knowledge on how to budget their monthly salary for their monthly utility bills.

2.4.2 Summary

The utility bill calculator is a great start to learn about financial planning as users can achieve the goal of budgeting for their monthly utility bill by using the calculator. User can also view their bill history and compare them on a month-by-month basis.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1 Feasibility Studies

3.1.1 Cost-Benefit Analysis

To estimate the cost and benefit of a project, several things are considered. For example, the cost of the system, the benefits that the project gives and if whether the benefits outweigh the project costs. If the project is economically feasible, then the project benefits is more than the project cost.

1. System Costs

    a) Development Cost

- There are two development costs in this project, equipment cost and software cost. The equipment required for this project is a laptop or a computer and an android smartphone. Both are readily available; therefore, the equipment cost is null. For the software cost, the software used in this project is Android Studio which is available to download for free by Google, therefore, the software cost is also null.

    b) Ongoing Cost

- The application will be maintained after the project is finished, new features such as internet bill calculator and mobile bill calculator might be added, and undiscovered bugs will be fixed. The tariff rates will also be updated accordingly.

2. Benefits

    a) Performance Benefits

- Using resources and documentation from the internet, the application will be developed swiftly with few error and bugs.

b) Cost Avoidance

- As most of the software and hardware require to develop the project is free and readily available, no new purchase is required.

c) Project Benefits

- The development of the project can benefit the developer by having to learn new method, technique on various android libraries, database usage and so forth. The application can also be used by the developer as a financial planner and finally, if the application is uploaded to Google Play Store, it can be monetized.

3.1.2 Technical Feasibility

In the technical feasibility, several questions are answered in this study and analysis, is the project practical? Is technology now available to develop the project? And are there enough time to develop the project?

1. Project Practicality

The project objective is to provide an easy way for Malaysians to calculate their monthly utility bills. It can also be used to calculate their monthly bills at any time by just looking at their electric or water meter. The project will be developed for android. Several aspects are considered when developing an Android application.

a) Expertise

- Android development requires the developer to know the language of Java and for Android libraries, the documentation can always be referred to during the project's development. Besides, Android has many developers that do tutorials and share resources over the internet.

b) Financial Support

- As analysed from the cost-benefit analysis, the cost requires to develop this project is practically non-existent as most of the hardware and software requires are readily available and free to use.

c) Requirements Gathering

- Requirements for the project are also gathered from the project proposal. All features required are stated in the project proposal.

2. Technology Availability

The technologies to develop the project are all available using Google's Android libraries. For the database, SQLite will be used and is integrated with Android development using Android Studio. Data, such as electricity bill, water bill and other spending are stored on the SQLite database.

3. Time

The time given to develop the project is six months. There is enough time to develop the project with proper planning and proper planning has been done for this project using Gantt Chart in project proposal. The actual development time is three months while documenting the proposal, literature review, analysis and design, development, and conclusion is another three months.

3.1.3 Operational Feasibility

Operational feasibility is to gauge the possibility of maintaining and supporting the project once it enters production. Several operational feasibility are considered developing this project.

1. Maintenance

a) Tools to Maintain the Project

- The only tools required to maintain the project is android studio and the source code.

b) Skills to Maintain the Project

- The project requires the developer to have adequate Java, Android Development, and SQLite knowledge.

c) Process to Update

- The project requires constant updating of tariff rates for both water and electricity bills.

d) Documentation

- The project is fully documented with the project proposal, literature review, and design and analysis.

2. Support Issues

a) User Documentation and Training

- The application will be developed with ease of use in mind. User interface design will be minimalistic and easy to navigate. User will be able to gauge what the application does through the user interface.

b) User Request and Bug Report

- User will be able to request or report any bugs by emailing the developer. The email will be provided in the application.

3.2 Requirement Methods

Requirements are important as it covers the methods, needs and conditions of the projects. In this requirement methods, data requirements, functional requirement, software requirement and hardware requirement be covered and fully discussed.

3.2.1 Data Requirement

The data captured by the software will be inputted by the user. The data are kWh and litres of water that the users wants to calculate using the application. Both data will be converted into Ringgit Malaysia using the latest tariff rates by the electricity and water provider.

3.2.2 Functional Requirement

1. Electricity Bill Calculator
    - Users should be able to input the kWh amount shown on their electric meter and the application will automatically convert it to Ringgit Malaysia based on the tariff rates of their provider.

2. Water Bill Calculator
    - User should be able to input the litres of water amount and the application will automatically convert it to Ringgit Malaysia based on the tariffs rate of their provider.

3. Other Spending Calculator
    - User will be able to input any other spending into the application to be saved and included in their total spending amount and history. For example, mobile bills, internet bills amongst others.

4. History
    - Users should be able to check their calculated history to compare the amount in a month-by-month basis. All calculated amount made by the users will be logged in the history and database of the application.

5. Tariff Rates
    - Users should be able to check the tariff rates using the application.

3.2.3 Software Requirement

1. Android Studio
    - Android studio is the official IDE for android operating system. It is based on JetBrains‘ IntelliJ IDEA. Both Kotlin and Java can be used on Android Studio but Kotlin is Google’s preferred language for Android application development. Java will be used to be develop the utility bill calculator application. It can be downloaded for free on Google’s developer website.

3.2.4 Hardware Requirement

1. Android Smartphone
    - As the application will be developed for Android, an android smartphone is required to run the application. The application does not require high-end component as it is a lightweight application.

Figure 3.1: Android Smartphone

2. Laptop or Computer
    - A computer or laptop is required to develop this project. A computer or laptop will be used to compile and install the application on the android smartphone.

Figure 3.2: Laptop

3.3 System Development Methods

3.3.1 Rapid Application Development

Rapid Application Development (RAD) model will be used for this project. The RAD model is one of the best methodologies to achieve high quality product with the shortest amount of time. The RAD model is also excellent in getting early feedback from end users from their requirements. RAD has four phase of development cycle derived from the standard System Development Life Cycle (SDLC). The four phases are planning, user design, construction, and cutover and testing.

1. Phase 1: Requirement Planning

This phase is to acknowledge about the project's problem statement, scope, and objective. The hardware and software are also acquired in this phase to achieve the objective of the project.

a) Problem Statement

- The project is developed as a financial planner, but it can also aid calculating user's utility bill every month if user does not receive their bill paper due to unforeseen circumstances.

b) Project Scope

- Project Deliverables: User will be able to calculate their utility bills by inputting the kWh and litres of water amount into the application. The application will then convert those value to Ringgit Malaysia.
- Project Deliverables: Users can track their monthly bill by looking at the calculation history from the application. The calculated values are stored using Android built-in SQLite database.
- Project Justification: For Malaysian to budget their monthly salary using an easy-to-use financial planner.
- Project Timeline: The analysis and design will be completed within three months and the project development will be completed within another three months.

c) Project Objective

- To provide an easy way for Malaysian to calculate monthly utility bills and budget their monthly salary accordingly.
- Calculating bills amount in Ringgit Malaysia from kWh and litres of water amount inputted by the user.
- Provides flexibility to user to calculate their utility bills at any time of the month as opposed to once a month.
- Simple and easy to use user interface to navigate around the application.

d) Software Requirement

- Android Studio: Android Studio will allow programmers to develop application for any devices running Google's Android operating system. It also contains an android emulator to run and test the software during the development phase.

e) Hardware Requirement

- Android Smartphone: The application will require a smartphone running Google's Android operating system. Users will be able to access all the features of the project using an Android Smartphone.
- Computer or Laptop: The development of the application will be done on a computer or a laptop. Any operating systems from Windows, macOS, Linux and Chrome OS can be used to develop Android application using Android Studio.

2. Phase 2: User Design

This phase is to give us better understanding of the project from the user's perspective. The application will be developed based on the flow process of the project. The initial design of the project is shown in figure 3.3 below.

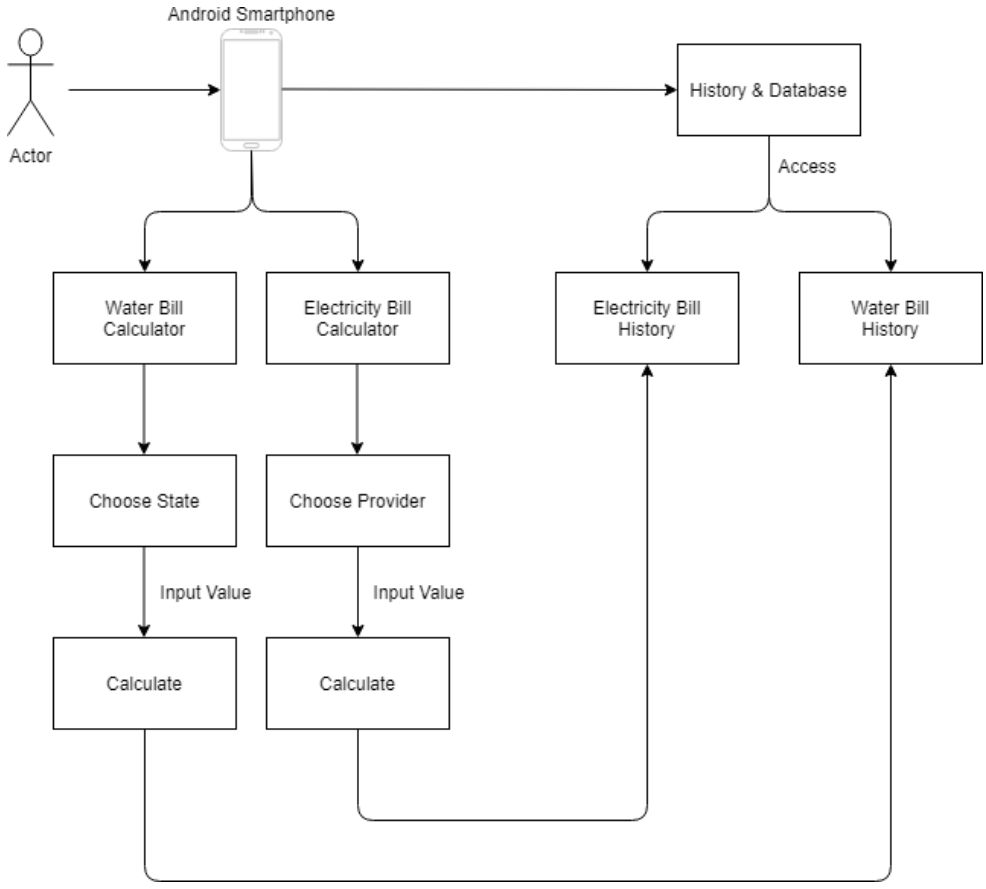


Figure 3.3: User Design

3. Phase 3: Construction

The proposed user design is constructed in this phase. The first part of construction is to develop the electric bill calculator with all providers available in Malaysia ensuring the calculation is correct from the latest tariff rates from all the providers. Next, the water calculator is developed with all the providers and tariff rates built-in and ensuring the calculation is correct. Finally, the database is developed storing values from previously calculated amount for users to check and compare.

4. Phase 4: Testing & Cutover

In this final phase, the project is then installed on an android smartphone, testing all the functions ensuring all the features, tariff rates, formulas and calculations are correct.

3.4 Data and Process Modelling Diagrams

Data flow diagram in this project will represent all the functions, process, manipulating, store, and distribution of data between all the components of a system. Figure 3.4 below shows the context level diagram and figure 3.5 shows the level 1 data flow diagram.

3.4.1 Context Level Diagram

In the context level diagram, there are five data flows which moves data from the user input to the utility bill calculator system and the data are then calculated and shown to the user again.

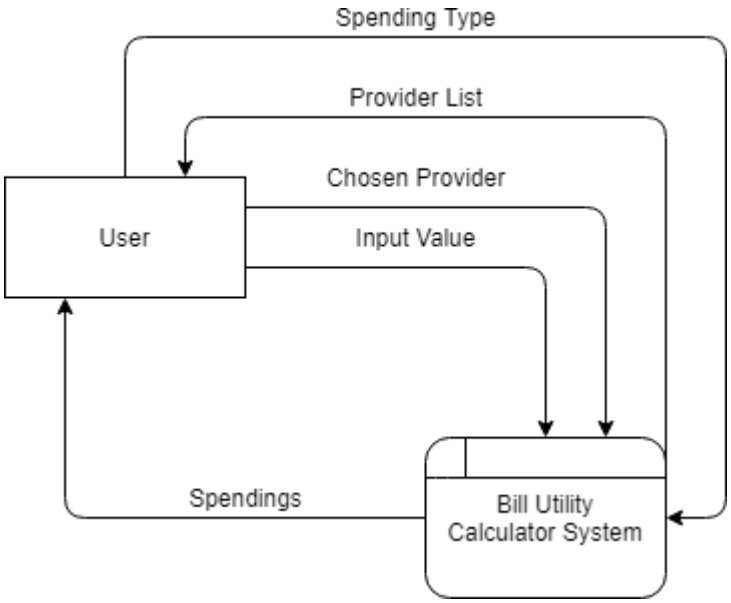


Figure 3.4: Context Level Diagram

3.4.2 Level 1 Data Flow Diagram

In the level 1 data flow diagram, the utility bill calculator system is expanded to show the technical details. First, user input the types of bill to calculate, afterwards the system then provides users with the provider list of the selected bill types. The user then selects the provider type, and the system programs the calculator to set the tariff rates for the selected provider. The user finally input the value to calculate which the system calculates and store it in the database. The final amount can then be accessed and viewed by the user.

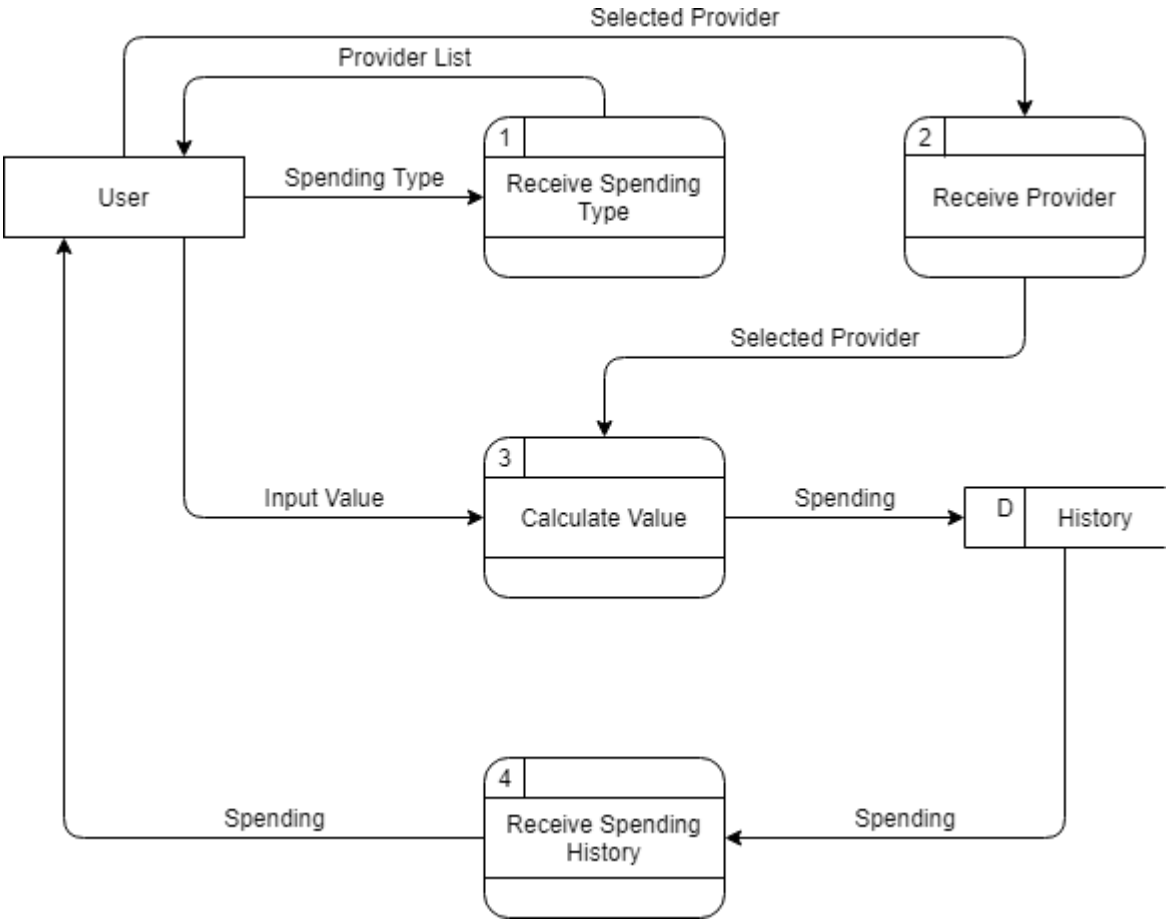


Figure 3.5: Level 1: Data Flow Diagram

3.5 Entity Relationship Diagram

Entity relationship diagram (ERD) is drawn to show the structural diagram for database design in this project. ERD also ensures high-quality database design is produced during database creation. For this project, only one ERD is required, Spending. The attributes for the Spending entity are SpendingID, type, name, date, toCalculate and cost. SpendingID is the primary key.

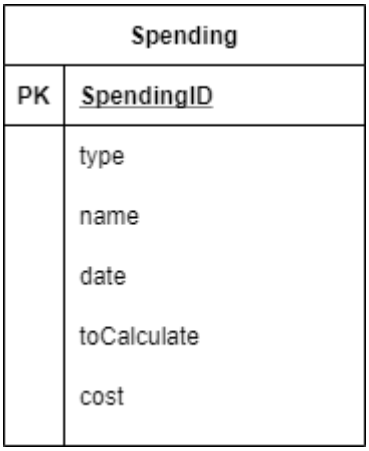


Figure 3.6: Entity Relationship Diagram

3.6 Database Normalization

Database normalization is used to discard redundant data and to ensure data dependencies. In this project, there is only one table to store data, the spending data. The database will be normalized.

3.6.1 First Normal Form & Second Normal Form

In the first normal form (1NF), there are three rules, the attributes value should only contain a single value, the stored value in each column must only be the same type and the column name should be unique. For the second normal form (2NF), the relation must be in first normal form and there should be no partial dependency. Partial dependency is when a non-primary key attribute is not dependant on the primary key in a table.

Table 3.1: Spending Table Example

SpendingID (PK)	**type**	**name**	**date**	**toCalculate**	**cost**
1	Electric	Tenaga Nasional	2020-12-01	683	279.84
2	Water	Kelantan	2020-11-05	44	36.33
3	Transportation	Bus	2020-11-03	NULL	50
4	Food	Lunch	2020-10-05	NULL	8

The table is already in first normal form where each attribute value only contains a single value, the stored value is the same type, and the column names are unique and the table is also already normalized to the second normal form where the table is in first normal form and all non-primary key attributes are dependant on the primary key, SpendingID.

3.6.2 Third Normal Form

The rules for third normal form (3NF) are the table must be in second normal form and it should not have transitive dependency. Transitive dependency happens when a B attribute depends on A and C attribute depends on B but does not depends on A. In this project, there is a calculated value which is toCalculate value to cost value.

Table 3.2: Spending Table Example

SpendingID (PK)	**type**	**name**	**date**	**toCalculate**	**cost**
1	Electric	Tenaga Nasional	2020-12-01	683	279.84
2	Water	Kelantan	2020-11-05	44	36.33
3	Transportation	Bus	2020-11-03	NULL	50
4	Food	Lunch	2020-10-05	NULL	8

The spending table does not satisfy the rule of no transitive dependency as the cost attribute is dependent on the toCalculate value while toCalculate value is dependent on the SpendingID value. As the cost attribute is calculated from the toCalculate attribute, preserving the third normal form is possible by calculating as needed and removing the cost attribute from the table but as user's spending increases, the application performance will deteriorate when a user views their spending history. For example, to populate the history list view of cost values, the application will have to calculate all the cost value again from the stored toCalculate value which slows down the application. Therefore, in this project, the Spending database will be denormalized because the calculated value (cost) will be stored in the database to preserve performance.

3.7 Unified Modelling Language

Unified model language (UML) is used to visualize, construct, and document the parts of a software systems by software developers. It is also used for business modelling and non-software systems. In this project, a UML use case diagram will be drawn to visualize and plan the project that will be used throughout the development.

3.7.1 Use Case Diagram

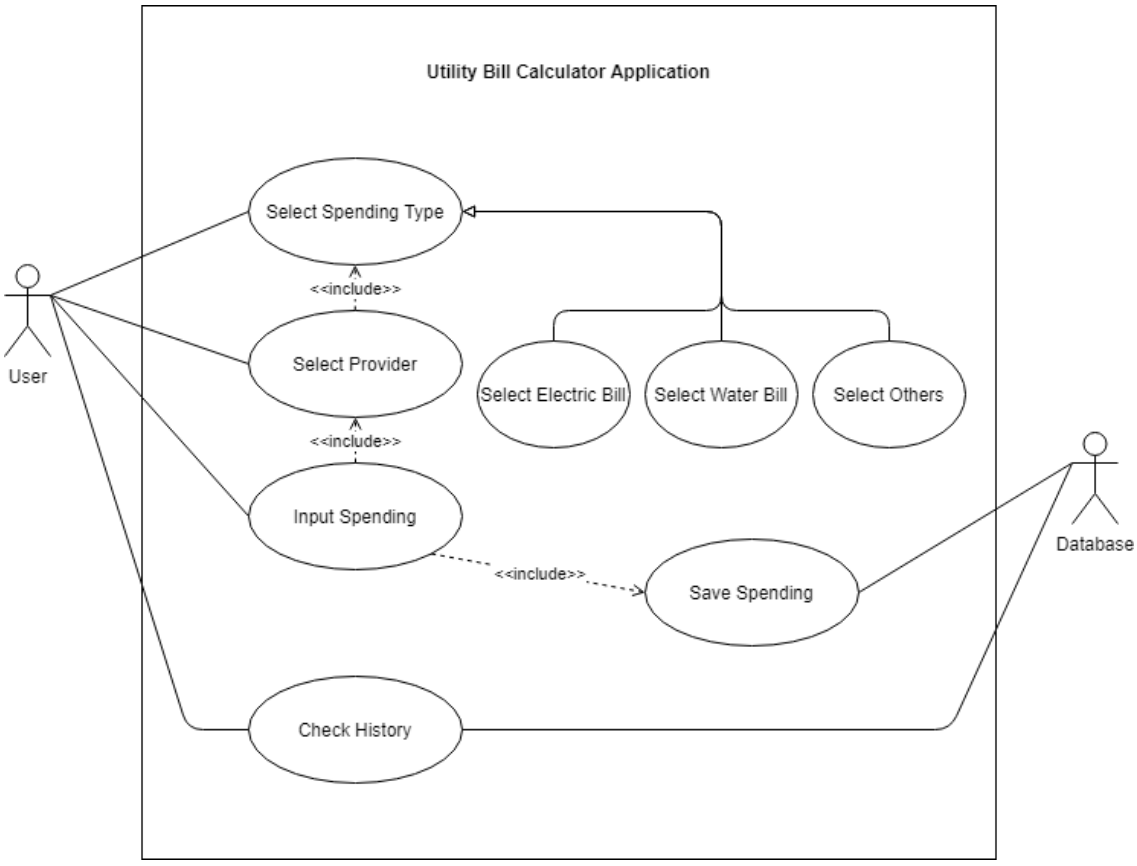


Figure 3.7: Use Case Diagram

1. Actors

- There are two actors in the use case diagram, user, and database. User will interact with the application while database store calculated value and retrieves calculated value.

2. Use Case

a) Select Spending Types

- User will choose the spending type here and generalization relationship exist in this use case where user select either the water bill, electricity bill or other spending.

b) Select Provider

- If user selects a bill type, user will choose the provider afterwards. Here, an include relationship exist to show user the proper provider for their selected bill types.

c) Input Spending

- For bill types, user will input the amount to calculate after selecting a provider. The application will then convert the toCalculate value to Ringgit Malaysia. For other spending, user will only input the spending cost, name and type. Two include relationship exist in this use case. First, the calculator will be programmed based on the chosen provider for bills spending. Second, the calculated value will be stored to the database.

d) Check History

- The spending value will be shown here in the history section of the application. The values are retrieved from the database.

CHAPTER 4

SYSTEM IMPLEMENTATION AND TESTING

4.1 System Manual

4.1.1 Main Page

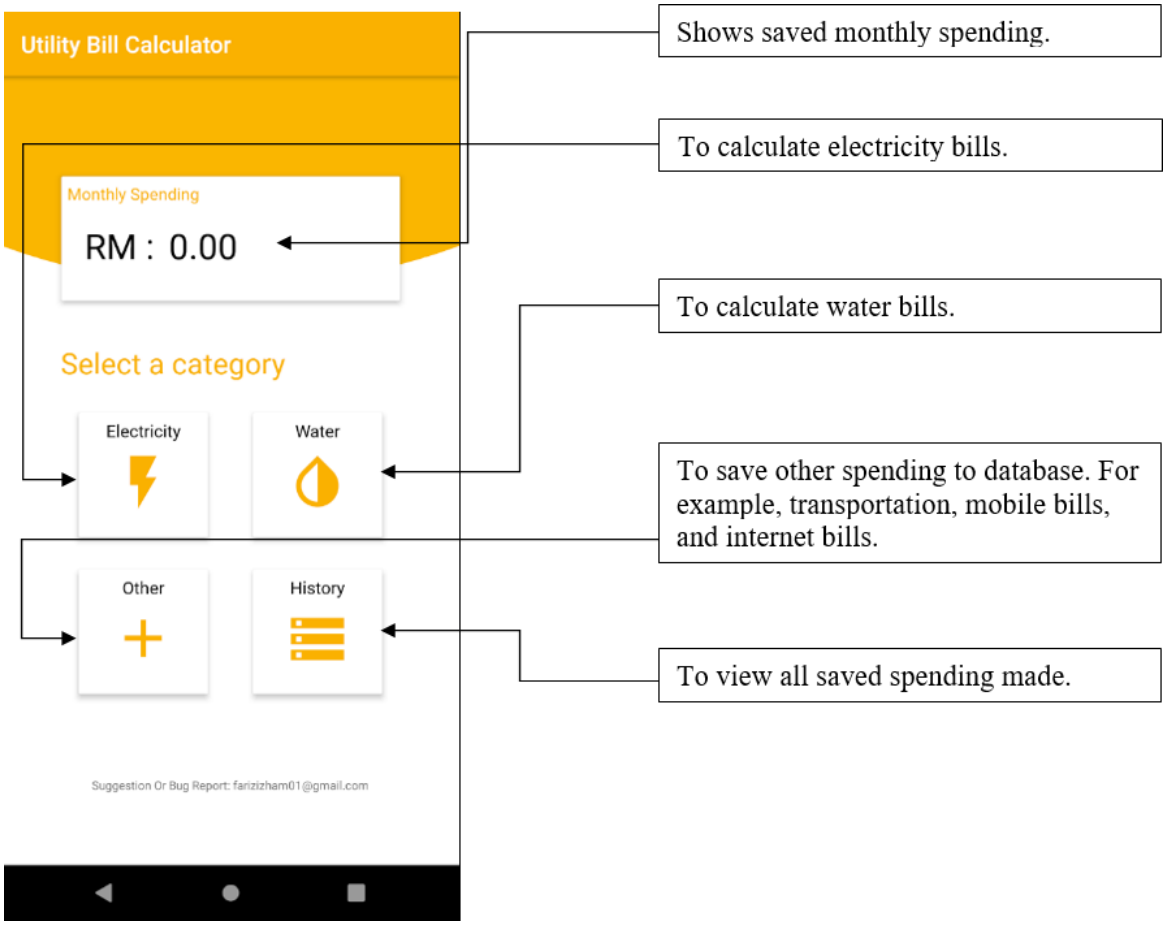


Figure 4.1: Application Homepage Manual

4.1.2 Electric & Water Provider & Calculator

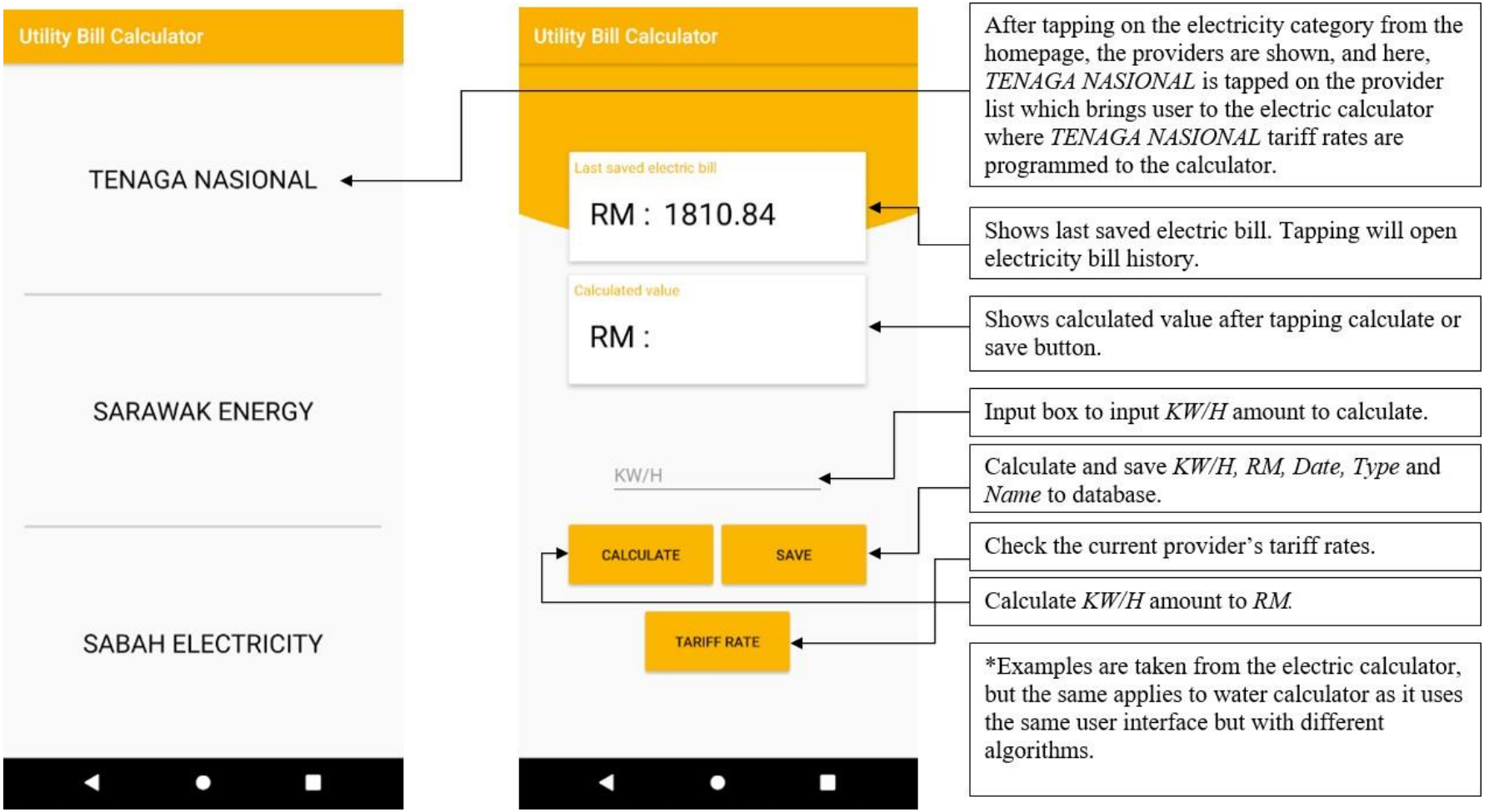


Figure 4.2: Electric & Water Provider & Calculator Manual

4.1.3 Other Spending Calculator

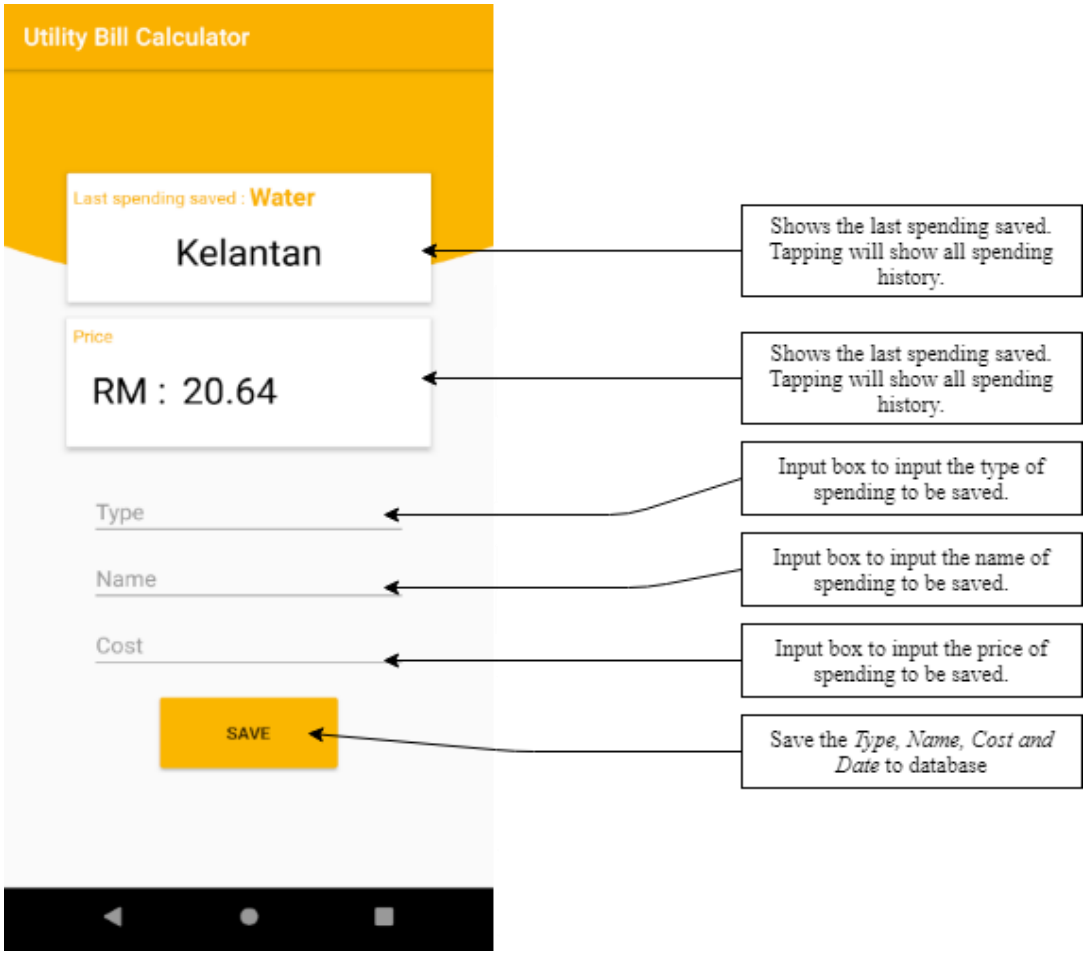


Figure 4.3: Other Spending Calculator Manual

4.1.4 History Choices

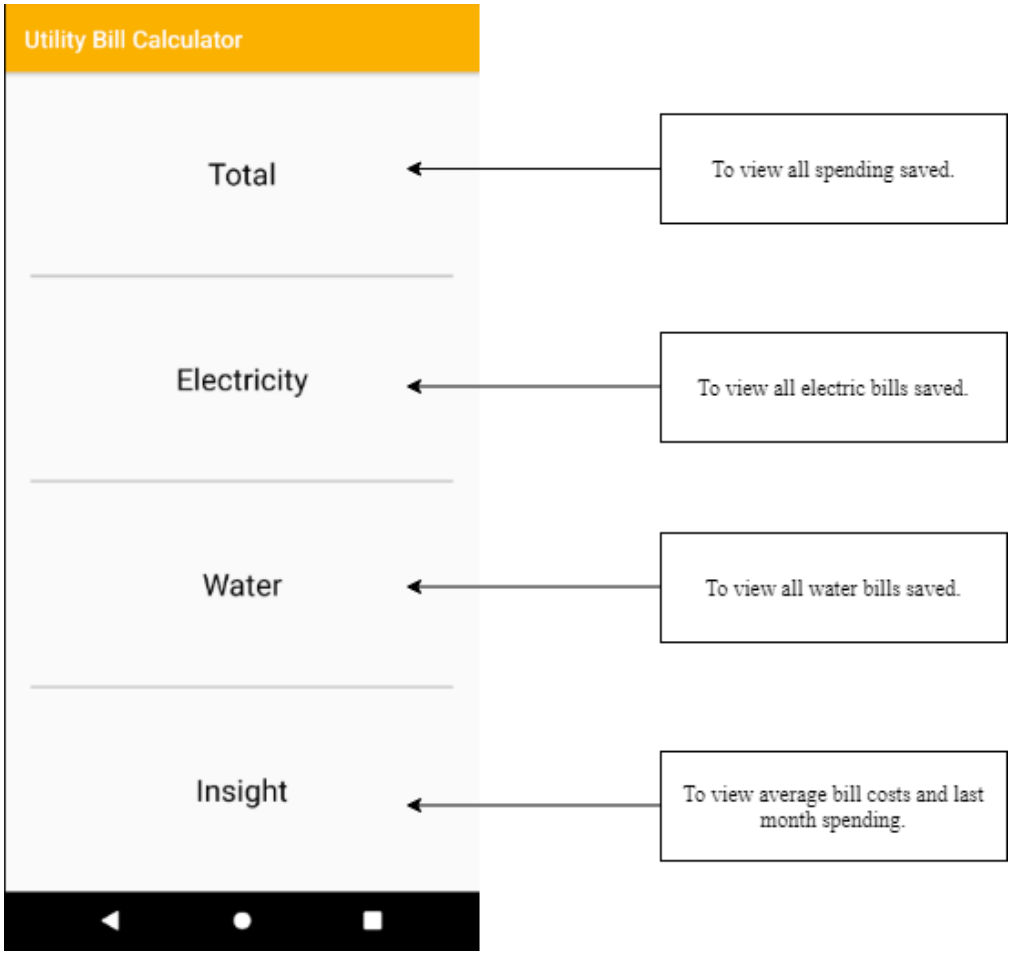


Figure 4.4: History Choices Manual

4.1.5 All Spending History

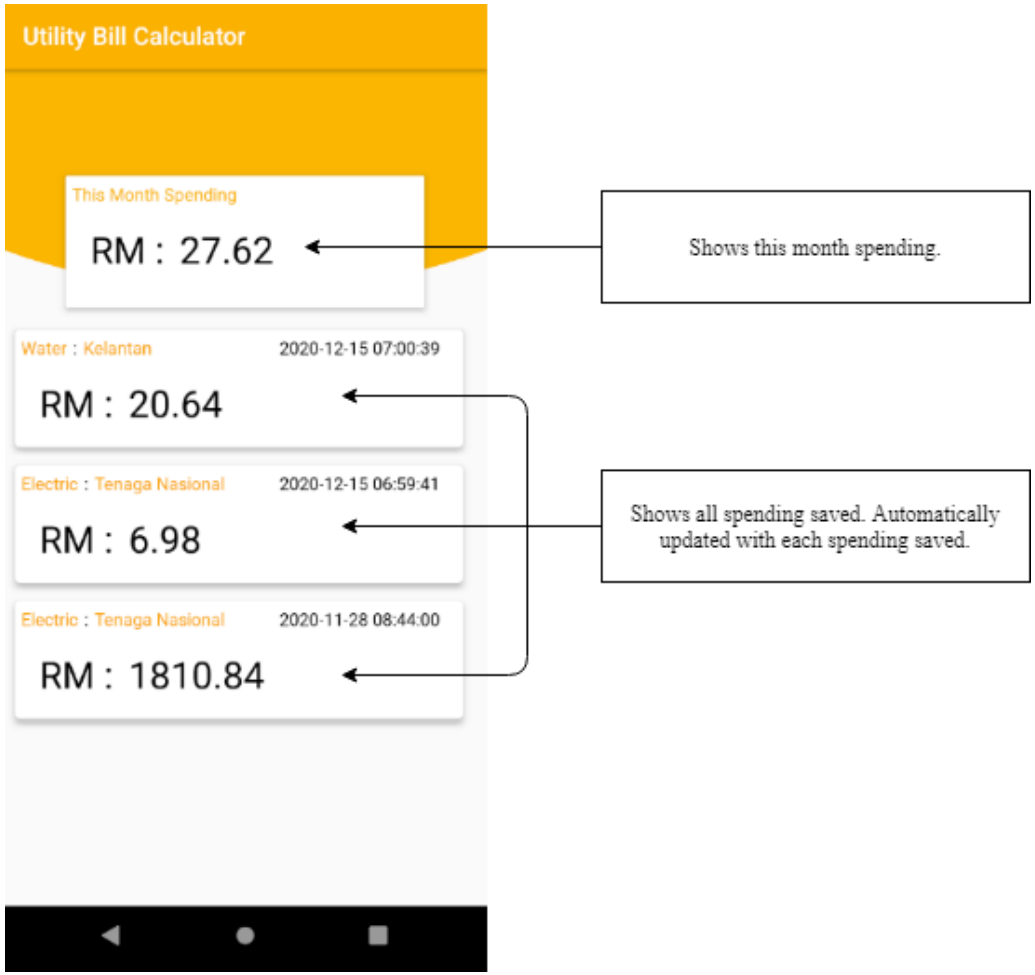


Figure 4.5: All Spending History Manual

4.1.6 Electric Bill History

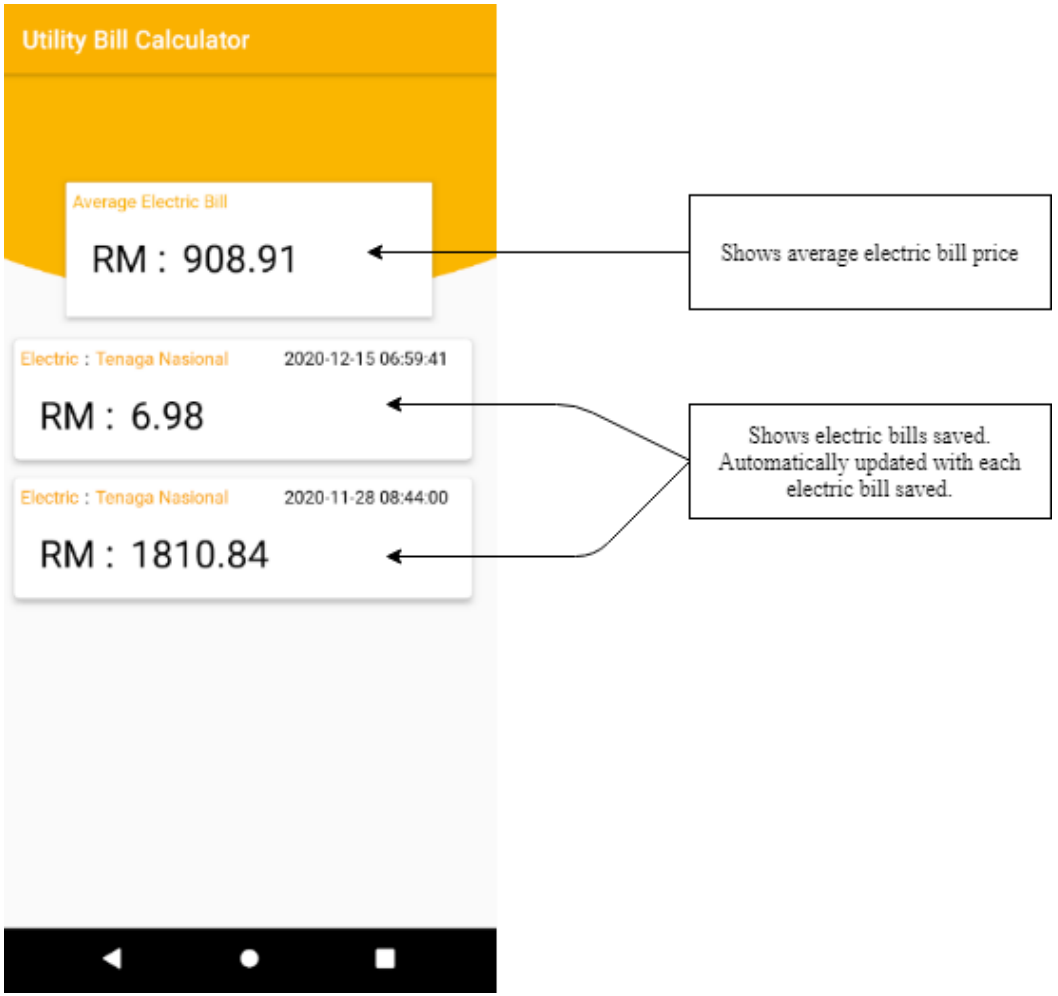


Figure 4.6: Electric Bill History Manual

4.1.7 Water Bill History

Figure 4.7: Water Bill History Manual

4.1.8 Spending Insight

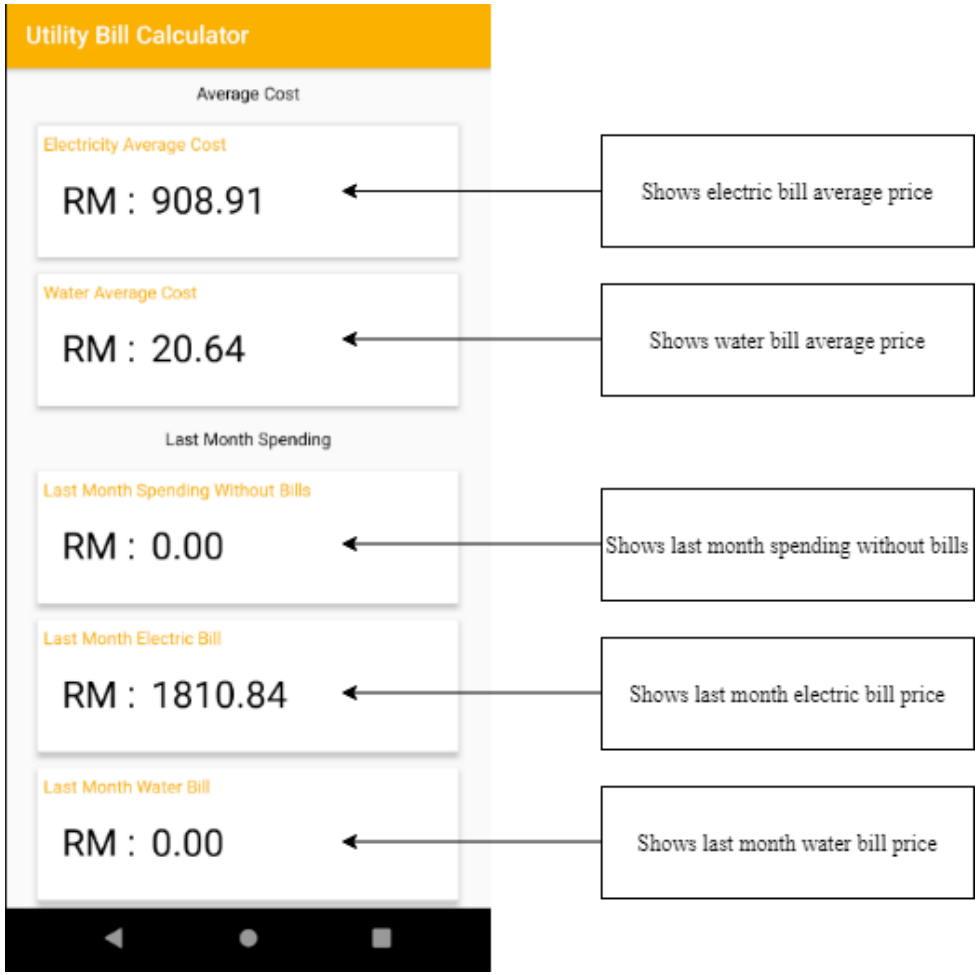


Figure 4.8: Spending Insight Manual

4.2 Installation Manual

Installing the application is easy as only the android package (APK) file is required. The steps are as follows:

1. Obtain the APK file.

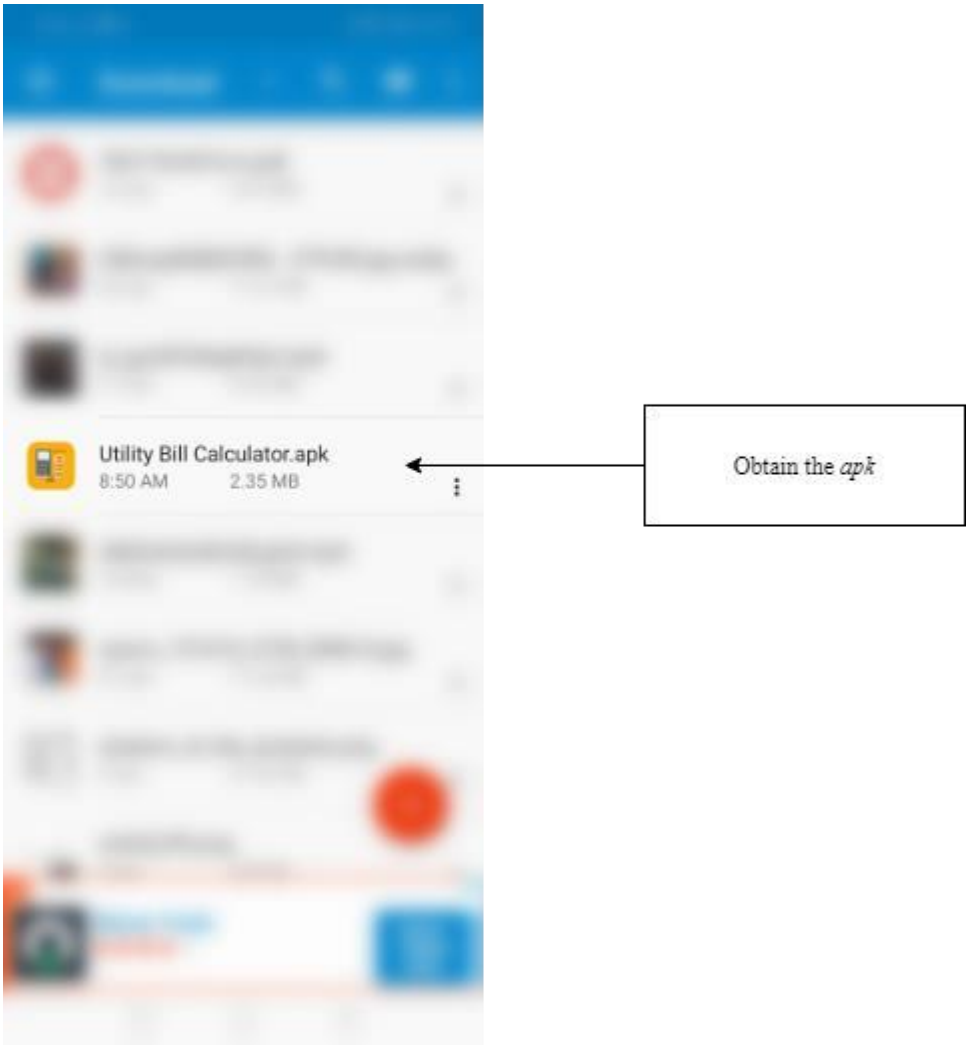


Figure 4.9: Obtain the APK File

2. Install

Figure 4.10: Install the Application

4.3 Testing Plan

4.3.1 Testing of Main Features

1. First Two Features

   The first step to developing the project is making sure all the features stated in project proposal are delivered. The first two features built are exactly as planned which are both, electric and water bill calculator. The plan was to test the algorithm so that the output is correct. To test this, all providers are tested using pre-existing bills and online calculator if available.

2. Database

   The next feature developed for the application is a way to save and view calculation history. This is achieved by using Android's built in SQLite local database.

With the database integrated into the application, all calculations can now be saved and analysed by users. The database is tested by inputting various amounts into the database and querying the value to the Android's built-in log class, toast, and showing all the queried values in the user-interface.

4.3.2 Handling Crashes, Invalid Value & Exception Error

After finishing the main features, the application is then tested for bugs, crashed, output value and exception error.

1. Exception Error

   One of the most critical part of using the Java programming language is to ensure all object are initialized properly before using them. In the project's example, if users left the input box empty in the bill calculator and tries to tap on calculate or save, it would cause the application to crash with an exception error. This is mitigated by using Java's try... catch block. Code snippet with try...catch block is given below from the calculator class and the full source code is given in Appendix A.

```
try {
    String toCalculate = userInputToCalculate.getText().toString();
    Double toCalculateAmount = Double.parseDouble(toCalculate);
    Spending spend = new Spending(toCalculateAmount, type, choice);
    String resultDouble = String.format("%.2f", spend.getCost());
    resultText.setText(resultDouble);
} catch (NumberFormatException e) {
    Toast.makeText(CalculatorBill.this, "Invalid Amount!", Toast.LENGTH_SHORT).show();
}
```

2. Invalid Value

   As Java calculate double to their exact value, calculated bill price will not be accurate. During testing, the calculated cost of bills will not be rounded to two decimal places. There are many solutions to this problem and in this project, String.format method is used to round up the final value to two decimal places. The String.format code snippet is shown below.

```
String resultDouble = String.format("%.2f", spend.getCost());
```

After the method is applied, the final value is now accurate. Figure 4.15 below shows the accurate value after applying the method.

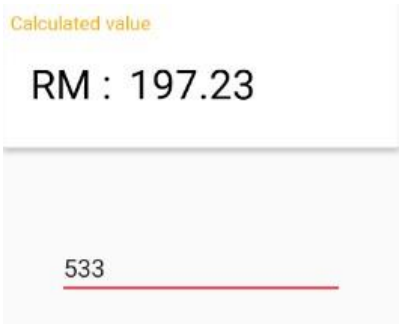


Figure 4.11: Accurate Calculated Value Example

4.4 Main Function Codes

4.4.1 Calculation Code

The code for the calculation of both bills is not in the calculator activity but instead, it is in its own class which follows Java's encapsulation to make code more concise. To calculate any electric or water amount, any other class has to create an object instance of "Spending" class and fill in the parameter of the amount to calculate, the type of the bill (water or electric) and the name of the provider. With all the information given, the method can then calculate using each provider's tariff rates. The snippet of the code for the calculation logic of two electric provider is given below from "Spending" class with the full source code of the class given in Appendix B.

```
//Calculation logic according to each provider
private Double setCalculatedCost(Double amountToCalculate, String type, String name) {
    Double result = 0.0;
    if (type.equals("Electric")) {
        Double tax = 0.0;
        if (name.equals("Tenaga Nasional")) {
            if (amountToCalculate < 1) {
                result = 300.0 / 100.0;
            } else if (amountToCalculate > 1 && amountToCalculate < 201) {
                result += (amountToCalculate * 21.80) / 100;
                if (result <= 3.00) {
                    result = 3.00;
                }
            } else if (amountToCalculate > 200 && amountToCalculate < 301) {
                amountToCalculate -= 200;
```

```
            result += ((amountToCalculate * 33.40) + (200 * 21.80)) / 100;
        } else if (amountToCalculate > 300 && amountToCalculate < 601) {
            amountToCalculate -= 300;
            result += ((amountToCalculate * 51.60) + (100 * 33.40) + (200 * 21.80)) / 100;
        } else if (amountToCalculate > 600 && amountToCalculate < 901) {
            amountToCalculate -= 600;
            tax += (amountToCalculate * 54.60 / 100) * 6 / 100;
            result += (((amountToCalculate * 54.60) + (100 * 33.40) + (200 * 21.80) + (300 * 51.60)) / 100) + tax;
        } else if (amountToCalculate > 900) {
            amountToCalculate -= 900;
            tax += ((amountToCalculate * 57.10 / 100) + (300 * 54.60 / 100)) * 6 / 100;
            result += (((amountToCalculate * 57.10) + (100 * 33.40) + (200 * 21.80) + (300 * 51.60) + (300 * 54.60)) / 100) +
tax;
        }
    } else if (name.equals("Sarawak Energy")) {
        if (amountToCalculate < 0) {
            result += 500.0 / 100.0;
        } else if (amountToCalculate > 0 && amountToCalculate < 151) {
            result += (amountToCalculate * 18) / 100;
            if (result <= 5.00) {
                result = 5.00;
            }
        } else if (amountToCalculate > 150 && amountToCalculate < 201) {
            result += (amountToCalculate * 22) / 100;
        } else if (amountToCalculate > 200 && amountToCalculate < 301) {
            result += (amountToCalculate * 25) / 100;
        } else if (amountToCalculate > 300 && amountToCalculate < 401) {
            result += (amountToCalculate * 27) / 100;
        } else if (amountToCalculate > 400 && amountToCalculate < 501) {
            result += (amountToCalculate * 29.5) / 100;
        } else if (amountToCalculate > 500 && amountToCalculate < 701) {
            if (amountToCalculate > 600) {
                double amountToCalculateHelper = amountToCalculate - 600;
                tax += (amountToCalculateHelper * 30) / 100 * 6 / 100;
            }
            result += ((amountToCalculate * 30) / 100) + tax;
        } else if (amountToCalculate > 700 && amountToCalculate < 801) {
            double amountToCalculateHelper = amountToCalculate - 600;
            tax += (amountToCalculateHelper * 30.5) / 100 * 6 / 100;
            result += ((amountToCalculate * 30.5) / 100) + tax;
        } else if (amountToCalculate > 800 && amountToCalculate < 1301) {
            double amountToCalculateHelper = amountToCalculate - 600;
            tax += (amountToCalculateHelper * 31) / 100 * 6 / 100;
            result += ((amountToCalculate * 31) / 100) + tax;
        } else if (amountToCalculate > 1300) {
            double amountToCalculateHelper = amountToCalculate - 600;
            tax += (amountToCalculateHelper * 31) / 100 * 6 / 100;
            result += ((amountToCalculate * 31.5) / 100) + tax;
        }
    }
```

4.4.2 Bill Calculator

As mentioned in the previous section, to calculate any type of bills from any providers, it requires three information, the amount to calculate, the type of the bills and name of the providers. In this project, the information is taken from which section users choose. For example, when user chooses the electric category, they are then presented with all electricity providers in Malaysia, the users can then choose any provider which brings them to the calculator. The application stores the information of which category and which providers user tapped on by using Android intent class. The intent class in the code snippet below does two things, first, it is used to launch another activity in this case, the calculator class and second, it stores two strings which is the type of category and provider that the user chose. After launching the stated activity in the intent class, the stored information can be then retrieved and used in the activity. The source code of the "Electric" choices class is in Appendix C. The "Electric" and "Water" classes use the same technique to get the type of category and name of the provider from users. Only the "Electric" class and one provider is used in the example below.

```
tnb.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent openCalculator = new Intent(getApplicationContext(), CalculatorBill.class);
        String choice = "Tenaga Nasional";
        openCalculator.putExtra("NAME", choice);
        openCalculator.putExtra("TYPE", TYPE);
        startActivity(openCalculator);
    }
});
```

The "setOnClickListener" method is used to "listen" to which providers user tapped on. For the "TYPE" string, a constant string is initialized to avoid error as shown below.

```
public static final String TYPE = "Electric";
```

The information stored in the intent class can be retrieved using the following code snippet.

```
//Intent is declared and used to receive information about which views user clicked on
final Intent intention = getIntent();
final String type = intention.getStringExtra("TYPE");
final String choice = intention.getStringExtra("NAME");
```

4.4.3 Database

The database built-in into Android is SQLite. The database requires a separate class to create and query the database itself. Code snippet below shows the creation of the database with the full source code of the database class given in Appendix F.

```
//Create the database
@Override
public void onCreate(SQLiteDatabase db) {
   String createTableStatement = "CREATE TABLE " + SPENDING_TABLE + " (" + COLUMN_ID + " INTEGER
PRIMARY KEY AUTOINCREMENT, " + COLUMN_SPENDING_TYPE + " TEXT, " + COLUMN_SPENDING_NAME + "
TEXT, " + COLUMN_DATE + " TEXT, " + COLUMN_TO_CALCULATE + " INT, " + COLUMN_COST + " INT) ";
   db.execSQL(createTableStatement);
}
```

The table name and column names are all stored in constant variables to avoid error. Code snippet below shows the constant variables initialization.

```
//Constant value declaration
public static final String SPENDING_TABLE = "SPENDING_TABLE";
public static final String COLUMN_SPENDING_TYPE = "SPENDING_TYPE";
public static final String COLUMN_SPENDING_NAME = "SPENDING_NAME";
public static final String COLUMN_DATE = "DATE";
public static final String COLUMN_TO_CALCULATE = "TO_CALCULATE";
public static final String COLUMN_COST = "COST";
public static final String COLUMN_ID = "ID";
```

The code snippet of adding one record to the database is shown below. All the information is received from an object of the “Spending” class from the calculator.

```
public boolean addOne(Spending spending) {

   SQLiteDatabase db = getWritableDatabase();
   ContentValues cv = new ContentValues();
   cv.put(COLUMN_SPENDING_TYPE, spending.getType());
   cv.put(COLUMN_SPENDING_NAME, spending.getName());
   cv.put(COLUMN_DATE, spending.getDate());
   cv.put(COLUMN_TO_CALCULATE, spending.getToCalculate());
   cv.put(COLUMN_COST, spending.getCost());

   long insert = db.insert(SPENDING_TABLE, null, cv);
   if (insert == -1) {
      return false;
   } else {
      return true;
```

```
    }
}
```

The code snippet below shows an example of querying the database to get all records from the database to be used in any other classes in the application. In the example, the code is used to populate the history of all spending saved. The records are stored in an "ArrayList" of the "Spending" class to be returned if required.

```
public ArrayList<Spending> getAllSpending() {
    ArrayList<Spending> spendingList = new ArrayList<>();

    String queryString = "SELECT * FROM " + SPENDING_TABLE;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor = db.rawQuery(queryString, null);

    if (cursor.moveToFirst()) {
        do {
            Spending = new Spending(cursor.getString(1), cursor.getString(2), cursor.getDouble(5), cursor.getString(3));
            spendingList.add(spending);
        } while (cursor.moveToNext());
    }
    db.close();
    cursor.close();
    return spendingList;
}
```

Below is a code snippet of the history class using the database method shown above. The full source code of the history class is given in Appendix E. In the code, if user chose to view all spending, the history will query the database method to get all spending and reverse the list in order to show from the latest to oldest spending.

```
if (choice.equals("All")) {
    ArrayList<Spending> spendingList = databaseHelper.getAllSpending();
    Collections.reverse(spendingList);
    lastMonthSpending.setText("This Month Spending");
    String averageBillCost = String.format("%.2f", databaseHelper.getMonthlyCost());
    averageSpending.setText(averageBillCost);
    mAdapter = new SpendingAdapter(spendingList);
}
```

CHAPTER 5

SUMMARY AND CONCLUSION

5.1 Summary of Main Findings

The application fulfilled the scope of being a financial planner built for Malaysians that can be used to calculate and track their monthly utility bills. The application automates the process of calculating by programming the tariff rates right into the calculator which can be saved into the application's database. Both water and electric bills are covered with the ability to view average bill prices, last bill price and comparing between all previously saved bills. The user interface is also easy to navigate with straightforward choices between each categories and providers of electric and water in Malaysia.

5.2 Discussion and Implications

From chapter 2, literature review, Java and Kotlin is discussed as the programming language of the project and Java is chosen because of the vast documentation and tutorials on various libraries and techniques of programming available online. By using Java, the main thing to look out for when programming the project is Null Safety and, in this project, there are many cases of having to use try…catch block in order to mitigate the NullPointerException problem. The project also uses the rapid application development model for its software development methodology successfully by prototyping each feature from the project proposal which ensures time efficiency is being kept while also producing high-quality product.

In chapter 3, the project is developed as planned except for time. All the design of the data and process modelling diagrams, database normalization and unified modelling language are developed as planned but the development and testing in chapter 4 took longer. In the planned design and analysis, the user interface is supposed to take 1 week but took longer than

expected with the added time of researching material design for android. The calculator also took longer to develop because in order to keep the number of calculator activity to one, a solution must be developed. The solution was to capture which provider user pressed on and keeps it in memory which can then be used in programming the calculator logic. The other spending calculator took shorter than the planned time because of the experience from building the previous bill calculator and the rest of the program is developed as planned afterwards.

5.3 Limitations of The System

5.3.1 Local Database

The application uses Android built-in SQLite database which is an offline local database that stores all the records in the internal storage of the phone. This means there are a few drawbacks to the application which is discussed below.

1. Unable to Sync

   Because the data is stored locally, there are no method to share the data across other systems, for example, if an application with the same objective for the iOS operating system is made, the data cannot be shared, and user cannot view their spending made on other devices.

2. Loss of Data

   The offline database also comes with another impact on storage of the data. For example, if users want to change phone or reset their phone, all the data from previous spending saved will be lost as a result.

5.3.2 Tariff Rates

Another limitation of the application is using manual tariff rates in both the “Spending” and “Tariff Rates” classes. There are many reasons as to why this is a limitation of the system. An example from “Tariff Rates” class is shown below which

shows the manual tariff rates in the code. The full source code of the “Tariff Rates” class is given in Appendix F.

```
//Setting all tariff values according to each provider
if (choice.equals("Tenaga Nasional")) {
    minimumChargeCost.setText("3.00");

    cat4Card.setVisibility(View.VISIBLE);
    cat5Card.setVisibility(View.VISIBLE);

    cat1Rm.setText("Cent : ");
    cat2Rm.setText("Cent : ");
    cat3Rm.setText("Cent : ");
    cat4Rm.setText("Cent : ");
    cat5Rm.setText("Cent : ");

    cat1Text.setText("1 - 200 kWh");
    cat1Cost.setText("21.80");
    cat2Text.setText("201 - 300 kWh");
    cat2Cost.setText("33.40");
    cat3Text.setText("301 - 600 kWh");
    cat3Cost.setText("51.60");
    cat4Text.setText("601 - 900 kWh");
    cat4Cost.setText("54.60");
    cat5Text.setText("901 kWh onwards");
    cat5Cost.setText("57.10");
}
```

By using manual tariff rates shown above, the code must be updated every time a provider updates its tariff rates which means the developer has to push a new build to the application that requires user to update the application manually. If the user does not update the application, if a provider updates its tariff rate, the calculation will not be accurate.

5.4 Future Development

5.4.1 Online Database

The application has many new features to be updated and of the most important one is using an online database. There are many online databases, for example MongoDB, MySQL, and Firebase among many others. In the case of the project, the most suitable database to be used is Firebase. Firebase is operated by Google to be used with mobile and web applications. It is free for a certain period of usage and is easily scaled with the growth of the project.

5.4.2 Synced Tariff Rates

By using an online database, the tariff rates can now be synced online using Firebase. Providers updating tariff rates no longer requires an application update but only numbers update from the database.

5.4.3 Other Calculation

Another feature that the application need is a calculator for other types of bills. At the moment, the application allows you to manually insert the type, name, and cost of your spending but an automatic process like the electric and water bill calculator is faster and more efficient. Other bill type includes mobile bill, internet bill amongst others.

REFERENCES

Alshamrani, A., & Bahattab, A. (2015). A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model. *International Journal of Computer Science Issues, 12*(1). Retrieved from https://www.ijcsi.org/papers/IJCSI-12-1-1-106-111.pdf

Bhatt, A. (2019, July 11). Introduction To Android. Retrieved from https://www.engineersgarage.com/article_page/introduction-to-android/

Bose, S., Kundu, A., Mukherjee, M., & Banerjee, M. (2018). A Comparative Study: Java Vs Kotlin Programming In Android Application Development. *International Journal of Advanced Research in Computer Science, 9*(3), 41-45. doi:10.26483/ijarcs.v9i3.5978

Callaham, J. (2020, February 19). The history of Android OS: Its name, origin and more. Retrieved from https://www.androidauthority.com/history-android-os-name-789433/

Daga, M. (2018, May 01). Java vs Kotlin: Which Programming Language Is Better for Android Developers? - DZone Mobile. Retrieved from https://dzone.com/articles/java-vs-kotlin-which-programming-language-is-better

Hershey, D. A., Jacobs-Lawson, J. M., Mcardle, J. J., & Hamagami, F. (2007). Psychological Foundations of Financial Planning for Retirement. *Journal of Adult Development, 14*(1-2), 26-36. doi:10.1007/s10804-007-9028-1

Jenkov, J. (n.d.). Java Constructors. Retrieved from http://tutorials.jenkov.com/java/constructors.html

Mgbemena, C. (2017, November 01). Kotlin From Scratch: Exception Handling. Retrieved from https://code.tutsplus.com/tutorials/kotlin-from-scratch-exception-handling--cms-29820

Moran, W. J. (2002). *U.S. Patent No. US 6,430,542 B1*. Washington, DC: U.S. Patent and Trademark Office.

MR, V. (2019, November 27). Kotlin vs Java:Which is the best Android Development language? Retrieved from https://www.edureka.co/blog/kotlin-vs-java/

Munassar, N. A., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. *International Journal of Computer Science Issues, 7*(5). Retrieved from https://www.researchgate.net/publication/258959806_A_Comparison_Between_Five_Models_Of_Software_Engineering

Raphael, J. (2020, June 26). Android versions: A living history from 1.0 to 11. Retrieved from https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html

Schildt, H. (2007). *Java: The complete reference*. New York: McGraw-Hill.

SDLC - Overview. (n.d.). Retrieved from https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

Singh, A. (2019, December 6). What Is Rapid Application Development (RAD)? Retrieved from https://blog.capterra.com/what-is-rapid-application-development/

APPENDICES

Appendix A

Calculator Class Source Code

```
package com.example.android.utilitybillcalculator.activities;

import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.cardview.widget.CardView;

import com.example.android.utilitybillcalculator.R;
import com.example.android.utilitybillcalculator.classes.Spending;
import com.example.android.utilitybillcalculator.database.DatabaseHelper;

public class CalculatorBill extends AppCompatActivity {

    //View component declaration
    EditText userInputToCalculate;
    Button calculateButton;
    Button saveButton;
    Button tariffRates;
    TextView resultText;
    TextView lastMonthText;
    TextView lastMonthCost;
    CardView historyCard;


    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.bill_calculator);

        //Attach all views to corresponding resources
        lastMonthText = findViewById(R.id.bill_last_month);
        lastMonthCost = findViewById(R.id.cost_last_month);
        calculateButton = findViewById(R.id.calculate);
        saveButton = findViewById(R.id.save);
        resultText = findViewById(R.id.result);
        userInputToCalculate = findViewById(R.id.toCalculate);
        historyCard = findViewById(R.id.last_month_card);
        tariffRates = findViewById(R.id.tariff_rate_button);

        //Database initialization
        final DatabaseHelper databaseHelper = new DatabaseHelper(CalculatorBill.this);
```

```
        //Intent is declared and used to receive information about which views user clicked on
        final Intent intention = getIntent();
        final String type = intention.getStringExtra("TYPE");
        final String choice = intention.getStringExtra("NAME");

        //Dynamically set the top card value depending on which category user tapped on
        if (type.equals("Electric")) {
            Double lastElectricCost = databaseHelper.getLastElectricCost();
            String formattedElectricCost = String.format("%.2f", lastElectricCost);
            lastMonthCost.setText(formattedElectricCost);
            lastMonthText.setText("Last saved electric bill");
            userInputToCalculate.setHint("KW/H");
        } else {
            Double lastWaterCost = databaseHelper.getLastWaterCost();
            String formattedWaterCost = String.format("%.2f", lastWaterCost);
            lastMonthCost.setText(formattedWaterCost);
            lastMonthText.setText("Last saved water bill");
            userInputToCalculate.setHint("M³");
        }

        calculateButton.setOnClickListener(new View.OnClickListener() {
            //This onClick method is used to receive and shows calculated user input values.
            @Override
            public void onClick(View v) {
                try {
                    String toCalculate = userInputToCalculate.getText().toString();
                    Double toCalculateAmount = Double.parseDouble(toCalculate);
                    Spending spend = new Spending(toCalculateAmount, type, choice);
                    String resultDouble = String.format("%.2f", spend.getCost());
                    resultText.setText(resultDouble);
                } catch (NumberFormatException e) {
                    Toast.makeText(CalculatorBill.this, "Invalid Amount!", Toast.LENGTH_SHORT).show();
                }
            }
        });

            /*
            setOnClickListener method is used to receive, calculated, show and save user input values.
            A new class is created based on user input values and calculated values which is then
            stored to the database.
            Toast is used to troubleshoot and show if storing is successful
             */
        saveButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                //Used to check for whether or not user has saved their bill this month.
                boolean checkIfSaved = false;
                if (type.equals("Electric")) {
                    checkIfSaved = databaseHelper.checkElectricBillSaved();
                } else {
                    checkIfSaved = databaseHelper.checkWaterBillSaved();
                }

                //If bill is saved, an alert will popup to warn user in order to ensure only one bill is saved per
month.
                if (checkIfSaved) {
                    AlertDialog.Builder checkDialog = new AlertDialog.Builder(CalculatorBill.this);
                    checkDialog.setMessage("Current month bill is saved, do you want to save another
one?")
```

```
                    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            try {
                                String toCalculate = userInputToCalculate.getText().toString();
                                Double toCalculateAmount = Double.parseDouble(toCalculate);
                                Spending spend = new Spending(toCalculateAmount, type, choice);
                                String resultDouble = String.format("%.2f", spend.getCost());
                                boolean success = databaseHelper.addOne(spend);
                                if (success) {
                                    Toast.makeText(CalculatorBill.this, "Saved",
Toast.LENGTH_SHORT).show();
                                }
                                resultText.setText(resultDouble);
                            } catch (NumberFormatException e) {
                                Toast.makeText(CalculatorBill.this, "Invalid Amount!",
Toast.LENGTH_SHORT).show();
                            }
                        }
                    })
                    .setNegativeButton("No", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            dialog.cancel();
                        }
                    });

                //Initialization of dialog box
                AlertDialog alert = checkDialog.create();
                alert.setTitle("Bill  Saved");
                alert.show();
            } else {
                try {
                    String toCalculate = userInputToCalculate.getText().toString();
                    Double toCalculateAmount = Double.parseDouble(toCalculate);
                    Spending spend = new Spending(toCalculateAmount, type, choice);
                    String resultDouble = String.format("%.2f", spend.getCost());
                    boolean success = databaseHelper.addOne(spend);
                    if (success) {
                        Toast.makeText(CalculatorBill.this, "Saved", Toast.LENGTH_SHORT).show();
                    }
                    resultText.setText(resultDouble);
                } catch (NumberFormatException e) {
                    Toast.makeText(CalculatorBill.this, "Invalid Amount!",
Toast.LENGTH_SHORT).show();
                }
            }
        }
    });

        historyCard.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick (View v){
        Intent openHistory = new Intent(getApplicationContext(), SpendingHistory.class);
        openHistory.putExtra("TYPE", type);
        startActivity(openHistory);
    }
    });

        tariffRates.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick (View v){
        Intent openTariffRates = new Intent(getApplicationContext(), TariffRates.class);
        openTariffRates.putExtra("NAME", choice);
        startActivity(openTariffRates);
    }
    });
}
}
```

Appendix B

Spending Class Source Code

```
package com.example.android.utilitybillcalculator.classes;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class Spending {

    private Integer ID;
    private String name;
    private String type;
    private String date;
    private Double toCalculate;
    private Double cost;

    //Four constructors in case of requiring null value
    public Spending(Double toCalculate, String type, String name) {
        this.name = name;
        this.type = type;
        this.date = setDate();
        this.toCalculate = toCalculate;
        this.cost = setCalculatedCost(toCalculate, type, name);
    }

    public Spending(String type, String name, Double cost, String date) {
        this.type = type;
        this.name = name;
        this.cost = cost;
        this.date = date;
    }

    public Spending(String type, String name, Double cost) {
        this.name = name;
        this.type = type;
        this.date = setDate();
        this.cost = cost;
        this.toCalculate = null;
    }

    public Spending(String type, String name, String date, Double cost) {
        this.type = type;
        this.name = name;
        this.date = date;
        this.cost = cost;
        this.toCalculate = null;
    }


    //Getter methods
    public String getName() {
        return name;
    }

    public String getType() {
```

```
        return type;
    }

    public String getDate() {
        return date;
    }

    public Double getToCalculate() {
        return this.toCalculate;
    }

    public Double getCost() {
        return cost;
    }

    //Calculation logic according to each provider
    private Double setCalculatedCost(Double amountToCalculate, String type, String name) {
        Double result = 0.0;
        if (type.equals("Electric")) {
            Double tax = 0.0;
            if (name.equals("Tenaga Nasional")) {
                if (amountToCalculate < 1) {
                    result = 300.0 / 100.0;
                } else if (amountToCalculate > 1 && amountToCalculate < 201) {
                    result += (amountToCalculate * 21.80) / 100;
                    if (result <= 3.00) {
                        result = 3.00;
                    }
                } else if (amountToCalculate > 200 && amountToCalculate < 301) {
                    amountToCalculate -= 200;
                    result += ((amountToCalculate * 33.40) + (200 * 21.80)) / 100;
                } else if (amountToCalculate > 300 && amountToCalculate < 601) {
                    amountToCalculate -= 300;
                    result += ((amountToCalculate * 51.60) + (100 * 33.40) + (200 * 21.80)) / 100;
                } else if (amountToCalculate > 600 && amountToCalculate < 901) {
                    amountToCalculate -= 600;
                    tax += (amountToCalculate * 54.60 / 100) * 6 / 100;
                    result += (((amountToCalculate * 54.60) + (100 * 33.40) + (200 * 21.80) + (300 * 51.60)) /
100) + tax;
                } else if (amountToCalculate > 900) {
                    amountToCalculate -= 900;
                    tax += ((amountToCalculate * 57.10 / 100) + (300 * 54.60 / 100)) * 6 / 100;
                    result += (((amountToCalculate * 57.10) + (100 * 33.40) + (200 * 21.80) + (300 * 51.60) +
(300 * 54.60)) / 100) + tax;
                }
            } else if (name.equals("Sarawak Energy")) {
                if (amountToCalculate < 0) {
                    result += 500.0 / 100.0;
                } else if (amountToCalculate > 0 && amountToCalculate < 151) {
                    result += (amountToCalculate * 18) / 100;
                    if (result <= 5.00) {
                        result = 5.00;
                    }
                } else if (amountToCalculate > 150 && amountToCalculate < 201) {
                    result += (amountToCalculate * 22) / 100;
                } else if (amountToCalculate > 200 && amountToCalculate < 301) {
                    result += (amountToCalculate * 25) / 100;
                } else if (amountToCalculate > 300 && amountToCalculate < 401) {
                    result += (amountToCalculate * 27) / 100;
                } else if (amountToCalculate > 400 && amountToCalculate < 501) {
```

```
                result += (amountToCalculate * 29.5) / 100;
            } else if (amountToCalculate > 500 && amountToCalculate < 701) {
                if (amountToCalculate > 600) {
                    double amountToCalculateHelper = amountToCalculate – 600;
                    tax += (amountToCalculateHelper * 30) / 100 * 6 / 100;
                }
                result += ((amountToCalculate * 30) / 100) + tax;
            } else if (amountToCalculate > 700 && amountToCalculate < 801) {
                double amountToCalculateHelper = amountToCalculate – 600;
                tax += (amountToCalculateHelper * 30.5) / 100 * 6 / 100;
                result += ((amountToCalculate * 30.5) / 100) + tax;
            } else if (amountToCalculate > 800 && amountToCalculate < 1301) {
                double amountToCalculateHelper = amountToCalculate – 600;
                tax += (amountToCalculateHelper * 31) / 100 * 6 / 100;
                result += ((amountToCalculate * 31) / 100) + tax;
            } else if (amountToCalculate > 1300) {
                double amountToCalculateHelper = amountToCalculate – 600;
                tax += (amountToCalculateHelper * 31) / 100 * 6 / 100;
                result += ((amountToCalculate * 31.5) / 100) + tax;
            }
        } else if (name.equals("Sabah Electric")) {
            if (amountToCalculate < 1) {
                result = 500.0 / 100.0;
            } else if (amountToCalculate > 1 && amountToCalculate < 101) {
                result += (amountToCalculate * 17.5) / 100;
                if (result <= 5.00) {
                    result = 5.00;
                }
            } else if (amountToCalculate > 100 && amountToCalculate < 201) {
                amountToCalculate -= 100;
                result += ((amountToCalculate * 18.5) + (100 * 17.5)) / 100;
            } else if (amountToCalculate > 200 && amountToCalculate < 301) {
                amountToCalculate -= 200;
                result += ((amountToCalculate * 33.0) + (100 * 17.5) + (100 * 18.5)) / 100;
            } else if (amountToCalculate > 300 && amountToCalculate < 501) {
                amountToCalculate -= 300;
                result += ((amountToCalculate * 44.5) + (100 * 17.5) + (100 * 18.5) + (100 * 33.0)) / 100;
            } else if (amountToCalculate > 500 && amountToCalculate < 1001) {
                if (amountToCalculate > 600) {
                    Double taxHelper = amountToCalculate – 600;
                    tax += (taxHelper * 45.0) / 100 * 6 / 100;
                }
                amountToCalculate -= 500;
                result += (((amountToCalculate * 45.0) + (100 * 17.5) + (100 * 18.5) + (100 * 33.0) + (200
* 44.50)) / 100) + tax;
            } else if (amountToCalculate > 1000) {
                Double firstTax = (400 * 45.0) / 100 * 6 / 100;
                Double taxHelper2 = amountToCalculate – 1000;
                tax += ((taxHelper2 * 47.0) / 100 * 6 / 100) + firstTax;
                amountToCalculate -= 1000;
                result += (((amountToCalculate * 47.0) + (100 * 17.5) + (100 * 18.5) + (100 * 33.0) + (200
* 44.50) + (500 * 45.0)) / 100) + tax;
            }
        }
    } else if (type.equals("Water")) {
        Double balance = 0.0;
        Double surcharge = 0.0;
        if (name.equals("Johor")) {
            if (amountToCalculate > 0 && amountToCalculate <= 20) {
                result += (amountToCalculate * 0.80);
```

```
    } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
        amountToCalculate -= 20;
        result += ((amountToCalculate * 2.00) + (20 * 0.80));
    } else if (amountToCalculate > 35) {
        amountToCalculate -= 35;
        result += ((amountToCalculate * 3.00) + (20 * 0.80) + (15 * 2.00));
    }

} else if (name.equals("Kedah")) {
    if (amountToCalculate > 0 && amountToCalculate <= 20) {
        result += (amountToCalculate * 0.50);
    } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
        amountToCalculate -= 20;
        result += ((amountToCalculate * 0.90) + (20 * 0.50));
    } else if (amountToCalculate > 35) {
        amountToCalculate -= 35;
        result += ((amountToCalculate * 1.30) + (20 * 0.50) + (15 * 0.90));
    }

} else if (name.equals("Kelantan")) {
    if (amountToCalculate > 0 && amountToCalculate <= 20) {
        result += (amountToCalculate * 0.45);
    } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
        amountToCalculate -= 20;
        result += ((amountToCalculate * 0.97) + (20 * 0.45));
    } else if (amountToCalculate > 35) {
        amountToCalculate -= 35;
        result += ((amountToCalculate * 1.42) + (20 * 0.45) + (15 * 0.97));
    }

} else if (name.equals("Melaka")) {
    if (amountToCalculate > 0 && amountToCalculate <= 20) {
        result += (amountToCalculate * 0.60);
    } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
        amountToCalculate -= 20;
        result += ((amountToCalculate * 0.95) + (20 * 0.60));
    } else if (amountToCalculate > 35) {
        amountToCalculate -= 35;
        result += ((amountToCalculate * 1.45) + (20 * 0.60) + (15 * 0.95));
    }

} else if (name.equals("Negeri Sembilan")) {
    if (amountToCalculate > 0 && amountToCalculate <= 20) {
        result += (amountToCalculate * 0.55);
    } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
        amountToCalculate -= 20;
        result += ((amountToCalculate * 0.85) + (20 * 0.55));
    } else if (amountToCalculate > 35) {
        amountToCalculate -= 35;
        result += ((amountToCalculate * 1.40) + (20 * 0.55) + (15 * 0.85));
    }

} else if (name.equals("Pahang")) {
    if (amountToCalculate > 0 && amountToCalculate <= 18) {
        result += (amountToCalculate * 0.37);
    } else if (amountToCalculate > 18 && amountToCalculate <= 45) {
        amountToCalculate -= 18;
        result += ((amountToCalculate * 0.79) + (18 * 0.37));
    } else if (amountToCalculate > 45) {
        amountToCalculate -= 45;
```

```
            result += ((amountToCalculate * 0.99) + (18 * 0.37) + (27 * 0.79));
        }

    } else if (name.equals("Penang")) {

        if (amountToCalculate > 0 && amountToCalculate <= 20) {
            result += (amountToCalculate * 0.22);
        } else if (amountToCalculate > 20 && amountToCalculate <= 40) {
            if (amountToCalculate > 35) {
                balance = amountToCalculate – 35;
                surcharge = balance * 0.48;
            }
            amountToCalculate -= 20;
            result += ((amountToCalculate * 0.46) + (20 * 0.22) + surcharge);
        } else if (amountToCalculate > 40 && amountToCalculate <= 60) {
            balance = amountToCalculate;
            balance -= 35;
            surcharge = balance * 0.48;
            amountToCalculate -= 40;
            result += ((amountToCalculate * 0.68) + (20 * 0.22) + (20 * 0.46) + surcharge);
        } else if (amountToCalculate > 60 && amountToCalculate <= 200) {
            balance = amountToCalculate;
            balance -= 35;
            surcharge = balance * 0.48;
            amountToCalculate -= 60;
            result += ((amountToCalculate * 1.17) + (20 * 0.22) + (20 * 0.46) + (20 * 0.68) +
surcharge);
        } else if (amountToCalculate > 200) {
            balance = amountToCalculate;
            balance -= 35;
            surcharge = balance * 0.48;
            amountToCalculate -= 200;
            result += ((amountToCalculate * 1.30) + (20 * 0.22) + (20 * 0.46) + (20 * 0.68) + (140 *
1.17) + surcharge);
        }

    } else if (name.equals("Perak")) {
        if (amountToCalculate > 0 && amountToCalculate <= 10) {
            result += (amountToCalculate * 0.30);
        } else if (amountToCalculate > 10 && amountToCalculate <= 20) {
            amountToCalculate -= 10;
            result += ((amountToCalculate * 0.70) + (10 * 0.30));
        } else if (amountToCalculate > 20) {
            amountToCalculate -= 20;
            result += ((amountToCalculate * 1.03) + (10 * 0.30) + (10 * 0.70));
        }

    } else if (name.equals("Perlis")) {
        if (amountToCalculate > 0 && amountToCalculate <= 15) {
            result += (amountToCalculate * 0.40);
        } else if (amountToCalculate > 15 && amountToCalculate <= 40) {
            amountToCalculate -= 15;
            result += ((amountToCalculate * 0.70) + (15 * 0.40));
        } else if (amountToCalculate > 40) {
            amountToCalculate -= 40;
            result += ((amountToCalculate * 1.10) + (15 * 0.40) + (25 * 0.70));
        }

    } else if (name.equals("Selangor")) {
        if (amountToCalculate > 0 && amountToCalculate <= 20) {
```

```
            result += (amountToCalculate * 0.57);
        } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
            amountToCalculate -= 20;
            result += ((amountToCalculate * 1.03) + (20 * 0.57));
        } else if (amountToCalculate > 35) {
            amountToCalculate -= 35;
            result += ((amountToCalculate * 2.00) + (20 * 0.57) + (15 * 1.03));
        }

    } else if (name.equals("Terengganu")) {
        if (amountToCalculate > 0 && amountToCalculate <= 20) {
            result += (amountToCalculate * 0.42);
        } else if (amountToCalculate > 20 && amountToCalculate <= 40) {
            amountToCalculate -= 20;
            result += ((amountToCalculate * 0.65) + (20 * 0.42));
        } else if (amountToCalculate > 40 && amountToCalculate <= 60) {
            amountToCalculate -= 40;
            result += ((amountToCalculate * 0.90) + (20 * 0.42) + (20 * 0.65));
        } else if (amountToCalculate > 60) {
            amountToCalculate -= 60;
            result += ((amountToCalculate * 1.00) + (20 * 0.42) + (20 * 0.65) + (20 * 0.90));
        }

    } else if (name.equals("Labuan")) {
        if (amountToCalculate > 0 && amountToCalculate <= 20) {
            result += (amountToCalculate * 0.57);
        } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
            amountToCalculate -= 20;
            result += ((amountToCalculate * 1.20) + (20 * 0.57));
        } else if (amountToCalculate > 35) {
            amountToCalculate -= 35;
            result += ((amountToCalculate * 1.70) + (20 * 0.57) + (15 * 1.20));
        }

    } else if (name.equals("Kuching")) {
        if (amountToCalculate > 0 && amountToCalculate <= 15) {
            result += (amountToCalculate * 0.48);
        } else if (amountToCalculate > 15 && amountToCalculate <= 50) {
            amountToCalculate -= 15;
            result += ((amountToCalculate * 0.72) + (15 * 0.48));
        } else if (amountToCalculate > 50) {
            amountToCalculate -= 50;
            result += ((amountToCalculate * 0.76) + (15 * 0.48) + (35 * 0.72));
        }

    } else if (name.equals("Sibu")) {
        if (amountToCalculate > 0 && amountToCalculate <= 15) {
            result += (amountToCalculate * 0.48);
        } else if (amountToCalculate > 15 && amountToCalculate <= 50) {
            amountToCalculate -= 15;
            result += ((amountToCalculate * 0.72) + (15 * 0.48));
        } else if (amountToCalculate > 50) {
            amountToCalculate -= 50;
            result += ((amountToCalculate * 0.76) + (15 * 0.48) + (35 * 0.72));
        }

    } else if (name.equals("Miri")) {
        if (amountToCalculate > 0 && amountToCalculate <= 15) {
            result += (amountToCalculate * 0.48);
        } else if (amountToCalculate > 15 && amountToCalculate <= 50) {
```

```
                amountToCalculate -= 15;
                result += ((amountToCalculate * 0.72) + (15 * 0.48));
            } else if (amountToCalculate > 50) {
                amountToCalculate -= 50;
                result += ((amountToCalculate * 0.76) + (15 * 0.48) + (35 * 0.72));
            }

        } else if (name.equals("Bintulu")) {
            if (amountToCalculate > 0 && amountToCalculate <= 14) {
                result = 6.60;
            } else if (amountToCalculate > 14 && amountToCalculate <= 45) {
                amountToCalculate -= 14;
                result += ((amountToCalculate * 0.61) + 6.60);
            } else if (amountToCalculate > 45) {
                amountToCalculate -= 45;
                result += ((amountToCalculate * 0.66) + 6.60 + (31 * 0.61));
            }

        } else if (name.equals("Sarawak")) {
            if (amountToCalculate > 0 && amountToCalculate <= 15) {
                result += (amountToCalculate * 0.44);
            } else if (amountToCalculate > 15 && amountToCalculate <= 50) {
                amountToCalculate -= 15;
                result += ((amountToCalculate * 0.65) + (15 * 0.44));
            } else if (amountToCalculate > 50) {
                amountToCalculate -= 50;
                result += ((amountToCalculate * 0.69) + (15 * 0.44) + (35 * 0.65));
            }

        } else if (name.equals("Sabah")) {
            if (amountToCalculate > 0 && amountToCalculate <= 10) {
                result += (amountToCalculate * 0.30);
            } else if (amountToCalculate > 10 && amountToCalculate <= 20) {
                amountToCalculate -= 10;
                result += ((amountToCalculate * 0.60) + (10 * 0.30));
            } else if (amountToCalculate > 20 && amountToCalculate <= 35) {
                amountToCalculate -= 20;
                result += ((amountToCalculate * 1.10) + (10 * 0.30) + (10 * 0.60));
            } else if (amountToCalculate > 35 && amountToCalculate <= 60) {
                amountToCalculate -= 35;
                result += ((amountToCalculate * 1.30) + (10 * 0.30) + (10 * 0.60) + (15 * 1.10));
            } else if (amountToCalculate > 60) {
                amountToCalculate -= 60;
                result += ((amountToCalculate * 1.80) + (10 * 0.30) + (10 * 0.60) + (15 * 1.10) + (25 *
1.30));
            }
        }
    }
    return result;
  }

  //Get current date
  private String setDate() {
    Calendar c = Calendar.getInstance();
    SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String formattedDate = df.format(c.getTime());
    return formattedDate;
  }

}
```

Appendix C

Electric Choice Class Source Code

```
package com.example.android.utilitybillcalculator.activities;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import com.example.android.utilitybillcalculator.R;

public class ElectricChoice extends AppCompatActivity {

    //View component declaration
    TextView tnb;
    TextView sarawakEnergy;
    TextView sabahElectric;

    //A final string variable is created to avoid mistyping as it is used multiple times.
    public static final String TYPE = "Electric";

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.choice_electricity);

        //Attaching views to layout resource.
        tnb = (TextView) findViewById(R.id.tnb);
        sarawakEnergy = (TextView) findViewById(R.id.sarawak_energy);
        sabahElectric = (TextView) findViewById(R.id.sabah_electric);

        /*
        setOnClickListener is used on each TextViews to store information on which views user
        tapped to program the calculator and start the calculator activity.
         */

        tnb.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openCalculator = new Intent(getApplicationContext(), CalculatorBill.class);
                String choice = "Tenaga Nasional";
                openCalculator.putExtra("NAME", choice);
                openCalculator.putExtra("TYPE", TYPE);
                startActivity(openCalculator);
            }
        });

        sarawakEnergy.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openCalculator = new Intent(getApplicationContext(), CalculatorBill.class);
                String choice = "Sarawak Energy";
                openCalculator.putExtra("NAME", choice);
                openCalculator.putExtra("TYPE", TYPE);
                startActivity(openCalculator);
```

```
        }
    });

    sabahElectric.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent openCalculator = new Intent(getApplicationContext(), CalculatorBill.class);
            String choice = "Sabah Electric";
            openCalculator.putExtra("NAME", choice);
            openCalculator.putExtra("TYPE", TYPE);
            startActivity(openCalculator);
        }
    });
  }
}
```

Appendix D

History Choice Class Source Code

```
package com.example.android.utilitybillcalculator.activities;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import com.example.android.utilitybillcalculator.R;

public class HistoryChoice extends AppCompatActivity {

    //View component declaration
    TextView allSpending;
    TextView electricSpending;
    TextView waterSpending;
    TextView insightSpending;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.choice_history);

        //Attaching views to layout resources.
        allSpending = findViewById(R.id.all_spending);
        electricSpending = findViewById(R.id.electric_spending);
        waterSpending = findViewById(R.id.water_spending);
        insightSpending = findViewById(R.id.insight_spending);

        /*
        setOnClickListener is used on each TextViews to store information on which views user
        tapped to program the spending history and start the spending history activity
         */

        allSpending.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openHistory = new Intent(getApplicationContext(), SpendingHistory.class);
                String choice = "All";
                openHistory.putExtra("TYPE", choice);
                startActivity(openHistory);
            }
        });

        electricSpending.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openHistory = new Intent(getApplicationContext(), SpendingHistory.class);
                String choice = "Electric";
                openHistory.putExtra("TYPE", choice);
                startActivity(openHistory);

            }
        });
```

```
        waterSpending.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openHistory = new Intent(getApplicationContext(), SpendingHistory.class);
                String choice = "Water";
                openHistory.putExtra("TYPE", choice);
                startActivity(openHistory);
            }
        });

        //No information is stored, only opens spending insight
        insightSpending.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openInsightHistory = new Intent(getApplicationContext(), SpendingInsight.class);
                startActivity(openInsightHistory);
            }
        });
    }
}
```

Appendix E

Spending History Class Source Code

```
package com.example.android.utilitybillcalculator.activities;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.cardview.widget.CardView;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.example.android.utilitybillcalculator.R;
import com.example.android.utilitybillcalculator.classes.Spending;
import com.example.android.utilitybillcalculator.classes.SpendingAdapter;
import com.example.android.utilitybillcalculator.database.DatabaseHelper;

import java.util.ArrayList;
import java.util.Collections;

public class SpendingHistory extends AppCompatActivity {

    //Views declaration
    private CardView spendingCard;

    private RecyclerView mRecyclerView;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager mLayoutManager;

    private TextView lastMonthSpending;
    private TextView averageSpending;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.history_spending);

        //Receive information from previous activity about which TextView user tapped on
        final Intent intent = getIntent();
        final String choice = intent.getStringExtra("TYPE");
        //Database declaration
        final DatabaseHelper databaseHelper = new DatabaseHelper(SpendingHistory.this);
        //Attaching views to layout resources
        lastMonthSpending = findViewById(R.id.last_month_spending);
        averageSpending = findViewById(R.id.last_month_cost);
        spendingCard = findViewById(R.id.spending_card);

        //Program the type of spending to show depending on which TextView user tapped on
        if (choice.equals("All")) {
            ArrayList<Spending> spendingList = databaseHelper.getAllSpending();
            Collections.reverse(spendingList);
            lastMonthSpending.setText("This Month Spending");
            String averageBillCost = String.format("%.2f", databaseHelper.getMonthlyCost());
```

```
            averageSpending.setText(averageBillCost);
            mAdapter = new SpendingAdapter(spendingList);
        } else if (choice.equals("Electric")) {
            ArrayList<Spending> spendingList = databaseHelper.getElectricSpending();
            Collections.reverse(spendingList);
            lastMonthSpending.setText("Average Electric Bill");
            String averageBillCost = String.format("%.2f", databaseHelper.getAverageElectricCost());
            if (averageBillCost.equals("0.00")) {
                averageSpending.setText("0.00");
            } else {
                averageSpending.setText(averageBillCost);
            }
            mAdapter = new SpendingAdapter(spendingList);
        } else if (choice.equals("Water")) {
            ArrayList<Spending> spendingList = databaseHelper.getWaterSpending();
            Collections.reverse(spendingList);
            lastMonthSpending.setText("Average Water Bill");
            String averageBillCost = String.format("%.2f", databaseHelper.getAverageWaterCost());
            averageSpending.setText(averageBillCost);
            mAdapter = new SpendingAdapter(spendingList);
        }

        //Tapping on top card shows spending insight
        spendingCard.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent openInsight = new Intent(getApplicationContext(), SpendingInsight.class);
                startActivity(openInsight);
            }
        });

        //RecyclerView for dynamically adding spending to a list
        mRecyclerView = findViewById(R.id.all_recyclerView);
        mLayoutManager = new LinearLayoutManager(this);
        mRecyclerView.setLayoutManager(mLayoutManager);
        mRecyclerView.setAdapter(mAdapter);

    }

}
```

Appendix F

Database Class

```
package com.example.android.utilitybillcalculator.database;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import com.example.android.utilitybillcalculator.classes.Spending;

import java.util.ArrayList;

public class DatabaseHelper extends SQLiteOpenHelper {

    //Constant value declaration
    public static final String SPENDING_TABLE = "SPENDING_TABLE";
    public static final String COLUMN_SPENDING_TYPE = "SPENDING_TYPE";
    public static final String COLUMN_SPENDING_NAME = "SPENDING_NAME";
    public static final String COLUMN_DATE = "DATE";
    public static final String COLUMN_TO_CALCULATE = "TO_CALCULATE";
    public static final String COLUMN_COST = "COST";
    public static final String COLUMN_ID = "ID";
    private static final int DATABASE_VERSION = 2;

    private static final String TAG = "DatabaseHelper";

    public DatabaseHelper(Context context) {
        super(context, "spending.db", null, DATABASE_VERSION);
    }

    //Create the database
    @Override
    public void onCreate(SQLiteDatabase db) {
        String createTableStatement = "CREATE TABLE " + SPENDING_TABLE + " (" + COLUMN_ID
+ " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMN_SPENDING_TYPE + " TEXT, " +
COLUMN_SPENDING_NAME + " TEXT, " + COLUMN_DATE + " TEXT, " +
COLUMN_TO_CALCULATE + " INT, " + COLUMN_COST + " INT) ";
        db.execSQL(createTableStatement);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + SPENDING_TABLE);
        onCreate(db);
    }

    public boolean addOne(Spending spending) {

        SQLiteDatabase db = getWritableDatabase();
        ContentValues cv = new ContentValues();

        cv.put(COLUMN_SPENDING_TYPE, spending.getType());
        cv.put(COLUMN_SPENDING_NAME, spending.getName());
        cv.put(COLUMN_DATE, spending.getDate());
```

```
        cv.put(COLUMN_TO_CALCULATE, spending.getToCalculate());
        cv.put(COLUMN_COST, spending.getCost());

        long insert = db.insert(SPENDING_TABLE, null, cv);

        if (insert == -1) {
            return false;
        } else {
            return true;
        }
    }

    public ArrayList<Spending> getAllSpending() {
        ArrayList<Spending> spendingList = new ArrayList<>();

        String queryString = "SELECT * FROM " + SPENDING_TABLE;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                Spending spending = new Spending(cursor.getString(1), cursor.getString(2),
cursor.getDouble(5), cursor.getString(3));
                spendingList.add(spending);
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        return spendingList;

    }

    public ArrayList<Spending> getElectricSpending() {
        ArrayList<Spending> spendingList = new ArrayList<>();

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " LIKE '%ELECTRIC%'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                Spending spending = new Spending(cursor.getString(1), cursor.getString(2),
cursor.getDouble(5), cursor.getString(3));
                spendingList.add(spending);
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        return spendingList;
    }

    public ArrayList<Spending> getWaterSpending() {
        ArrayList<Spending> spendingList = new ArrayList<>();

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
```

```
COLUMN_SPENDING_TYPE + " LIKE '%WATER%'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                Spending spending = new Spending(cursor.getString(1), cursor.getString(2),
cursor.getDouble(5), cursor.getString(3));
                spendingList.add(spending);
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        return spendingList;
    }

    public Double getMonthlyCost() {
        Double sum = 0.0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_DATE + " BETWEEN " + COLUMN_DATE + "('now','start of month') AND '" +
COLUMN_DATE + "(now)'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {

            do {

                sum += cursor.getDouble(5);

            } while (cursor.moveToNext());

        }

        db.close();
        cursor.close();

        return sum;

    }

    public Double getLastElectricCost() {
        Double electricCost = 0.0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " = " + "'Electric'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToLast()) {
            electricCost = cursor.getDouble(5);
        }

        db.close();
        cursor.close();

        return electricCost;
```

```
    }

    public Double getLastWaterCost() {
        Double waterCost = 0.0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " = " + "'Water'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToLast()) {
            waterCost = cursor.getDouble(5);
        }

        db.close();
        cursor.close();

        return waterCost;

    }

    public Spending getLastRecord() {
        Spending spendObject = null;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " + COLUMN_ID +
" = (SELECT MAX(" + COLUMN_ID + ") FROM " + SPENDING_TABLE + ");";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToLast()) {
            spendObject = new Spending(cursor.getString(1), cursor.getString(2), cursor.getDouble(5));
        }

        db.close();
        cursor.close();

        return spendObject;
    }

    public Double getAverageElectricCost() {
        double sum = 0.0;
        double average = 0.0;
        int count = 0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " LIKE '%ELECTRIC%'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                sum += cursor.getDouble(5);
                count++;
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();
```

```
        if (sum == 0) {
            return 0.00;
        }

        average = (double) sum / count;
        return average;
    }

    public Double getAverageWaterCost() {
        double sum = 0.0;
        double average = 0.0;
        int count = 0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " LIKE '%WATER%'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                sum += cursor.getDouble(5);
                count++;
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        if (sum == 0.0) {
            return 0.00;
        }

        average = (double) sum / count;
        return average;
    }

    public Double getAverageSpending() {
        double sum = 0.0;
        double average = 0.0;
        int count = 0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                sum += cursor.getDouble(5);
                count++;
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        if (sum == 0.0) {
            return 0.00;
        }

        average = (double) sum / count;
```

```
        return average;
    }

    public Double getAverageSpendingWithoutBills() {
        double sum = 0.0;
        double average = 0.0;
        int count = 0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " NOT LIKE '%Electric%' OR " + COLUMN_SPENDING_TYPE + "
NOT LIKE '%Water%'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                sum += cursor.getDouble(5);
                count++;
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        if (sum == 0.0) {
            return 0.00;
        }

        average = (double) sum / count;
        return average;
    }

    public Double getLastMonthSpendingWithoutBills() {
        double sum = 0.0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_DATE + " BETWEEN " + COLUMN_DATE + "('now', 'start of month', '-1 month') AND "
+ COLUMN_DATE + "('now', 'start of month', '-1 day')";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            do {
                String billCheck = cursor.getString(1);
                if (billCheck.equals("Water") || billCheck.equals("Electric")) {
                    continue;
                }
                sum += cursor.getDouble(5);
            } while (cursor.moveToNext());
        }

        db.close();
        cursor.close();

        return sum;
    }

    public boolean checkElectricBillSaved() {
        boolean checked = false;
```

```
        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " LIKE '%Electric%' AND " + COLUMN_DATE + " BETWEEN " +
COLUMN_DATE + "('now','start of month') AND '" + COLUMN_DATE + "(now)'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            checked = true;
        }

        Log.d(TAG, "True: " + checked);

        db.close();
        cursor.close();

        return checked;
    }

    public boolean checkWaterBillSaved() {
        boolean checked = false;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " LIKE '%Water%' AND " + COLUMN_DATE + " BETWEEN " +
COLUMN_DATE + "('now','start of month') AND '" + COLUMN_DATE + "(now)'";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToFirst()) {
            checked = true;
        }

        db.close();
        cursor.close();

        return checked;
    }

    public Double getLastMonthElectricBill() {
        Double bill = 0.0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
COLUMN_SPENDING_TYPE + " LIKE '%Electric%' AND " + COLUMN_DATE + " BETWEEN " +
COLUMN_DATE + "('now', 'start of month', '-1 month') AND " + COLUMN_DATE + "('now', 'start
of month', '-1 day')";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToLast()) {
            bill = cursor.getDouble(5);
        }

        db.close();
        cursor.close();
        return bill;
    }

    public Double getLastMonthWaterBill() {
        Double bill = 0.0;

        String queryString = "SELECT * FROM " + SPENDING_TABLE + " WHERE " +
```

```
COLUMN_SPENDING_TYPE + " LIKE '%Water%' AND " + COLUMN_DATE + " BETWEEN " +
COLUMN_DATE + "('now', 'start of month', '-1 month') AND " + COLUMN_DATE + "('now', 'start
of month', '-1 day')";
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(queryString, null);

        if (cursor.moveToLast()) {
            bill = cursor.getDouble(5);
        }

        db.close();
        cursor.close();
        return bill;
    }

}
```