Final Report

Supervisor: Dr Max Garagnani

Project Team: Lyubomir Stoyanov

ABSTRACT

The motivation for ‘Bills’ came from my personal need for better home budgeting. Instead of using Excel for home budget accountancy I thought of creating an application which does it automatically. After the idea formed a survey was conducted on similar applications and the survey discovered several similar products on the market but they all had some distinctive differences from Bills. The mobile applications which the survey discovered where: Every Dollar, Fudget, Mint, YNAB, Moneyfy, Expense manager, Daily Expense manager. All of the mobile applications which were explored in the survey are more or less well organized in tracking ones expenses on a daily bases, however some are more budget oriented than expense tracking and none of them have the option to automatically categorize the items from a receipt which could be really helpful for expense tracking.

The idea of the application is to help people of all ages to have a better home budgeting and money management. The application potentially will be able to provide a qualitative data to both users(customers) and brands(sellers)

The application will prompt the users to make a registration and login with their username and password credentials. In this way all information which is accumulated throughout the use of the application will be saved on their accounts on a safe database. After been logged in a user will be able to scan QR code or a barcode on a receipt from a store and the information from the receipt will be automatically transferred onto their account. The application should be able to distinguish the different items bought and it will separate them in different groups(Food, Drinks, Alcohol, Household goods, Travel etc.). Once the QR code or barcode is scanned the application should put the different expenses in the different categories. The application will total the values of the different groups separately and together and the user will be able to see in a clear and comprehensive manner how much and on what they have spent their money for a particular period of time(A month, week, day). The information might be provided in percentage or in a different data visualization way and this will be decided in the process of development. Ultimately I envision a data visualization in the form of map populated with all the stores which the user uses and if clicked on a particular store information about how much and on what the user has spent.

ACKNOWLEDGEMENTS

I would like to thank the Department of Computing at Goldsmiths University of London, for the opportunity to work on this project.

I would also like to thank all those who participated in providing data and feedback, through participation in surveys and focus groups which has helped me shape and evolve the direction of this project.

Final Report 1
ABSTRACT 1
ACKNOWLEDGEMENTS 1
1 INTRODUCTION 4
1.1 Motivation 4
1.2 Scope and Objectives 4
1.3 Intended Audience 4
1.4 Coverage 4
2 LITERATURE REVIEW 6
2.1 Background 6
2.2 Android Studio 6
3 PROJECT DEVELOPMENT PROCESS 7
3.1 Methodologies 7
3.2 Agile Model 8
3.3 Source Code Management 9
4 ANALYSIS 9
4.1 Specifications 9
4.2 Use case diagrams 12
5 DESIGN 15
5.1 Architectural Design 15
5.2 Design Models 16
5.3 Database Design 18
6 IMPLEMENTATION 20
6.1 Approach 20
6.2 Frontend development 21
6.3 Graphical User Interface (Templates) 24
6.4 Backend development 27
6.5 Unforeseen challenges & solutions 33
7 TESTING 33

7.1 Debugging 33
7.2 Functional Requirements Testing 34
7.3 UI Testing 35
8 PROJECT CONCLUSION 36
8.1 Reflection 36
8.2 Future Work 36
9 GITLAB REPOSITORY 36
REFERENCES 36

1 INTRODUCTION

1.1 Motivation

The motivation came for my personal need for better home budgeting. Instead of using Excel for home budget accountancy I thought of creating an application which does it automatically. After the idea formed a survey was conducted on similar applications and the survey discovered several similar products on the market but they all had some distinctive differences from Bills. The mobile applications which the survey discovered where: Every Dollar, Fudget, Mint, YNAB, Moneyfy, Expense manager, Daily Expense manager. All of the mobile applications which were explored in the survey are more or less well organized in tracking ones expenses on a daily bases, however some are more budget oriented than expense tracking and none of them have the option to automatically categorize the items from a receipt which could be really helpful for expense tracking. In general Every Dollar and Expense manager are the closest applications to my idea, however their menus are differently organized which provides different functionality from what I envision. In case I am unable to implement the automatic categorization I will use real pictures of the receipts to track the expenses. This is an option which Expense manger offers but since it’s an android application I couldn’t explore it in more detail than what I saw in a video review of the app.

.

1.2 Scope and Objectives

Bills is an Android mobile application where the objective is to create a platform for people who would like to have a better knowledge over their expenses. The application would require a user to create an account in order to be able to start using the application.

The application will prompt the users to make a registration and login with their username and password credentials. In this way all information which is accumulated throughout the use of the application will be saved on their accounts on a safe database. After been logged in a user will be able to scan QR code or a barcode on a receipt from a store and the information from the receipt will be automatically transferred on their account. The application should be able to distinguish the different items bought and it will separate them in different groups(Food, Drinks, Alcohol, Household goods, Travel etc.). Once the QR code or barcode is scanned the application should put the different expenses in the different categories. The application will total the values of the different groups separately and together and the user will be able to see in a clear and comprehensive manner how much and on what they have spent their money for a particular period of time(A month, week, day). The information might be provided in percentage or in a different data visualization way and this will be decided in the process of development. Ultimately I envision a data visualization in the form of map populated with all the stores which the user uses and if clicked on a particular store information about how much and on what the user has spent. The information could be presented in a bar chart or a histogram. The different categories could give information for the total amount spent for this category as well as a percent of the total amount spent a particular month, year etc. The data visualization will play a key role in the functionality of the application, however as it is very time consuming and requires a deeper investigations my aim is going to be to develop a registration and login system and to implement a functioning QR code scanning and to be able to save each user data into a data base and very rudimentary to visualize the data.
The minimal viable product will be the creation of the menus with easy transition between them and a registration and login system with a rudimentary database attached to them.

1.3 Intended Audience

This report is intended for all individuals participating and supervising the project and presumes some technical knowledge and assumes the reader has read the initial project proposal.

1.4 Coverage

This report includes seven chapters, excluding the introduction:

Chapter 2 - Literature Review: provides a brief overview on what was outputted in the project proposal and includes theoretical background and research into how Android Studio works.

Chapter 3 - Project Development Process: covers project development methodology providing an overview of the development model adopted for this project and source code management.

Chapter 4 - Analysis: requirements specification covering functionalities of the project and requirements analysis through use case interactions.

Chapter 5 - Design: describes the architectural design and artefacts of the web application.

Chapter 6 - Implementation: covers how the application was implemented.

Chapter 7 - Evaluation & QA Testing: formative evaluation and functional testing.

Chapter 8 - Project Conclusions: provides reflective conclusions of the project and proposed future work.

2 LITERATURE REVIEW

In this section, the background to the project is discussed where I review the foundations and vision of my project. I also consider the technologies and theories used to develop the application including the basic concepts of Android Studio.

2.1 Background

This report builds on my project proposal where I laid the foundations and vision behind my project - establishing the stakeholders for conducting market research as well as social and marketplace trends. I used Android Studio for the development of the project instead of using xcode and swift as initially intended because more useful information about the implementation of the application was discovered post the preliminary research.

2.2 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.

As an IDE then, Android Studio's job is to provide the interface for you to create your apps and to handle much of the complicated file-management behind the scenes. The programming language you will be using is Java and this will be installed separately on your machine. Android Studio is simply where you will write, edit and save your projects and the files that comprise said projects. At the same time, Android Studio will give you access to the Android SDK or 'Software Development Kit'. Think of this as an extension to the Java code that allows it to run smoothly on Android devices and take advantage of the native hardware. Java is needed to write the programs, the Android SDK is needed to make those programs run on Android and Android Studio has the job of putting it all together for you. At the same time, Android Studio also enables you to run your code, either through an emulator or through a piece of hardware connected to your machine. You'll then also be able to 'debug' the program as it runs and get feedback explaining crashes etc. so that you can more quickly solve the problem.

2.2.1 Activities

Android Studio makes life significantly easier compared with non-specialist software, but is still has a little way to go before it can claim to be a completely intuitive and smooth experience. For complete beginners, there is an awful lot to learn here and much of the information available – even through official channels – is either out of date or too dense to make head or tails of.

2.2.2 Layout

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects.

2.2.3 DatabaseHelper

The DatabaseHelper class contains the code which defines the data base tables, the relationship between them and the functions for inserting new data into the database.

2.2.4 SQLite

SQLite is a open source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features.

2.2.4 Widgets

The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.

3 PROJECT DEVELOPMENT PROCESS

3.1 Methodologies

The project development process included a set number of stages to my app's evolutionary lifecycle [1]. First I began with the requirements phase which outlined the goals of what my application will be capable of doing. Next, was the design phase which provided a blueprint to what the app's menus would look like. The implementation phase is where functionalities outlined by the design of the system were implemented. However, due to the strange times the last few months I was able to implement only the minimal viable product without any further initially considered developments. Finally comes evaluation and testing phase which allowed for iterations of the app to be tested and evaluated, whilst also addressing bugs to ensure the fulfilment of project requirements.

There are various development models that currently exist in software engineering [2] that consist of an abstract representation of a software process through various conventions. I considered various models and approaches of software development but difficult to say that I have used one particular method. I built the web application on small increments and in this sense my development process resembles the incremental model. However, each incremental development built upon previous functionalities and each release tested for quality assurance and in this sense the development process I used resembles the agile method, since the application was always operational and only evolved over time.

The Waterfall Model [3] offers a linear-sequential lifecycle model, where each phase must be completed fully before the next phase can begin. At the end of each a phase a review takes place that determines if the project can progress to the next stage. This can work well for smaller projects where requirements are clearly understood. However this model is unsuitable for long and ongoing projects where requirements are subject to changing and it cannot produce a working application until later in the development lifecycle. Lastly, when the app reaches the testing phase, this model makes it quite difficult to go back and change something that might have been flawed in the concept phase.

The Incremental Model [4] approaches software development by dividing requirements into various builds. This consists of multiple development cycles divided into smaller more manageable modules, where each module goes through the requirements, design, implementation and testing phases with a working version of the application produced in the first module. Subsequent modules adds functionality to the previous release. This process continues until all project requirements are fulfilled. While this model is flexible and produces a working application quickly, it does however require good planning and design of the entire system from the get go before it can be implemented incrementally.

The Spiral Model [5] is quite similar to the incremental model, with the inclusion of risk analysis. It includes, planning, risk analysis, engineering & evaluation. Software development process is presented in a spiral based on iterations. Subsequent spirals build on the baseline spiral. Requirements are gathered during the planning phase, then a process is undertaken to identify risks with alternate solutions proposed during the risk analysis phase. A prototype is produced at the end of the risk analysis phase. This model works well for larger projects where the costs of development is an important factor, with avoidance of risks playing an important role. However risk analysis requires great expertise in that parameter and project success highly dependent on the outcome of risk analysis.

3.2 Agile Model

The Agile Model [6] is another variant of the incremental model. Software development process is done in incremental, rapid cycles, with small incremental releases with each release building on previous functionality. Each release is tested for quality assurance. This model has several advantages including frequent delivery. The Agile Model was chosen by me as the best approach to the development process of this project. One of the most important reasons for this is because this model allows for changes to the project requirements giving scope to respond to formative evaluation with the users and provides frequent incremental releases and iterations of the development. Furthermore, with frequent 'sprint planning', it allows to develop the application in increments with each one building and improving on previous release. In each increment, there will be some changes to the requirements and improvements to the design leading up to new software development cycles. Furthermore the application will need to be tested on multiple devices. This creates a need for a software development methodology which can handle change and provide a way to meet the rapid development requirements. The phases of the agile model are followed in the development process of project, with the following taking place:

Requirements Specification: Description of the behavior of the system to be developed and fulfilled.
Design: Blueprints of the system's architecture and its important components
Implementation: Design artefacts and blueprints implemented
Evaluation & Testing: Test cases to ensure the system has met its specification and evaluation of successes of previous phases with modifications made to them when applicable.

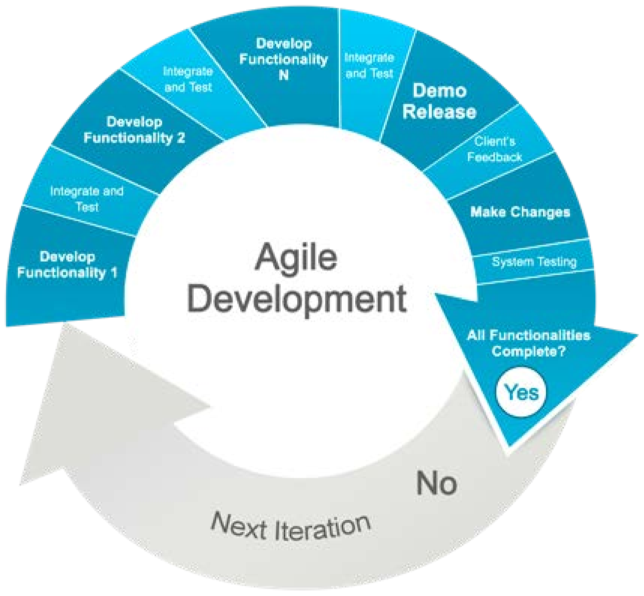


Figure 1: Agile Development diagram

3.3 Source Code Management

An important part of the development process of an application is how a team chooses to manage their application's source code. In recent years, advances in version control systems such as git have provided powerful tools allowing for concurrent development and traceability.

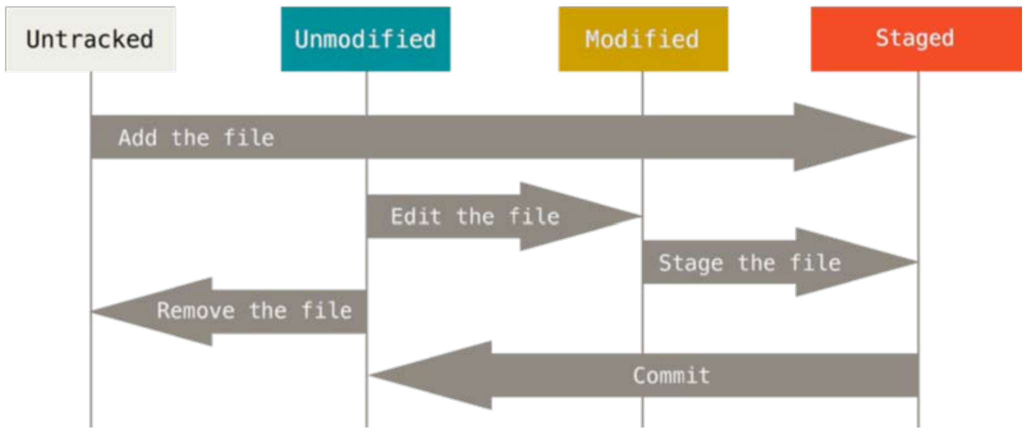


Figure 2: Gitlab source code management diagram

I used gitlab [7] - a source code repository which provides storage, management and tracking of changes made to the code throughout implementation of the system but most of the work was done on my local machine since there was no need to share the code with anybody and gitlab was used only for sharing my work at the end.

4 ANALYSIS

I have already established a set of functional requirements derived from my project proposal for my application to fulfil. Following some formative evaluation with the potential users via online questioner - I made some refinements and additions to the requirements.

4.1 Specifications

My system requirements are divided between functional and non-functional requirements. I looked to various conventions defined in software engineering to aid me in the analysis of the requirements of the system under development. [8]

4.1.1 Functional Requirements

Functional requirements defines the functionality and components of the system.

Requirement ID	Requirement Name	Statement	Must/Want
1	Register Username, Password. Logging In.	The user shall be able to login to use, work and access information. For example to add a new expense or simple check their current financial situation. The user shall be able to register before providing any sort of information.	Must
2	Uploading Photos	The user shall be able to upload photos for their profile picture.	Must/Want
3	Edit Profile Info	The user shall be able to edit their profile information once they have registered. They should be able to edit their profile information & picture to identify themselves.	Must/Want
4	Adding an Expense Category	The user should be able to add a new expense category	Must
5	Adding an Expense Category	The user should be able to select an expense category for adding an expense	Must
6	Selecting and Expense	The user should be able to select an expense in order to add a picture of a receipt.	Must
7	Adding a picture of a receipt for a certain expense	The user should be able when adding a expense to add a picture accompanying the expense.	Must/Want
8	Help	Should contain all the information one should need regarding the use of the application.	Must/Want
9	About	A general information about the application.	Want
10	Search an expense by date	A user should be able to search for an expense within a certain category by date.	Must/Want
11	Logging out	The user shall be able to sign out once they're done with the application or if they are inactive. This will insure efficient security and robust inactivity.	Must/Want

Table 1: Functional Specification

4.1.2 Non-functional Requirements

Non-functional requirements support the functional requirements - describes how the system will fulfil them. For example covering areas important to the users such as usability, reliability, security, performance and costs. These are important requirements to the users which were indicated by a focus groups.

Requirement ID	Description	Statement
1	The system will be able to run on all devices.	The system will be developed to run on all devices such as mobiles/tablets.
2	The system must be designed to be able to evolve with new functionality and operations.	The system must be designed in a way that can be added new functionalities and features to the source code.
3	The graphical user interface must be intuitive and dynamic to fit all screen sizes.	The layout of the application should be universal to all screen sizes. The UI experience should be simple and consist for our users across.
4	The system must lack bugs and inform the user of any wrong operations encountered.	With each incremental release the system must be thoroughly tested with error message logs implemented to inform a user of any errors in operations encountered.
5	The system will have a fast response time.	The system should run operations quickly. Operations that require longer execution times will present the user a 'loading icon' until the operation is fulfilled.

Table 2: Non-functional Requirements

4.2 Use case diagrams

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

In this project, the actors are the primary users from the stakeholders defined in the proposal which describe a set of operations and interactions that users can perform with the system.

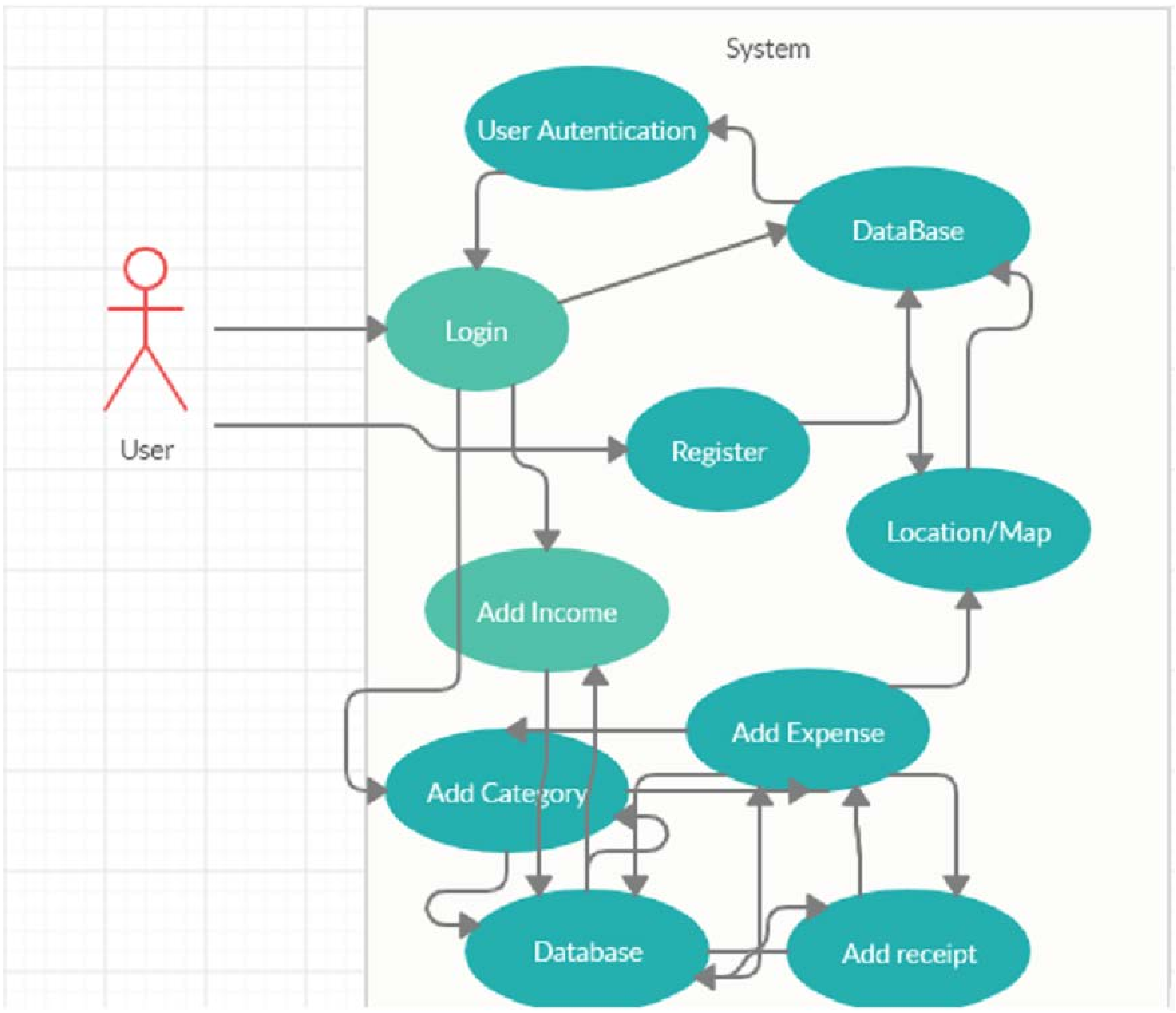


Figure 3: Profile use case diagram

4.2.1 Use Case Scenarios

I was able to describe the use case interactions with the system in more detail in form of use case scenarios. Every use case represents a user scenario and provides information on the name, the actors, the components, a description, an abstract basic flow, a precondition and its trigger. This enables traceability of the requirement and helps to evaluate its outcome.

User Case Name	Upload picture of a receipt
Context	Upload receipt to the server.
Primary Actor	User
Precondition	The user has to be logged in in order to access and upload it to the right server.
Success Post Condition	The receipt will be uploaded and there will be a dialogue message telling the user that the receipt is uploaded.
Trigger	The user wants to upload a receipt
Main Success Scenario	1.The user directs to the upload navigation bar 2.The user pressed the upload receipt button 3.The system will display the storage of the user's current device 4. The user will choose one the receipt they want to select and hits upload 5.The storage option will close and the process of uploading a receipt will occur. a. If the photo upload is successful, a dialogue will appear and it says "Upload successful". b. If the receipt upload is not successful, a dialogue will appear with an error message and what caused it.

Table 3: picture use case scenario

User Case Name	Login
Context	Login with credentials
Primary Actor	User
Precondition	The user has to be registered in order to be able to login.
Success Post Condition	The user will log in and there will be a change in the UI suggesting successful login.
Trigger	The user wants to use the platform to either check their current financial situation or add a new expense.
Main Success Scenario	1.The user opened the application. 2. The system will display the login screen. 3. The user will enter their username and password. 4. The login screen will disappear and tailored for the type of user screen will appear. a. If the login is successful a change in the UI will suggest success. b. If the login is not successful, a dialogue will appear with an error message and what caused it.

Table 4: Login case scenario

User Case Name	Register user
Context	Registration
Primary Actor	User
Precondition	The user has open the application and has to click on the Register button.
Success Post Condition	When the registration is successful a message will indicate that and the user will be transferred to the Login screen.
Trigger	The user wants to register an account with the system.
Main Success Scenario	1. The user directs to the register screen. 2. The user fills in the registration form correctly. 3. The user clicks on the button register. 4. The system will display a message indication a successful registration and the user will be transferred to the Login screen. a. If the registration is successful, a dialogue will appear which says “Registration successful”. b. If the registration is not successful, a dialogue will appear with an error message and what caused it.

Table 5: Registration use case scenario

User Case Name	Add new expense category
Context	Add new expense category
Primary Actor	User
Precondition	The user has to be logged in order to add a new expense category.
Success Post Condition	After clicking on the button “Add” under the section Expense categories the user will be able to add a new expense category, the planned budget for it, the spent money on the category and the remaining from the planned expense.
Trigger	The user wants add a new expense category.
Main Success Scenario	1. The user directs to Categories menu and presses “Add” under the Expense Category section. 2. The user fills in the required fields, planned expense, spent, . 3. The user presses add expense category. a. If the expense category is added successfully, a dialogue will appear that says “Category has been added successfully!” b. If the category adding is not successful, a dialogue will appear with an error message and what caused it.

Table 6: Create a CV/Update a CV use case scenario

User Case Name	Add expense
Context	Add expense
Primary Actor	User
Precondition	The user has to be logged in order to access the expenses of a certain category.
Success Post Condition	The expense will be added to a certain category and there will be a dialogue message telling the user that the expense is added.
Trigger	The user wants to add a new expense to a certain category.
Main Success Scenario	1. The user directs to a certain expense category. 2. The user fills in the required fields. 3. The user presses the add button and the expense is then added to acertain category. a. If the expense is posted successfully, a dialogue will appear that says “The expense has been posted successfully!” b. If the expense post is not successful, a dialogue will appear with an error message and what caused it.

Table 7: New Job Offer use case scenario

5 DESIGN

In this section I discuss the design of the application and its database with the purpose of meeting the requirements set out in Section 4. I started by addressing the application’s architectural design followed by its design models providing an overview of the final design of the web application.

5.1 Architectural Design

When was thought about the architectural design of the project, it needed to be simple and easy to understand so that it aids in the development of a reusable and commonly understandable application. Furthermore, the architecture required a minimum dimension of modularity to ensure that this application can be developed in increments that can be expanded and refined in each of its interactions - allowing for an agile approach project development process.

5.1.1 Activities [10]

An activity represents a single screen with a user interface just like window or frame of Java.Android activity is the subclass of ContextThemeWrapper class.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram:

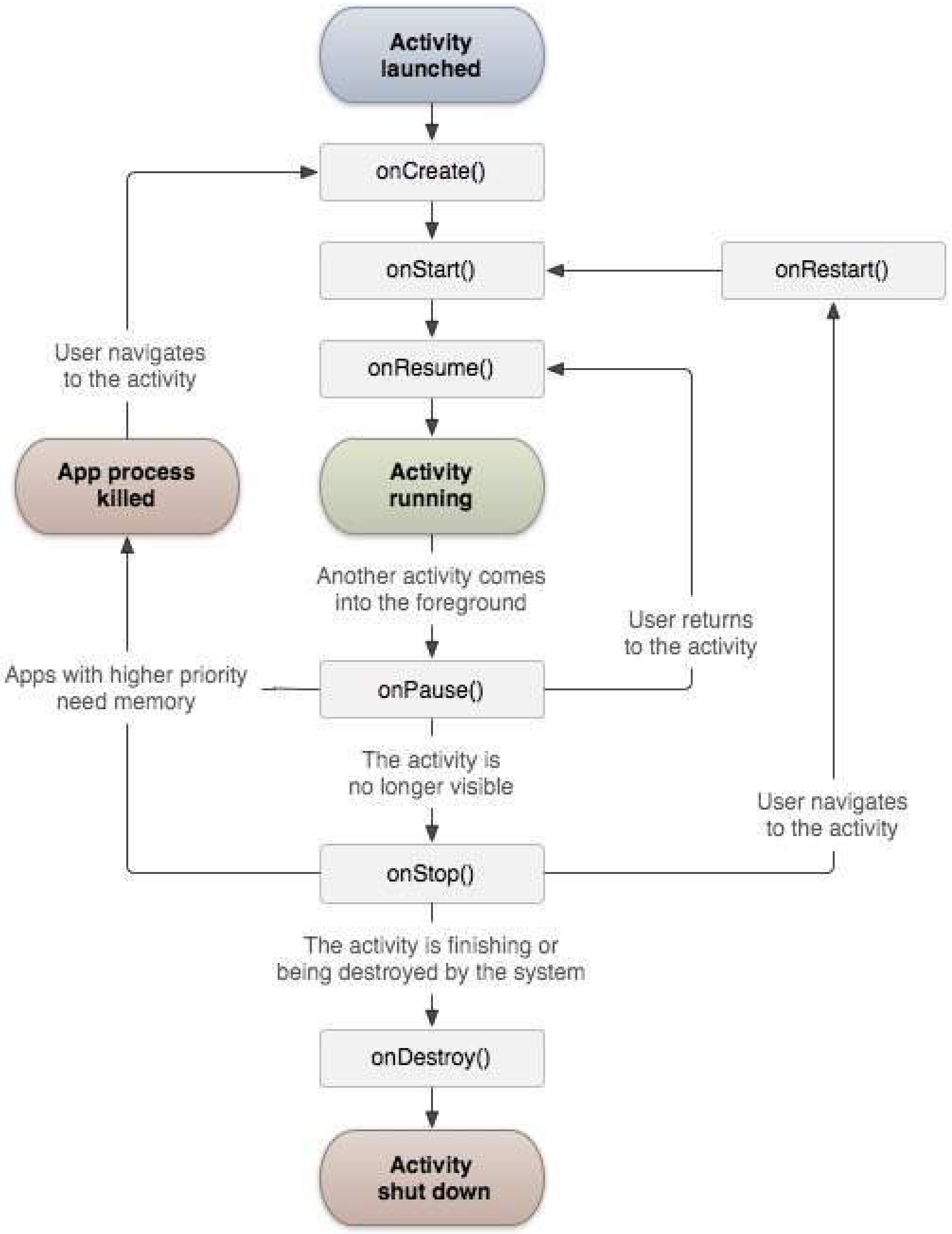

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

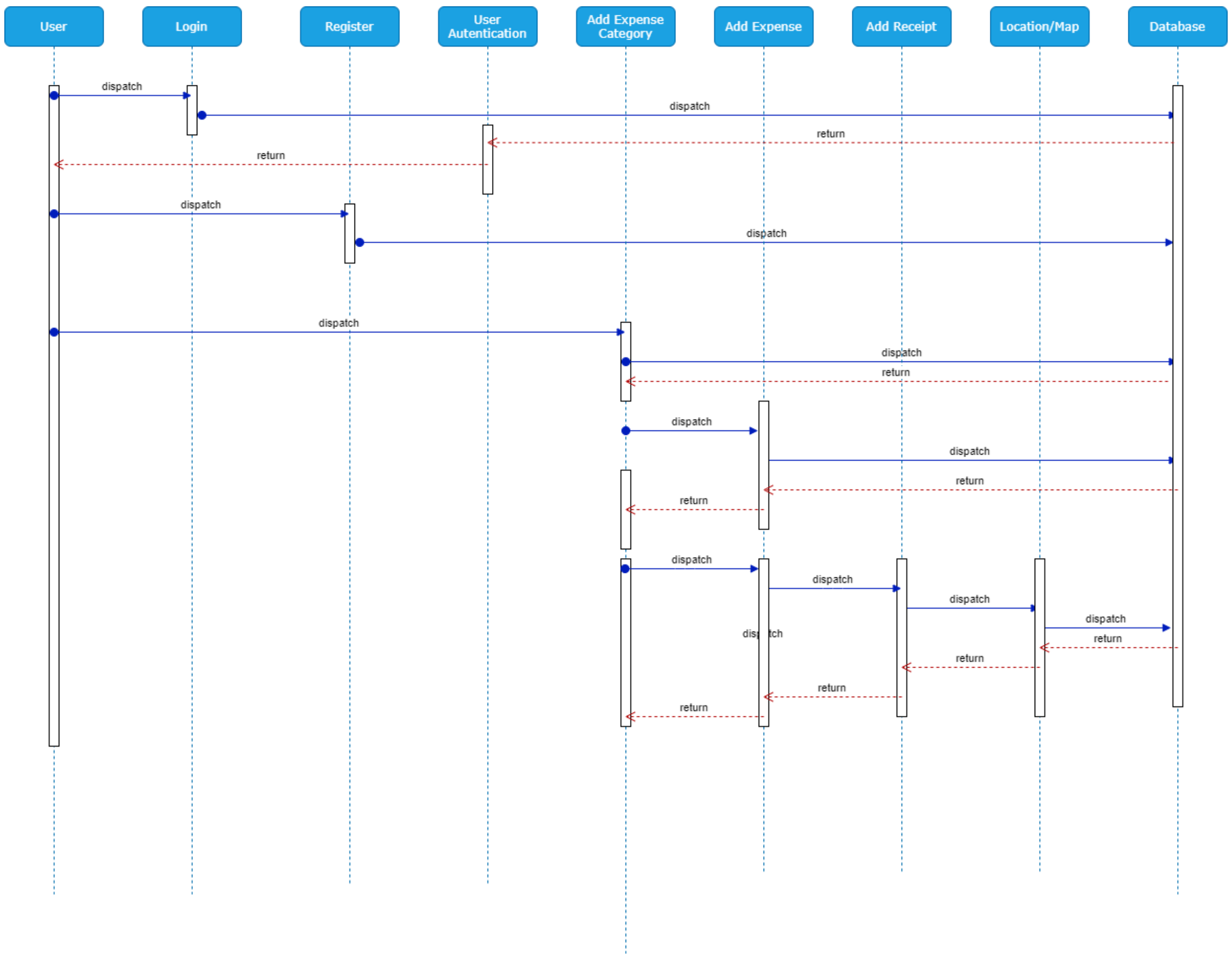
Figure 4: Activity diagram

5.2 Design Models

With the architectural design established, I could begin to describe the designs around the structure of the system.

5.2.2 Sequence Diagram [11]: A sequence diagram represents the behavior and interactions of the system by showing the interactions between objects in sequential order. Sequence diagrams dynamically focus on the 'lifelines' of an object and their communication in performing a function before a lifeline ends. We look at interactions around if the user isn't logged in and the main activities.

Figure 5: Sequence diagram encompassing the main dynamic interactions of the system



5.3 Database Design

Below shows the structure of the database implemented for this project. The table *users* is used to store the users. The table *Application* is used to store all applications that have been created by *candidates* and is matched with the User table by the User.id. The table *Job Offer* is matched to *User* by the User.id and is used to store all Job Offers by the different employers.

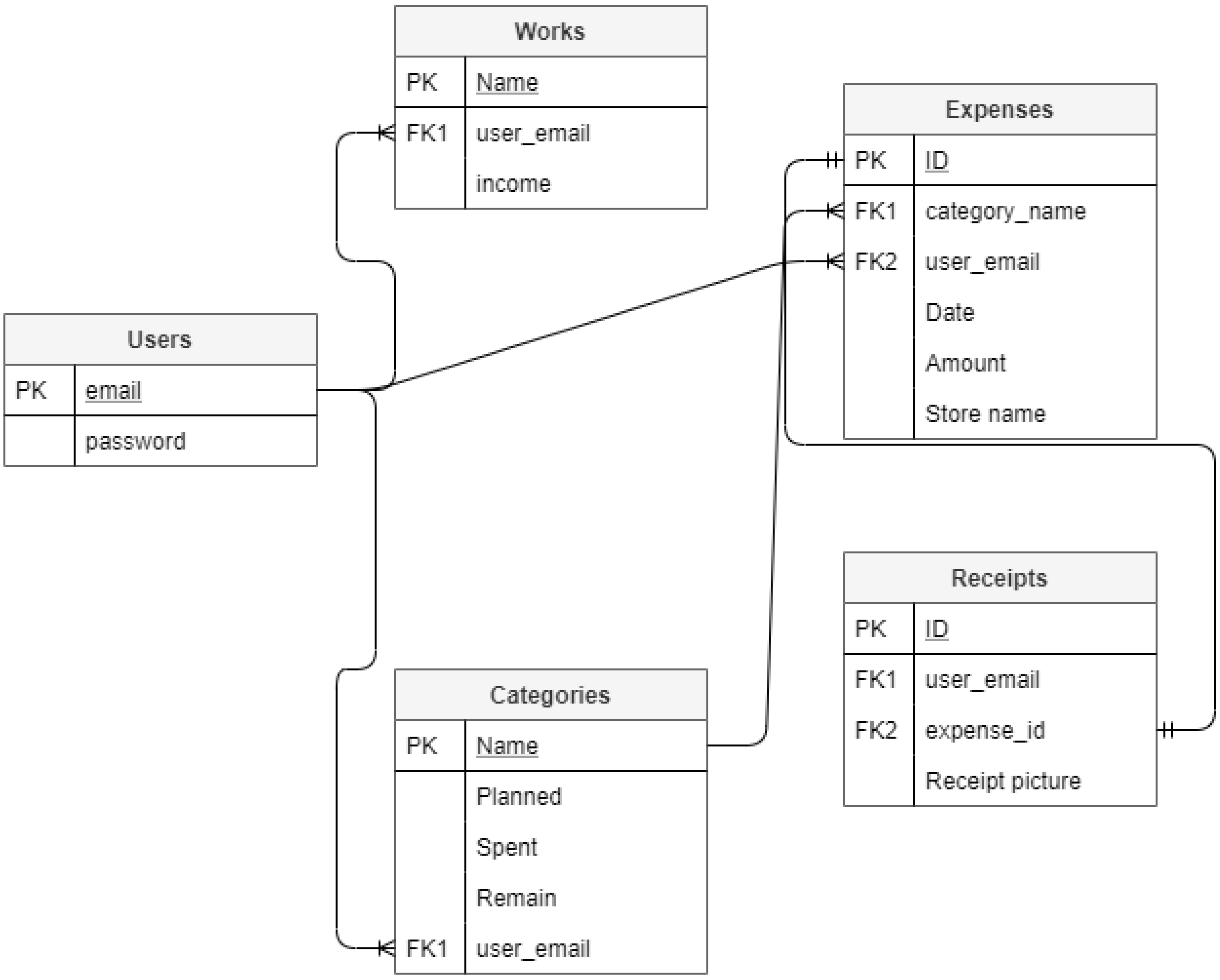


Figure 6: simplified entity-relation diagram

All the tables in the database contain a primary key. The lines in the diagram represent a foreign key relation, which means the foreign key of one table points to a primary key in another table. The use of the foreign keys is also important in the database design as it allows operations like insertion and deletion to be more efficient. The tables that will have foreign keys are all tables except Users which will hold the login credentials of a user.

User

Attributes	Data type	Description
Email (PRIMARY KEY)	STR(120)	The unique email of the user.
password	STR(60)	The password of the user.

Table 8: Users’ database table attributes

All Expenses of a particular Category will be stored into one table. When an Expense is posted, it will have its own *id* as a primary key and will have the *user_email and category_name* as its foreign key. It is then inserted into the Expenses table with the date-time recorded as well. This way all Expenses with the same *user_email* and category_name will be collected and can also be ordered by the time when the Expense was made. If an Expense was to be deleted this will not affect any other Expense by the same user. The same is applicable when a user creates a new Category. Subsequent attempts will only result in updating the already existing Expense or it’s deletion if that’s what the user wants.

Categories

Attributes	Data type	Description
Name (PRIMARY KEY)	STR(20)	The name of the category
Planned	INT	The amount a user is willing to spent on particular category
Spent	INT	The amount spent on a particular category
Remain	INT	The amount left to spent on a particular category
user_email (FOREIGN KEY)	INT	The user’s id for the CV.

Table 9: Categories table attributes

Expenses

Attributes	Data type	Description
id (PRIMARY KEY)	INT	The unique id for the expense.
Date_posted	DATETIME	The time when the expense was made.
Store name	STR(20)	The name of the store in which the expense was made.
Spent	INT	The amount of the expense
category_name (FOREIGN KEY)	INT	The category which the expense belong to.
user_email (FOREIGN KEY)	INT	The unique email address of the user.

Table 10: Expenses table attributes

Works

Attributes	Data type	Description
Work name (PRIMARY KEY)	STR(20)	The name of the income
Income	INT	The amount of the income
user_email (FOREIGN KEY)	INT	The unique email address of the user.

Table 11: Works table attributes

Receipts

Attributes	Data type	Description
ID (PRIMARY KEY)	INT	The name of the income
Receipt Picture	INT	Picture's number(name)
Expense_id (FOREIGN KEY)	INT	The unique expense id.
user_email (FOREIGN KEY)	INT	The unique email address of the user.

Table 12: Works table attributes

6 IMPLEMENTATION

This section outlines the implementation process that was undertaken - utilizing the Adroid Studio framework and the development and design methodologies discussed in previous sections to forge a functional application. I'll also discuss unforeseen problems that I encountered during development and some of the solutions to them.

6.1 Approach

To develop the application an agile iterative approach was chosen as discussed in section 3.2. Each iteration of the development cycle covers new components with its outcome forming a new increment of the application.

Iteration 1: front-end development undertaken with the setup and deployment of the project and the development environment including tools provided by Adroid Studio.

Iteration 2: design and building of simple templates for the different stages of use of the application.

Iteration 3: development and deployment of the backend including connection to a local database.

6.1.1 Basic vs core Implementation

Given time constraints and dependencies around implementation - I took a conscious decision to divide the development of the application into multiple phases. I prioritized basic development where I looked to implement essential functionality vs core development which involved extending the application's functionality. This was denoted in my functional specification in Section 4.1.1.

6.2 Frontend development

The components that are part of the Android Studio framework used for development are outlined in

Section 2.2.

6.2.1 Android Studio Components

Activity:

An activity class loads all the UI component using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity_main.xml file*:

```
setContentView(R.layout.activity_main);
```

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
      <activity android:name=".MainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

If either the MAIN action or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

Adroid Studio Emulator:

The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device.

The emulator provides almost all of the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.

Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.

The emulator comes with predefined configurations for various Android phone, tablet, Wear OS, and Android TV devices.

User input:

Getting a users input is followed by few steps. First in the design of an activity is created an EdithText field using the Android Studio palate with prebuilt buttons, icons and text fields. A specific and unique id is set for this particular EdithText field so that after an event happens, for example a button is clicked, this particular EdithText can be addressed and the information retrieved.

After the design of the EdithText is done a Java class responsible for the evoking of the particular activity with the EdithText field is created and an EdithText is declared with the unique id set in the design of the activity. Then when an event happens, a click on a particular element, an onClick event is triggered and a function getText() on the already created EdithText is evoked and the data is saved in the appropriate data type(String, Integer, etc.)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="100">


    <EditText
        android:id="@+id/addExpenseCategory"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:layout_gravity="center"
        android:layout_marginHorizontal="75dp"
        android:ems="10"
        android:hint="Expense Category"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/addPlannedToSpend"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginHorizontal="75dp"
        android:layout_marginTop="5dp"
        android:ems="10"
        android:hint="Planned"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/addSpent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginHorizontal="75dp"
        android:layout_marginTop="5dp"
        android:ems="10"
        android:hint="Spent"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/addRemain"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginHorizontal="75dp"
        android:layout_marginTop="5dp"
        android:ems="10"
        android:hint="Remain"
        android:inputType="textPersonName" />

    <Button
        android:id="@+id/addExpense"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:layout_gravity="center"
        android:text="Add Category" />

</LinearLayout>


public class add_expense_category extends AppCompatActivity {

    private TextView category;
    private TextView planned;
    private TextView spent;
    private TextView remain;
    private Button button;

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

        category = (EditText)findViewById(R.id.addExpenseCategory);
        planned = (EditText)findViewById(R.id.addPlannedToSpend);
        spent = (EditText)findViewById(R.id.addSpent);
        remain = (EditText)findViewById(R.id.addRemain);

        button = (Button)findViewById(R.id.addExpense);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                String category1 = category.getEditableText().toString();
                String planned1 = planned.getEditableText().toString();
                String spent1 = spent.getEditableText().toString();
                String remain1 = remain.getEditableText().toString();
```

Database with SQLite:

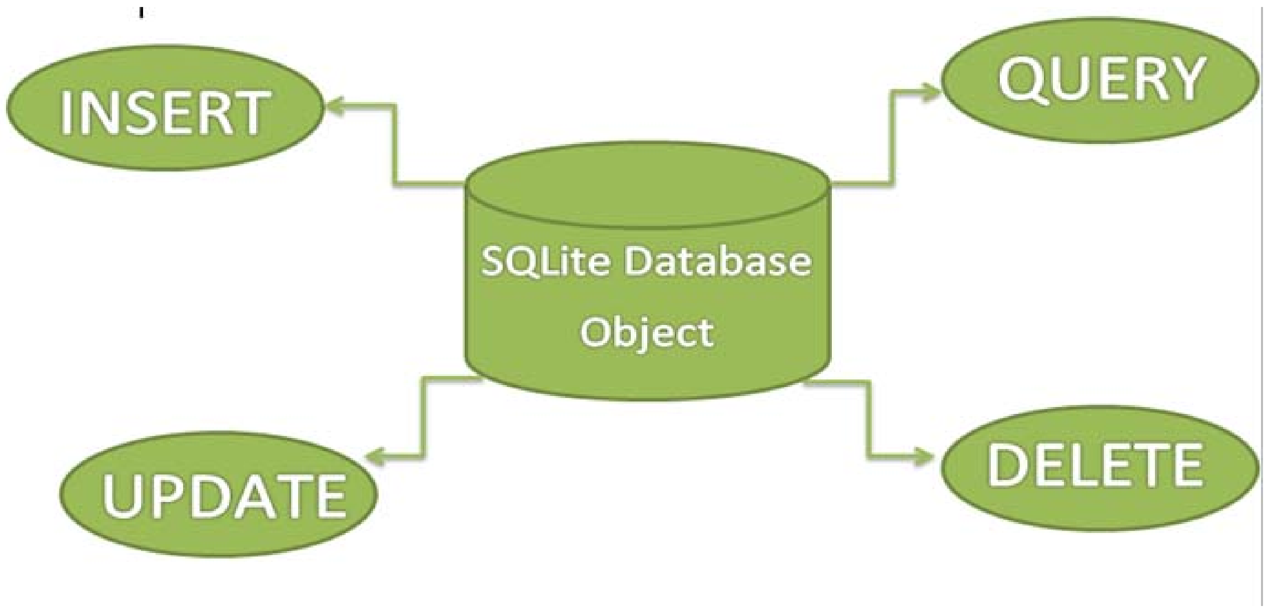


Figure 7: SQLite database structure

Whenever an application needs to store large amount of data then using sqlite is more preferable than other repository system like SharedPreferences or saving data in files.

Android has built in SQLite database implementation. It is available locally over the device(mobile & tablet) and contain data in text format. It carry light weight data and suitable with many languages. So, it doesn't required any administration or setup procedure of the database.

6.3 Graphical User Interface (Templates)

In this case the role of graphical user interface play the design of the activity XML files which are carefully constructed to accommodate the needed functionality and in the same time to be eye pleasing and intuitive and not over complicated for implementation in order future improvements to be done easier.

Write the XML:

Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

Layouts:

Each layout file must contain exactly one root element, which must be a View or ViewGroup object. Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout. For example, here's an XML layout that uses a vertical LinearLayout to hold a TextView and a Button:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

After you've declared your layout in XML, save the file with the .xml extension, in your Android project's res/layout/ directory, so it will properly compile.

More information about the syntax for a layout XML file is available in the Layout Resources document.

When you compile your app, each XML layout file is compiled into a View resource. You should load the layout resource from your app code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.*layout_file_name*. For example, if your XML layout is saved as main_layout.xml, you would load it for your Activity like so:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

The onCreate() callback method in your Activity is called by the Android framework when your Activity is launched.

XML layout attributes named layout_*something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides.

Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group. As you can see in figure 2, the parent view group defines layout parameters for each child view (including the child view group).

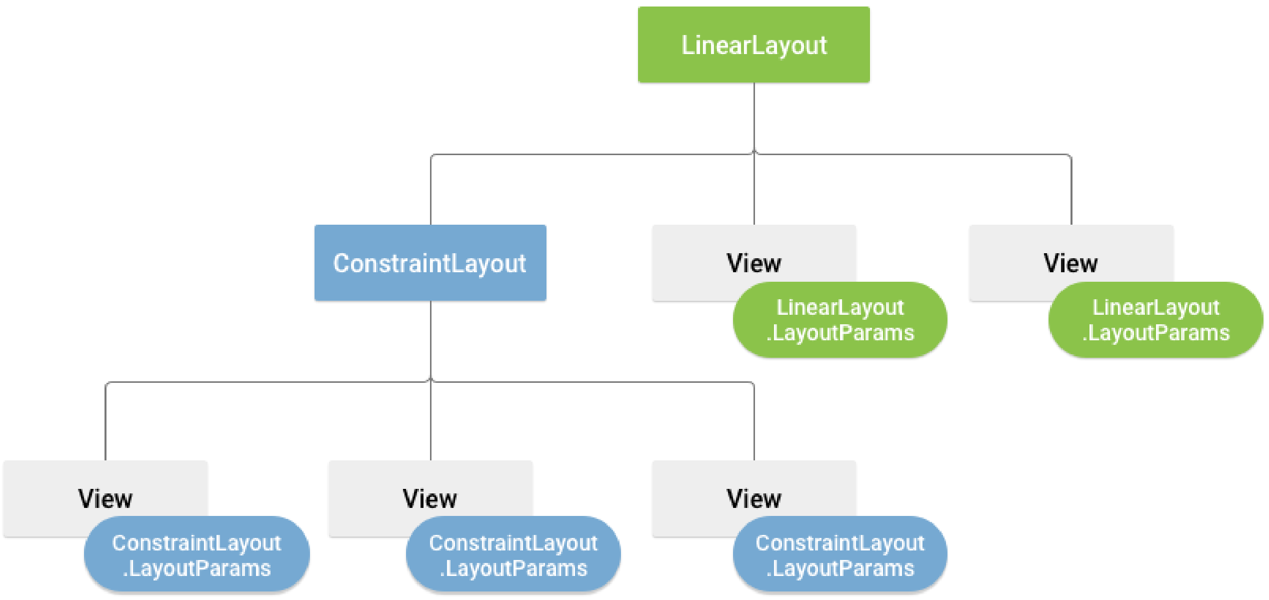

LinearLayout
ConstraintLayout
View
LinearLayout .LayoutParams
View
LinearLayout .LayoutParams
View
ConstraintLayout .LayoutParams
View
ConstraintLayout .LayoutParams
View
ConstraintLayout .LayoutParams

Figure 8: Visualization of a view hierarchy with layout parameters associated with each view

Note that every LayoutParams subclass has its own syntax for setting values. Each child element must define LayoutParams that are appropriate for its parent, though it may also define different LayoutParams for its own children.

All view groups include a width and height (layout_width and layout_height), and each view is required to define them. Many LayoutParams also include optional margins and borders.

You can specify width and height with exact measurements, though you probably won't want to do this often. More often, you will use one of these constants to set the width or height:

- ***wrap_content*** tells your view to size itself to the dimensions required by its content.
- ***match_parent*** tells your view to become as big as its parent view group will allow.

In general, specifying a layout width and height using absolute units such as pixels is not recommended. Instead, using relative measurements such as density-independent pixel units (***dp***), ***wrap_content***, or ***match_parent***, is a better approach, because it helps ensure that your app will display properly across a variety of device screen sizes. The accepted measurement types are defined in the Available Resources document.

Widgets:

The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.

You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

  You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.
- **Instantiate layout elements at runtime**. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

Declaring your UI in XML allows you to separate the presentation of your app from the code that controls its behavior. Using XML files also makes it easy to provide different layouts for different screen sizes and orientations

The Android framework gives you the flexibility to use either or both of these methods to build your app's UI. For example, you can declare your app's default layouts in XML, and then modify the layout at runtime.

6.4 Backend development

The back-end in Android Studio in terms of working with SQLite is usually implanted in a separate class but within the same project. As such the back-end of the application wasn't developed explicitly separately from everything else. The back-end is everything that isn't a template which a user can see. From the database models to the different classes behind each activity to the handling of the user authentication and the routes managing. In that spirit the back-end was developed might've been developed parallel with the templates, however as it was my first time working with Android Studio I developed the templates first along with the onClick activities and only after I started working on the database.

6.4.1 Changes on initial ideas:

The initial idea was for the development of fully operational mobile application with a QR scanning system or receipt images processing system for separating each product in the receipt into the right category and a database to save all the information. However, after a research about each of initially thought features a decision was made that nor QR scanning system or receipt image processing system will be implemented but a simple image upload with manual entering of an expense due to time constraints and possible unforeseen technical challenges which might lead to possible missing of the deadline.

6.4.2 Current state of the back-end and achieving user-authentication and registration using SQLite.

Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is “baked into” the Android runtime, so every Android application can create its own SQLite databases.

Android SQLite native API is not JDBC, as JDBC might be too much overhead for a memory-limited smartphone. Once a database is created successfully its located in **data/data//databases/** accessible from Android Device Monitor.

SQLite is a typical **relational database**, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a **schema**.

For this particular mobile application for the purpose of creating a registration form capable to save the a new user's credentials into a database a DatabaseHelper class was created which to take care of the creation of the database table required for the purpose. The DatabaseHelper class also consists of functions for inserting new data into the database and for searching into the database about a certain user's information and retrieving it and retrieving the expense categories of a user and the expenses of a certain category.

```
// Database helper class responsible for the creation of the different tables
and the functions for inserting and retreiving data from the database
public class DatabaseHelperTest extends SQLiteOpenHelper {


    public DatabaseHelperTest(@Nullable Context context) {
        super(context, "bills.db", null, 1);
    }

    //Creating the tables in the database with the relations between them
    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL("PRAGMA foreign_keys=ON");

        db.execSQL("Create table users(email text primary key, password
text)");
        db.execSQL("Create table categories(name TEXT primary key , planned
INTEGER, spent INTEGER, remain INTEGER, user_email TEXT, FOREIGN
KEY(user_email) REFERENCES users(email))");
        db.execSQL("Create table expenses(id INTEGER primary key AUTOINCREMENT,
name TEXT, amount INTEGER, user_email TEXT, category_name TEXT, FOREIGN
KEY(user_email) REFERENCES users(email), FOREIGN KEY(category_name) REFERENCES
categories(name))");

    }

    private static void setForeignKeyConstraintsEnabled(@NonNull SQLiteDatabase
db) {
        if (!db.isReadOnly()) {
            db.execSQL("PRAGMA foreign_keys=1;");
        }
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int i, int i1) {

        db.execSQL("drop table if exists users");
    }

    //Iserting new user into the database
    public boolean insertUser(String email, String password){

        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("email",email);
        contentValues.put("password", password);

        long ins = db.insert("users", "null", contentValues);
        if(ins == -1){

            return false;

        }else{

            return true;
        }
    }

    //Iserting a category to the categories table of a certain user
    public boolean insertCategory(String name, int planned, int spent, int
remain, String user_email){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", name);
        contentValues.put("planned", planned);
        contentValues.put("spent", spent);
        contentValues.put("remain", remain);
        contentValues.put("user_email", user_email);

        long ins = db.insert("categories", "null", contentValues);
        if(ins == -1){
            return false;
        }else{
            return true;
        }
    }

    //Iserting an expense into the exepnses table of a certain category of a
ceratin user
    public boolean insertExpense(String name, int amount, String user_email,
String category_name){

        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("name", name);
        contentValues.put("amount", amount);
        contentValues.put("user_email", user_email);
        contentValues.put("category_name", category_name);

        long ins = db.insert("expenses", "null", contentValues);

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

    //Checking if an email has already been registered
    public boolean checkmail(String email){
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery("Select * from users where email=?", new
String[]{email});
        if(cursor.getCount()>0){

            return false;

        }else{

            return true;

        }
    }

    //Checking for a record in the database with certain email and password
    public Boolean emailPasword(String email, String password){

        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery("select * from users where email=? and
password=?",new String[]{email,password});
        if(cursor.getCount() > 0){
            db.close();
            return true;
        }
        else{
            db.close();
            return false;
        }
    }
    //Getting all the categories added by a certaing user
    public Cursor getCategories(String user_email){

        String query = "SELECT * FROM categories WHERE user_email=?";
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.rawQuery("select * from categories where 
user_email=?", new String[]{user_email});

        return cursor;
    }

    //Getting all the expenses added to certaing category by a certain user
    public Cursor getExpenses(String user_email, String category){

        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.rawQuery("select * from expenses where user_email=? 
and category_name=?", new String[]{user_email, category});

        return cursor;
    }
}
```

The Login authentication is done in a separate class which is responsible for the loading of the Login activity which represents the front-end, the menu appearance of the Login form. The class is conveniently called Login. When the application is initially started the Login activity opens and the Login class is activated automatically. A database is created and if a user tries to Login and their credentials are in the database the user is allowed enter the application.

```
public class Login extends AppCompatActivity {

  private TextView registerHere;
  private Button loginBtn;
  private EditText email_login, password_login;
  DatabaseHelper db;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);
    db = new DatabaseHelper(this);

    email_login = (EditText)findViewById(R.id.email_login);
    password_login = (EditText)findViewById(R.id.passowrd_login);

    registerHere = (TextView) findViewById(R.id.register_login);
    loginBtn = (Button) findViewById(R.id.login);
```

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

            @Override
            public void onClick(View view) {

                openRegisterActivity();

            }
        });

        loginBtn.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                String email_login1 = email_login.getText().toString();
                String password_login1 = password_login.getText().toString();

                Boolean checkEmailPass = db.emailPasword(email_login1, password_login1);
                if(checkEmailPass == true){

                    openMainScreenActivity();
                }else{

                    Toast.makeText(getApplicationContext(),"Ivalid credentials!",
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }

    private void openMainScreenActivity(){

        Intent intent = new Intent(this, planned.class);
        startActivity(intent);
    }

    private void openRegisterActivity(){

        Intent intent = new Intent(this, Register.class);
        startActivity(intent);
    }
}
```

All back-end software is currently configured to run either on a local machine. First

6.4.3 Development:

First I started by creating a few very simple sketches of what I envisioned the application would look like in its essence and what the different menus' functionalities would be. Consequently a research about other similar applications was conducted which concluded that there are already multiple applications similar to Bills. After the differences and similarities between Bills and the already existing applications on the market were established some changes were made on the initial model to separate it as much as possible from the already existing similar apps for better fit in the market. When all changes on the preliminary sketches were done a high fidelity prototype was created in order the stakeholders to gain a better perspective into what the application is about and its functionalities. The prototype was only a mock up of what the real application was envisioned to be with automated transition between the different menus, however with no option for entering real data, but it served a perfect purpose to give accurate perspective into what the capabilities of the application would be. Now I had to find the right tools to build up a real working mobile application. First I thought of using xcode and swift but the amount of information on the internet wasn’t enough by my opinion to freely explore and built the desired functionalities without to endanger the on time delivery of the project, so I searched for alternatives. I had already thought about using Android Studio but I still wasn’t convinced if that was the right choice so I continued searching for the right platform. I came across videos on youtube about a platform called Flutter which videos showed a great capabilities by the platform but still the amount of videos and the text available on the internet about the different functionalities which I envisioned for Bills wasn’t enough so I decided that I would have to use Android Studio. Installing Android Studio and making my first design of a menu wasn’t that difficult, however it was very time consuming. Finding the right information took some time but the really time consuming part was getting familiar with the emulator and some meaningful ways to debug the code in a case of malfunctioning. Starting to link the different menus together so that a user can transition between them was the next part of the development. When I knew how to get the data provided by a user I had to find how to transition it from one menu to another. Adding new elements into the design dynamically on click it turned out to be not that difficult at all but it took a lot of time to find the right source about how to do it. The last part was adding a database to hold the information about each user and setting up the registration and the login systems. This again turned out to be not that difficult to learn but the amount of time I spent on looking for the right information comprised of most of the effort to accomplish that part of the project. I’m thankful of all the videos on YouTube about mobile application development with Android Studio and the multitude of answers of people to different problems on stackoverflow and the mupltiple Android Studio websites.

6.4.4 Limitations of the current implementation and future considerations:

The current application as it is doesn’t have much capabilities but it is the ground work for future development. The database in place is local only, the data stored in the database isn’t encrypted which is significantly important for securing the information of the users. Further the application as it is in the moment doesn’t have an option to logout an user.

Additionally due to the time limitation on the creation of the project there isn't a working solution for searching for an expense and uploading a receipt picture. However, it is thought what the search engine should be like and is a paramount when implemented to leave the door open for future improvements if the search criteria change over time.

6.5 Unforeseen challenges & solutions

During the implementation phase I encountered a number of challenges that hindered development and solutions considered.

6.5.1 Adding elements dynamically

This was one of the most cumbersome parts in the development of the application as the information on the internet was in most of the cases not exactly what I needed for this functionality and I had to either come up with the solution myself or search more extensively which in both cases was very time consuming. However, I couldn't find the exact solution so I came up with my own based on the solutions which I went through.

6.5.2 Transferring information from one screen to another

Transferring data from one screen to another isn't in itself a difficult operation, however because of the way the Expense Category menus is structured the transfer of information had to be done in a way in which doesn't interrupt the flow of code execution and finding a solution to this was a time consuming as a I had to swift trough the solutions of similar problems online and find the one that was closest to mine and adapt it to fit my problem.

6.5.3 Time and Technical Constraints

One of the biggest challenges encountered was the time and technical constraints during the implementation phase. Learning Android Studio by the functionalities needed to be implemented wasn't the difficult part but finding the right sources to learn from was. It was very time consuming to find the right information and then although this information in most of the cases was correct as a solution to the problems which were solving it wasn't exactly what I needed and I had to adopt it to the problems I had which additionally extended the time needed to implement the functionality.

7 TESTING

This section presents the evaluation and testing of the project. Android Studio doesn't really give the developer the option to check whether everything works the way it should but with the use of Toast(onclick flashing on screen messages).

7.1 Debugging

The Toast messages get displayed not only on the screen but in the console as well which is the place where errors get displayed as well. It isn't extremely helpful to debug the code using only Toasts but since there isn’t option to print message on screen when a certain point in the flow of execution and didn’t have the time explore how the debugger works I had to work using Toasts only.

7.2 Functional Requirements Testing

The testing of all functional requirements was done by me manually as the project isn’t that complex and if a function works for one user this means that it should work for all other users too. The testing was done sequentially by using the list of functional requirements mentioned in **Section 4** and marking out every function which has already been tested.

Test Reference	1
Requirement	1
Content	Register Username, Password. Logging In.
Input	User enters credentials in the required fields
Pass Criteria	Application displays a loading screen and successfully transports the user to home screen.
Test Reference	2
Requirement	2
Content	Adding Expense Category
Input	The user goes to the Expense Category page and clicks “Add”. Then the user is transferred to another page where they can add a name, planned budget, currently spent money and remaining budget for a new category.
Pass Criteria	After the operation is done the new category appears instantly on the screen.
Test Reference	3
Requirement	3
Content	Add Expense for a certain category.
Input	User goes to the expense category screen and clicks on certain category then is prompted to a new menu where by clicking on a button new expense the user can enter the date, the name of the store and the amount spent.

Pass Criteria	New expense appears instantly into selected category.

Test Reference	**Result**
1	**PASS**
2	***FAIL/Not Implemented***
3	***FAIL/Not Implemented***
4	**PASS**
5	**PASS**
6	**PASS**
7	***FAIL/Not Implemented***
8	***FAIL/Not Implemented***
9	***FAIL/Not Implemented***
10	***FAIL/Not Implemented***
11	***FAIL/Not Implemented***

Table 11: FR Test Case Results

Table: outlines the test results of the test cases. Not all functional requirements were met during the implementation phases. This is because some of the functionalities require more time and men power to be developed which I didn't have.

7.3 UI Testing

In addition to unit testing of functional components of the application it's also vital to test the usability and behavior of the application UI running on diverse set of devices. This fundamentally ensures that the application returns the correct output and response to a sequence of user actions.

There are several approaches one can follow to test the UI of a system. However, I conducted the tests manually.

Test Reference	**Content**	**Result**
1	Checks if the application UI gives scope for keyboard to allow input in EditText objects found in forms such as username, password etc.	**PASS**
2	Checks if the application UI handles the transition between menus correctly.	**PASS**

3	Progress dialogs shown correctly when loading data from a database.	**PASS**
4	Checks if the application UI shows all context menus when items are selected.	**PASS**
5	Checks if the application UI shows buttons correctly in every screen	**PASS**

Table 12: UI Test Cases

8 PROJECT CONCLUSION

8.1 Reflection

The project had an ambitious goal of revolutionizing the way we do home budgeting and even though it's current stage is far from the lofty goal set in the beginning it gave me different technical expertise and the opportunity to work on a project in Android Studio. For me it wasn't the first time I learnt about approaching the software development ecosystem - starting with conceptual ideas to software engineering conventions to software development such as agile and UML modelling, right through to the implementation but it was the first time I worked on a mobile application and gave me a better understanding on the whole development process. The other dimension to this project was how to manage the time well enough to finish the project on time and in the same time go through all obligatory steps of the development.

8.2 Future Work

Having taken an iterative approach combined with strong ambitions and a clear vision around the output of the project - there is scope for expansion with new functionalities and functionalities that were thought about but not implemented for some reasons, taking the project to new heights in the areas of statistical analysis

9 GITLAB REPOSITORY

git@gitlab.com:lstoyanov/bills-app.git

REFERENCES

[1] Software Development Process (2013) - http://www.computerhope.com/jargon/softdeve.htm

[2] Ian Sommerville, Software Engineering, 8th Edition, 2007

[3] Waterfall Model (2015) - http://istqbexamcertification.com/what-is-waterfall-model

[4] Incremental Model (2015) - http://istqbexamcertification.com/what-is-incremental-model

[5] Spiral Model (2015) - http://istqbexamcertification.com/what-is-spiral-model

[6] Agile Model (2015) - http://istqbexamcertification.com/what-is-agile-model

[7] GIT Source code management - https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-theRepository

[8] Software Engineering, A Practitioners Approach, 8th Edition, Pressman & Maxim

[9] UML Distilled 3rd Edition, Martin Fowler

[10] Android Developer Tools - https://developer.android.com/

[11] Learn Android, Application Development - https://www.tutorialspoint.com/android

[12] Website for programming skills for different programming languages - https://www.thecrazyprogrammer.com/2016/12

[13] Tutorials for developing mobile applications with Android Studio - https://abhiandroid.com/

[14] Android SQLite Database Example Tutorial - https://www.journaldev.com/9438/android-sqlite-database-example-tutorial

[15] Multiple videos sources on YouTube for developing mobile applications with Android Studio.