ASHESI

ASHESI UNIVERSITY COLLEGE

IMPLEMENTATION OF A WEB-BASED CODE GENERATOR FOR THE ANDROID MOBILE PLATFORM

APPLIED PROJECT

B.Sc. Computer Science

Constant Likudie Komla

2018

ASHESI UNIVERSITY COLLEGE

Implementation of a Web-based Code Generator for The Android Mobile Platform

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

Constant Likudie Komla

April 2018

DECLARATION

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……………………………………………………………………………………………

Candidate's Name:

……………………………………………………………………………………………

Date:

……………………………………………………………………………………………

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied project laid down by Ashesi University College.

Supervisor's Signature:

……………………………………………………………………………………………

Supervisor's Name:

……………………………………………………………………………………………

Date:

……………………………………………………………………………………………

Acknowledgements

I would like to honour God, my family, friends and all who stood by me in completing this project.

A special gratitude to my supervisor Dr. Nathan Amanquah for his academic guidance and dedication to this project.

Abstract

Producing software compels programmers to have in-depth knowledge of several programming languages, tools and frameworks. This time-consuming process demands dedication and mastery which are typically the foremost reasons most novice programmers give up a little into the software development process. This project seeks to eliminate learning programming languages and tools to create Android applications and reduce the time used in creating Android applications.

This paper presents a comprehensive implementation of a code generator intended for skilled and novice programmers to build Android applications through drag and drop. Likened to other code generators, using this code generator does not require knowledge of a programming language. A key advantage of using this code generator is the ability to download the source code of the generated Android application for customization, a feature that other code generators do not provide.

Regarding outputs, the code generator creates an APK that can be installed on an Android device or uploaded to the Google Play store, backend and frontend pages with a database to save all the data of the generated Android application.

Table of Contents

List of figures

Chapter 1: Introduction

1.1 Aim

This project aims at developing a code generator that automatically writes the source code for an Android application by creating an intuitive drag and drop interface where users can build their projects without having to write much or any code at all. This application is targeted at software developers who have programming experience and novices who have never written a line of code. The code generator also seeks to eliminate the repetitive tasks involved in writing data management-oriented software applications. Experienced developers may further customize the generated source code if they so wish but would typically be unnecessary.

1.2 Background

The influx of computers, smartphones and tablets in the twenty-first century has undoubtedly enhanced the usage of such devices as millions of households can boast of at least one. The abundance of smart devices such as phones and tablets has necessitated the creation of software applications for the masses. The enormous number of applications in the Google Play store and Apple store are testament to the progress of software creation. Although there are over a million applications in the mobile app stores, software development demands an enormous amount of time and a lot of repetitive code. "App developers are currently required to write copious amount of boilerplate code, scripts, organize complex directories, and author actual functionality" (Barnett, Grundy, & Vasa, 2015). The challenge of writing huge chunks of code for each application developed is indeed a pain and if too much time is spent on writing codes for configurations rather than the actual application, there is a high probability that it will result in slippage in delivery time. The current solution to this predicament is developing programs that automatically generate code for the mundane tasks of software development such as database connections, creating configuration files, database migration files and so on. In so doing, the developer can focus on developing the actual software.

The process of making software development easier started with making integrated development environments (IDEs) used for developing software applications “smarter”. IDEs such as NetBeans and Eclipse just to mention a few, automatically create folder structures and boilerplate code which most times are just HTML pages with header tags. These simple pages are not very useful when it comes to the overall implementation of a software project. Overtime, software developers started writing libraries to make the process easier, but this was also not very effective. “Modern libraries alleviate these issues, but they can only do so much. Many components need to be wired up correctly to render data to a list and each instance has slight differences” (Barnett, Grundy, & Vasa, 2015). Lately, some web application frameworks such as Ruby on Rails, Laravel and .Net Core provide software developers with scaffolds to make them write code faster and focus on the most important aspects of the software being developed. This however is not the case for mobile application development. The most used mobile application platforms, Android and iOS do not come with extensive scaffolding abilities as compared to the web application frameworks and hence make the developer code almost the entire application by hand.

The ability to automatically generate code for programs is deeply rooted in understanding how most applications work. A critical look at all data-driven applications will reveal that most applications are created based on the idea of CRUD. CRUD means create, read, update and delete. At the core of every data-driven application, what really is happening is that a user is either creating a new resource, updating, reading, deleting or searching for a resource. Code generators or scaffolding applications simply examine the parameters that the user wants in the application and then create a database, database tables, models, controllers and views based on the user’s requirements. As useful as this may be, developers often do a lot of refactoring to the scaffolds that were generated to make the application useful. Another disadvantage of the current scaffolding method is that it creates just a small part of the application and hence the developer must put the various pieces of the code together to integrate the components of the application and make the application do exactly what it is supposed to do. A typical example of this is Laravel, a PHP framework. For non-technical users however, even the generation of scaffold code still does not solve the problem of being unable to edit the applications since there is no way they can step through the code to tweak it and make it work as they desire.

This paper details how a complete Android mobile application can be created without having to write a line of code. The approach introduced in this paper makes the process of software development effortless such that novices can create applications on the go. For experienced software developers, the process is also simplified to the extent that they do not have to piece all the different segments of the generated code together before it becomes a complete application. They could also view the underlying code to make tweaks if they so desire.

1.3 Related Work

Developing software for production is no easy task and as affirmed by Shinde & Sun (2016), creating software applications require a lot repetitive code. To solve this problem, the two proposed “a template-based code generation framework to automate the database-oriented web services for improving code quality” (Shinde & Sun, 2016). According to Shinde & Sun (2016), given that clients want their applications developed faster, it is prudent to have a tool that enables developers generate applications in a shorter time frame. Using a code generator makes developers focus on the main functionalities of the application while the repetitive aspects of the code are dealt with by the code generator. Their template-based code generator ensures that programmers adhere to standards that make their programs more secure and maintainable. Security has been a major issue in most code generators as the boilerplate code generated is the same each time. This means that if a hacker can bypass one system that was built using a code generator, chances are that the same hacker can bypass all systems that have used that code generator since they will be the same code at the core.

For users who are tech savvy, there exists a tool for generating code that uses domain specific language. This means that for the code generator to generate the much-needed underlying code, the user must specify the specifications of the application in a higher-level language called RAPPT DSL. The code generator known as RAPPT, developed by Vasa, Grundy & Barnett (2015), passes the code generated by the RAPPT DSL through a parser and then through an inference engine before it finally generates the program that the programmer specified. The advantage of this method is that the code generator does not generate just any generic code but rather generates code based on the user's specifications. Though this is an improvement on previous code generators, its target users are limited. Users who have no programming skills may find it difficult to use RAPPT because they must get an in-depth understanding of how RAPPT DSL works. The code generator also does not generate a ready-to-ship application as its primary features are to generate the database and then some higher-level code. This means that the user still needs to piece the code together to make it work as a full application.

An advanced approach to code generation has been developed by Franky & Pavlich-Mariscal (2012) which makes use of regular expressions substitution to generate code based on previous projects that the user has worked on. This code generator requires a user to have some past projects. Running regular expression substitution on the past project, the codes of the old project that are no longer required in the new project are replaced. Generating code from the programmer's previous code projects means that the programmer will be familiar with the code base that is generated by the code generator. Although this code generator has immense benefits, it is not ideal for users who have no coding experience because they might not have previous projects that the regular expression code generator will be run on. It also means that if there were some flaws in the programmer’s previous project, the flaws will most likely be found in the new code that is generated by the code generator since they will have the same code base.

As per the examples provided above, it is evident that a lot of effort has been put into code generators. However, a new wave of code generators is beginning to spring up. Unlike all the previous versions of code generators, this new wave of code generators uses artificial intelligence to generate their code. A typical example is the Airbnb sketching interfaces where the front-end code of an application is generated by an AI observing wireframes drawn on paper (Wilkins, n.d.). What all these code generators have in common which is a limitation that my paper seeks to address is that they all generate code for the user interfaces and do not go any further. This means that even an expert in writing computer programming will still have to modify the code for it to communicate with some database to make it useful.

In Ashesi University College, two students previously worked on code generators that were primarily used for data collection. In this project, the limitations of the previously worked on code generators is addressed. These limitations are the lack of an intuitive user interface for users to build their application and the burden of users having to compile their own code using the terminal. The code generator implemented in this paper automatically compiles the generated Android source code into an APK. This code generator also provides an intuitive interface where users build their applications through drag and drop unlike the previous implementation where the users had to choose the user interface elements needed for their application from a list.

The MIT App Inventor is another platform for generating code through drag and drop. With its powerful features such as giving access to most of the phone's capabilities and allowing users to write logic for their applications using code blocks, it has a disadvantage that this code generator seeks to address. The disadvantage of App Inventor that this code generator seeks to address is the inability to download the source code of the generated application.

In this paper, a code generator that can generate front-end, backend code and database thereby creating a finished data collection application, is presented. As most business applications revolve around collection of data, the code generator presented in this paper does not only build user interfaces but also generates a database for every Android application generated to give the application data collection abilities. Applications that can be built using this code generator include but not limited to visitors' log book, record of sales and record of deliveries.

The code generator presented in this paper will reside on the web making it platform independent. A major advantage of using this code generator is that the source code of the generated Android application can be downloaded and modified by the user, a feature which is non-existent in other code generators. Figure 1.1 shows a conceptual overview of the code generator implementation in this paper.

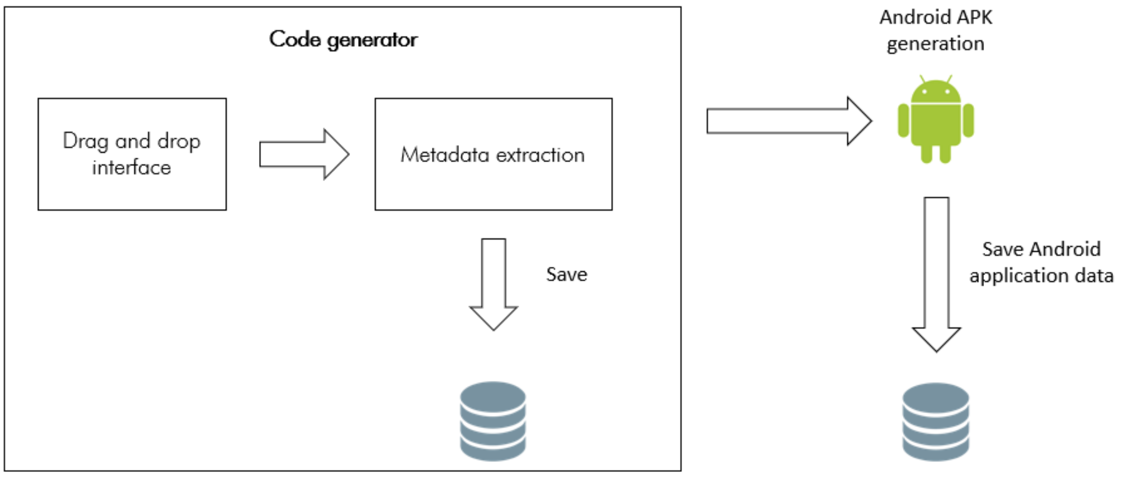


Figure 1.1 Diagram showing a conceptual overview of the code generator.

1.4 Objectives

To complete this project, the following objectives need to be met:

1. To create a web-based application that enables a user to create an android application.
2. To create a drag and drop user interface that is usable by novices.
3. With respect to the code generator, metadata refers to the set of data that describes the Android application to be created. As such, an objective is to extract the metadata that was generated through dragging and dropping.
4. To store the extracted metadata for future editing/or to permit storage and retrieval of metadata.
5. To create an application for generating the code.

6. To package the generated Android application and make it ready for deployment.

Chapter 2: Requirement Analysis

2.1 User Requirements

2.1.1 Approach

To complete the code generator, there was a need to understand the stakeholders of the project and know exactly what they wanted the application to do. The stakeholders as identified during the user requirements gathering are experienced programmers who would like to use the application to quickly generate start up code to make their work faster. The other stakeholders are users who do not have any programming knowledge at all but have some experience using the computer and can perform basic functionalities such as drag and drop. The stakeholders were interviewed and their responses to the interview questions have been added in Appendix A for reference.

2.1.2 Scenarios

The data collected, and insights drawn from interviewing the stakeholders of the code generator application were used to formulate the following scenarios under which the code generator will be used.

First Scenario: Kwame an experienced programmer working in a well-established software company in Ghana has been given the task of building a data collection application by his boss. This new application was impromptu, and Kwame is already hard pressed for time since he has unfinished tasks from a project the company is currently working on. Deadlines are rushing at him fast and he needs a faster way of creating this new application assigned to him by his boss. Kwame launches his web browser and then visits the URL of the code generator. He builds the application faster by using the drag and drop feature provided by the code generator. Upon completion, Kwame realizes that he needs a few tweaks to the application. He goes on to download the source code that was generated by the code generator and launches it in Android Studio where he makes his tweaks and then packages his app as an APK and hands it over to his boss in a couple of minutes. He is satisfied and goes back to his original tasks for the day.

Second Scenario: Antoine, a Business Administration major in Ashesi University wants to build an Android application for a business he is running. Unfortunately, the semester is almost getting to and end and most of his friends who can quickly build the application for him are seriously studying for examinations and working on their final projects. This means that he has no option but to build the application himself. He signs onto the code generator platform and quickly starts building his application through drag and drop. In a few minutes, Antoine has built his application and successfully uploaded the application to the Play store.

2.1.3 Use Case Diagram

The use case diagram in figure 2.1 was generated using the data collected from the interviews. The diagram outlines each user's usage of the application and will aid in measuring the project's completion.

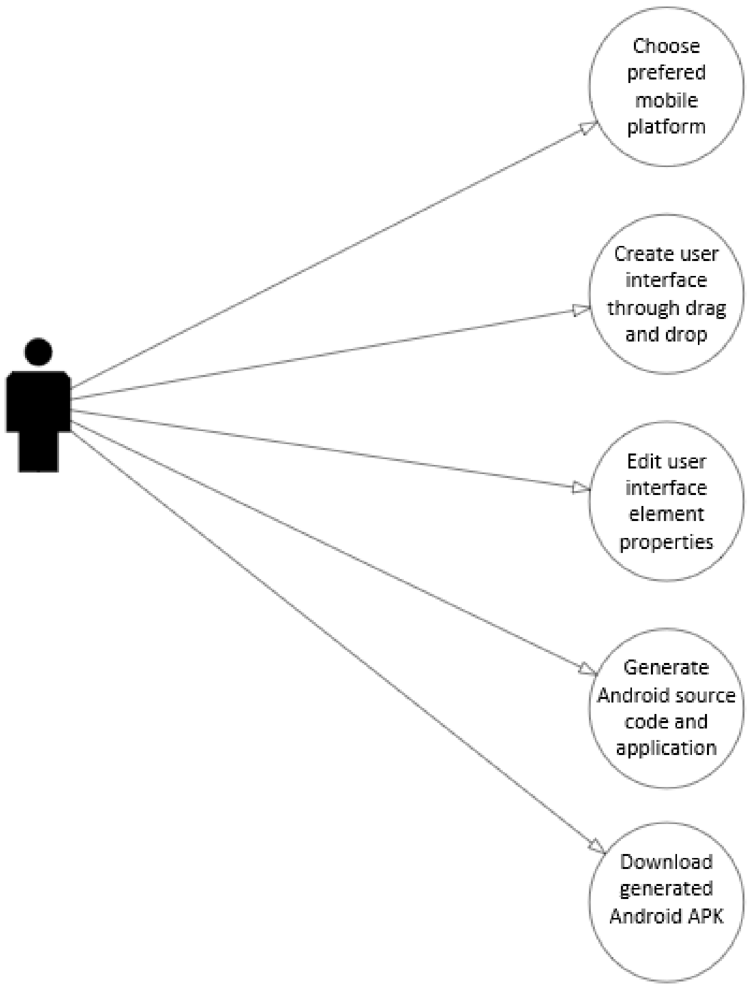

Choose prefered mobile platform
Create user interface through drag and drop
Edit user interface element properties
Generate Android source code and application
Download generated Android APK

Figure 2.1 Use case diagram.

2.1.4 Functional requirements

Figure 2.2 outlines the step-by-step process involved in the operation of the code generator. The various actions in the diagram influenced the functional requirements that have been outlined below.

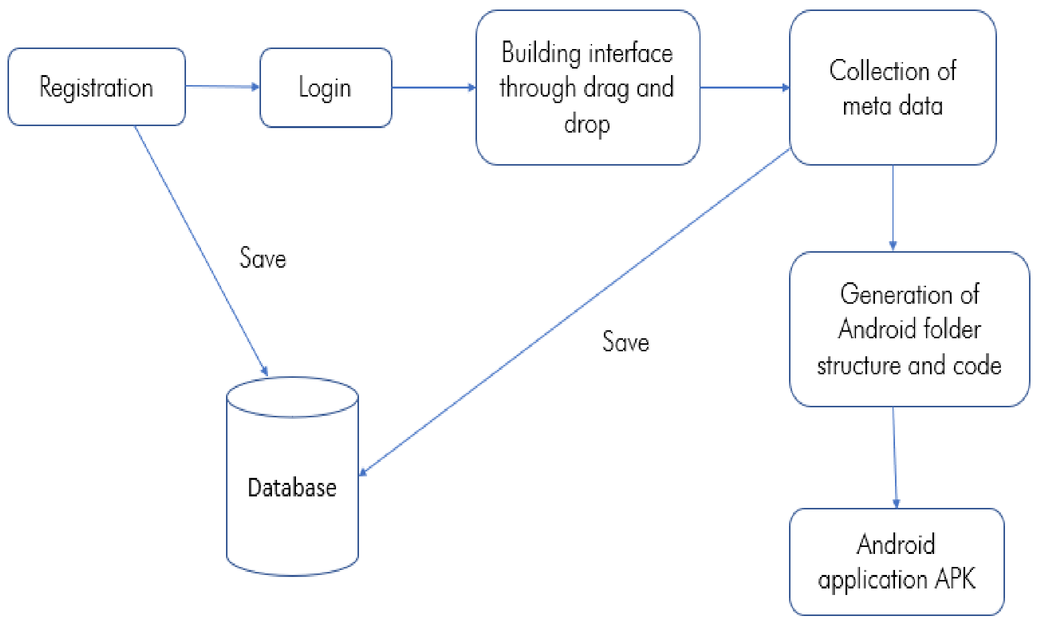

Registration
Login
Building interface through drag and drop
Collection of meta data
Save
Save
Generation of Android folder structure and code
Database
Android application APK

Figure 2.2 Code generator architecture.

2.1.4.1 User registration and log in

A user should be able to register and log onto the platform. This will help him/her get a personalized feel of the application and grant him/her the ability to view older projects that he/she has completed.

2.1.4.2 Platform selection

A user of the code generator should have the ability to select which mobile platform he/she wants to build his/her application for. The available options should cover the major mobile platforms which include but not limited to Android, iOS and Windows.

2.1.4.3 Drag and Drop Interface

To create his/her desired application, a user should be given the luxury of dragging and dropping the various UI elements unto a mobile-like interface of the code generator. This makes the process faster, less prone to errors and he/she does not need to memorize any long lines of syntax of a programming language to build the application.

2.1.4.4 User Interface Elements Edits

A user of the code generator should have the ability to edit the details of the various user interface elements that he/she used in building the application. This involves specifying his/her own activity names, specifying the name, labels and ID given to each element used in the creation of the application. However, the application should work even without editing the details of the various user interface elements.

2.1.4.5 Application Creation

After a user has designed his/her application through drag and drop, the metadata created from each activity should be used to generate the Android folder structure, XML files for the Android application's layout and the Android application's Java code.

2.1.4.6 Application Compilation

The generated Android source code should be compiled into an APK that can be installed on a mobile device or an emulator and uploaded to the Google Play store if need be.

2.1.4.7 Application APK Download

A user should have the ability to download the APK generated. Alternatively, for multiple downloads, a user should be sent a link where he/she can download the APK anytime.

2.1.4.8 Past Projects Retrieval

The code generator should give a user the ability to retrieve past projects. This involves searching through the database and retrieving the projects that correspond to the ID of the logged in user.

2.1.5 Non-Functional Requirements

2.1.5.1 Product

1. The user should be able to use the code generator to design an Android interface.
2. The code generator should be able to package the user's application as an Android APK.

2.1.5.2 External

1. The code generated by the code generator should be secure enough so that users can confidently move the application from development to production mode.

2.2 System Requirements

The section below is a description of what the system requires for the functional requirements to be met.

2.2.1 User interface

1. The user interface of the code generator should be friendly, clean and intuitive so a user can easily build his/her desired application.
2. The user interface of the code generator must have all the components required in developing an Android application
3. The user must have a computer and an active internet connection to use the application.

Chapter 3: Architecture and Design

3.1 Project Overview

The aim of this project is to create a web application that will help users with little to no technical background in computer programming quickly create applications that can be installed on a phone or uploaded to the Google Play store. This is done by providing a nice and intuitive interface where users only need to drag and drop the various UI elements that make up their Android application onto a mobile interface provided by the code generator. With a button click, an APK file that can be installed on a phone or uploaded to the Google Play store is generated.

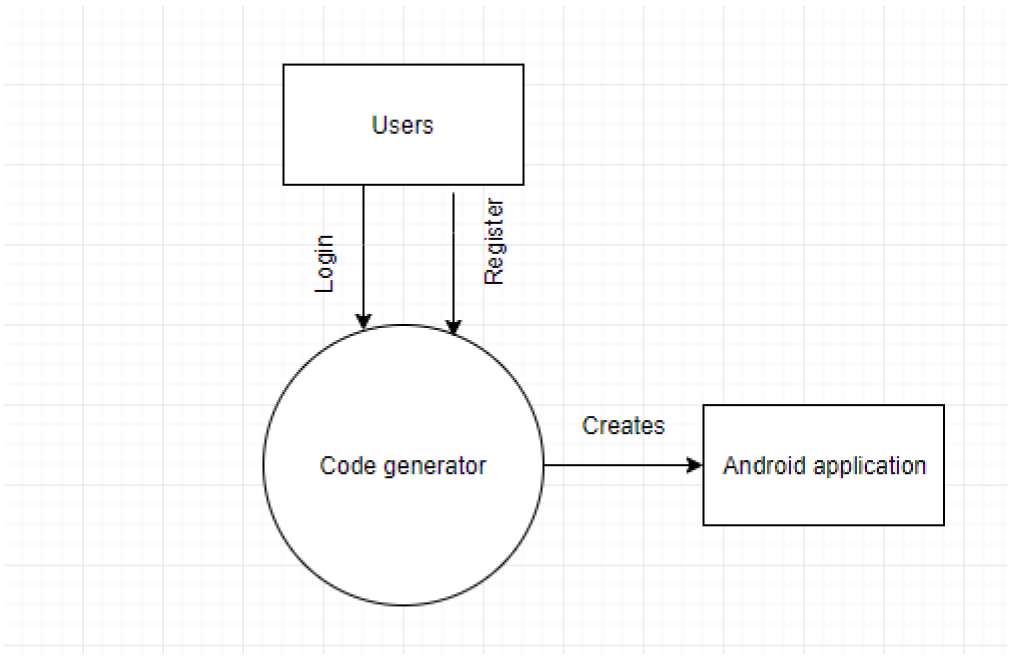


Figure 3.1 Context diagram

3.2 Software Process Model

To complete the project, the agile process of software development was adopted. In this method, “increments are small and, typically new releases of the system involve customers in the development process to get rapid feedback on changing requirements”(Sommervile, 2011). The steps involved in the agile process are outlined in figure 3.1. The main advantages of this software process model are as follows:

1. New features are quickly delivered and frequently.
2. It is easy to integrate changes anytime in the project.
3. The users are the focus of the applications created.
4. Breakdown of the project into manageable units ensure the product is of high quality

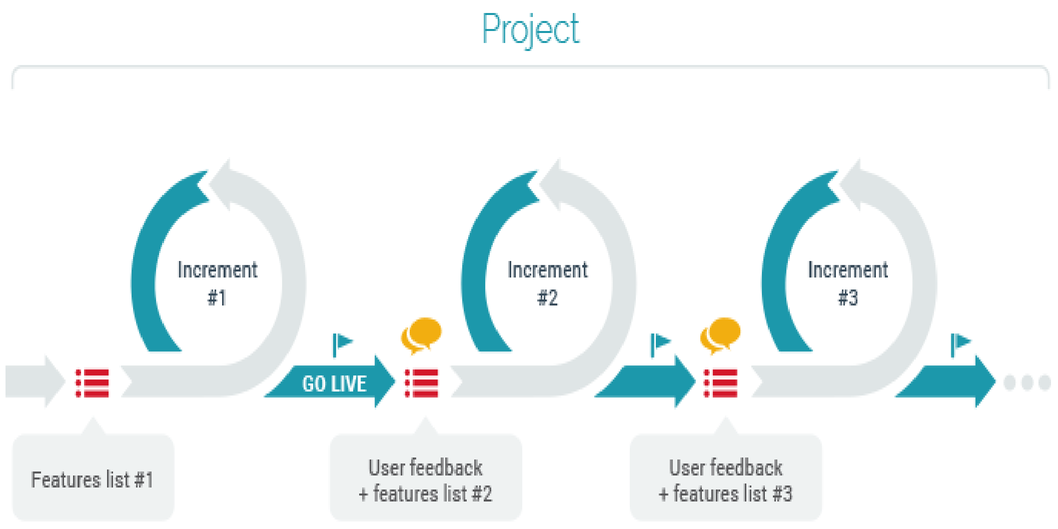


Figure 3.2 Agile software process model.

3.3 Project Modules

For the code generator to function effectively, the different tasks it performs are broken down into different modules. The modules that make up the code generator are presented below:

- User module
- Android Application Creation Module

- Folder generation module
- Android manifest code generation module
- Java code generation module
- XML code generation module
- Values resources generation module

Layered Architecture

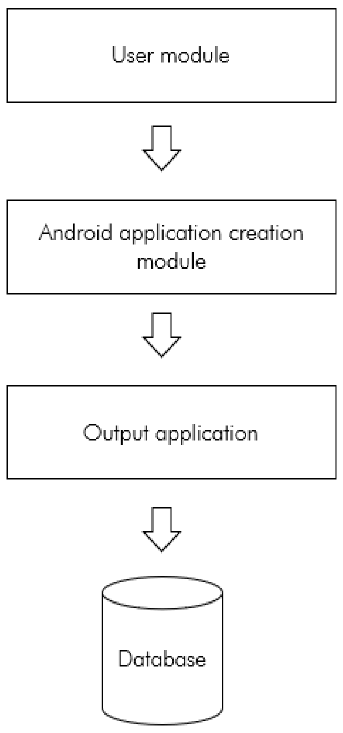


Figure 3.3 Layered architecture.

Figure 3.3 is a layered architecture of the code generator. The first layer, the user module, is where all the interactions between the user and the code generator take place. This module contains user registration, login and project specification, the page where the user provides details about the application to be built. The details required at the project specification page are the application name, description and the platform of choice, be it Android, iOS or Windows. The last page in this module is the builder page. This is where the dragging and dropping of user interface elements is done. This module also has a database where information about users and their projects are stored. Figure 3.3 is a diagrammatic representation of the code generator database schema.

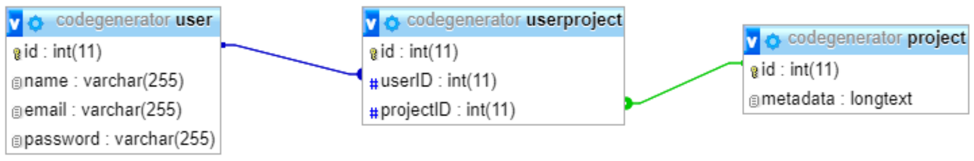
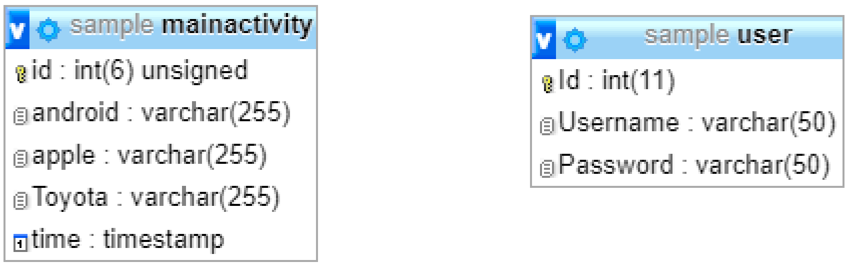


Figure 3.4 Code generator database schema.

The application creation module is where the dragged and dropped user interface elements are translated into Android code. In this module, the UI elements are extracted from the code generator and passed to the backend in the form of JSON for an Android application to be built. This module starts by generating the appropriate folder structures followed by the Android source code. Automatic compilation of the source code into an APK is done by running a gradle script on the terminal.

The generated Android APK with its source code can be downloaded by a user for further tweaking or installation on a phone. Once the generated Android application is launched, it initiates the last layer by creating a database unique to each generated application. The generated application name is assigned as the database name while the tables are the names of the activities in the application. The attributes of each table are the UI elements that are editable such as edit texts and combo boxes, just to mention a few . In the diagram below, the database was generated for an application called Sample, that had two activities, mainactivity and user. The user table had username and password as attributes and these were dynamically generated from edit text elements in Android. The schema for the Sample Android project is described in figure 3.4.



sample mainactivity
- id : int(6) unsigned
- android : varchar(255)
- apple : varchar(255)
- Toyota : varchar(255)
- time : timestamp

sample user
- Id : int(11)
- Username : varchar(50)
- Password : varchar(50)

Figure 3.5 Sample generated Android application database schema.

3.3.1 User Module

The user of the code generator interacts directly with the user module. Using this module, the user can build the user interface for their desired Android application. The users build the user interface using the drag and drop feature provided by the user interface module. The user module is built following an MVC web architecture. The various aspects of the MVC architecture are described in figure 3.6.

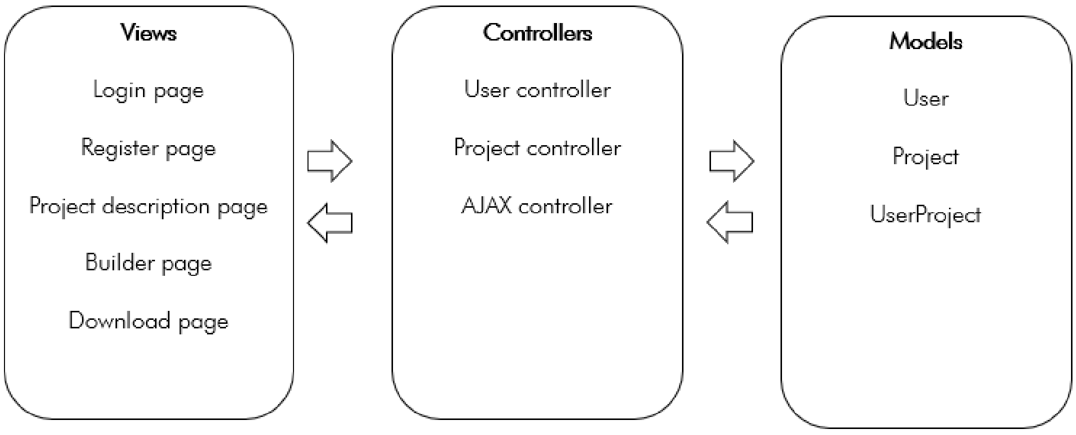


Figure 3.6 MVC architecture for user module.

3.3.1.1 Model

The model contains all the files that describe the data of the application to be created by the user. The models of the user module are described below:

3.3.1.1.1 User

The user model specifies the details of a user that is registered or logged onto the code generator. The attributes of the user are as follows:

a) ID: An autogenerated identification number assigned to the user at the point of registration.
b) Email: The user's unique identifier. Example, likudie@gmail.com
c) Password: An encrypted string that is required during the login.

3.3.1.1.2 Project

The project model specifies the details of a project that is created using the code generator. The attributes of the project are as follows:

a) ID: An autogenerated identification number assigned to a project upon creation.
b) Metadata: A string representation of the metadata collected from the application upon creation.

3.3.1.1.3 UserProject

The UserProject model specifies the details of a project in relation to its user. Below are attributes of the UserProject model.

a) ID: An autogenerated identification number assigned to a user's project upon creation.
b) UserID: The ID of a user to whom a project belongs.
c) ProjectID: The ID of a project that is being associated to a user.

3.3.1.2 View

The view contains all the files that define the user interface of the code generator. Below are the various screens that make up the code generator:

3.3.1.2.1 Registration page

On this page, a user will provide his/her name, email address and password to be registered on the code generator.

3.3.1.2.2 Login page

On this page, a user will provide his/her email address and password to be given access to the full functionalities of the code generator.

3.3.1.2.3 Project description page

On this page, a user will give brief descriptions about the project that he/she is about to create. This includes specifying the application name, a succinct description about the project and then the platform for which the application should be built for.

3.3.1.2.4 Builder page

This is the page where the dragging and dropping of UI elements take place. A user can add more screens and edit element properties.

3.3.1.2.5 Download page

This page has a button which when clicked will download a zipped folder containing the source code and the APK file of the generated application.

3.3.1.3 Controller

3.3.1.3.1 User Controller

This controller is responsible for the communication between the user model and the user view (registration and login)

3.3.1.3.1 Project Controller

This controller is responsible for deserializing the JSON containing the metadata of the application to be built. It is also responsible for issuing commands that generate the folder structure and the source code.

3.3.1.3.1 AJAX Controller

The AJAX controller enables communication between the front-end code and the backend.

Figure 3.7 is an activity diagram which describes the various dynamic aspects of the code generator.

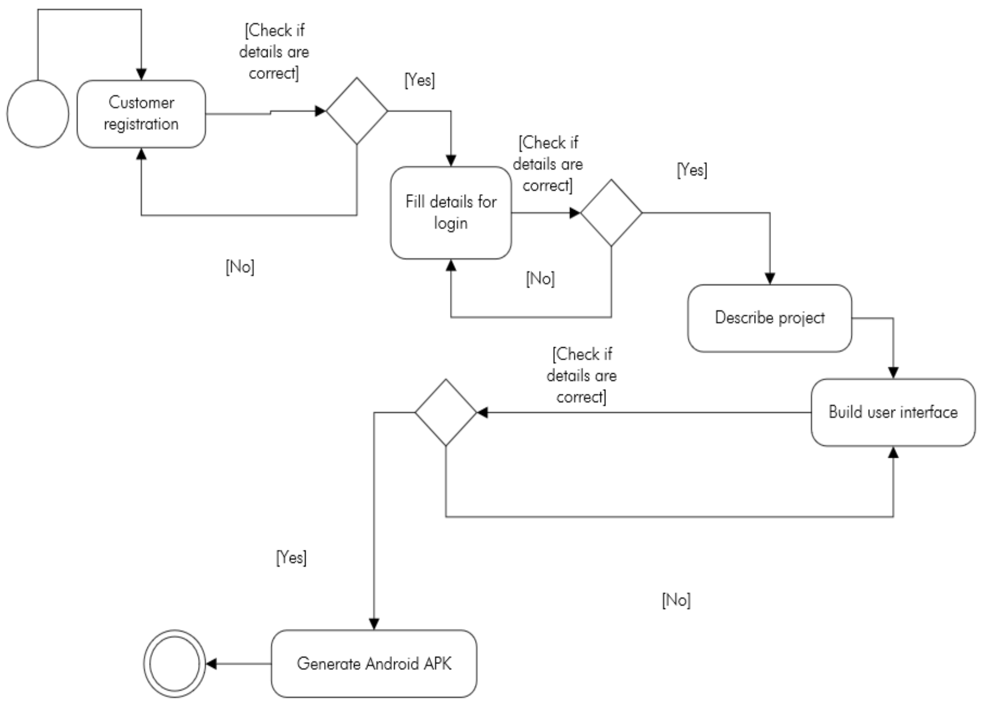


Figure 3.7 Activity diagram.

3.3.2 Android Application Creation Module

Creating an Android application without Android Studio or Eclipse involves replicating the Android folder structure, writing the XML and Java code. Replicating the folder structure involves creating the lib and src folders which will hold the various libraries and source codes respectively. The values folder contains files that specify the colour and strings to be used throughout the Android application. The next stage is generating the layout files which define the look and feel of the Android application to be created. The logic and backend of the Android application is handled by the Java code. The manifest file contains information about the application being created. In the manifest file, various permissions and dependencies are added to ensure that the Android application runs smoothly.

The Android application creation module will contain most of the logic of the code generator as illustrated in figure 3.8. To achieve abstraction, this module is broken down into five submodules. The role of this module is to create the Android application that was designed using the user module of the code generator. The functions of the submodules of the Android Application Creation Module are outlined below:

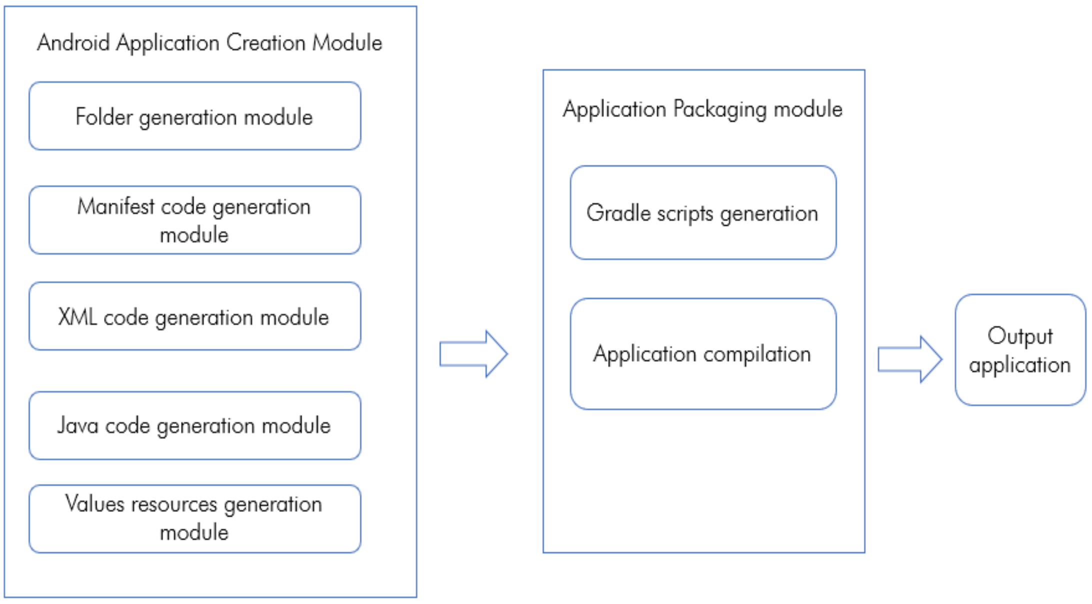

Android Application Creation Module
Folder generation module
Manifest code generation module
XML code generation module
Java code generation module
Values resources generation module
Application Packaging module
Gradle scripts generation
Application compilation
Output application

Figure 3.8 Code generator modules.

3.3.2.1 Folder Generation Module

The folder generation module is responsible for generating the Android folder structure. This includes generating the src, libs and all subfolders.

3.3.2.2 Values Resources Generation Module

This module is responsible for generating all files and code in the values folder of a generated Android project. This includes specifying the application theme, strings and colours that define the user interface of a generated Android application.

3.3.2.3 Manifest Code Generation Module

The manifest code generation module is responsible for generating the code in the Android manifest file. This module specifies the various permissions and activities that make up the Android project.

3.3.2.4 XML Code Generation Module

The XML code generation module is responsible for generating all the layout files that make up the user interface of a generated Android project. The tasks of this module include specifying the details of all the UI elements that make up the various activities in the application to be created.

3.3.2.5 Java Code Generation Module

The Java code generation module is responsible for generating the code that makes up the logic of the application. This module is responsible for initializing UI elements and specifying the on click behaviours of the various elements among many other functions.

Chapter 4: Implementation

4.1 Overview

In this section, the processes involved in implementing the code generator are described in detail. The chapter is divided into sections that describe the implementation process of the code generator web application, creation of Android code, generation of the Android APK and the code generator database.

4.2 Implementation Process

4.2.1 Code Generator Web Application

The code generator is a web application that can be accessed from any location when connected to the internet through a web browser. By drag and drop, a user can design and build a complete android application ready for production using this platform. A sample project that can be built using the code generator is a traffic offence collection application.

4.2.1.1 Registration

A user must be registered onto the code generator platform to gain access to its functionality. Registration involves providing a username, email and a password, which is hashed before saved into the database. Registering enhances an effective retrieval of a user's past projects. For instance, if a user wants to edit a past project, he/she does not have to start from scratch.

4.2.1.2 Login

The code generator requires users to be authenticated before starting any project. As such, an email and a password are required of a user during login. Once these details are verified to be correct, the authenticated user is allowed access to the full functionality of the code generator.

4.2.1.3 Project Specification

The project specification page of the code generator shown in figure 4.1, requires a user to provide a name, description and choose the platform of choice for the application he/she wishes to build. The application name and description provided by a user are included in the metadata for generating the Android application. Currently, the available options for the desired mobile platforms are Android, iOS and Windows, although only Android has been implemented.

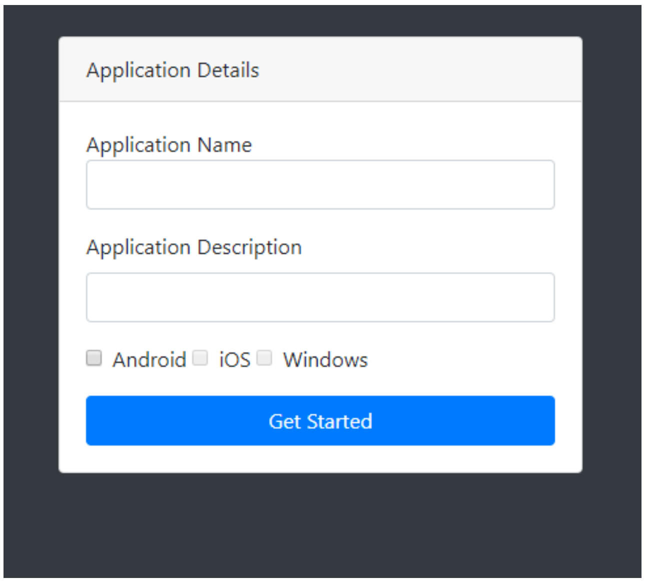


Figure 4.1 Project specification page.

4.2.1.4 Application Design

On the application design page, a user designs the interface of the application that he/she wishes to build. The left sidebar contains elements used in building an Android mobile interface. Currently, the available UI elements are label, textbox, radio button, spinner, checkbox and a text area. The main section of the application has a single screen resembling a mobile interface with two buttons underneath. The button labelled "Add New

Screen” adds an extra mobile interface to the default one in instances where the user wants to build multiple screens.

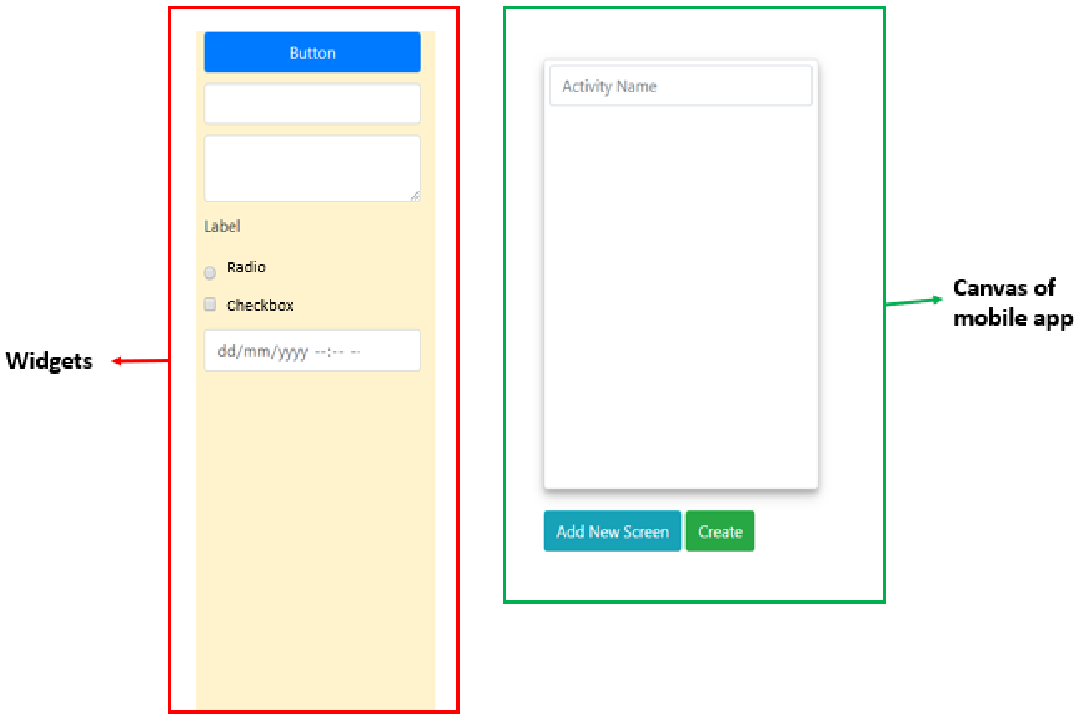


Figure 4.2 Builder page.

4.2.1.4.1 Drag and Drop

With the use of a mouse, a user drags the various elements from the sidebar unto the phone screen. A unique ID is automatically generated and assigned to every dropped element to avoid elements being assigned the same ID. This step is needed to avoid the generated Android application failing during the build stage.

4.2.1.4.2 Edit UI Element Properties

A user is given the ability to edit the default values of each UI element. Figure 4.3 is a modal that pops up when a user clicks on a UI element to be edited. The editable values include the id, name, the display text and if the element calls another activity in the application, the name of the activity.

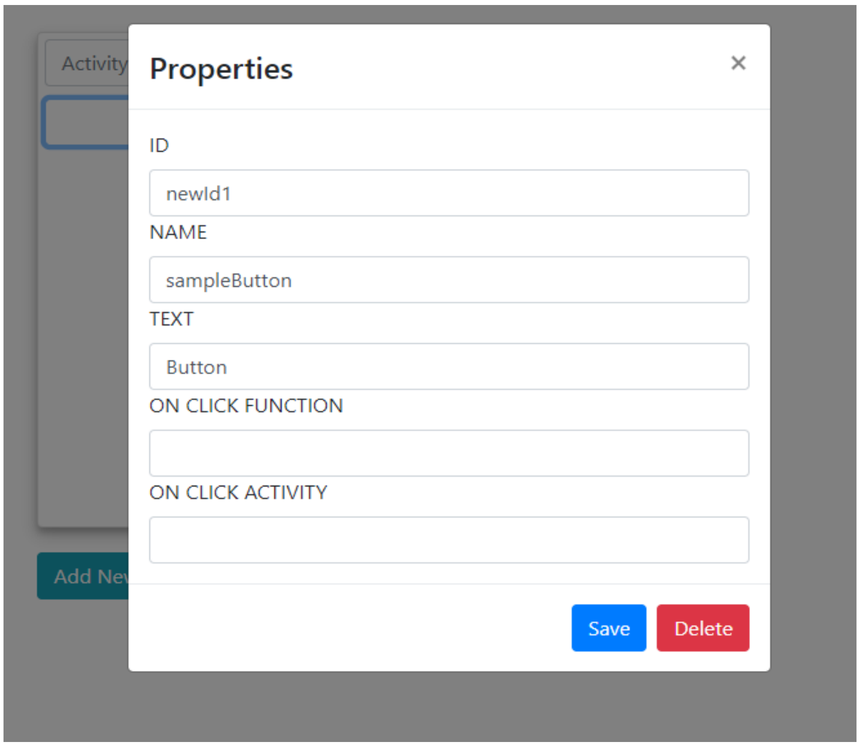


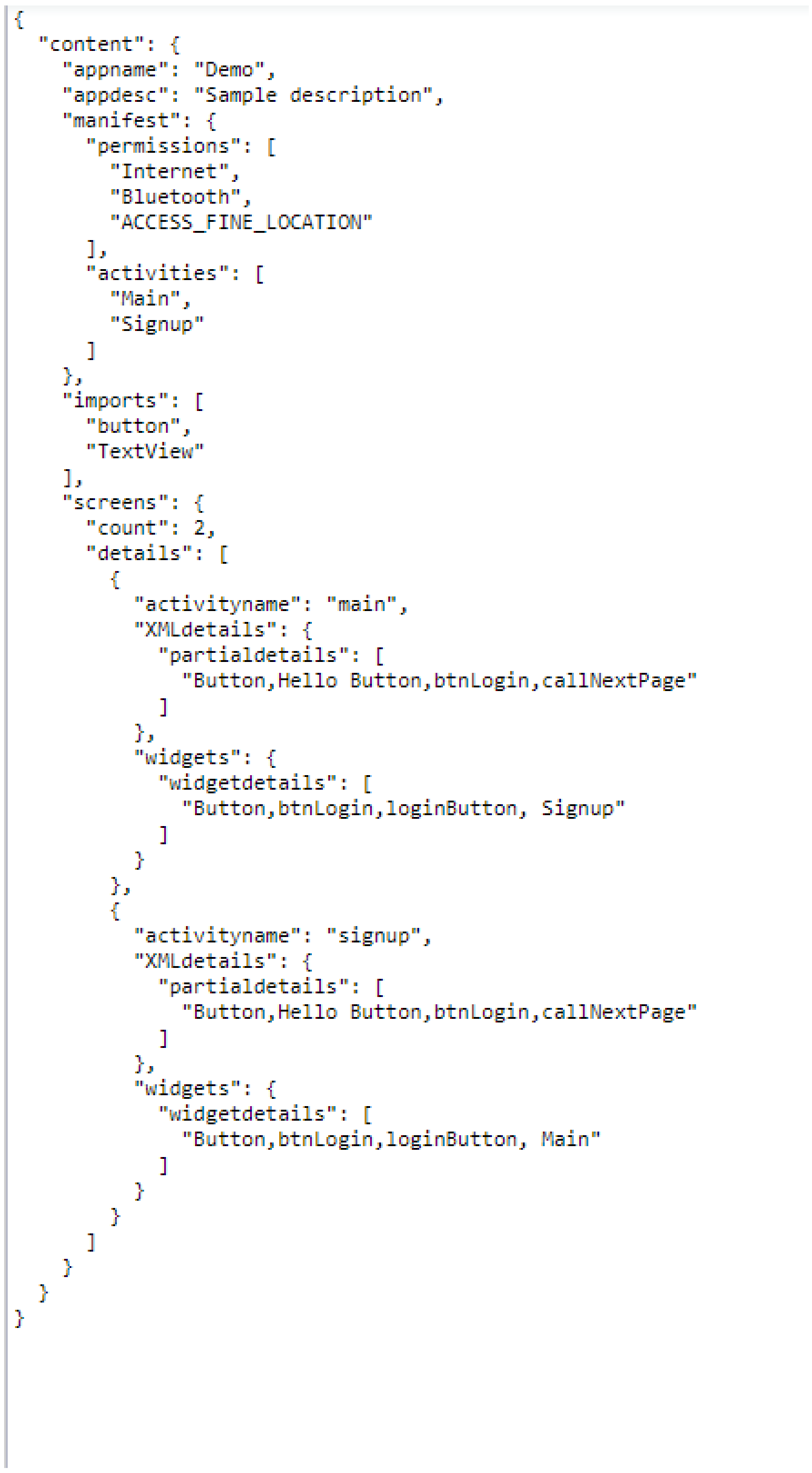
Figure 4.3 Edit UI element properties modal.

4.2.1.4.3 Adding More Screens

The code generator provides a user the ability to add extra screens to the default one. More screens can be added by clicking on the “Add New Screen” button.

4.2.1.4.3 Metadata Extraction

In generating the Android application, the HTML elements on the web application must be extracted and translated into actual Android UI elements. The extraction of metadata is done by counting the number of screens a user has built. Using this count and the unique IDs of each screen, the HTML elements are extracted and converted into JSON to be saved in the code generator database. The metadata is passed to the backend and translated into Android UI elements. Figure 4.5 is a sample metadata formatted as JSON for an Android application to be generated.



```
{
  "content": {
    "appname": "Demo",
    "appdesc": "Sample description",
    "manifest": {
      "permissions": [
        "Internet",
        "Bluetooth",
        "ACCESS_FINE_LOCATION"
      ],
      "activities": [
        "Main",
        "Signup"
      ]
    },
    "imports": [
      "button",
      "TextView"
    ],
    "screens": {
      "count": 2,
      "details": [
        {
          "activityname": "main",
          "XMLdetails": {
            "partialdetails": [
              "Button,Hello Button,btnLogin,callNextPage"
            ]
          },
          "widgets": {
            "widgetdetails": [
              "Button,btnLogin,loginButton, Signup"
            ]
          }
        },
        {
          "activityname": "signup",
          "XMLdetails": {
            "partialdetails": [
              "Button,Hello Button,btnLogin,callNextPage"
            ]
          },
          "widgets": {
            "widgetdetails": [
              "Button,btnLogin,loginButton, Main"
            ]
          }
        }
      ]
    }
  }
}
```

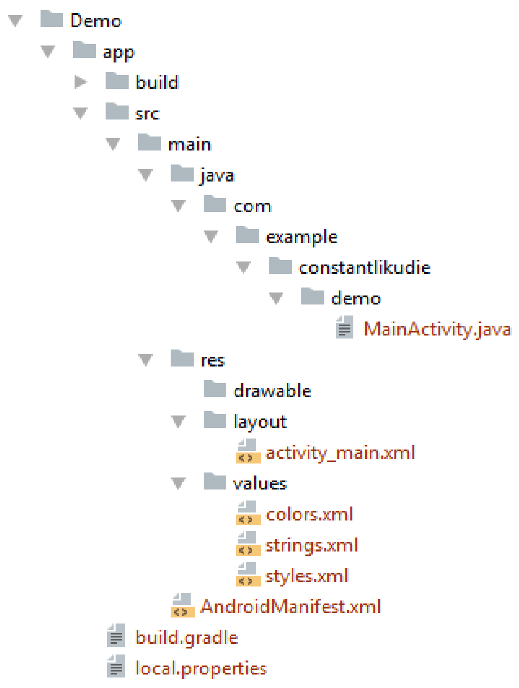
Figure 4.5 Metadata structure.

4.2.2 Android Code Generation

The JSON passed to the backend is deserialized to generate the Android folder structure and code. The steps involved in the creating the Android source code are:

1. Generating Android folder structure
2. Generating Android manifest file
3. Generating layout XML files
4. Generating Java code

The Android folder specifies where all files go. For instance, the Java codes are in the src folder while the layout XML files are in the res folder. Replicating the Android folder structure enhances an easy location of files by developers who wish to customize the generated code. Figure 4.6 is a screenshot of the folder structure automatically generated by the code generator for a sample Android project.



```
▼ Demo
  ▼ app
    ▶ build
    ▼ src
      ▼ main
        ▼ java
          ▼ com
            ▼ example
              ▼ constantlikudie
                ▼ demo
                    MainActivity.java
        ▼ res
            drawable
          ▼ layout
              activity_main.xml
          ▼ values
              colors.xml
              strings.xml
              styles.xml
          AndroidManifest.xml
    build.gradle
    local.properties
```

Figure 4.6 Generated application folder structure.

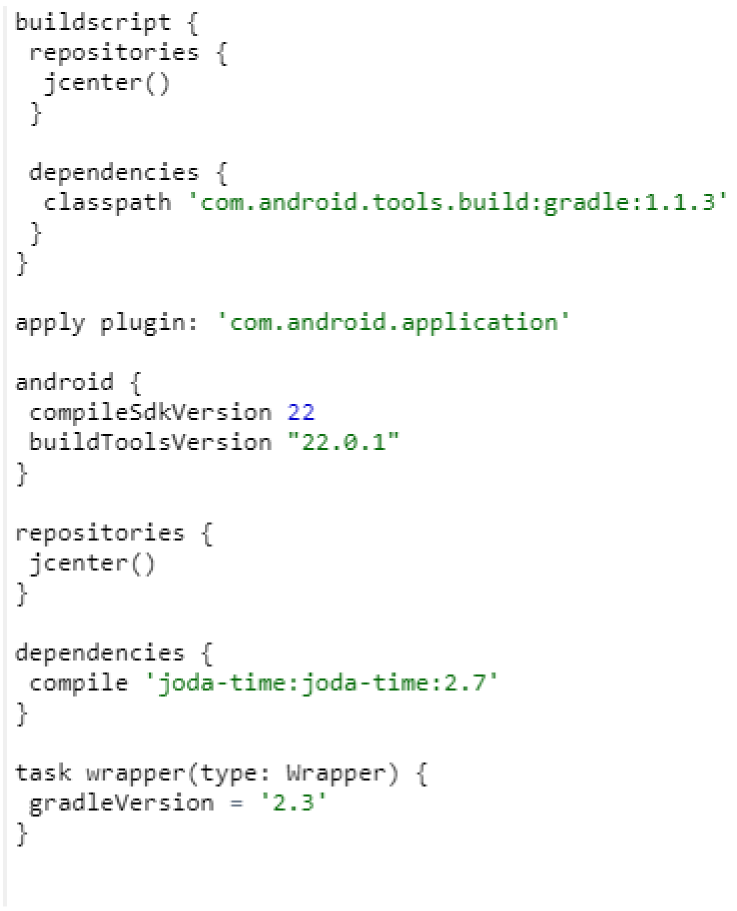
The manifest file contains various permissions that the generated Android application requires to execute. It also contains a list of all activities that make up the application and libraries that will help the Android code execute.

The layout XML file specifies the user interface of the generated Android application. It also specifies the orientation and position of each element making up the application.

The Java code contains the logic of the application. All computational tasks are done here.

4.2.3 Android APK Generation

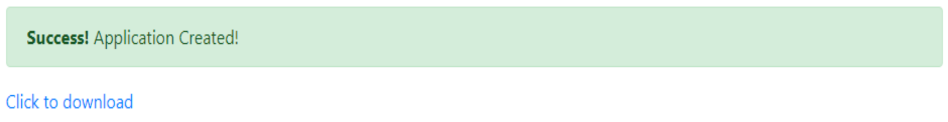
To build an Android APK, gradle.build and local.properties files are added to the generated Android source code. The local.properties file contains the path to the Android sdk while the gradle.build is “based on a Domain Specific Language that supports the automatic download and configuration of dependencies or other libraries”(Vogel & Scholz, 2016). Running gradle build in the terminal downloads all the necessary libraries needed by the Android application and chooses a target sdk for which the application should be built. Running this command also generates an APK file that can be installed on a device or an emulator. Figure 4.7 is a screenshot of a sample gradle script automatically generated by the code generator. It is required by each Android source code to generate an APK.



```
buildscript {
 repositories {
  jcenter()
 }

 dependencies {
  classpath 'com.android.tools.build:gradle:1.1.3'
 }
}

apply plugin: 'com.android.application'

android {
 compileSdkVersion 22
 buildToolsVersion "22.0.1"
}

repositories {
 jcenter()
}

dependencies {
 compile 'joda-time:joda-time:2.7'
}

task wrapper(type: Wrapper) {
 gradleVersion = '2.3'
}
```

Figure 4.7 Gradle file structure.

A user is redirected to the download page on a successful Android application build. When “Click to download” in figure 4.8 is clicked, a zipped folder containing the Android source code and APK is downloaded onto a user’s device.



Success! Application Created!

Click to download

Figure 4.8 Application creation success.

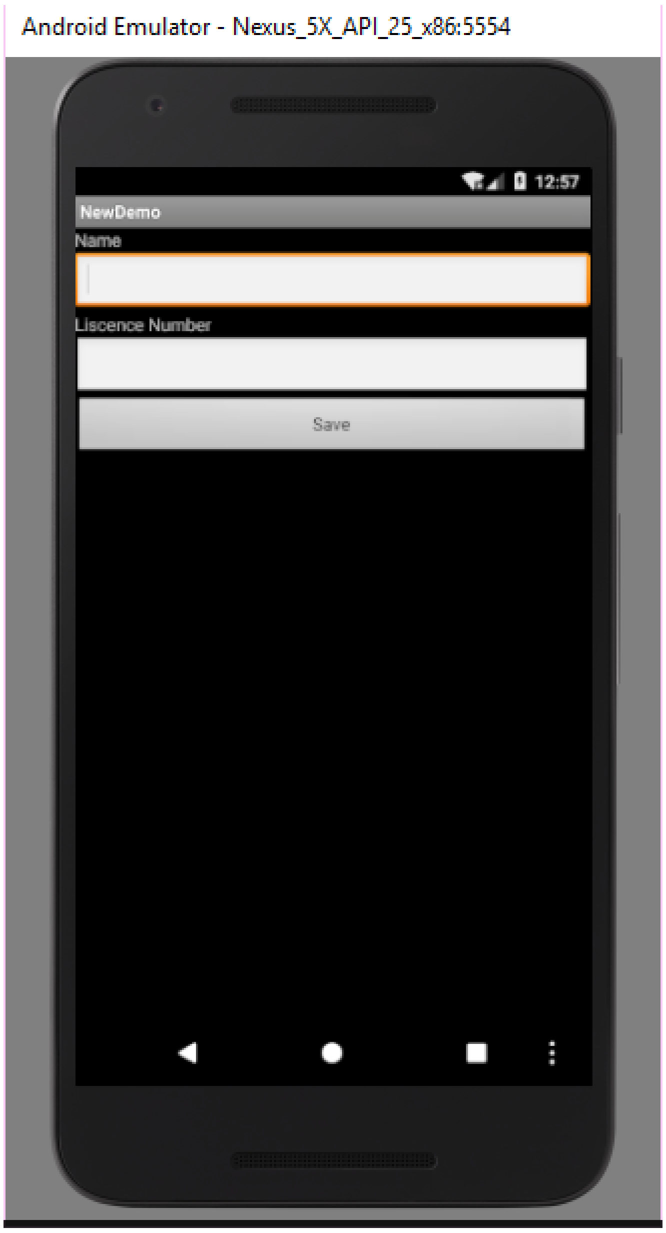


Figure 4.9 Sample generated application.

Figure 4.9 is a screenshot of an application generated by the code generator. The APK in the downloaded zipped folder was installed on an Android emulator and tested.

The sample Android application in figure 4.9 above was generated using two labels, two textboxes and a button. The labels were renamed from their default names to name and license number as illustrated in the output above. The default name of the button was also changed to Save. This was done via the builder page of the code generator. Additional controls for building applications using the code generator include radio buttons, checkboxes, a spinner and datetime widget.

4.2.4 Code Generator Database

The code generator database stores the details of a user and the projects he/she builds. It is made up of three tables; the user table, the project table and the userProject table. ID, name and password are the attributes of the user while the project table has ID and metadata as attributes. The userProject serves as the link table allowing inter-operation between the user and the project tables has ID, userID and projectID as its attributes.

4.3 Implementation Tools

The front end of the application was built using Hypertext Mark-up Language (HTML) and Bootstrap. HTML is a standard mark-up language for creating webpages and applications. Bootstrap was used to style the frontend of the code generator. The various interactions and validations on the various pages including the drag and drop and cloning features were all implemented using JavaScript and jQuery. Cloning was necessary because the UI elements had to be dragged and dropped multiple times in order to build the user interface of the Android application to be generated. As such, the duplication of UI elements was achieved using JavaScript cloning. jQuery is a JavaScript framework that simplifies writing code in JavaScript by using concise methods. Hypertext Preprocessor is a server-side scripting language that handles all the backend logic of the code generator. Java and XML were the main languages with which the Android application was built and the gradle.build file was written in Groovy, a language based on the Java.

Chapter 5: Testing

5.1 Overview

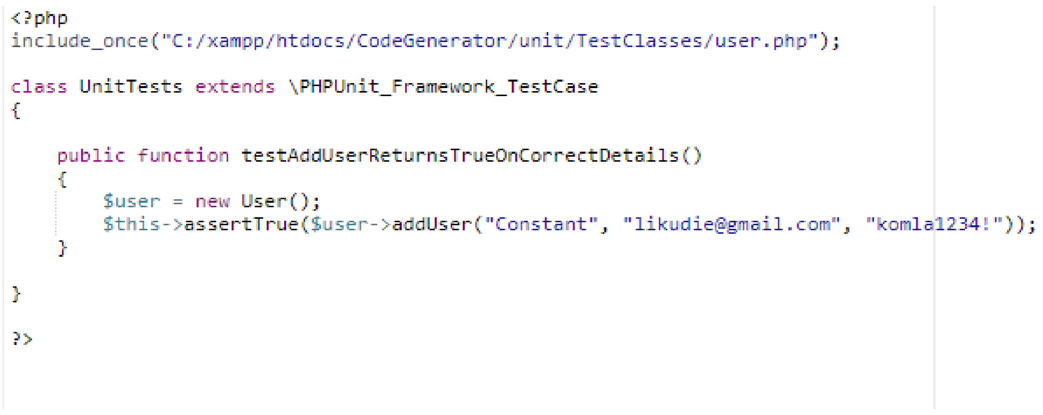
Tests were conducted during the development of the code generator. These tests were conducted to ensure and validate that the requirement specification of the project has been met. The tests covered in this chapter are component, unit and system testing. Unit tests are conducted to ensure that individual functions of the code generator work as expected whereas in component testing, each component is tested to ensure all components work as expected. System testing was done on the complete code generator with all parts integrated. This section presents the test cases, results and analysis.

5.2 Development Testing

This section of the chapter focuses on the tests that were conducted by the developer of the Code Generator during development.

5.2.1 Unit Testing

Unit tests are conducted to ensure that individual functions perform the tasks that they are supposed to and return the appropriate results. PHPUnit was used for all unit tests discussed in this chapter. Figure 5.1 is a screenshot of the registration test.



```
<?php
include_once("C:/xampp/htdocs/CodeGenerator/unit/TestClasses/user.php");

class UnitTests extends \PHPUnit_Framework_TestCase
{

    public function testAddUserReturnsTrueOnCorrectDetails()
    {
        $user = new User();
        $this->assertTrue($user->addUser("Constant", "likudie@gmail.com", "komla1234!"));
    }

}

?>
```

Figure 5.1 Sample PHP unit test code.

5.2.1.1 Testing Registration

Password Match Test

This test was done to verify the password provided in the password field matches; the password in the confirm password field. This test yielded the desired output as different passwords returned an error message. This test was done to make sure that the intended password of the user is saved in the database and not one that was entered in error.

Number	Test	Response	Status
1	Different password and confirm password values	Passwords do not match	Fail
2	Password and confirm password values left blank	Please provide password values	Fail
3	Same password and confirm password values	Passwords match	Success

Existing Email

In this test, the developer tried registering with an existing email. The test returned an error since the email of a user is unique and can only be used by an individual. The test yielded the desired result of returning an error message. A new email that hasn't been previously registered was used in registration. This process succeeded.

Number	Test	Response	Status
1	Existing email	Email already exists	Fail
2	New email	Registration successful	Success

5.2.1.2 Testing Login

Unregistered User

In this test, an email and password that had not being registered was used to login into the Code Generator. The login failed returning an error message indicating the email was not registered.

Wrong Password

In this test, a valid email was provided but with an incorrect password to log in to the Code Generator. The login failed returning an error message validating that an incorrect password with a correct email combination could not be used to log into the system.

Number	Test	Response	Status
1	Non-existent email	Email does not exist	Fail
2	Non-existent email and password	Email and password combination not found	Fail
3	Correct email and password	Login successful	Success

5.2.1.2 Testing Project Specification

In this test, the developer started building an application using the code generator without specifying the name, description and the platform for the desired application. The developer was not routed to the builder page as the he/she was asked to provide the missing information about the application.

Number	Test	Response	Status
1	No project name	Provide project name	Fail
2	No project description	Provide project description	Fail
3	No platform selected	Please choose a platform	Fail

4	Project name, description and platform specified	Redirected to builder page	Success

5.2.1.3 Testing Project Specification

Incomplete Screen Details

In this test, some details needed by an activity were omitted. For instance, the activity name was not filled, and the user clicked on "Create". The code generator responded appropriately by asking the user to fill in the name of the activity before proceeding.

Same IDs

In this test, two elements on the same screen or activity were given the same ID and the user clicked on the "Create" button. The code generator responded by asking the user to change one of the IDs.

5.2 Component Testing

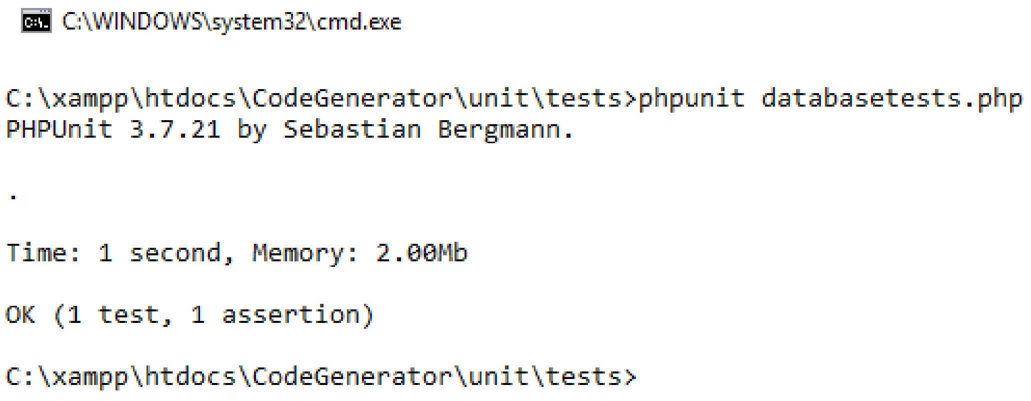
This section describes the tests that were done to ensure the various components of the code generator worked as desired.

Testing Android Folder Generation Module

This test was done to make sure that the folder generation module of the Code Generator created the appropriate Android folders. This test was done by providing the Code Generator with the name of the application to be created. This yielded the desired results as the folder structure was created using the application name provided.

Testing Android Code Generation Module

This test was done to ensure that given the appropriate description of all the screen elements, an appropriate Android code was generated. This was done by passing a JSON containing the description of the application to the code generation module. This test yielded the appropriate result as the appropriate Android code was created based on the description in the JSON.



```
C:\WINDOWS\system32\cmd.exe

C:\xampp\htdocs\CodeGenerator\unit\tests>phpunit databasetests.php
PHPUnit 3.7.21 by Sebastian Bergmann.

.

Time: 1 second, Memory: 2.00Mb

OK (1 test, 1 assertion)

C:\xampp\htdocs\CodeGenerator\unit\tests>
```

Figure 5.2 PHP unit test result.

5.3 System Testing

System test was performed on the fully functional code generator platform. In this test, all the various components and modules of the code generator were integrated to enable communication between them. The test yielded varying results as sometimes the data passed from one component to the other was slightly modified or was not in the appropriate format as per the design of the recipient module.

Chapter 6: Conclusion and Recommendation

This paper describes how a code generator for building Android applications through drag and drop was implemented. With this application, both software programmers and novices will build Android applications without having to write any code. The process of creating the application using this code generator is also very intuitive and simple. Currently, the code generator meets the following functional requirements:

1. A user should be able to sign onto the code generator platform.
2. A user should be able to log into the code generator platform.
3. A user of the application can select his/her preferred mobile platform.
4. A user can drag and drop the various UI elements onto a phone screen interface, so he/she can visualize the application as he/she builds along.
5. The extracted metadata is saved into a database.

To improve this project and make it more effective, the following tasks are outlined as future works:

1. Ability to do real time code edit: Although this project seeks to eliminate writing of code completely, some experienced developers would like to further tweak the code to perform more complicated tasks. The current setup is that such users will have to download the project folder before loading it in their IDE of choice to make edits. To eliminate this, users can be given the ability to edit the codebase in the code generator.
2. Addition of more UI elements: The current implementation of the code generator has just a few UI elements. To make the application more useful, more elements must be added to what is currently present. The current elements available on the code generator tool are textview, button, radio button, spinner and checkbox.

3. Addition of sample projects: A user should be able to build upon a sample project. This will reduce development time as the user will not be required to build the entire application by drag and drop.
4. Building for other mobile platforms: The current implementation of the code generator supports only Android code generation. To make the code generator more useful, more mobile platforms such as Windows, Cross platform and iOS can be added.

In conclusion, this project has an enormous potential to be the code generator of choice among programmers and novices worldwide and will no doubt be a pain reliever to all programmers across the world.

References

Barnett, S., Vasa, R., & Grundy, J. (2015). Bootstrapping Mobile App Development. *Proceedings - International Conference on Software Engineering*, *2*, 657–660. https://doi.org/10.1109/ICSE.2015.216

Franky, M. C., & Pavlich-mariscal, J. A. (2012). Improving implementation of code generators: Regular-Expression Approach.

Mbogo, C., Blake, E., & Suleman, H. (2013). A mobile scaffolding application to support novice learners of computer programming. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/2517899.2517941

Schlee, M., & Vanderdonckt, J. (2004). Generative Programming of Graphical User Interfaces. *Proceedings of the Working Conference on Advanced Visual Interfaces AVI 04*, 4. https://doi.org/10.1145/989863.989936

Shinde, K., & Sun, Y. (2016). Template-Based Code Generation Framework for Data-Driven Software Development, 55–60. https://doi.org/10.1109/ACIT-CSII-BCD.2016.22

Sommervile, I. (2011). *Software engineering* (9th ed.). Boston: Pearson Education, Inc.,.

Vogel, L., & Scholz, S. (2016). The gradle build system- tutorial. Retrieved March 16, 2018, from http://www.vogella.com/tutorials/Gradle/article.html

Wilkins, B. (n.d.). Sketching interfaces; Generating code from low fidelity wireframes. Retrieved March 24, 2018, from https://airbnb.design/sketching-interfaces/

Appendix

A. Requirements Gathering

A.1 Interview Questions

In gathering requirements for this project, the following interview questions were asked.

1. Do you have any experience with Android programming? If answer is no, proceed to question 10.
2. What is your first reaction to Android application programming?
3. Overall, how satisfied or dissatisfied are you with tools for creating Android applications?
4. What do you like most about Android Studio?
5. What about Android Studio are you dissatisfied with?
6. What other IDEs do you consider when programming Android applications?
7. How would you rate the quality of Android Studio substitutes?
8. What changes would most improve competing IDEs from other companies?
9. When you're considering integrated development environments (IDEs), what are the top two things you generally consider?
10. What do you think of a platform that builds Android applications without having to write any code?

Interview Insights

In total, twenty-two people were interviewed out of which 13 had programmed an Android application while the other 9 had never. The insights gathered from the interview are summarized below.

1. From the interviews, it was evident that interviewees who had no exposure to programming were enthusiastic about a platform that will easily help them create applications without having to memorize syntax or be proficient in a programming language.
2. Most users also wanted to see only the things of major concern to them. For example, the interviewees who had used Android studio before complained about how complex the interface was and the difficulty in locating some important tabs. As such they opted for a platform with minimal user interface design where only the most useful tabs are displayed.
3. Users also wanted a platform where they did not have to install the software for each of their operating systems. If they had three different computers all running different operating systems, they did not want the situation where they had to install the application on all three of them. Hence most of the interviewees opted for a platform independent system.

Summary and charts of responses

Question: Do you have any experience with Android programming?

Responses	**Number**	**Percentage**
Yes	13	59.09%
No	9	40.91%

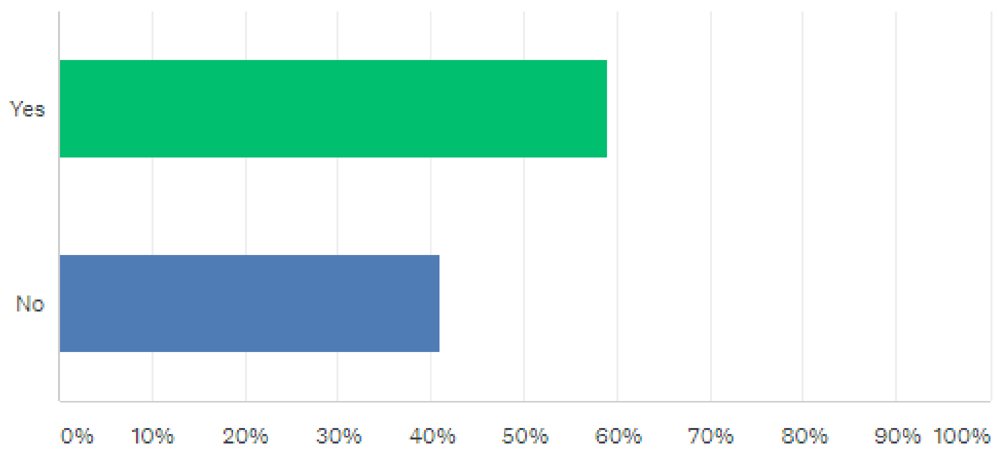


Chart showing respondents with Android programming experience and those without.

Question: Overall, how satisfied or dissatisfied are you with tools for creating Android applications?

Responses	**Number**	**Percentage**
Very satisfied	1	7.69%
Somewhat satisfied	7	53.85%
Neither satisfied nor dissatisfied	1	7.69%
Somewhat dissatisfied	4	30.77%
Very dissatisfied	0	0%

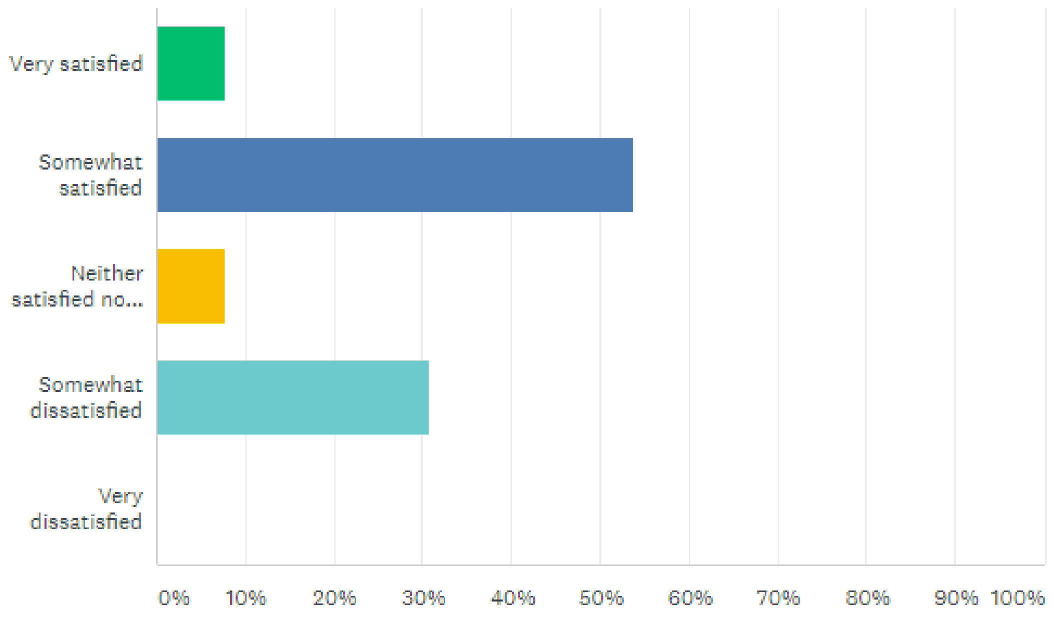


Diagram showing Android Studio satisfaction level.

Question: How would you rate the quality of Android Studio substitutes?

Responses	**Number**	**Percentage**
Very high quality	1	7.69%
High quality	4	30.77%
Neither high nor low quality	7	53.85%
Low quality	1	7.69%
Very low quality	0	0%

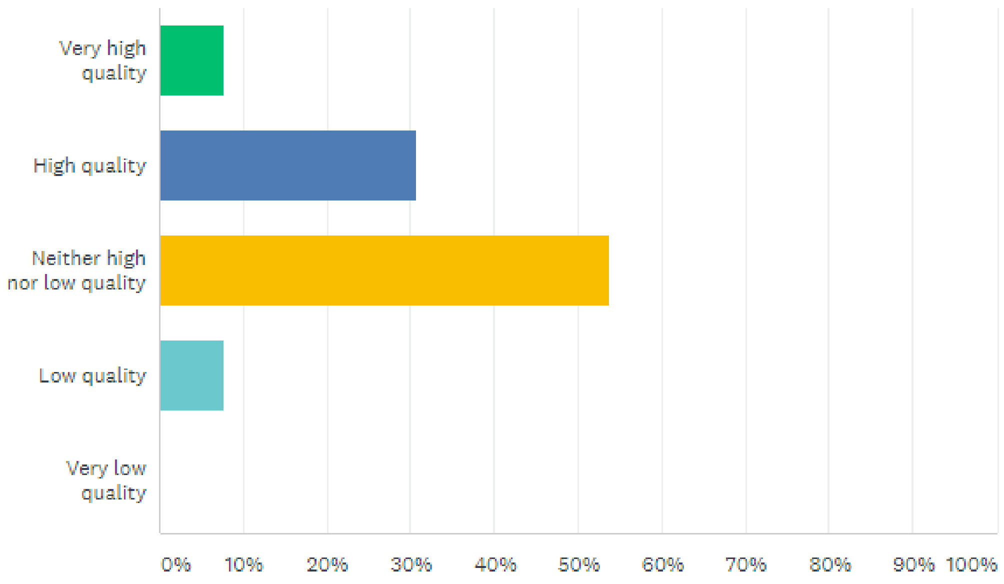


Diagram showing rating of Android Studio substitutes.

Question: When you’re considering integrated development environments (IDEs), what are the top two things you generally consider?

Responses	**Number**	**Percentage**
Intuitive design	8	61.54%
Code completion	8	61.54%
Multiple language support	7	53.84%
Ease of use	9	69.23%
Lightweight	5	38.46%
Other	1	7.69%

B. Class Diagram

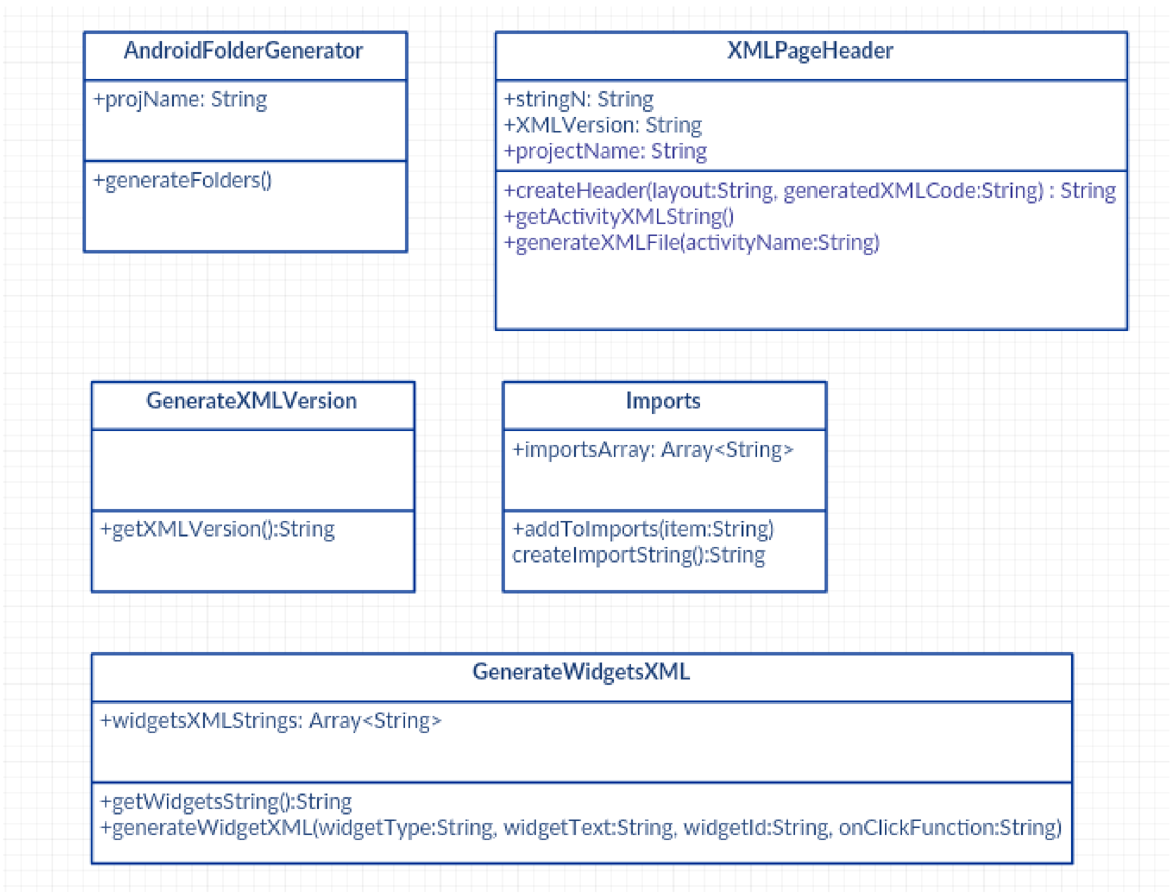

AndroidFolderGenerator
+projName: String
+generateFolders()
XMLPageHeader
+stringN: String
+XMLVersion: String
+projectName: String
+createHeader(layout:String, generatedXMLCode:String) : String
+getActivityXMLString()
+generateXMLFile(activityName:String)
GenerateXMLVersion
+getXMLVersion():String
Imports
+importsArray: Array<String>
+addToImports(item:String)
createImportString():String
GenerateWidgetsXML
+widgetsXMLStrings: Array<String>
+getWidgetsString():String
+generateWidgetXML(widgetType:String, widgetText:String, widgetId:String, onClickFunction:String)

Figure showing class diagram for five classes of the code generator.