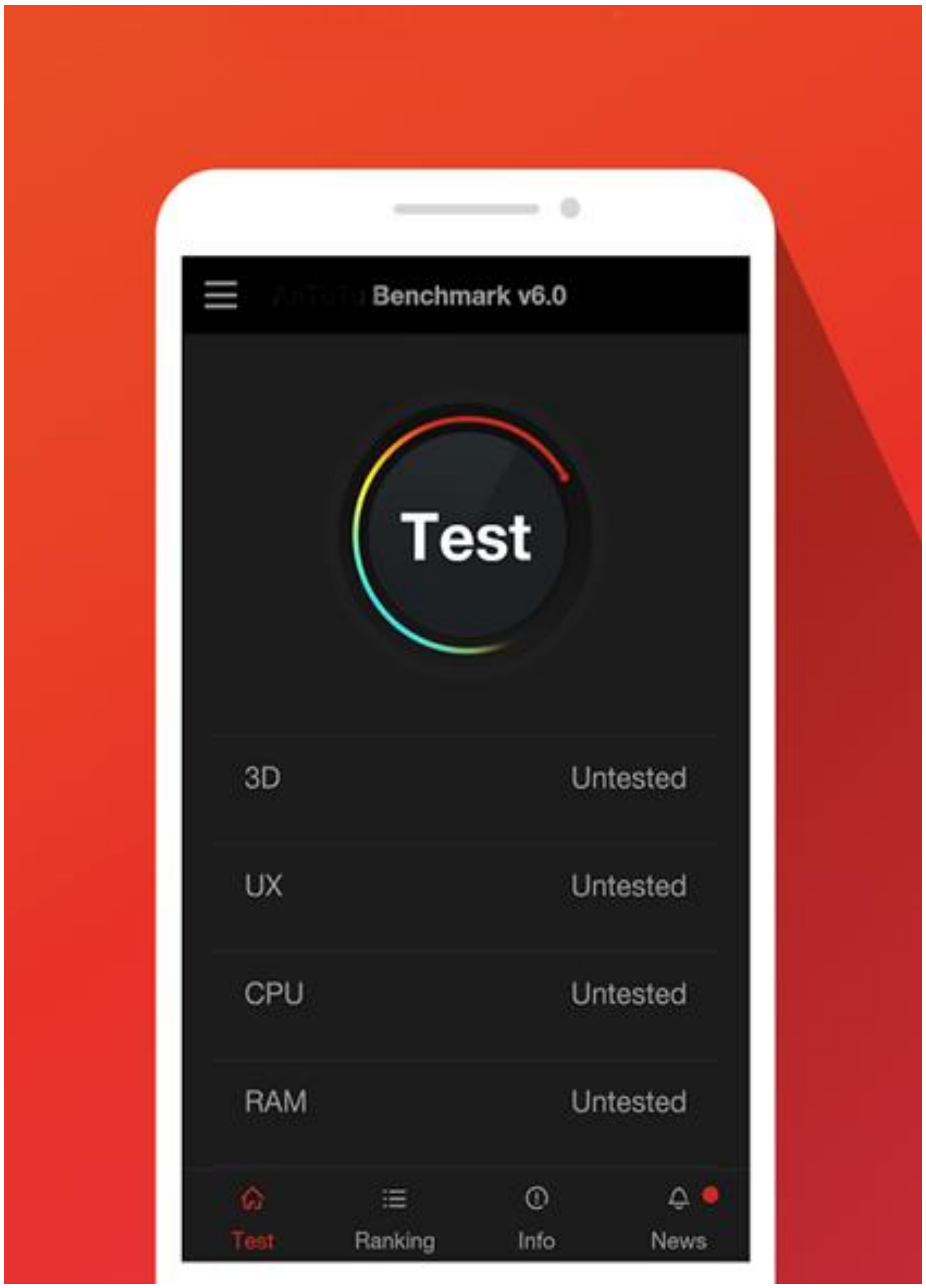


Benchmark for Android Devices

Andreica Ioan-Paul | Structure of Computer Systems | T.U. Cluj-Napoca | 2021

Contents

1. Introduction

1.1 CONTEXT

The goal of this project is to implement a mobile solution for fast and easy testing of core hardware features (CPU, STORAGE, GPU).

In implementing the tests, a variety of methods will be used based on the tested component.

For CPU testing we will test the processor power by using various mathematical operations meant to stress the processor. The time in which the operations complete will provide the score of the processor under test. Some operations include: calculation of Fibonacci function, calculation of PI, calculation of prime numbers, encryption techniques like SHA-1 etc.

For the GPU we will test the performance of a graphics virtualization by issuing many draw calls using various techniques learned from graphics processing, like Bezier Curves, particles, using OpenGL. The score will be calculated based on the FPS (frames-per-second) in correlation with the number of draw calls and object on screen.

For STORAGE testing we will use multiple read and write calls and compare the speed of the operations.

The application can be used by anyone who wants to measure the performance of their android phone and compare it to other devices.

1.2 SPECIFICATIONS

The program will run on mobile devices supporting recent android versions. It will be written in a combination of Java and C++ using the Android Studio IDE. Simulation using android emulators on PCs will most likely be possible and allow us to test various mobile phone configurations to build a relevant database of scores that the user can later check to compare its own device.

For storing the user scores, Firebase or a similar database solution will be used.

An advanced solution would be to create a separate storage microservice using Java or .NET that will handle storing the scores from the user benchmarks in a MongoDB database, the clients will only need to make simple API calls to the service, and we can use a free solution like Heroku to host the service and ensure it is alive at any time.

1.3 OBJECTIVES

Implement reliable testing algorithms for the mentioned hardware components, create a user-friendly interface that is straight forward and still appealing, ensure that the data generated by the tests is always saved and stored safely, since a benchmark application that uses a comparison method to give meaning to the benchmark scores is useless without a comprehensive database.

2. Bibliographic study

A mobile benchmark application requires 3 major components: a good UI that makes it easy for everyone to test their devices, reliable testing algorithms, and a data store to compare the test scores.

The most difficult part will be testing the GPU since the only solutions I found are using a combination of C++ for creating and drawing the objects in the environment, and Java for hosting the environment. The challenge comes from the fact that I never worked with graphical processing on Android before, and I never integrated C++ classes in Java applications. Other than that, the solutions found are straight forward: make draw calls to the GPU, constantly measure the FPS and increase the draw calls progressively to stress the GPU. The average FPS will provide the final score.

3. Analysis

3.1 ALGORITHMS

CPU:

The main algorithm that will be used to test the CPU performance will be the computation of an SHA-1. We will run this somewhere in the range of tens of thousands of times to stress the CPU and obtain a reliable score.

GPU:

As previously mentioned, we will use the OpenGL engine to render many entities on the screen and constantly measure the FPS. The average FPS corelated with the rendered entities count will provide the score for the GPU tests.

STORAGE:

For the storage tests we will simply measure the read and write speeds in different variations (small files, large files, images etc.).

3.2 TECHNOLOGIES

The application will run on Android devices and the codebase will be made in Java using the Android Studio IDE. The GPU code used to draw images will be written in C++ and used in the main Java application. For the server side we will have a .NET service written in C# and hosted in the Cloud, using a free service provider so that the app will be available for everyone. The database will use MongoDB and their service Atlas for hosting the database remotely and online.

3.3 ARCHITECTURE

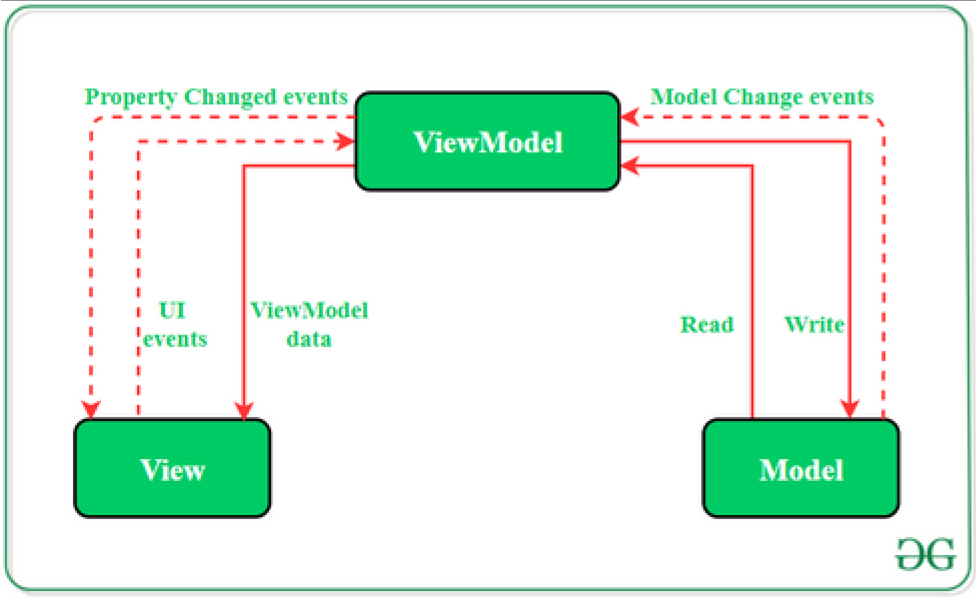
The project will follow the MVVM (Model-View-ViewModel) pattern.

Justification:

Developers always prefer a clean and structured code for the projects. By organizing the codes according to a design pattern helps in the maintenance of the software. By having knowledge of all crucial logic parts of the android application, it is easier to add and remove app features. Further, design patterns also assure that all the codes get covered in Unit Testing without the interference of other classes. **Model — View — ViewModel (MVVM)** is the industry-recognized software architecture pattern that overcomes all drawbacks of MVP and MVC design patterns. MVVM suggests separating the data presentation logic (Views or UI) from the core business logic part of the application.

The separate code layers of MVVM are:

- **Model:** This layer is responsible for the abstraction of the data sources. Model and ViewModel work together to get and save the data.
- **View:** The purpose of this layer is to inform the ViewModel about the user's action. This layer observes the ViewModel and does not contain any kind of application logic.
- **ViewModel:** It exposes those data streams which are relevant to the View. Moreover, it serves as a link between the Model and the View.



MVVM pattern has some similarities with the MVP (Model — View — Presenter) design pattern as the Presenter role is played by ViewModel. However, the drawbacks of the MVP pattern have been solved by MVVM in the following ways:

1. ViewModel does not hold any kind of reference to the View.
2. Many to 1 relationship exist between View and ViewModel.
3. No triggering methods to update the View.

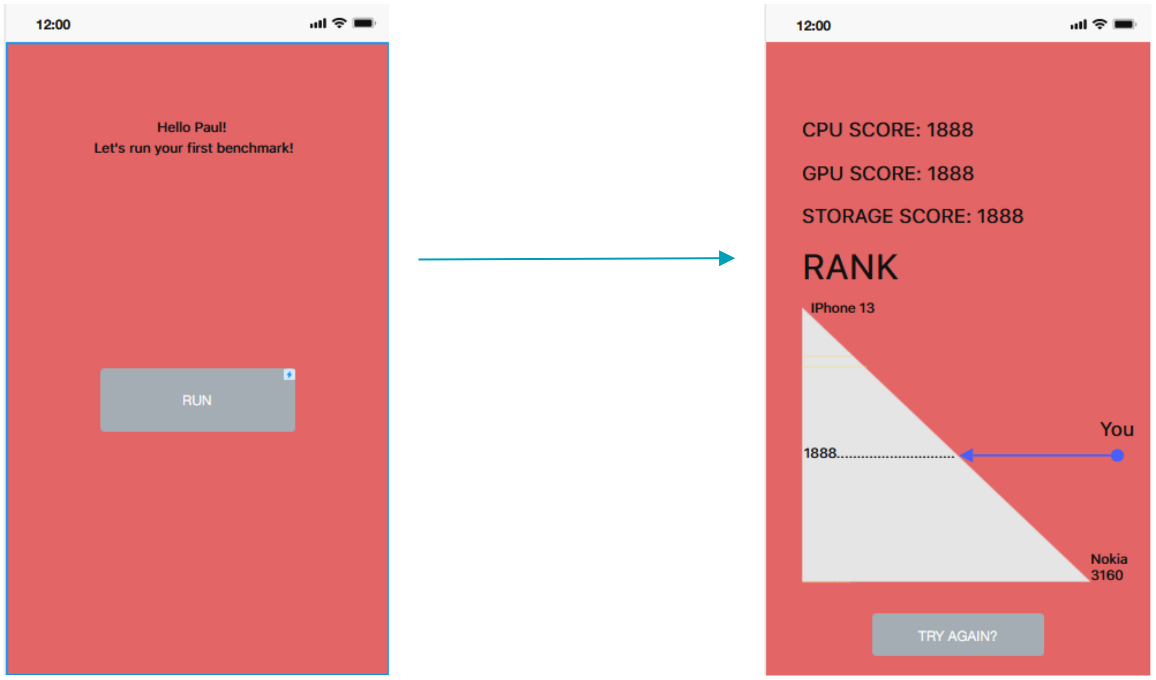
4. Design

4.1 USER INTERFACE

The user will only need to press a button to run a full benchmark test, alternatively he can select only a component to test, since the tests don't depend on each other and can be run individually. During the tests, the user will be provided with some status updates, or other visual indication that the app is still running. After the tests are finished the user will see the final scores for each component and a chart rating his device for every individual component or for the overall score.

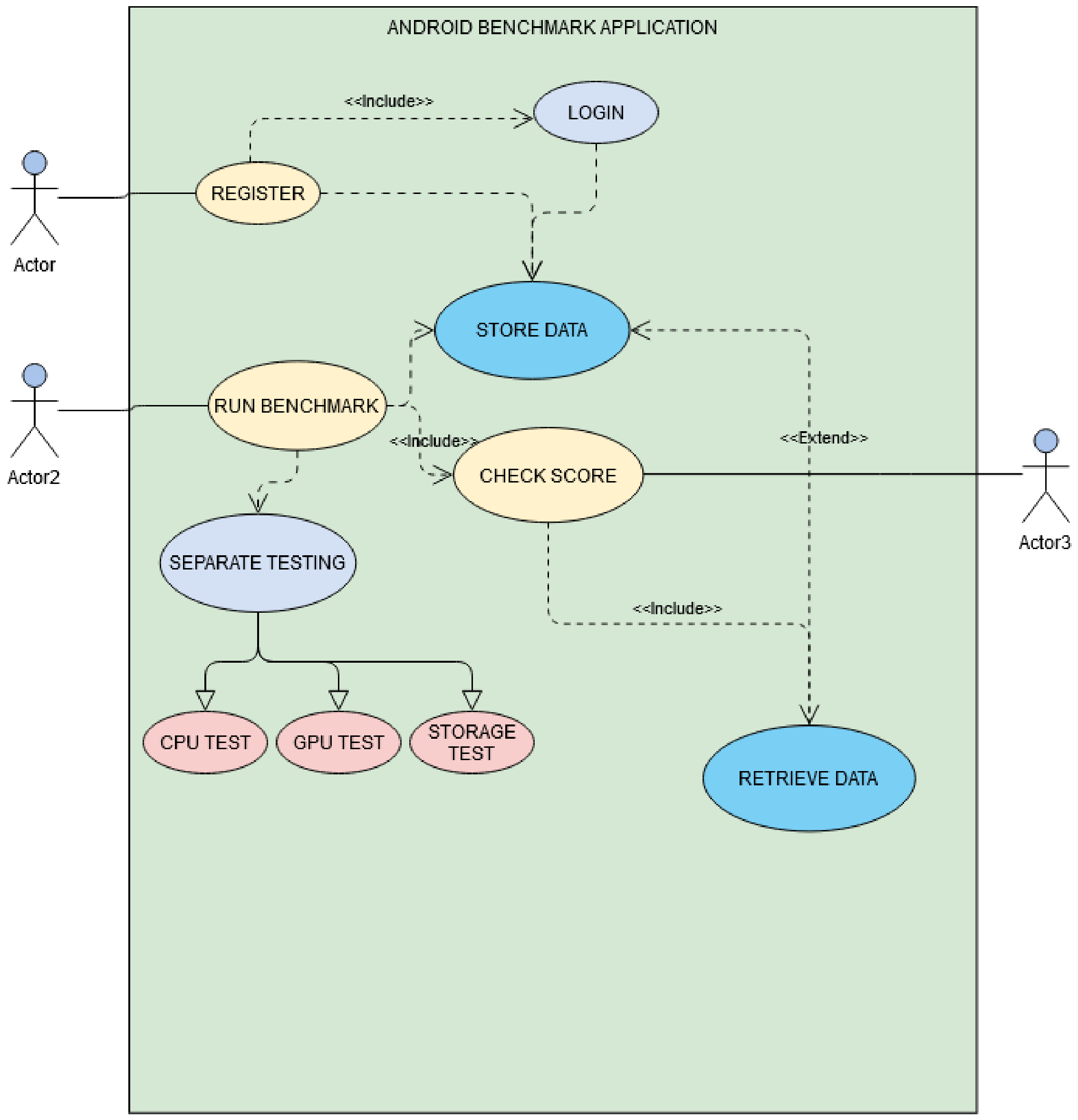
A user profile will also be created when the app is first started that will contain a device ID and the device details (manufacturer, model, configuration, hardware resources). Additionally, the user may create a profile to save his benchmark results if he ever needs them in the future, this profile requires a simple email and password configuration.

4.2 UX:



This is the first design ideea. The app will also have a login screen and buttons to allow only some benchmarks to be run at minimum.

4.3 USE CASES

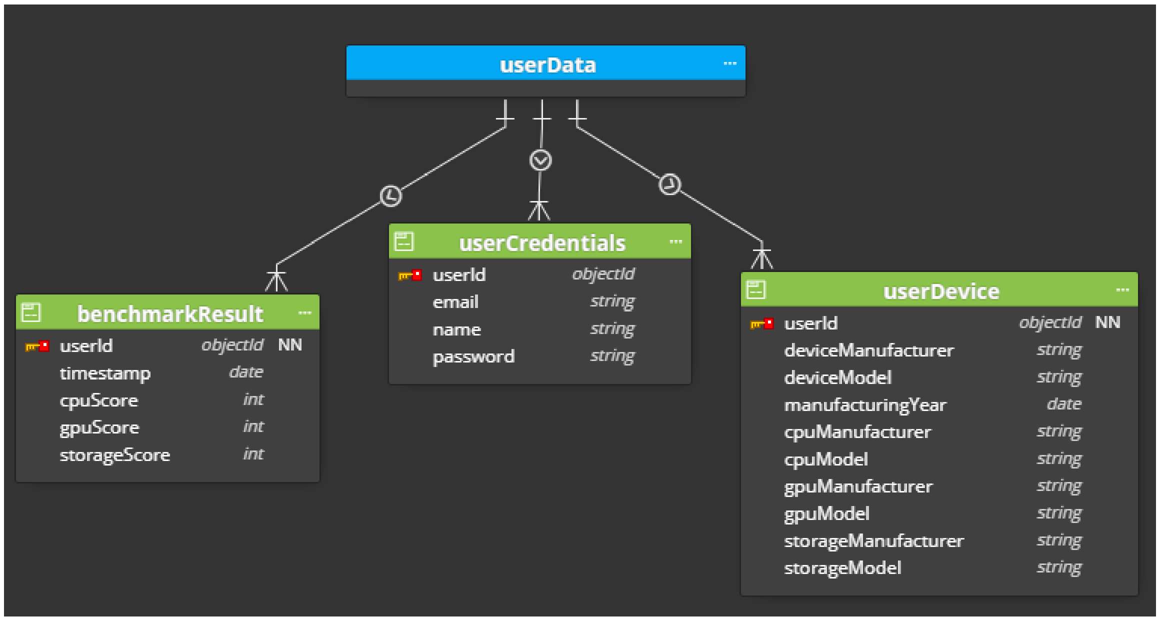


4.4 DATABASE DESIGN

Since the database will be very simple with no complicated queries or relationships, we can use a non-relational database to store our data as JSON documents. For this I chose the most used solution: MongoDB.

MongoDB provides free cloud service for data storing and accessibility along with a reliable GUI to see, modify and create datastore entries. Since the tables in mongo are just simple JSON documents we can easily insert any data we want without worrying about complex SQL statements or syntax. This allows us to work with data as code and provides vertical scaling which is much needed considering this application datastore will continue to grow with every usage.

Below is a quick design of the required data, of course this is not a real representation of the database since the actual design will comprise of simple JSON documents managed under a Collection:



benchmarkResult		
userId	objectId	NN
timestamp	date	
cpuScore	int	
gpuScore	int	
storageScore	int	

userCredentials	
userId	objectId
email	string
name	string
password	string

userDevice		
userId	objectId	NN
deviceManufacturer	string	
deviceModel	string	
manufacturingYear	date	
cpuManufacturer	string	
cpuModel	string	
gpuManufacturer	string	
gpuModel	string	
storageManufacturer	string	
storageModel	string	

As you can see, we have the main collection called "userData" which will store the coresponding documents "userCredentials", "benchmarkResult", "userDevice".

"userCredentials":

- This will be the first created document and a new entry will be added each time a user logs in with an email, name and password.
- The 'userId' will be assigned then and it will be used to make the correlations between the user and it's "benchmarkResult" or "userDevice"

"userDevice":

- This document will be created after the user logs in based on data collected from its device.
- The purpose of storing this data is so that we can create leaderboards and classifications based on the "benchmarkResult", with which the users can compare the performance of their device.

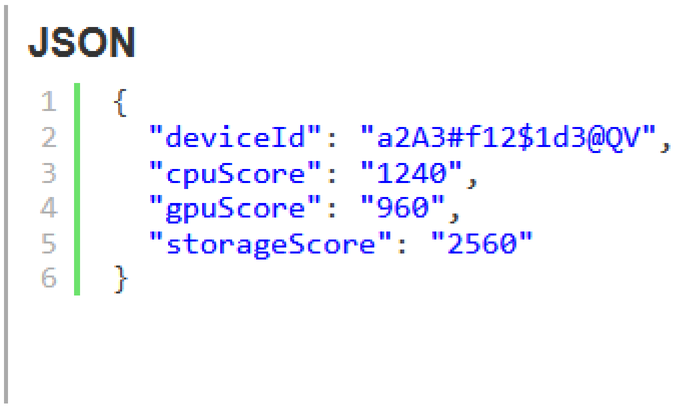
- A single entry will be added in this tabel for each user.

"benchmarkResult":

- This document will be used to store the scores that each user recieves after running a benchmark.
- There will be multiple entries in this document for each user coresponding to each succesful benchmark run. They will be uniquely identified by the userId coresponding to the user that ran the bechmark and the timestamp. This will be sufficient since a benchmark will take at least 1 minute to run so we can't have 2 identical timestamps for one user.

In the end we will corelate the entries from "benchmarkResult" with the data from "userDevice" to create some useful charts regarding the most performant processors, GPUs or Storage Devices. A lot more data analisys can be done here considering the available data set, but for a minimum viable product this will sufice.

How an actual benchmark result entry will look like:



JSON

```json
{
  "deviceId": "a2A3#f12$1d3@QV",
  "cpuScore": "1240",
  "gpuScore": "960",
  "storageScore": "2560"
}
```

Since it's a JSON file it will be trivial to send it from the mobile device to the backend server that handles the data store, regardless of what frameworks we use on the client and server.

5. Bibliography

- “Benchmark (computing)” [Online]: https://en.wikipedia.org/wiki/Benchmark_(computing)
- ” How to write an Android CPU benchmark tool” [Online]: https://www.androidauthority.com/write-an-android-cpu-benchmark-part-1-679929/
- “CPU Benchmark: Measure the power of Android devices” [Online]: https://hackernoon.com/cpu-benchmark-measure-the-power-of-android-devices-4aadb595cb61
- “GPU Emulation Stress Test” [Online]: https://github.com/google/gpu-emulation-stress-test
- “Use case design” [Online]: https://online.visual-paradigm.com/app/diagrams/#diagram:proj=0&type=UseCaseDiagram&gallery=/repository/514b156d-0257-4a05-bf77-1636e1c0815b.xml&name=Use%20Case%20Diagram%20Template
- “Mockup design” [Online]: https://mockitt.wondershare.com/
- ” MVVM (Model View ViewModel) Architecture Pattern in Android” [Online]: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/