Final Year Project:

Clever Carpooling

Testing Manual

16th May 2020

Authors:

Shaun Carey *16450454*

Nigel Guven *14493422*

Supervisor:

Ray Walshe

CLEVER CARPOOL

Table of Contents

1. Introduction

1.1 Policy and Strategy

This document is a supplementary document to support our testing formulae, which allows our development to publish our project with the goals of correctness, completeness, and consistency in mind. A rigid testing strategy is key to delivering successful software and this document is proof that we have established our project with testing as a fundamental part of the CA400 Final Year project.

Our strategy involves testing in four critical stages. We want to perform unit and integration testing, so the functionality of our project is correct and reliable and provides redundancy measures. A key aspect relating to our project's development came in inspiration from "Software Engineering" by Ian Sommerville specifically the topic of Dependability and Security whereby a system should be specifically available, secure, and reliable. This book was provided as a referential document to our CA4004 module.

We also want to provide performance testing and ad hoc testing. Performance testing relates to how well the application operates with the underlying system. Ad Hoc testing has been performed throughout the duration of the development stage and is mainly unplanned work which refers to bugs which creep up during development and are fixed when necessary rather than being provided with further analysis.

JUnit4 is to be used to provide the tools for testing on Android Studio for unit, instrumentation, and integration tests. These automated test cases are written separately to the main application under debugging. Manual cases are handled for critical features and are a proponent of ad hoc testing. Performance testing is monitored by analysing the resource constraints on the underlying system. We are using Android Profiler to monitor resource usage at certain points of the application in the hopes that we can refactor functions to optimise application activities. API calls and read/write operation are expensive so we must examine how we can measure the intensity and reduce the laborious functions.

1.2 Requirements Traceability Matrix

The requirements traceability matrix is a document which maps and traces user requirements with test cases and specific outcome results. All functionality within the system is available in the matrix. We are using the requirements traceability matrix to validate the user test cases by listing all coverage for major features across the system and includes all areas of testing. Our variant of the matrix includes tests to make sure that all tests have passed and cleared by a certain date.

Requirement No.	Description	Date of Validation
01	Splash Screen Displays	15th January 2020
02	Log In	23rd January 2020
03	Log In Bad Credentials	23rd January 2020
04	Forgot Password	15th February 2020
05	Register Driver/Passenger	29th January 2020
06	Register Name	29th January 2020
07	Invalid Date of Birth	13rd February 2020
08	Valid Date of Birth	13rd February 2020
09	Register Valid Address	18th February 2020
10	Register Valid Mobile Number	18th February 2020
11	Invalid Mobile Number	18th February 2020
12	Take Profile Photo	16th February 2020
13	Valid Email and Password	19th February 2020
14	Invalid Email AND/OR Password	19th February 2020
15	Validate Driver License	23rd February 2020
16	Validate Car Registration	25th February 2020
17	Check If Logged In	3rd March 2020
18	Home Profile Correct View	26th February 2020
19	Home Profile Side Bar	29th February 2020
20	Log Out Validation	5th March 2020
21	Dashboard Displays Posts	8th March 2020
22	Map Creates Route	9th March 2020
23	Date Time Dialogs	4th March 2020
24	Correct User and Post Counts	7th March 2020
25	Post Details Correct Functionality	9th March 2020
26	View Profile Correct View	12th March 2020
27	Send Request	15th March 2020
28	Cancel Request - Sender	15th March 2020
29	Accept Request	18th March 2020
30	Cancel Request – Receiver	1st May 2020
31	Peer-to-Peer Send Message	27th April 2020
32	Peer-to-Peer Send Image	27th April 2020
33	Peer-to-Peer Send PDF	29th April 2020
34	Create Group Chat	30th April 2020
35	Group Chat Send Message	1st May 2020
36	Delete Message for Sender	27th April 2020
37	Delete Message for Everyone	27th April 2020
38	Last Seen Correct Functionality	10th April 2020
39	Current Online Functionality	10th April 2020
40	Manage Trips Correct View	21st April 2020
41	Search Bar Correct Functionality	26th April 2020
42	Search Results Correct View	27th April 2020
43	Search Results Sort by Date	20th April 2020
44	Search Results Sort by Relevance	1st May 2020

1.3 Glossary

Requirements Traceability Matrix – An RTM is a document that maps and traces user requirements with test cases. It captures the entire set of requirements proposed by a client.

JUnit – JUnit is a testing framework for the Java programming language. It comes installed on Android Studio and is useful for creating functional tests.

Android Profiler – Android Profiler tools provide real-time data to help you to understand how your app uses CPU, memory, network, and battery resources.

3C Task Manager - 3C Task Manager is an Android application which monitors CPU and memory usage of Android applications like Windows Task Manager.

Performance Testing – Performance testing is the process of determining how a system uses its available resources, what its max requirements are and how it can handle stress and volume.

Ad Hoc Testing – Ad Hoc testing is a method of spontaneous testing which lacks documentation and rather serves as a quick determination of whether a function works or not.

Unit Testing – Unit testing is the testing of the smallest components in an application, generally the low-level functions and methods which are used by the application.

Integration Testing - In the case of Android, integration testing refers to how non-native functions operate within the application, known as content providers.

Instrumentation Tests – Instrumentation tests are running on an emulator as opposed to the native Android application and are useful for checking if certain devices do not match system requirements.

Content Provider – Content providers provide functionality to external applications. API's, web scrapers and databases are examples.

Logcat – Logcat is a command line tool which dumps a log of system messages, including stack traces.

JSoup – A Java purpose-built library for interacting with HTML pages. It provides code for cleaning HTML, returning the underlying data.

Gitlab – Gitlab is a DevOps platform which provides version control, CI/CD, and issue tracking.

Firebase ML Kit – ML Kit is a mobile SDK that brings machine learning capabilities to Android and iOS applications.

Fused Location API – Fused Location is an API in Google Play services which uses the platform's operating system GPS module to retrieve the users' location.

Firestore – Cloud Firestore is a flexible, scalable database for development using Firebase and Google Cloud Platform

2. Functional Testing

Our testing of functionality was performed in Android Studio using JUnit4 and performed on a local Android device. We were able to utilise Android Studio to automate test cases and were test cases to fail, we could refactor code or provide extra checks to fix the problem.

Unit testing in the application was performed on functions and classes. Most of the functions which were being utilised by multiple classes were refactored into the Functions class file so we could test the input against expected output parameters. Some of the main unit testing was also performed on the various algorithms which had to be refactored to be tested effectively. Through the writing of multiple user tests, we were able to refactor various functions to fix the handling of their inputs.

Integration testing on Android meant that we had to examine how external content providers affected the application. This included testing API's and web scraping functions. We created test suites specifically for analysing various inputs and outputs. A problem we had concerned JSoup and its testability, where the MotorCheck website would deny access to too many requests from the same location. This would display as a failure inside of Android Studio, but when reading from Logcat, we could see that there was a connection request error outputted by JSoup.

Testing for licence validation and scraping

```
@Test
public void scrapeAPI() throws IOException
{
    //API Timeout Refuse Connection
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("07D27644"), true);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("08D2764"), true);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("07D244"), true);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("07D24"), true);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("07c44"), true);

    Assert.assertEquals(LevenshteinDistance.scrapeAPI("09FF27"), false);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("09G27"), false);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("09789"), false);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("07"), false);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("27"), false);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI("09v27"), false);
    Assert.assertEquals(LevenshteinDistance.scrapeAPI(""), false);
}

@Test
public void isValidLicense()
{
    Assert.assertEquals(true, LevenshteinDistance.isValidLicense("CEADUNAS TIOMANA
DRIVING LICENCE"));
    Assert.assertEquals(true, LevenshteinDistance.isValidLicense("CEADUNAS
TIOMANA"));
    Assert.assertEquals(true, LevenshteinDistance.isValidLicense("DRIVING
LICENCE"));
    Assert.assertEquals(true, LevenshteinDistance.isValidLicense("TIOMANA
DRIVING"));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("CDNS TMN DRVNG
LCNC"));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("CEADUNAS
DRIVING "));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense(""));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("CEAD FOGHLAMORA
LEARNER PERMIT"));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("FOGHLAMORA
LEARNER"));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("CEAD"));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("LEARNER
PERMIT"));
    Assert.assertEquals(false, LevenshteinDistance.isValidLicense("CD FGHLMR LRNR
PRMT"));
}
```

Testing for Levenshtein Distance and Longest Substrings

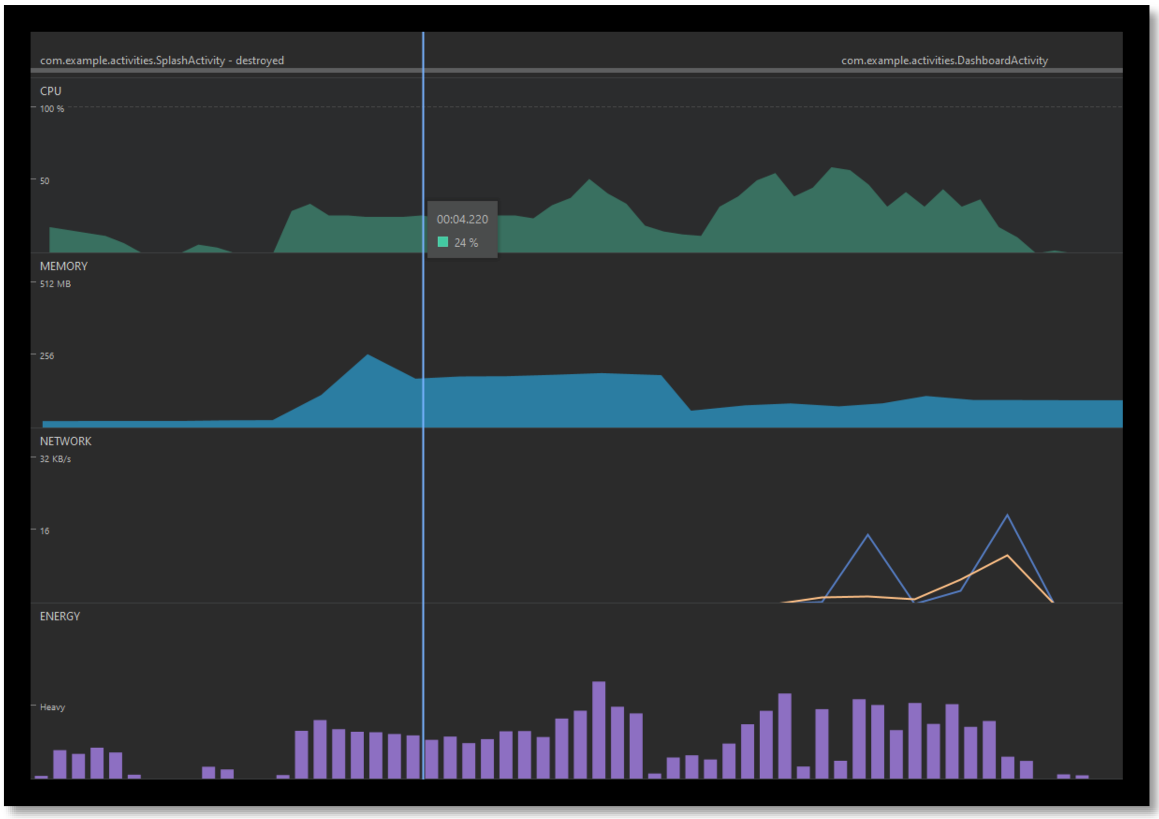
```
@Test
public void returnDistance()
{
    System.out.println("Results: " + LevenshteinDistance.returnDistance("Hello
Wold", "Hello World"));
    System.out.println("Results: " + LevenshteinDistance.returnDistance("", "Hello
World"));
    System.out.println("Results: " +
LevenshteinDistance.returnDistance("HelloWorld", "Hello World"));
    System.out.println("Results: " + LevenshteinDistance.returnDistance("Hello
Wrld", "Hello World"));
    System.out.println("Results: " + LevenshteinDistance.returnDistance("Hello",
"Hello World"));
}

@Test
public void longestSubstr()
{
    Assert.assertSame(2, LevenshteinDistance.longestSubstr("Hi", "hi"));
    Assert.assertNotSame(4, LevenshteinDistance.longestSubstr("Hitt", "hi2"));
    Assert.assertEquals(6, LevenshteinDistance.longestSubstr(" Hello ", "Hello
World"));
    Assert.assertEquals(0, LevenshteinDistance.longestSubstr("", "Hello World"));
    Assert.assertEquals(5, LevenshteinDistance.longestSubstr("HelloWorld", "Hello
World"));
    Assert.assertEquals(7, LevenshteinDistance.longestSubstr("Hello Wrld", "Hello
World"));
    Assert.assertEquals(6, LevenshteinDistance.longestSubstr("Hello ", "Hello
World"));
}
```

3. Performance Testing

Android Studio provided in-house tools to measure application performance in the areas of CPU, memory, network, and energy resources. The goal of performance testing is to analyse how the system will operate under heavy load, handling stressful situations and what factors will lead to a maximum volume of resources. We decided to make a trial run of the application and utilise the main activities of our system, from logging in to sending messages. Below contains a 2-minute, 39 sec monitoring session of the main resource.

Fig A.



From Figure A, we can see that the application is intensive on all resources when it must load the application from the beginning. Once data is loaded onto the interface, the resource requirements begin to stabilise as can be seen once the Dashboard Activity has loaded its elements from Firestore. We assume that the large network spike is the retrieval of data from Firestore. Luckily, Firestore saves data offline by caching so there are no requirements for unnecessary. The application is quite heavy on the battery at start up which does the threshold but stabilises once the dashboard interface is displayed.

Fig B.

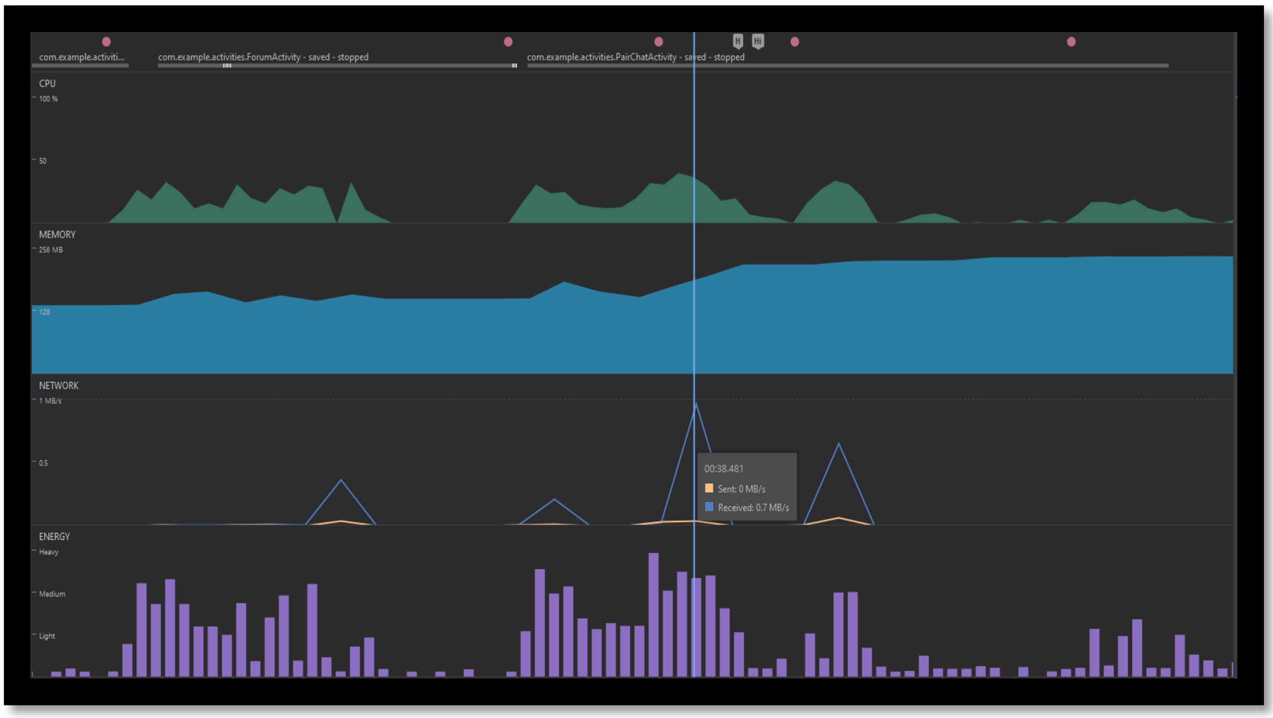
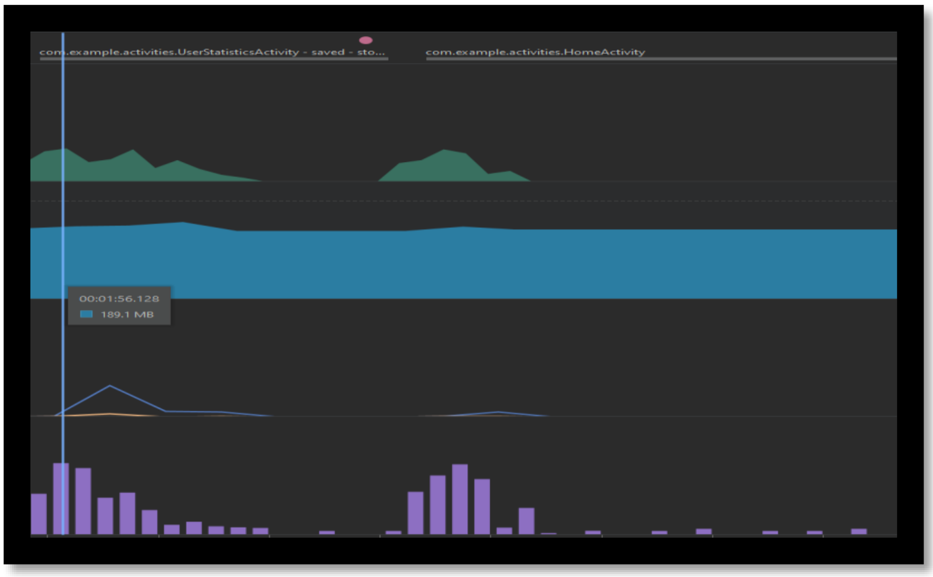


Figure B. tests the sending of different message types in the built-in messaging system. The highest network speed was calculated from the sending of a PDF document. Its counterpart to the right was the sending of an image, and to the left a text message. We noticed a large increase in the amount of memory required as each message was sent. We assume this because the specific activity requires the holding of messages at runtime in its cache. Again, there is heavy usage of the battery.

Figure C.


com.example.activities.UserStatisticsActivity - saved - sto...
com.example.activities.HomeActivity
00:01:56.128
189.1 MB

We can see from this analysis that the User Trips Section and Home profile use little amounts of resources as they do not provide critical functionality.

Figure D.

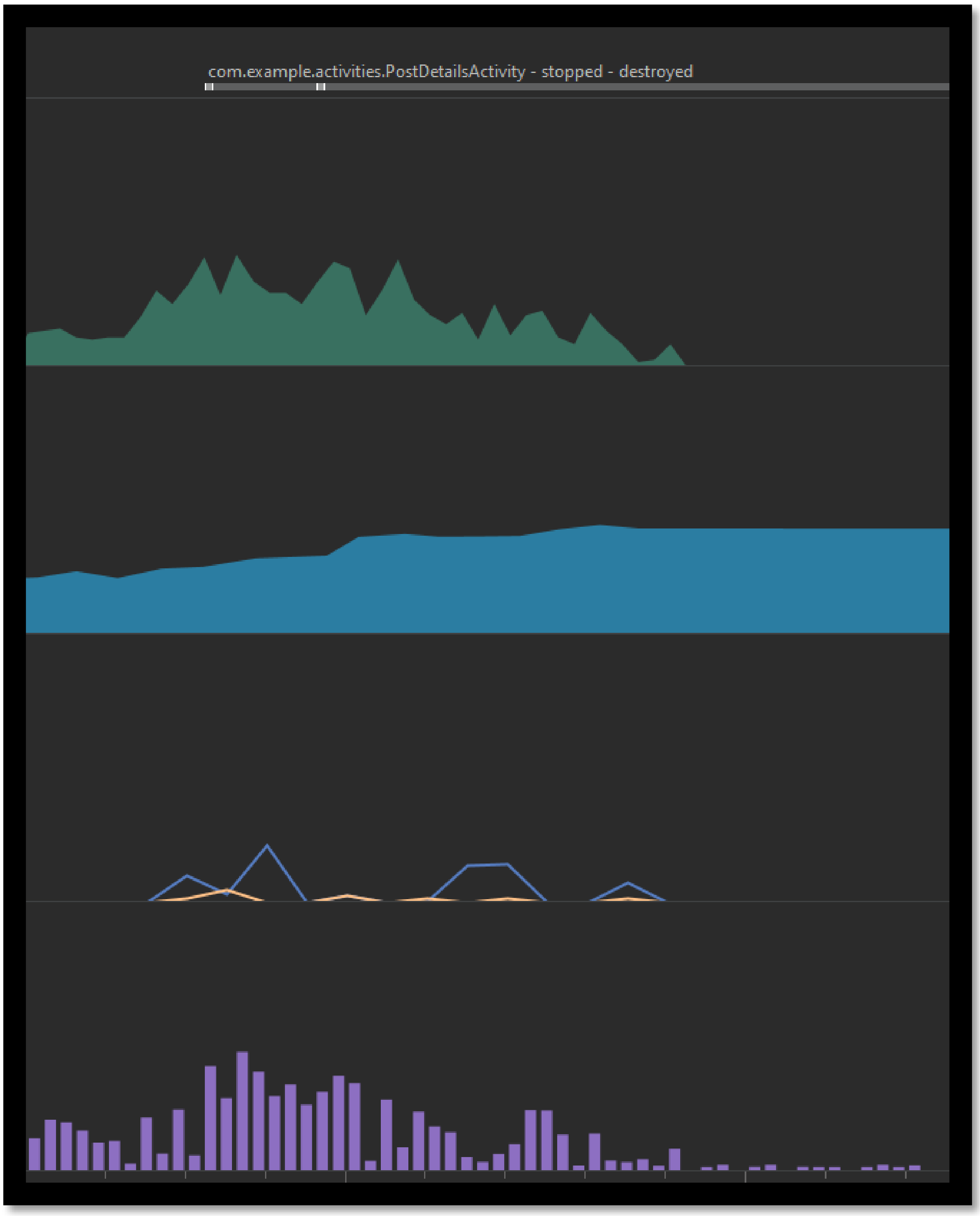
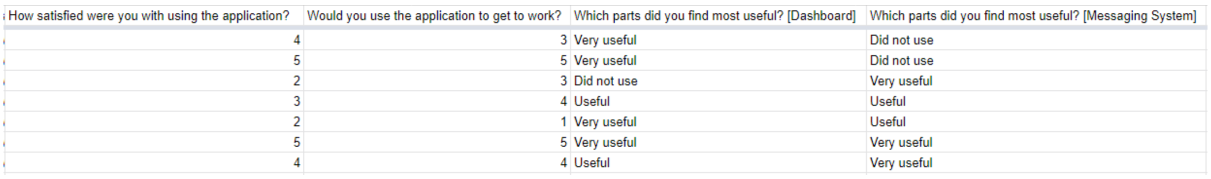

com.example.activities.PostDetailsActivity - stopped - destroyed

Figure D comes from what we assume is the most labour-intensive activity. We assume that this activity is the most intensive since it requires the calling of a Google Maps API, Fused Location API and 3 Directions API's. It requires a lot of resources in its onCreate method but once it the data it requires from API calls and database references, it stabilises.

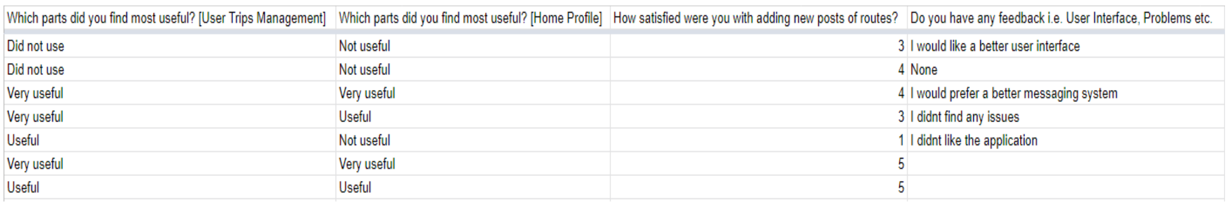
4. User Testing

We created a Google Form which asked for user feedback in using the application. We received 7 responses from the application survey, and while this is minimal, given the current circumstances, it was to harder to retrieve a larger test population. As of the 12th May, we still do not have Ethics approval, instead we are providing users with access to the application under pseudonyms such as famous celebrities. They were able to use the central system such as posting routes on the dashboard, searching for routes, and sending requests and retrieving messages from other users. We were able to analyse the feedback from our user base to see what we could improve in the application before the due date.

Figure E.



How satisfied were you with using the application?	Would you use the application to get to work?	Which parts did you find most useful? [Dashboard]	Which parts did you find most useful? [Messaging System]
4	3	Very useful	Did not use
5	5	Very useful	Did not use
2	3	Did not use	Very useful
3	4	Useful	Useful
2	1	Very useful	Useful
5	5	Very useful	Very useful
4	4	Useful	Very useful



Which parts did you find most useful? [User Trips Management]	Which parts did you find most useful? [Home Profile]	How satisfied were you with adding new posts of routes?	Do you have any feedback i.e. User Interface, Problems etc.
Did not use	Not useful	3	I would like a better user interface
Did not use	Not useful	4	None
Very useful	Very useful	4	I would prefer a better messaging system
Very useful	Useful	3	I didnt find any issues
Useful	Not useful	1	I didnt like the application
Very useful	Very useful	5	
Useful	Useful	5	

Figure E shows the results of our survey in excel format. We asked users how satisfied they were with using the application and which of the main subsystems of the application they found most useful. We specifically asked users on their opinion of creating routes. Finally, users were able to provide feedback regarding any problems they encountered or if they had some advice on ideas that we could implement.

Users were most satisfied with the dashboard and the messaging system. They were however not satisfied with their home profile, but we assume this is because of a lack of user functionality in this interface.

5. Ad Hoc Testing

A lot of our testing was performed in a style of Ad Hoc testing. This was usually performed at each stage of incremental development. For example, we began a thorough Ad Hoc testing of a driver licence retrieval which utilises Firebase machine learning API's. We began testing by taking various pictures which included text and from this we could come up with an algorithm to filter out invalid driving licences.

Ad Hoc testing covered all areas of our application from the user interface functionality to formatting user form data and firebase notifications. With each error encountered, we reimplemented designs to update the application. This was aided by the version control repository of Gitlab which aided us at times when a feature had been broken and required a full reconversion of a class or method.

6. Appendices

Figure F. – Testing plan UML

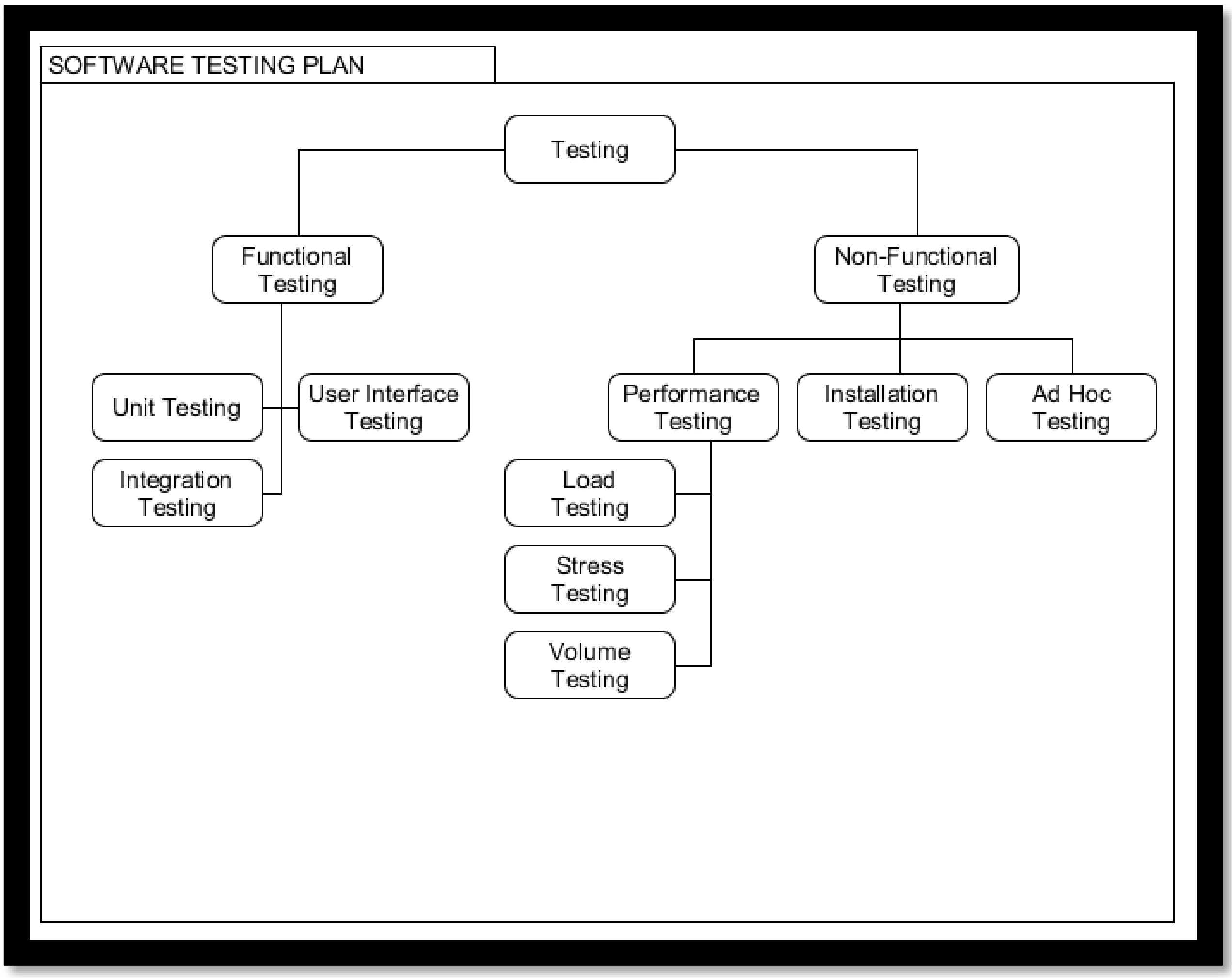


Figure G. Testing Functions in Login System

```
@Test
public void containsDigit()
{
    assertTrue(Functions.containsDigit("fgasfsafd7afdf"));
    assertFalse(Functions.containsDigit("hellowogfsdfgrldgmaicom"));
}

@Test
public void getAddress()
{
    assertTrue("DCU, 630 Collins Ave Ext, Whitehall, Dublin, D09 W6F4", true);
    assertFalse("dCU, 630 Collins Ave Ext, Whitehall, Dublin, D09 W6F4", false);
}

@Test
public void format_home_address()
{
    assertEquals(Functions.format_home_address("DCU, 630 Collins Ave ext,
Whitehall, Dublin"), Functions.format_home_address("dCU, 630 collins Ave Ext,
Whitehall, dublin"));
```

```java
    assertEquals(Functions.format_home_address("dCU, 630 collins ave ext,
whitehall, dublin"), Functions.format_home_address("dCU, 630 collins Ave Ext,
Whitehall, Dublin"));
    assertNotEquals(Functions.format_home_address("dCU, 630 Collins Ave ext,
whitehall, dublin"), Functions.format_home_address("dCU, 630 Collins AVe Ext,
Whitehall, Dublin"));
}

@Test
public void capitalize()
{
    //Should be Correct
    assertTrue("Helloworld", true);
    assertTrue("Hello World", true);
    assertTrue("2Hello World", true);

    //Should be Incorrect
    assertFalse("hello World", false);
}

@Test
public void getImage()
{
    //Should be Correct
    assertEquals(Functions.getImage(true), R.drawable.driver_post);
    assertEquals(Functions.getImage(false), R.drawable.passenger_post);

    //Should be Incorrect
    assertNotEquals(Functions.getImage(false), R.drawable.driver_post);
    assertNotEquals(Functions.getImage(true), R.drawable.passenger_post);
}

@Test
public void isValidEmail()
{
    //Should be Correct
    assertTrue(Functions.isValidEmail("helloworld@gmail.com"));
    assertTrue(Functions.isValidEmail("helloworld@ii.com"));
    assertTrue(Functions.isValidEmail("helloworld@homeemail.com"));
    assertTrue(Functions.isValidEmail("helloworld@1233numbers.com"));
    assertTrue(Functions.isValidEmail("helloworld@hotmail.com"));
    assertTrue(Functions.isValidEmail("helloworld@firebase.com"));
    assertTrue(Functions.isValidEmail("helloworld@outlook.com"));
    assertTrue(Functions.isValidEmail("heldfg234234orld@gm.com"));
    assertTrue(Functions.isValidEmail("heldfg234234orld@gm.com"));

    //Should be Incorrect
    assertFalse(Functions.isValidEmail("helloworldgmail.com"));
    assertFalse(Functions.isValidEmail("hello@@worldmail.com"));
    assertFalse(Functions.isValidEmail("helloworldgmail.com"));
    assertFalse(Functions.isValidEmail("helloworld@com"));
    assertFalse(Functions.isValidEmail("helhd345345ld@out345k+com"));
    assertFalse(Functions.isValidEmail("helloworldg345com"));
}
```

7. References

Android - https://developer.android.com/

Android Profiler - https://developer.android.com/studio/profile/android-profiler/

JUnit - https://junit.org/junit4/

CA400 - https://www.computing.dcu.ie/~pclarke/ca400/

Mockito - https://site.mockito.org/

UMLet - https://www.umlet.com/

3C Task Manager - https://play.google.com/store/apps/details?id=ccc71.tm&hl=en_IE

Types of Software Testing - https://hackr.io/blog/types-of-software-testing

The Art of Software Testing, Glenford J. Myers - http://barbie.uta.edu/~mehra/Book1_The%20Art%20of%20Software%20Testing.pdf

Gitlab - https://about.gitlab.com/

Firebase ML Kit - https://firebase.google.com/docs/ml-kit

Google Forms - https://www.google.com/forms/about/

Fused Location API - https://developers.google.com/location-context/fused-location-provider

Firestore - https://firebase.google.com/docs/firestore

-End of Document-