



Final Report

Project 117: Mobile Car Messaging System

By Alex Hutchings 1412529

CM3203 One Semester Individual Project, 40 Credits

Supervisor: George Theodorakopoulos

Moderator: David Walker

Table of Contents

1. Introduction	3
2. Background	4
2.1. Applications that I will use throughout the project.....	5
2.2. Libraries and modules	6
2.3 Current Solutions	7
2.2 Constraints	10
2.3 Potential Stakeholders	10
3. Specification and Design	11
3.1 Aims and Objectives.....	11
3.2 System Design	14
3.4 Database Design	20
3.5 Use Case Diagrams	21
3.6 System Flow Chart	22
4. Implementation	24
4.1 Changes from the initial plan	24
4.2 Execution of the Approach.....	25
4.3 Implementation concerns	38
5. Evaluation	40
5.1 Testing	40
5.2 Evaluation of the Implementation	44
5.3 Next Steps	48
5.4 Device Performance	49
6. Reflection.....	51
6.1 Self-Evaluation	52
7. Conclusion.....	53
8. References	54
9. Appendix.....	56
9.1 User interface designs.....	56
9.2 Test Case Screenshots.....	60

Table of Figures

FIGURE 1.0– ANDROID MARKET SHARE COMPARED TO OTHER PLATFORMS	5
FIGURE 2.1 – ANDROID APPLICATION STRUCTURE	7
FIGURE 3.1– THE LOGIN SCREEN DESIGN	14
FIGURE 3.2 – THE CREATE ACCOUNT DESIGN	15
FIGURE 3.3 – THE MAIN MENU SCREEN DESIGN	16
FIGURE 3.4 – THE INBOX SCREEN DESIGN	17
FIGURE 3.5 – THE SEND MESSAGE SCREEN DESIGN	18
FIGURE 3.6 – THE REPORT ACCIDENT SCREEN DESIGN	19
FIGURE 3.7 – THE TRAFFIC INFORMATION SCREEN DESIGN	20
FIGURE 3.8 – DIAGRAM OF THE DATABASE TABLES AND THEIR FIELDS	21
FIGURE 3.9 – USE CASE DIAGRAM OF THE INTENDED SEND MESSAGE ACTIVITY	21
FIGURE 3.10 – INTENDED INBOX ACTIVITY USE CASE DIAGRAM	22
FIGURE 3.11 – FLOW CHART OF THE INTENDED APPLICATION	23
FIGURE 4.1 – CLIENT ACTIVITY CODE	25
FIGURE 4.2 – IMPLEMENTATION OF CLIENT ACTIVITY	26
FIGURE 4.3- SERVER ACTIVITY CODE	26
FIGURE 4.4 – SERVER ACTIVITY IMPLEMENTATION	27
FIGURE 4.5 - DATABASE ADAPTER METHODS	28
FIGURE 4.6 - BUTTON LISTENER METHODS: (ONE FOR EACH IMAGE BUTTON ON THE FORM)	29
FIGURE 4.7 – THE SEND MESSAGE ACTIVITY IMPLEMENTATION	30
FIGURE 4.8 – ENCRYPTION FUNCTION	30
FIGURE 4.9 – THE ENCRYPTION AND DECRYPTION FUNCTIONS THAT HAVE BEEN IMPLEMENTED.	31
FIGURE 4.10 – DECRYPTION FUNCTION	31
FIGURE 4.11 – SECRET KEY GENERATOR FUNCTION	32
FIGURE 4.12 – INBOX ACTIVITY FUNCTION	32
FIGURE 4.13 – INBOX DIALOG FUNCTION	33
FIGURE 4.14 – THE INBOX ACTIVITY IMPLEMENTATION	33
FIGURE 4.15 – HOMESCREEN LOGINBUTTON FUNCTION	34
FIGURE 4.16 – THE HOME ACTIVITY IMPLEMENTATION	35
FIGURE 4.17 – TRAFFIC INFORMATION POPULATE ACCIDENTS FUNCTION	35
FIGURE 4.18 – TRAFFIC INFORMATION ACTIVITY IMPLEMENTATION	36
FIGURE 4.19 – REPORT ACCIDENT PERMISSION AND MAP LOADER FUNCTIONS	36
FIGURE 4.2 – THE REPORT ACCIDENT ACTIVITY IMPLEMENTATION	37
FIGURE 4.21 – ACTION BAR FUNCTIONS AND XML	38
FIGURE 4.22 – THE ACTION BAR IMPLEMENTATION	38
FIGURE 5.1 – FLOW OF EVENTS FOR SENDING A MESSAGE IN THE IMPLEMENTED SOLUTION	45
FIGURE 5.2 – FUTURE ENCRYPTION USE CASE	47
FIGURE 5.3 – FUTURE DECRYPTION USE CASE DIAGRAM	48
FIGURE 5.4 – TRAFFIC INFORMATION ACTIVITY AND ITS MONITOR LEVELS	49
FIGURE 5.5 – REPORT ACCIDENT ACTIVITY AND ITS MONITOR LEVELS.	50

1. Introduction

Have you ever had a brake light that has stopped working, but you haven't spotted it for several weeks? Have you ever noticed another driver's headlights not on, and thought they could cause an accident if they didn't turn them on? Have you ever seen another vehicle's brake lights not working? If you answered yes to any of these questions, then you are not alone. These are common problems that happen every day for a driver. Personally, I see multiple cars with problems on a day to day basis, and always think to myself, if only I could contact them and let them know there is a problem, they can go ahead and fix it.

This project is about creating an application that allows drivers to communicate with each other regarding the status of their respective vehicles. The application will be designed specifically for Android mobile devices and will allow users to create, send and read messages regarding the fault of either their car or another driver's car respectively. A mobile application is the ideal solution for providing drivers and other car-users the ability to actively communicate these issues to one another. Mobile phones are ideal because they allow for fast real time communication on the move.

The motivation for this project stems from new cars being deployed with built-in error reporting functionality that notifies drivers of issues with their vehicle, for example, brake lights not working, flat tyres and headlights out. This creates a gap as generations of older cars do not have any indicator that alerts the driver to faults with their car. Therefore, drivers of these older cars lack the technology that could help identify and bring the drivers attention to these types of issues. Creating a mobile application that allows drivers to communicate would bridge that gap by allowing other road-users who spot these issues to report them directly to the vehicle's driver. An application of this ilk would allow drivers to communicate in a safe and simple way, making the unaware, aware of their issues, and ultimately helping them to avoid penalties and fines from law enforcement agencies.

2. Background

Overview

There are many vehicles on the street that have minor faults with their components, whether that is a faulty reverse light, brake light, or their headlights are not switched on or a tyre that is very flat etc. These faults could potentially cause an accident or get a driver fined for an issue that they are not aware of. The problem that needs to be solved, is the inability to communicate between drivers, telling them that there is a fault within the other person's car. The only way to currently notify a driver of a fault is to physically get out of your car and tell them. The application that I am proposing would solve the issue of alerting other driver's to a problem with their vehicle as you could send a simple message between mobile devices.

Modern vehicles have built in technologies that provide status updates on issues within the vehicle and alert the driver to them. This project will focus on cars that are built on a low-cost budget, or older vehicles that do not contain this technology.

"36.5 million vehicles licensed for use on the roads in Great Britain. The average car licensed for use on the road was 8 years old." [1] (Vehicle Licensing Statistics 2015). This statistic shows the magnitude of vehicles that potentially have no access to this technology in the UK. Therefore, the potential use for the rest of the world is astronomical, if an application like the one proposed was readily available in a safe environment for drivers to communicate with each other.

The project being proposed is different to those that are currently available or that are being developed, as discussed in section 2.3. The solution will be an application that can be used by any person inside the vehicle. It does not have to be the driver that spots a fault within another driver's vehicle, it could be the passenger, or someone in the back seat. The available solutions all focus on the driver being the main target for their ideas. This project also focusses on the driver being the main user of the application, but they are not limited to being the sole user.

The project is going to be used for Android mobile devices only. In the future, there could be other implementations designed for wearable technology, tablets and eventually IOS devices but for this project Android is the only developing platform that will be used. This is because of the market share of Android users compared to IOS users. Figure 1.0, taken from businessInsider.com, [3], shows Android is clearly the more popular choice for users across the globe and has been increasingly so over the last decade since its release which helped influence the motivation of using Android. Other factors include owning an Android device and that the programming language for Android development is very similar to Java, which I have had experience with at university. Android itself is an open-source operating system that was introduced by Google and is considered the World's most popular mobile operating system. [2] For this project, the application created will be developed within the Android Studio Development package. The package has many different libraries and API's that can be called and used within the project. The aim is to integrate many of these libraries along with new code to create an application that gives drivers the ability to communicate with each other in order to find faults within their respective vehicles.

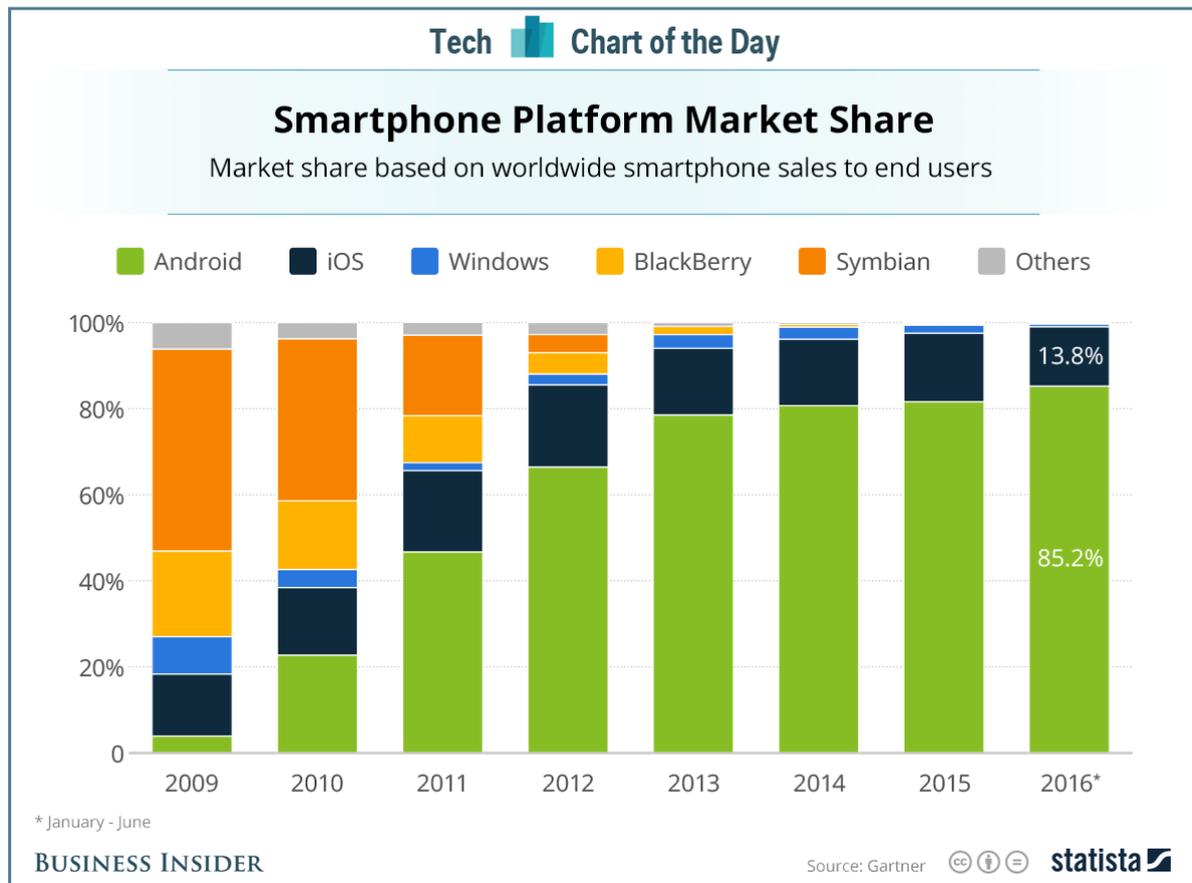


Figure 1.0– Android Market Share compared to other platforms [3]

2.1. Applications that I will use throughout the project

Android SDK justification

The Android Studio Development Kit software program is the dedicated platform for android application developments. It is constantly being updated by the Google Android team and provides the latest packages ready to use for any developers. It is now the official Integrated Development Environment (IDE), providing lots of support, tutorials and videos which can provide detailed documentation and help for someone developing applications in the software. It also provides the ability to plug in your own android devices to test current builds of your application and to test functionalities directly rather than using an emulator. The programming language that will be used is the Android programming language. This language is very similar to Java and incorporates its syntax and structure into its language, with the added functionality of working on an Android device. The studio provides an in-depth IDE that allows developers to create user interface elements in XML or by dragging and dropping components on to the screen, accommodating for both novice and expert users, this will help in the development of the solution to the problem being solved.

This is in direct contrast to the Eclipse IDE. Eclipse is no longer receiving support for Android developer tools which would be vital for maintaining and developing an active application within the market. Eclipse will not be receiving the latest updates from the Google team as they are directly working with Android Studio instead. This was the main reason for choosing to use Android Studio over Eclipse as there are more benefits for Android Studio over Eclipse in the current working environment.

SQLite Justification

In this project, I will be making use of a database that can act as the verification and storage for user accounts and the messages sent between users. I have opted to use SQLite because it has many built-in functionalities for development within Android Studio itself. There are many tutorials on the Android Studio website that can be used should the need arise to familiarise myself with the package. It also provides numerous classes and instances that can be used as a starting point, for example, establishing connections from the application to the database, rather than having to create the code from scratch. It uses the SQL language which I am familiar with from experience at university. The program also comes with a downloadable application that has a very user-friendly interface that you can use instead of having to create tables and records within the command shell.

2.2. Libraries and modules

Android and Java Libraries

The Android Developer website provides lots of Application Programming Interfaces (API) and frameworks, with complete classes and methods, that will be very useful for developing an Android application. This will be helpful during the project as I am new to programming for Android and mobile development. Below are some of the main frameworks and APIs that will be the most useful throughout the project.

A library that can be used for this project is the **Java.Security** module which has many classes and functions that will provide the security framework of the application. As discovered during research, the current Vehicle to Vehicle (V2V) communication standards use public key cryptography security methods to encrypt vehicle communications. Therefore, by using this framework in the Java language classes such as “KeyPairGenerator” offer the ability to create a key pair that can be used to send encrypted messages over the server. There are multitudes of other useful classes and functions within the Java.Security framework that could be used throughout this project.

Another library that could be used is the **Android WiFi Peer-to-Peer API**. This API contains magnitudes of methods and listeners that will allow development of an application that uses fast connections over Wi-Fi to connect to the internet, for example, when connecting to the database or to the server. This API has multiple classes that could be useful, i.e. the “WifiP2pManager” class that provides methods that would effectively allow the connection of devices. This is important for the application if I take the approach that will allow users to directly send messages to each other through Wi-Fi direct or similar methodology, rather than directly going through the server.

Figure 2.1 below shows the typical layout of an Android application. These applications are not just comprised of your typical “code files”, they include resource files and a manifest file as well. Resource files dictate how your application is laid out, what menu options are available etc. Android manifest files include a number of options for the application such as permissions, any third party libraries that need to be used and a list of all your activities within the application. The application will follow the typical Android application layout in the style of this diagram.

An Android Application takes the following format:

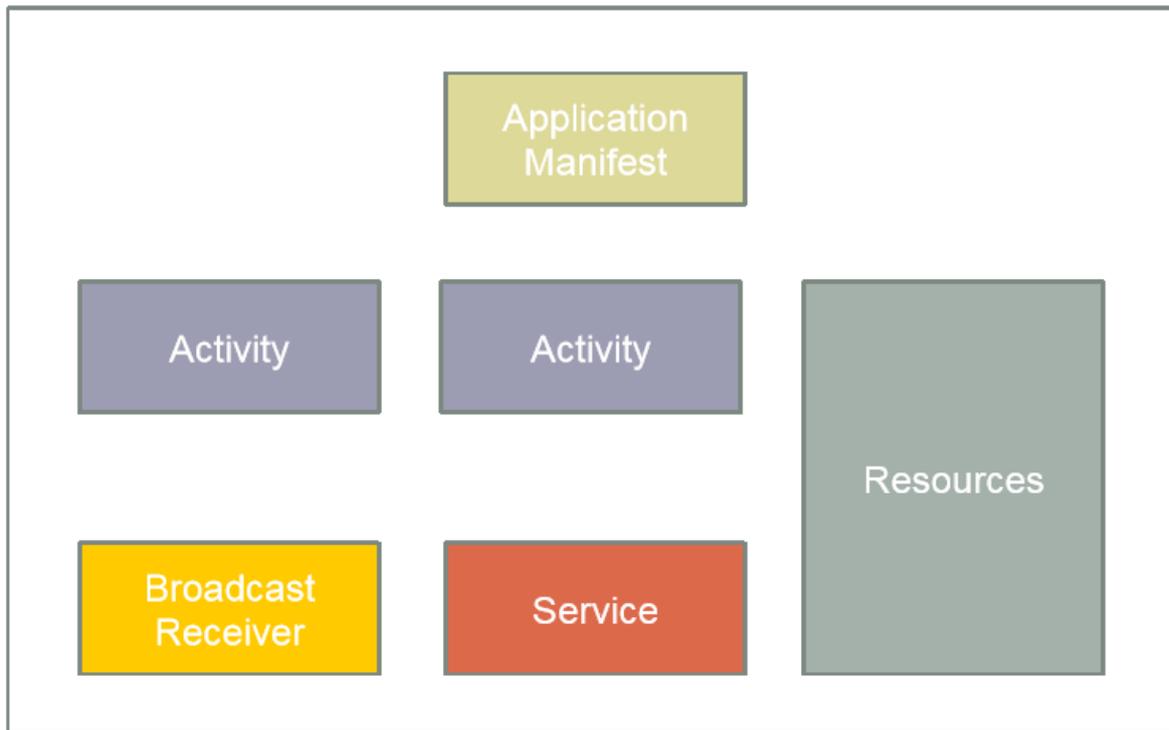


Figure 2.1 – Android Application Structure [4]

2.3 Current Solutions

There are a limited number of solutions in practice today that are implemented using a device within a vehicle allowing you to communicate with other drivers. There is a large focus on providing communication within vehicles to relay traffic information and important updates such as accidents and maintenance, but only one solution that actually looked to communicate between drivers themselves. This solution is called The Bump Network.[5]

The Bump Network is similar to the project proposed as it tries to solve the lack of status updates from your car to the driver in older vehicles. Bump uses an application to scan a registration plate, which forms a unique address with an email address you use to sign up to the network, therefore, allowing drivers to contact other drivers via text, email or voice messaging. Both users would have to be signed up to the network to make use of the application. This solution is different to the idea that I am proposing as this allows free communication between the drivers, literally so that they can send anything they want. The solution being implemented will aim to provide ready to use statements that will prohibit offensive comments and messages being sent, something that Bump does not seem to restrict.

Bump provides location specific details that relate to the location of your vehicle which allows friends or family to monitor your location. This is something that this project will not consider because that data is private and the application that provides that data could become quite sinister if people can access your location without your knowledge. Bump is currently only available within the USA which is different to the solution proposed as it aims to be implemented within the UK. Bump is a great idea as it offers a way of communicating between drivers. The idea of the photo application to scan the registration plate is a very good idea and something that could possibly be adapted for this solution. Usage in regards to the photo aspect of the Bump network was unclear during research as it did not

specify, whether it is through a mobile device or through a camera on the vehicle itself. If it is through a mobile application, then there are clearly safety risks that need to be addressed that were not publicly available. There seems to be very little update on the project after 2013 which leads to speculation that the company is still trying to implement their solution, focusing on driverless cars or that the company no longer exists.

Another solution that aims to implement vehicle to vehicle communication is the Car 2 Car Consortium. This is a non-profit industry driven organisation that is supported by many vehicle manufacturers, equipment suppliers and researchers. They are supported by vehicle manufacturers such as Land Rover, Audi, Honda and BMW. However, they are more focussed on increasing road safety and efficiency by introducing Co-operative Intelligent Transport Systems (C-ITS) throughout Europe, with the overall objective of introducing validation processes for vehicle to vehicle communication systems. This company has a lot of international backing and could pave the way for solutions like the application proposed to become readily available in a safe environment for drivers and citizens alike. As the Car 2 Car Consortium is an globally recognised company, they will be able to gain international backing and increase the probability of legislation being created and passed that will allow for vehicle to vehicle communication systems. They are not currently implementing any way for drivers to communicate with each other, but are more focussed on legislation allowing communication between vehicles. [\[6\]](#)

There was some consideration into whether a server would be the best way to send messages between drivers for the intended application. Roadside Units(RSU) and adhoc vehicle networks are a possibility for creating the communication hubs that relay information between drivers and the server. A research paper, written by Andre B. Reis et al, a Computer Communications and Computer Security researcher, that I considered provided some clear issues with using such methods.

“When a vehicular network is formed, the low density of vehicles under such conditions leads to network sparsity.” (Reis et al. 2014) [\[7\]](#)

This quote suggests that if a low number of vehicles are using the networked RSUs at any one time, then the messages between them could lead to delays in communication. The paper does go on to discuss benefits of using connected networks of RSUs and how that can have a great improvement on the delay time caused between sending messages between nodes on a highway. This could be implemented in the future, with the application taking advantage of RSUs, instead of the server acting on behalf of an agency like the DVLA. The RSUs would form the backbone of the server and directly connect to the DVLA servers, allowing them to communicate messages to drivers based on their travel location rather than directly contacting their account on the system. This would be based on RSUs being built along every road, so it is unrealistic for the foreseeable future because of the expense to do so, as stated by Reis et al. 2014 [\[8\]](#),

“the cost of deploying and supporting RSUs in vehicular environments can be very high.”

Currently, using a server fits the aim of the proposed solution and does not have to factor in potential delays in message communication because the application will be mainly focussed on issues that are not life-threatening for example, a brake light being broken.

Vehicular Ad-Hoc Networks (VANETs) seem to be increasing in popularity and more and more research is being done into this field to try and build up towards implementing Vehicle to Vehicle (V2V)

communication systems. Using vehicles as nodes for Wi-Fi networks is definitely something that could be considered in years to come, however for this project it is not currently viable, therefore, using a server to act as a database is the best approach.

Transportation Technology and Mobility Editor, Pete Biglow, sets the scene in his blog for the future implementation of V2V communication, [\[9\]](#)

in Washington, D.C., federal Department of Transportation (DOT) officials unveiled a long-awaited notice of proposed rulemaking. If a final rule reaches fruition, it will mandate that all new vehicles contain equipment that permits vehicle-to-vehicle (V2V) communications. Regulators say this connected-car technology could one day prevent thousands of deaths and tens of thousands of crashes every year by offering drivers real-time alerts of imminent dangers ahead.

The blog article certainly paints an image of possible implementations in the future that could be used to build on existing solutions like, The Bump Network and Car2Car Consortium. Current solutions make use of the public key security method in their V2V communication systems. This practice will be taken forward in this project and be implemented as the communication system with the use of certificates to authenticate the messages between drivers.

Car manufacturers are already implementing high-tech dashboard and display units within their latest car models. These display units allow drivers to see lots of information regarding the vehicle's faults, i.e. broken lighting systems, flat tyres, software updates required for latest onboard technologies like Bluetooth etc. The latest Ford Focus has an array of technology features that helps detect all sorts of problems, that the proposed application could not hope to implement at this stage of the project.

"Ford SYNC 2 with Voice Control and Touchscreen takes the existing infotainment system a step further with the addition of an optional 8" coloured touchscreen. This high-resolution display gives intuitive control of several key functions including phone, audio, climate control and optional navigation via voice or touch." [\[10\]](#)

The display units demonstrated by Ford are a vital step forward to the future of V2V communication as they could allow for applications like the one proposed to be situated within, to allow drivers to communicate with each other. However, these new technologies and systems in modern cars also present a problem regarding the electronics and how a single fault in the system could cause these features to stop working entirely. Where as if there was an application, like the one being proposed, that operates without relying on the features of the car working correctly then there is always that chance of receiving updates regarding the status of your car from other drivers who notice issues with said vehicle. A challenge to overcome with the proposed solution and the solutions developed is how to implement communication features in a safe environment without distracting the driver. Having an 8" screen will be great and help drivers to find errors within their vehicles but it is a big distraction that takes the driver's focus away from the road and their fellow drivers. There are still safety risks to be resolved within the newest technologies that are being implemented by car manufacturers. The benefit of what the intended solution and the Bump Network set out to achieve is communication between drivers and not just communication between the vehicle and yourself, the driver.

2.2 Constraints

There are a few constraints that must be considered with this project. The application that will be created is targeting vehicle owners and mainly their drivers. As it is a mobile phone application that will simulate potential use for the application to sit inside a car's technological capabilities, it is not possible to undergo a testing period in an actual car. This is due to the safety risks it could entail with the driver and their fellow road users. Not to mention the possible consequences for a road user who is testing the application should they encounter law enforcement agencies. With the prohibition of mobile phone use in vehicles, the project will opt to use mobile phones as "vehicles", simulating the real application functionality. The application will be tested vigorously in an environment suitable to the project, i.e. using a number of instance applications that will be "vehicles" for testing purposes.

2.3 Potential Stakeholders

There are many potential stakeholders within this project's scope. For example, drivers who drive everyday through traffic filled streets with lots of cars on the road. These drivers can provide lots of information to other drivers regarding the status of their cars, as well as being informed about the status of their own vehicle. This could result in multitudes of information being passed from driver to driver, providing fixes to minor issues within their vehicles. The alternative being that they find the problem themselves after someone points it out to them, i.e. a family member informs them or they are pulled over and made aware of the issue by a member of law enforcement. The latter being the worst-case scenario, if they were then given a penalty or a fine rather than a caution.

Other stakeholders include organisations like the DVLA or Police Force who can use the application to potentially deal with unlawful drivers. For example, if someone fails to fix a brake light after being warned by another driver after a certain time-period passes, then that vehicle owner could be fined. This would take the application in a different direction than intended, but could be considered in the future.

Another stakeholder could be vehicle manufacturers as they can incorporate the application into future car models. The application could sit within their own car based systems that would allow their drivers to pass on messages to other vehicle owners who have older cars.

3. Specification and Design

This section of the report will detail the intended user interface of the application at a high level as well as system diagrams such as flow charts, database designs and why these options have been considered. Most importantly it will document the projects specification, aims and objectives and how the project will set about achieving those aims.

3.1 Aims and Objectives

The objectives of this project have been thoughtfully considered and aim to provide the best working solution to the problem that is being solved. The original objectives described in the initial plan have changed slightly but the overall objective of achieving driver to driver communication over mobile devices remains the same. The main changes from the initial plan will be discussed in the [Implementation](#) section of this report and all reasons for making those changes will be justified there.

To re-iterate the problem in more detail. The problem is that there is no way for drivers to communicate with each other if they spot a fault or a distraction that may help another driver. For example, if there is a vehicle with both brake lights that are not working, a driver can send that message to the recipient's application who can then do something about that problem. Similarly, if a driver reports an accident that they are currently stuck in traffic because of, they can alert other drivers to these accidents so that less congestion builds up. This is a problem that many motorists see every day, with many drivers assumingly unaware of the faults within their vehicles. The ability to communicate between drivers, would seemingly solve this problem.

Below are the aims and objectives of this project. Broken down and described in detail so that the project's goals are clear. These aims and objectives are the criteria that will be used to solve the problem at hand.

1. Provide a complete application that provides users with a fully functional car messaging system.
2. Allow users to send messages from one device to another device regarding the status of their respective vehicles.
3. Provide a simple and easy interface for drivers to communicate with each other.
4. Store all data used within the application securely and in a way that does not compromise the performance of the application itself.

Aim 1 – Provide a complete application that provides users with a fully functional car messaging system.

This is an important aim of the project as it will shape the entire functionality of the application by giving users the features that they would expect with the given solution. For example, the project aims to provide users with the ability to send messages between drivers, to view their own messages within the application and to report and monitor traffic incidents. This functionality is important for the application because it allows drivers to communicate with each other, which is the fundamental aim of the project. Reporting and monitoring traffic incidents would allow drivers to change their desired route if they discover a problem on the road they were going to take because of an accident, instead of sitting in traffic because of that issue.

Aim 2 – Allow users to send messages from one device to another device regarding the status of their respective vehicles.

This objective is the main function of the application. If this is achieved in the implementation then the project could be deemed a success because of how important it is. The way this project will set

about achieving this objective is by implementing a server with the ability to pass on messages from one device to another device. The idea for this objective is that one user sees another driver's vehicle fault and reports it by sending a message, using the application to select a fault and input the registration number of the other vehicle. When they send the message, they will connect to the other phone through the server and the message will be displayed for the other user, notifying them of the fault within their vehicle.

Aim 3 – Provide a simple and easy interface for drivers to communicate with each other.

If the interface is not easy to use, then drivers will not use the application. The interface must be simple, so that users can use the application without previously experiencing the application. A good interface should have clear labels and text, useful hints where appropriate and its functions should be self-explanatory. For example, if a button says, "send new message" then that button should be related to sending a message. If it is not then it will confuse users and that will fail this objective. The application will follow several design heuristics that will maintain a clear and simple user interface. These heuristics are adapted from Rosa Yáñez Gómez et al, a Researcher from the University of Sevilla, [\[11\]](#)

- User control and Freedom
 - Users that choose a function within the system may get lost, there should be adequate ways for them to return to previous screens, undo or redo the last action.
 - E.g. a user entered the wrong form when they meant to click the send message form. This should be easily reversible.
- Help users recover, diagnose and recover from errors
 - A helpful error message should be displayed to inform the user how to correct their mistake. Whether it was a system fault or the user fault should also be displayed.
 - E.g. Check user enters a registration number while sending a message, and if not then display a notification telling them that they must include this field to successfully send a message.
- Aesthetic and minimalist design
 - Only information that is relevant to the screen that the user is on should be displayed for the user. There is no need to display irrelevant information that could confuse the user, or takes away the focus of the relevant information.
 - E.g. For sending a message, there should be no mention of how to read your messages, the form should focus on how to send a message only.
- Consistency and standards
 - Platform standards should be followed to achieve consistency and standards of the application. Different words, or actions should not be used as they could cause ambiguity and confuse the user.
 - E.g. Does "Send Message" mean the same as "Create Message" or "New Message" and if they are, then one phrase should be used throughout.
- Match between the system and the real world.
 - Familiarity is important in the application so that words and phrases are similar to those that are used in real world situations.
 - E.g. Registration number should be used rather than License plate as the application is based in the UK. Information should be displayed in a logical and natural order so that users can follow it easily.

These heuristics will be followed to ensure that the application meets this objective.

Aim 4 - Store all data used within the application securely and in a way that does not compromise the performance of the application itself.

The application will be storing some sensitive data such as registration numbers and surnames. That makes it vital to that there are adequate security measures in place within the application to prevent data being stolen or apprehended during transfers between users and the database. To complete this objective, encryption and decryption measures, for example, the Java.Security API, AES and padding will be used to protect data during transmission from users to the database and from user to user. This will look to stop eavesdropping attempts and unauthorised access to the data. Passwords will also need to be stored securely to avoid unauthorised access to the application itself.

Non-functional requirements of the project are required to ensure that the application runs to the standard that users expect on their devices, without overloading current device specifications, therefore, keeping the device running smoothly. The non-functional requirements that I hope to achieve in the project are:

1. Performance
2. Security
3. Reliability

Performance will be measured by the CPU, Memory and Network usage that the application requires to run effectively. Android Studio provides a monitoring system for connected devices that check these statistics to show the current levels being exuberated by the application on the device. I can monitor the hardware usages to ensure that standards are maintained and these parameters are not overly high and cause issues for the user and their device.

Storing the data securely and protecting the data being transferred between users is a vital aspect of this project. As well as ensuring devices are protected from other devices accessing their data through the client to server connections that can be formed. For example, if I implement client to server architecture for sending messages between users on different devices, the user sending the message from their client device to the server device should not be able to access any components on the server device and vice versa. The connection should be terminated after the initial message has been sent from one device to the other.

Reliability is an important feature for the application. It is important that the application is consistently available for the users and does not contain errors or bugs that cause the software to function incorrectly. The functionality of the software should be responsive and available to the user throughout their experience. And if for some reason it is not, then appropriate messages should be displayed to the users detailing why they cannot access a specific feature.

3.2 System Design

The user interface has been designed using software called Balsamiq. The software allowed me to create a number of interface diagrams that used a real mobile device template to show a realistic interface for the application itself.

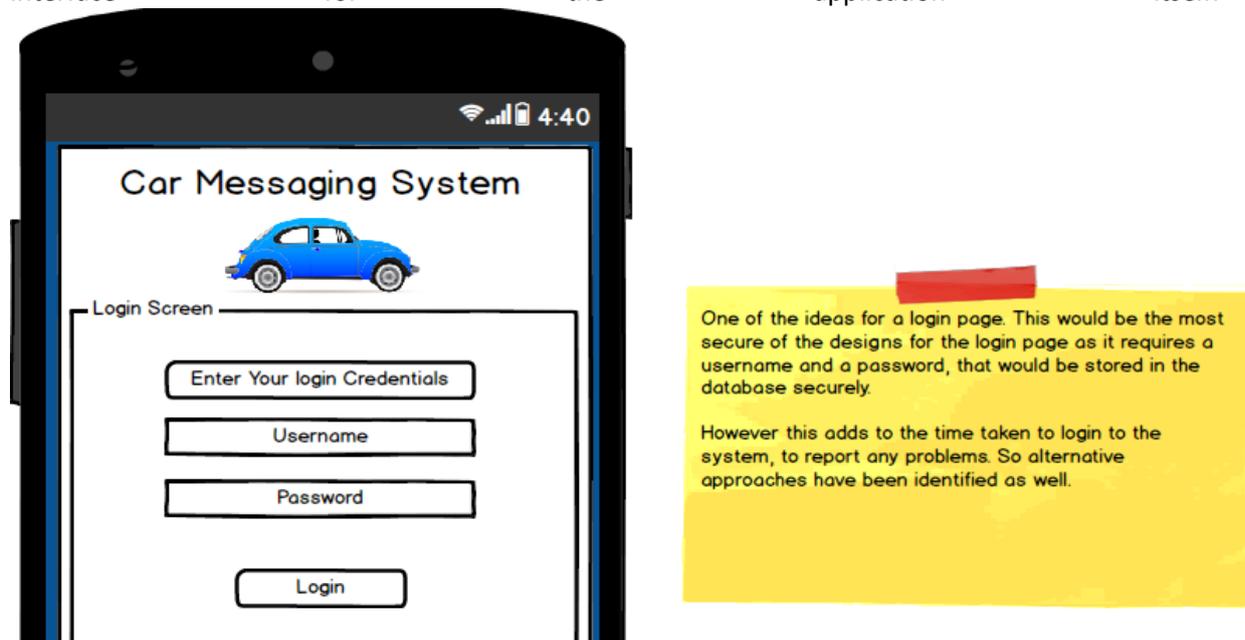


Figure 3.1– The Login screen design

This design for the login activity is a classic example of login screens seen across multiple applications and websites. The username and password approach is the most familiar method in which users will log into applications, therefore, making it a suitable design choice. The credentials required on this activity will be set by the user in the create account activity. Both of these fields will be stored in the database and will correspond to a user when they log into the system. The password field will be hashed out when a user types in their word, as is typical with many password textboxes across websites and applications, for example, password would be *****. The username field will be automatically generated on the create account activity, which is discussed in the next design and the reasoning behind this approach.

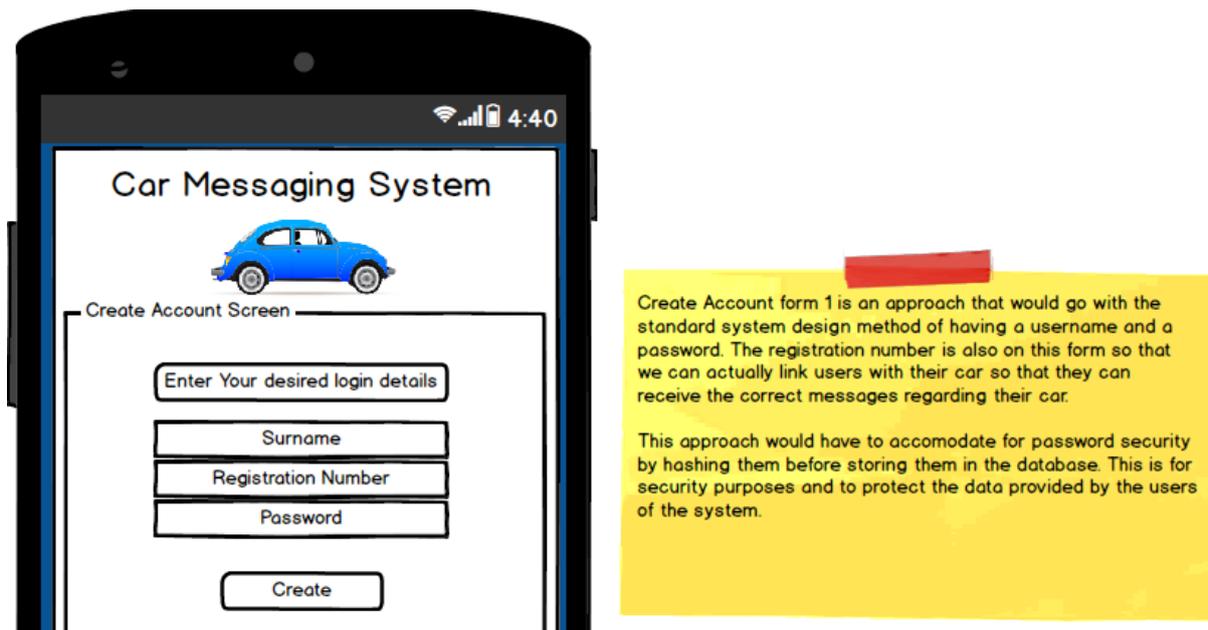


Figure 3.2 – The Create account design

The create account activity is an approach that follows the login screen design provided in Figure 3.2 as it asks users to fill in the necessary fields. As stated in the side note for Figure 3.2, the registration number is a necessary requirement as it will be required to link users to their cars. A field that will also be required on this form is the Surname field. This field is vital in the generation of the UniqueID field, which is a combination of the registration number and the surname. The uniqueID field will be used to log into the application. The choice of including this UniqueID that is generated by the system rather than the user is purely for reasoning of making unique usernames that are relevant to the user. It would make it easier to remember and is not seen by other users so it won't be deemed giving away sensitive data. For example, a UniqueID formed from the surname Hutchings and AB11CDE would be Hutchings-AB11CDE. This is easy to remember for users as it is just their surname and their registration number.

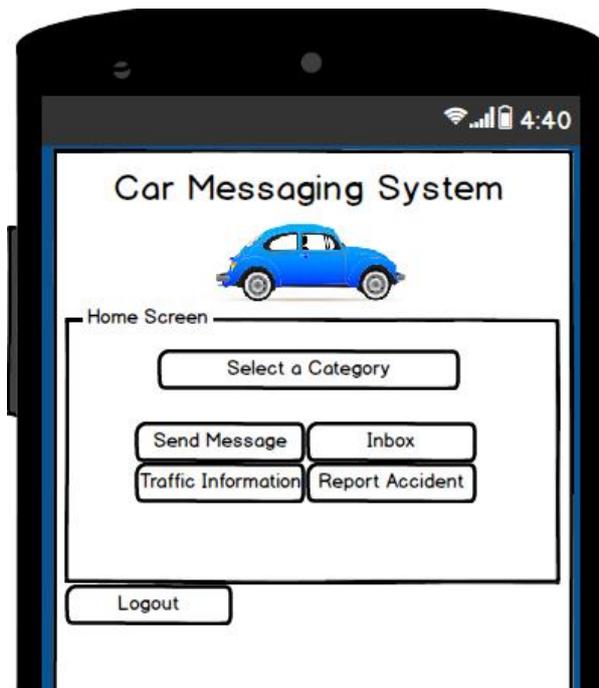


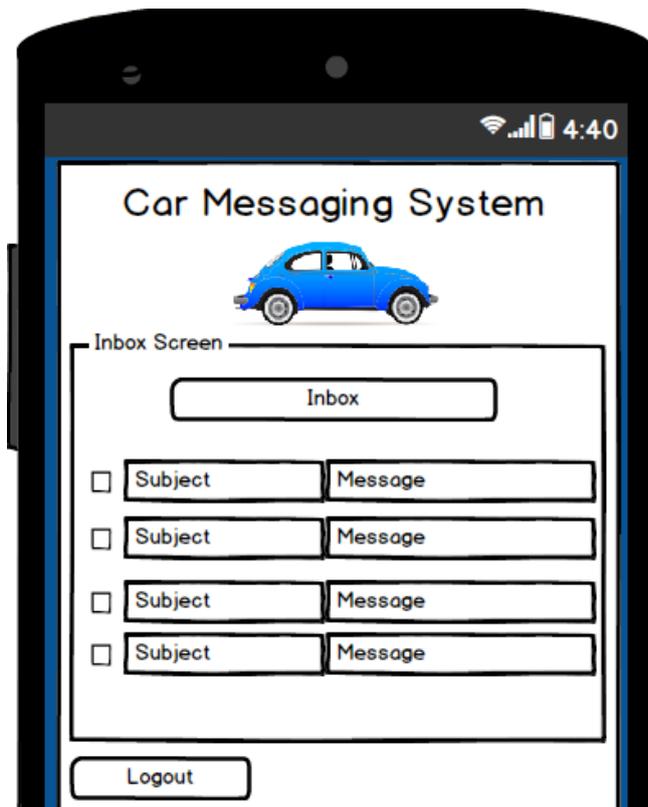
Figure 3.3 – The Main menu screen design

An alternative approach to the menu form. This is the advanced menu page that would be available if I have the necessary time to fully implement all of the functionality of the system.

The Traffic information and Report Accident functions would provide more usability and greater information to the users through use of the application.

The reasoning behind this approach, is that I want to create a very simple and basic user interface that provides all the necessary functions for users, without any complex designs that might take the focus away from the application's main functionality.

As stated in Figure 3.3's side note, the project aims to implement a very simple interface and menu screen that allows users to go to the activity they want quickly and without having to wonder how to navigate to a certain screen. The boxes are informative and tell users exactly what happens by clicking on them and provides a clear way of navigating the application, following the design heuristics discussed in section 3.1. The buttons use phrases that would be expected by the users, containing information that is relevant to the users displayed on the form and the consistency is kept between the text size and box size on the form. The consistency is evident from the logo and the title header that is on show at the top of the activity.



The simple design of the Inbox screen is based on Google's Gmail account design. It is simple and effective. Providing users with all the information they need on screen in an easy-readable manner.

The message formatting will be put together in a script, taking the information from the database table that stores the information received from incoming messages

Figure 3.4 – The Inbox screen design

The Inbox design is based on Google's Gmail account setup, as stated in Figure 3.4's side note. The simple design provides an effective layout of the activity and creates an easy reading experience for the user. As the messages are all pre-set and can be displayed on a single line. The message will include the problem with the car, the registration number and the time and date the message was sent. These are important for the user to be able to diagnose the problem with their car and work out when it was from, as they may have already fixed it by this point.

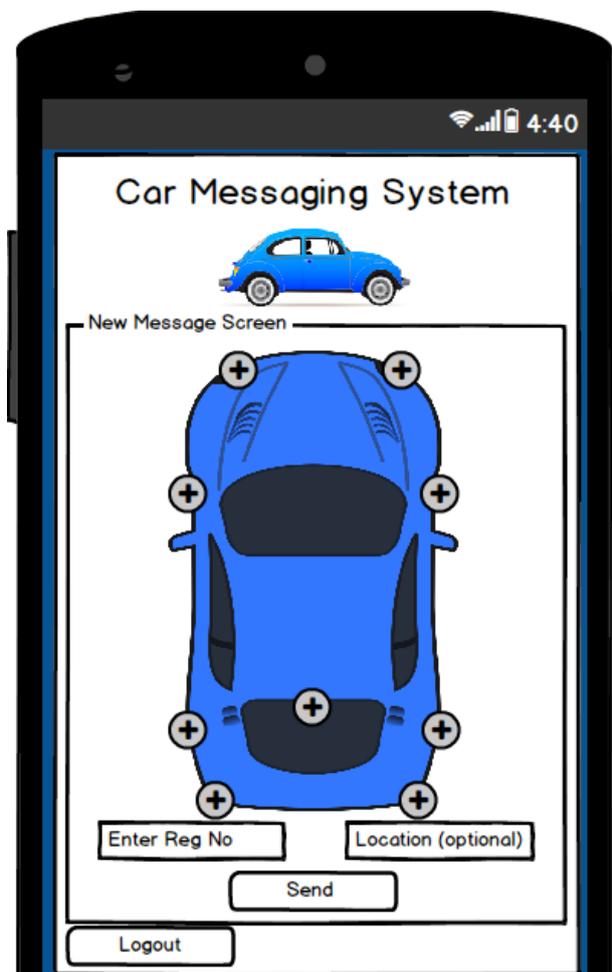


Figure 3.5 – The send message screen design

The best approach that I think for the new message screen is this one.
An interactive image of a car where users will be able to select a fault from touching that part of the car. By touching the plus icon on the image, that will give users a preset fault with the car that they have selected.

This provides the easiest approach to the messaging screen. Having a picture also provides the user with the easiest method of selecting the part of the car that had the fault. Let's say you've just noticed the brake light of the vehicle in front of you is not working. By the time you use the application you might not remember which side it was on, but by seeing an image of the car itself, that may remind you because you can physically see the object.

This is the new message screen that will be created within Android Studio for the application. It is an image of a car where users will be able to select parts of the car that are not working, for example selecting the front left plus icon image, automatically setting the message to be "Front Left headlight is faulty". This gives users pre-set messages that they can select to send about another car. The reasoning for this design choice is that, when creating the application, the option to allow users to write their own messages seemed like a bad idea because it can quickly become a messaging service that supports abusive messages, spam messages or any kind of inappropriate messaging. This way it restricts users to the options provided and limits their choices to focus on the problem. This is different to the Bump Network's approach (see [Background section 2.3](#)) because they allow for a user to send any kind of message that they want, which allows for rude or abusive comments. This method would eliminate any kind of abusive messaging. This design also provides the simplest and most effective way for users to actually select a fault. It means that permutations of the selected fault do not become too ambiguous or vague, for example, "Headlight not working" or "Left light broken", etc. It provides a uniform approach to sending an informative message and lets users relay the message quickly. The only challenge with this approach, is providing enough options for users to report the faults they see. This can be overcome by providing logical faults that a user is likely to be able to see.

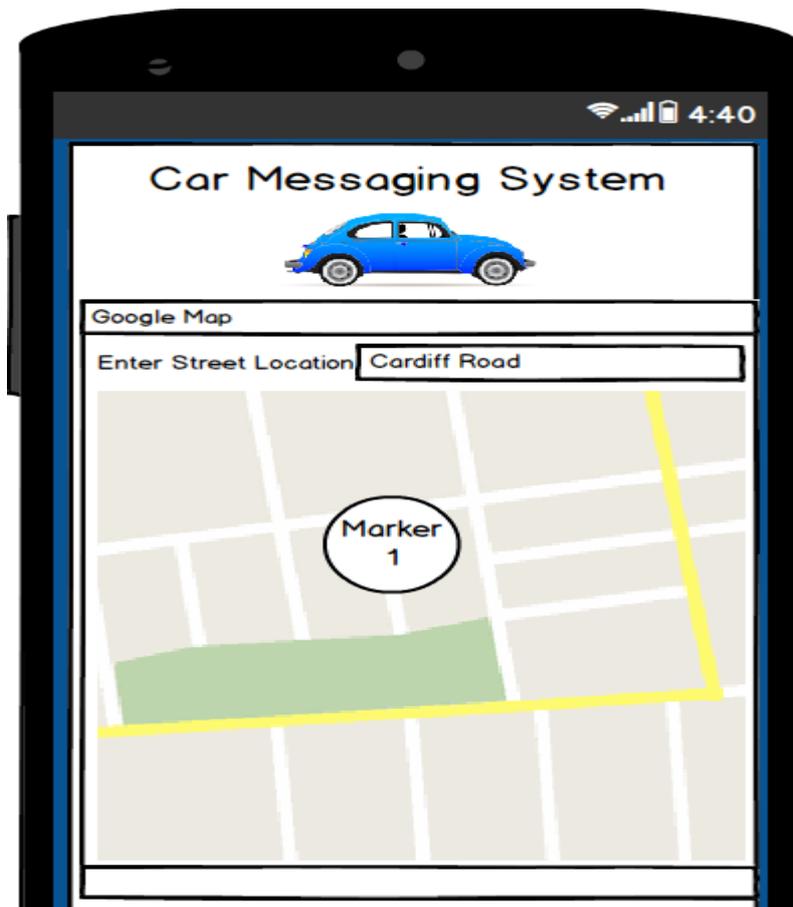


Figure 3.6 – The Report Accident screen design

The Report Accident design shows how the application will implement the use of Google Maps so that users can insert their own markers showing other drivers, where accidents have occurred. The design for this activity was selected to provide users with a familiar looking screen helping to make the application more user friendly and increasing ease of use. By providing a Google Maps interface, users will not have to learn how to interact with this activity, they can use previous experience gained with the Google application. The user will be able to easily tap where they saw the accident and a marker will appear at that location. This is a simple and effective design that makes use of APIs available within the Android Mobile development platform. The design will make use of the user's location if they provide that permission. This way the user's location can be pinpointed by the application rather than them having to find themselves on the map, further increasing ease of use and reducing interaction by the user which could cause distractions.

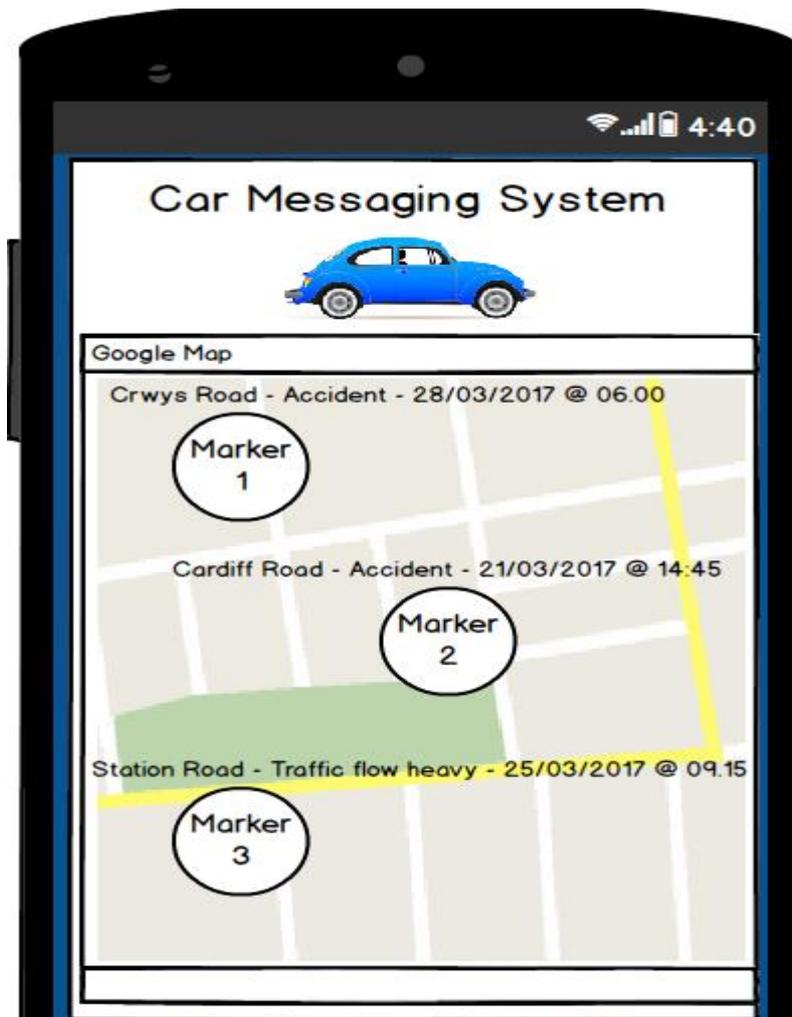


Figure 3.7 – the Traffic Information screen design

The Traffic Information activity design demonstrates how the users will be able to see the reported accidents or traffic information from the Report Accident activity. Each marker will be shown with an information panel above that shows the road name, the problem and the date and time that the issue occurred. The design is simple and informative as it is just a display feature for users to view.

3.4 Database Design

The database design consists of three main tables that will be used throughout the application. The Identifier table which will store all the user's information, the Messages table which will store all the messages received from users and the AccidentInformation table which will store all information about reported accidents.

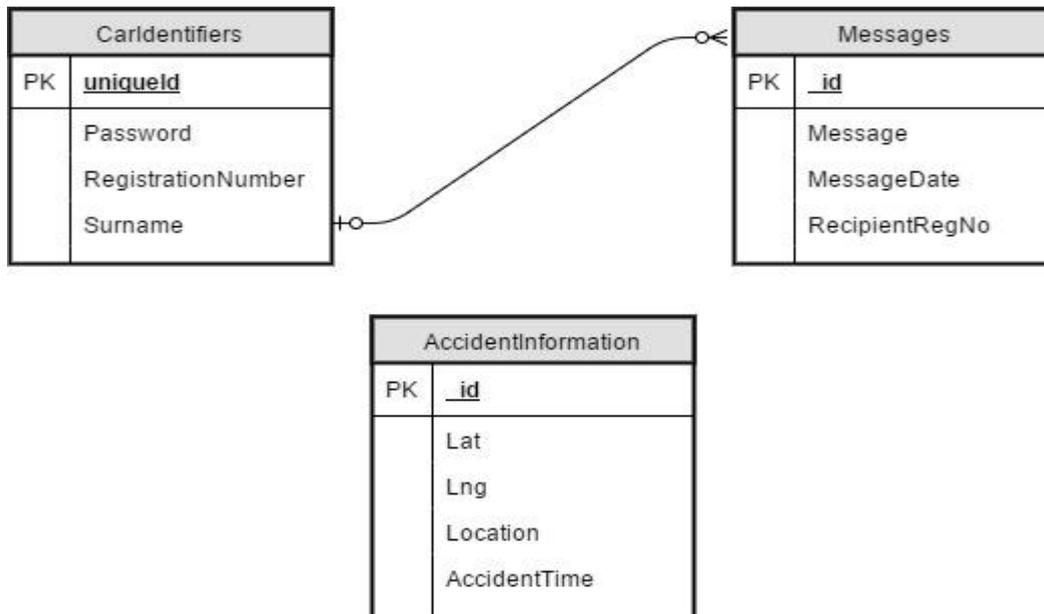


Figure 3.8 – Diagram of the Database tables and their fields

The Identifiers table will have a one-to-many entity relationship because a single user can have many messages at any given time. Users could also have zero messages hence why it is an “optional many” rather than a mandatory one. The RegistrationNumber and Surname fields make the UniqueID field which is a requirement in that table as it is the primary key.

The AccidentInformation table requires latitude and longitude coordinates to pinpoint the location of the accident reported by a user, and the location and time will be for reference to other drivers so that they can see the exact timings and area where the accident will have taken place.

Similarly, with the Messages table, the message sent to the user will have to include a message field which will describe the issue with the car that has been reported, the date of the message that has been sent so that the user has an idea of the relevance of the message, and the RecipientRegNo is required to identify the car that has the fault. These three tables will be created in an SQLite database that can be accessed through the Android Studio software and will provide a direct link to the application. Please see Figure 3.8 for further database structure.

3.5 Use Case Diagrams

The use case diagrams in this report look at the main functions of the intended user interaction within the application.

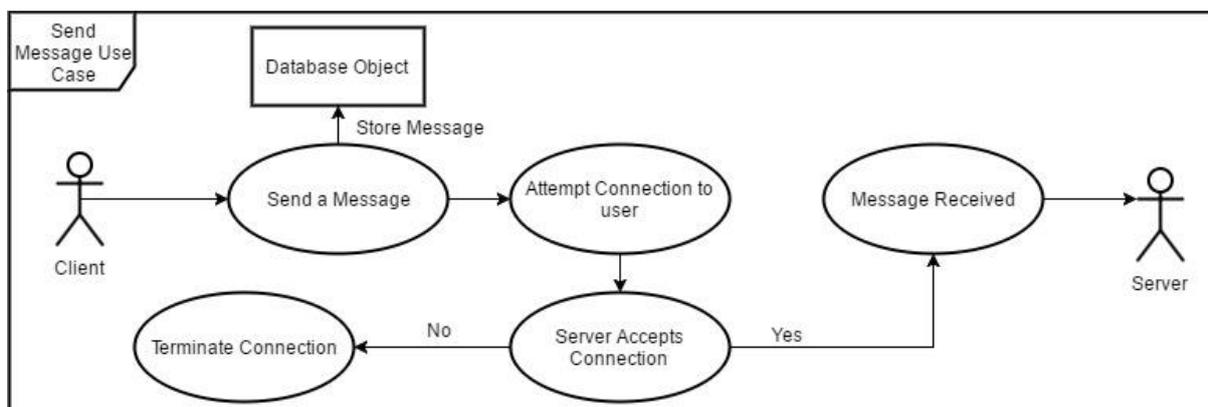


Figure 3.9 – Use case diagram of the intended Send Message Activity

The send message use case in Figure 3.9, illustrates the main functionality of the application and how the solution intends to work. The user will send the message, which is stored in the database regardless of whether the server user accepts to receive the message or not. This message will be available in the server user’s inbox even if they do not accept the connection. This approach will allow for a driver to view the message at a more convenient time for them without distracting them from the road. When the client user connects to the server user, they will connect via a WiFi-direct peer-2-peer connection that is formed from nearby devices. These devices appear on the list of available devices for the client to make the connection. Once that connection is established, the user’s message will be delivered to the server user who will view the message that has been sent.

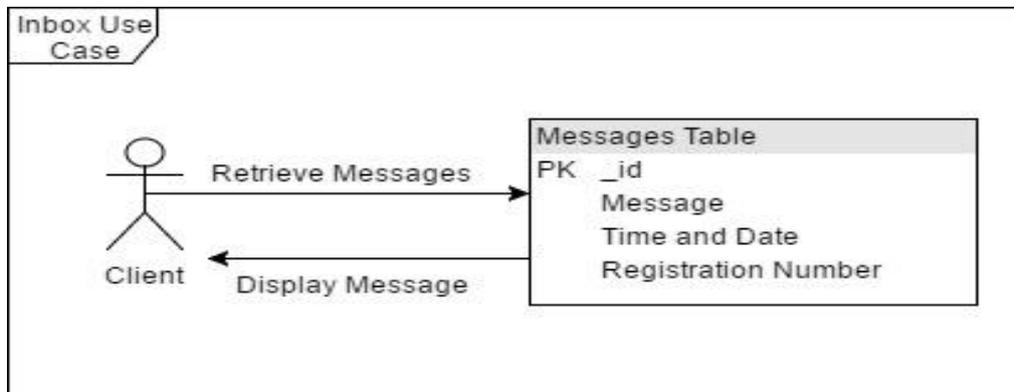


Figure 3.10 – Intended Inbox Activity use case diagram

The inbox use case, in Figure 3.10, demonstrates how the user will retrieve their messages from the database. The messages are stored in the database from the other users who send messages to this client user. The client will check their Inbox in the application to find a list of all the available messages that they can view. The messages will be displayed for the user so that they can see what faults their vehicle has.

3.6 System Flow Chart

The flow chart (Figure 3.11) gives a high-level overview of the intended solution and the processes that can be carried out. The user will always start in the Login Activity and from there will be able to navigate to the two screens that branch from it. The flow chart covers all possible interactions for the user and how each activity will be linked together, presenting the possible outcomes of implementing a complete Car Messaging application, discussed in Aim-1 in the Approach section. The complete application would have these extra functionalities such as the Reporting Accidents and viewing traffic information activities.

The flow chart demonstrates a typical experience for the user. They must log in successfully to be able to access the features of the application. The login activity will check the user-input against the credentials stored in the database to ensure that they are an authorised user. If not authorised, the user cannot log into the system. From the main menu, users have a choice as to what functionality they want to carry out while logged in. They can check their inbox for new messages, send a new message, report an accident or check for traffic information. The send message and inbox activities are directly linked because of their relationship. New messages are sent to the database and to the user directly, by checking the inbox, any new messages will be displayed. Similarly, the Report Accident and Traffic Information activities are linked as all new accidents will be displayed in the traffic information screen. The server activity correlates with the send message activity, because the server activity is where a user will listen for messages from other users. If a user is trying to connect to the server activity to send a message directly to the user then the other user needs to be in the server activity.

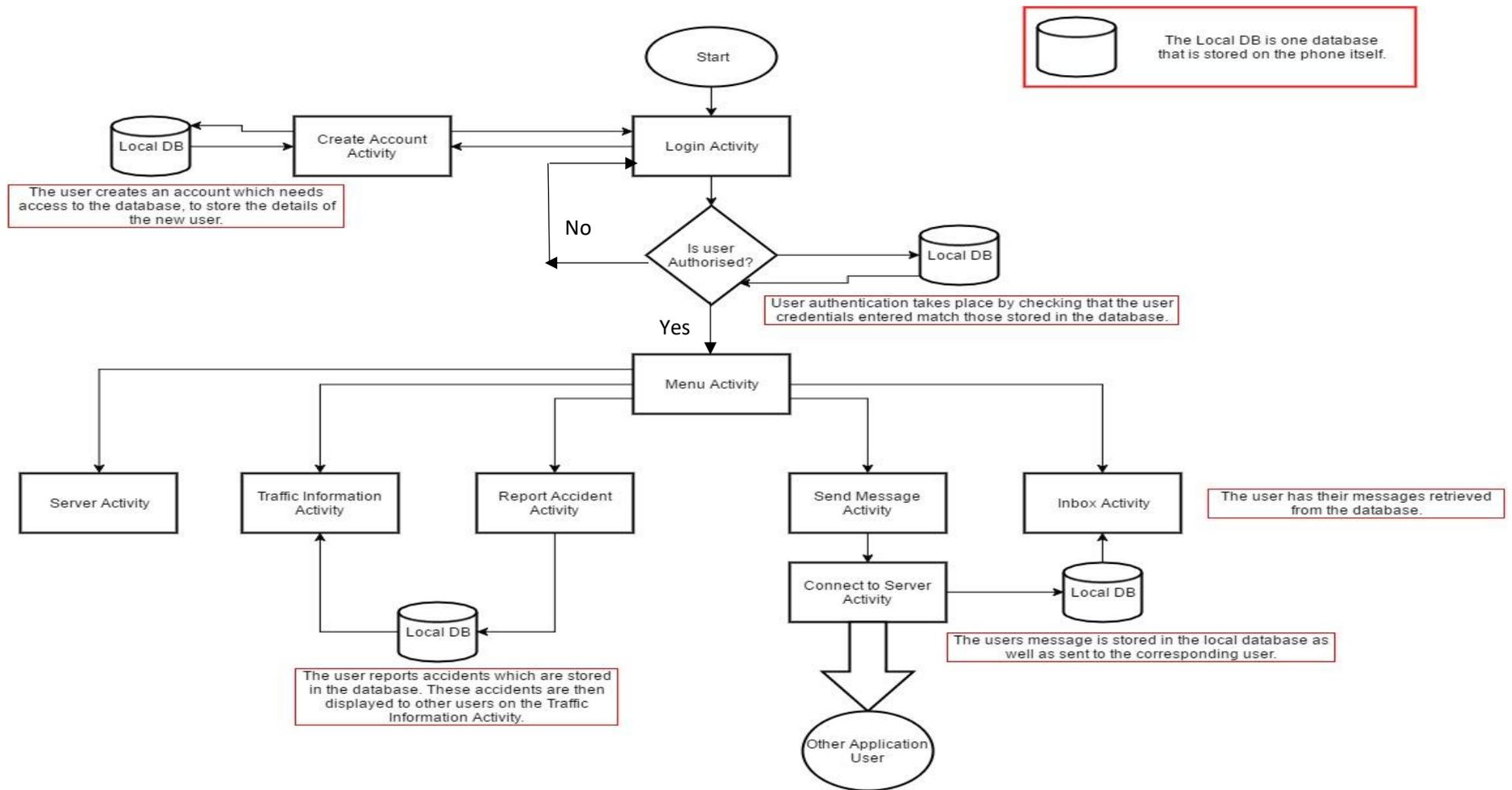


Figure 3.11 – Flow chart of the Intended Application

4. Implementation

This section of the report describes the execution of the approach and methodologies used to solve the problem. The main aspects of the code that I produced will be detailed in this section, with reference to why some parts of the approach were not created and reasons why they were omitted.

4.1 Changes from the initial plan

The changes from the initial project plan and the design phase are listed in the table below, with a quick summary of what has been implemented in their place.

Initial Plan Objective	Approach Objective	What has been implemented
Develop a functioning mobile application which allows users to create, send and receive messages regarding the status of their vehicle	Allow users to send messages from one device to another device regarding the status of their respective vehicles.	Functionality within the application that uses client-to-server architecture to send messages between devices
Create a server that will create certificates for registered drivers, and relay messages between drivers.		The server is created within the application on the device, and is used to listen for incoming messages.
A function within the server that allows an agency or "Governing Body" to send traffic updates and maintenance schedules to drivers on the road.	Provide a complete application that provides users with a fully functional car messaging system.	Functionality within the application that allows users to check for traffic information and accidents that have occurred on a Google Maps activity.
Allow non-driver registered users to send messages regarding the status of other vehicles on the road but not to receive messages.		This initial objective has not been included in the final implementation, because it was not relevant to the problem aforementioned.
Create a feedback function that allows drivers to rate how helpful the message they received from another driver was.		This has been implemented in the way of dialog boxes that appear when a user clicks on a message and they can then fill out a star rating bar and leave a comment with their feedback.
Third party information that can be used by drivers to fix any problems with their vehicles. E.g. A link to a garage or high-street shop like Halfords that sells parts for a fault that has been raised.		This initial objective has not been included in the final implementation.
	Provide a simple and easy interface for drivers to communicate with each other.	The entire application is very minimalistic, everything that the user needs to input is explained and only the relevant information is displayed on the screen for them.
	Store all data used within the application securely and in a	The data is not encrypted but there is a security activity that

	way that does not compromise the performance of the application itself.	shows a possible route for encrypting data in the future.
--	---	---

The passer-by camera optional objective from the initial plan was not implemented because the whole application is dependent on a user's registration number and being able to log in to identify that user. This could be changed in the future to allow for the passer-by camera screen but in the current implementation it does not fit and makes no contribution to solving the problem.

The other objective from the initial plan that was not implemented was the third-party information that is provided to users based on problems found within their car. This would have been a useful addition to the project but did not correlate with the main objectives of the project.

4.2 Execution of the Approach

An important feature that was overlooked during the implementation stage of the project was that SQLite is only a local database that is stored on the device or emulator that you use to run your application. This was realised late on in the implementation stage and that I wouldn't have time to recreate the database in a software package that allowed for the database storage to be across multiple devices such as MySQL or phpMyAdmin. The impact of this flaw was large and affected how I could spend the rest of the implementation cycle because the device to device communication was a priority rather than implement a new database and change the methods that had the database functionality. A more suitable database software package that allowed communication to all users across the application rather than local device storage could have been chosen. This would mean that I could have used python and php scripts to connect the database to the Android Studio project like I mentioned in the approach, instead of taking the different approach that I did.

Client Activity Code:

```

100     public void run() {
101         Looper.prepare();
102         mHandler = new Handler();
103
104         try {
105             InetAddress serverAddr = InetAddress.getByName(serverIpAddress);
106             Log.d("ClientActivity", "C: Connecting...");
107             Socket socket = new Socket(serverAddr, ServerActivity.SERVERPORT);
108             connected = true;
109             while (connected) {
110                 try {
111                     Log.d("ClientActivity", "C: Sending command.");
112                     PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket
113                         .getOutputStream()), true));
114                     // WHERE YOU ISSUE THE COMMANDS
115                     out.println(message + " " + regNum);
116                     Log.d("ClientActivity", "C: Sent.");
117                     Log.d(message, "Message Recieved");
118                     out.flush();
119                     break;

```

Figure 4.1 – Client Activity Code

The Client Activity is one of the most important features of the application that has been created. It allows users to send a message from their device to a server device that is listening for commands. It was a challenge to implement because I had originally planned to use the WiFi P2P communication method, in which you connect to other devices in the local area through Wi-Fi direct. However, using the Genymotion emulator software prevented me from carrying out this technique successfully. The devices I created to use as emulators were not Wi-Fi direct compatible and the devices would not find any other compatible device on the network. Therefore, the decision to implement socket programming using a client and server device respectively to forward messages between the users

was chosen. Having to learn and understand about socket programming from scratch was a challenging experience because I had no prior knowledge of how socket programming worked and had not implemented anything of the sort before. The code in Figure 4.1 is a client thread that is run in the background of the application as a service. It checks that the inputted IP address matches a listening server device and if it does, it posts the message that the user sent from the client device onto the server device and stores the message in the server's database. (Adapted from code taken from thinkAndroid, "Incorporating Socket Programming into your Applications." [\[12\]](#))

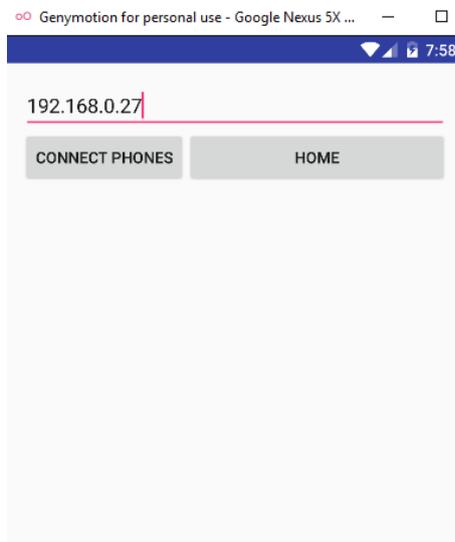


Figure 4.2 – Implementation of client activity

Server Activity Code:

```
68         while (true) {
69             // LISTEN FOR INCOMING CLIENTS
70             Socket client = serverSocket.accept();
71             handler.post(new Runnable() {
72                 @Override
73                 public void run() {
74                     serverStatus.setText("Connected.");
75                     serverUpdates.setText("Incoming Messages...");
76                 }
77             });
78         });
79
80         try {
81             db = new DBAdapter(ServerActivity.this);
82             BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
83             String line = null;
84             while ((line = in.readLine()) != null) {
85                 Log.d("ServerActivity", line);
86                 serverUpdates.setText("Message Received - " + line.toString());
87                 Calendar c = Calendar.getInstance();
88                 SimpleDateFormat df = new SimpleDateFormat("yyyy-mm-dd HH:mm:ss");
89                 String currentDateTimeString = df.format(c.getTime());
90                 db.insertMessage(line.substring(0, 23), currentDateTimeString, line.substring(24));
91                 handler.post(new Runnable() {
92                     @Override
93                     public void run() {
94                         //get messages from client
95
96                         Toast.makeText(ServerActivity.this, "New Message", Toast.LENGTH_SHORT).show();
```

Figure 4.3- Server Activity Code

Similar to the Client activity code, the Server code shown in Figure 4.3, waits for the connection to come in from the client device and then creates the connection between the devices. While they are connected, the server takes the command from the client device, which is the message, and posts it

to the screen of the server device displaying the message to the user. This was an area of difficulty because I had to get the message that was sent from the client device to be stored in the server devices local database, effectively synching the two databases after the new message had been sent. In reference to the SQLite problem discussed, this would not have been so difficult to implement if the database had capabilities to store data across multiple devices rather than just locally. An alternative approach would have been to use a database that would store the message on the client's side of the communication and then use socket programming to display the message to the server user, rather than having to store the message on both ends to synchronise the database. Socket programming in Java and Android was challenging to understand as it is not class knowledge that was taught at university, which meant learning how the sockets worked was difficult, and was required before the code from ThinkAndroid [\[13\]](#) could be adapted to meet the project requirements.

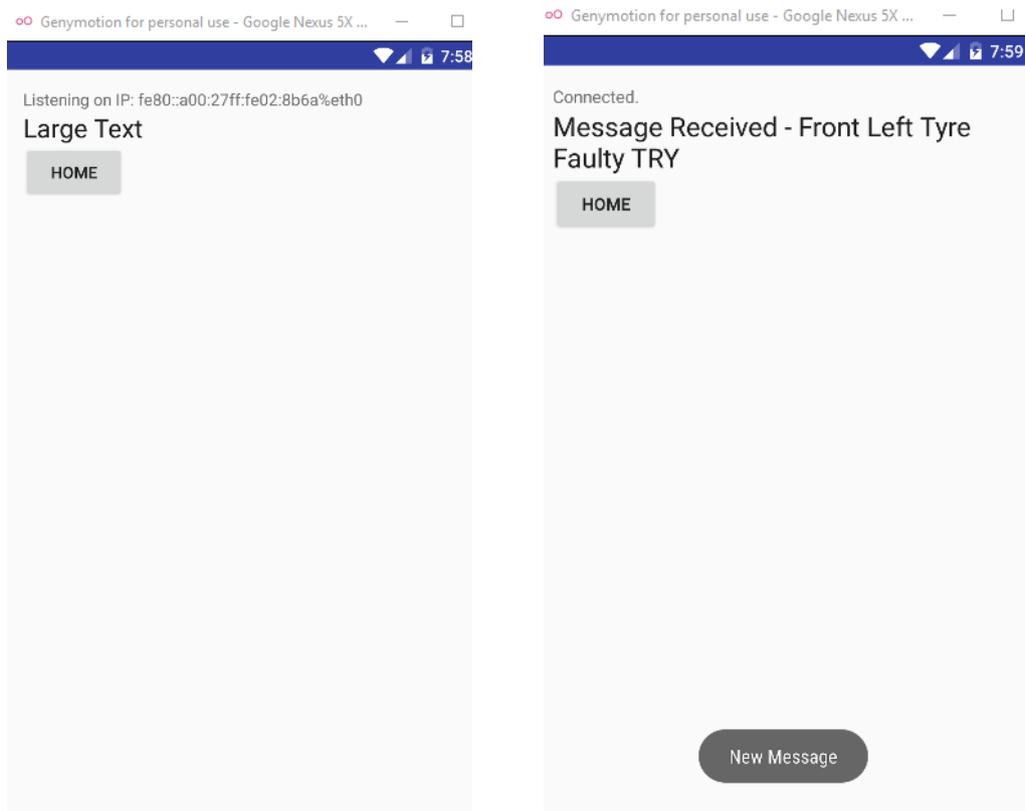


Figure 4.4 – Server activity implementation

DB Adapter Activity

```
214     public Cursor getAllRows(String regNum){
215         String where = regNum;
216         Cursor c = db.query(DATABASE_TABLE2, new String[] {KEY_ID,KEY_MESSAGE, KEY_DATE, KEY_REGNO},
217             KEY_REGNO + "=?", new String[]{(regNum.toUpperCase())}, null, null, null, null);
218         if(c !=null){
219             c.moveToFirst();
220         }
221         return c;
222     }
223 }
224 public String getRegNo(String uniqueID) {
225     Cursor cursor = db.query(DATABASE_TABLE, new String[]{KEY_REGISTRATIONNUMBER}, KEY_UNIQUEID + "=?",
226         new String[]{(uniqueID)},null, null, null, null);
227     if(cursor !=null){
228         cursor.moveToFirst();
229         String returnRegistration = cursor.getString(cursor.getColumnIndex(KEY_REGISTRATIONNUMBER));
230         return returnRegistration;
231     }
232     else{
233         return "Sorry no registration numbers exist for that record";
234     }
235 }
```

Figure 4.5 - (Database Adapter methods)

The DB Adapter activity, shown in Figure 4.5, is important for the functionality of the application that was created. The DBAdapter controls the application's interactions with the SQLite database stored on the phone. These methods were used to get the messages for the specific user logged into the application making sure only their messages appeared in the inbox. Because of how the database tables were created, i.e. the Messages table was separate from the CarIdentifiers table, there needed to be a way to link the two so that the specific messages for that user who was logged in could be retrieved. The getRegNo method is used first in the application to match the uniqueID with the registration number of that user. The query retrieves the registration number of the user and returns that registration number, using that to query the Messages table with. The method, getAllRows(), gets all the messages for the specific registration number that is passed as the parameter, which is returned from the getRegNo method. This took some time to implement correctly because the tables were not connected in anyway by a foreign key. Once the correct queries were set up to pull the required data from the database, the functionality of this class was complete.

Send Message Activity:

This activity is one of the fundamental objectives for the system. It is the main function of the implementation as it allows users to send messages about other user's vehicles to a database, based on a fault that they have spotted within the other user's vehicle.

```
54  sendMessageFrontLeftLightImage.setOnClickListener((view) -> {
56      sendMessageFrontRightLightImage.setImageResource(R.drawable.ic_stars_black_18dp);
57      sendMessageBackRightLight.setImageResource(R.drawable.ic_stars_black_18dp);
58      sendMessageBackMiddleLight.setImageResource(R.drawable.ic_stars_black_18dp);
59      sendMessageBackLeftLight.setImageResource(R.drawable.ic_stars_black_18dp);
60      frontLeftTyre.setImageResource(R.drawable.ic_stars_black_18dp);
61      frontRightTyre.setImageResource(R.drawable.ic_stars_black_18dp);
62      backLeftTyre.setImageResource(R.drawable.ic_stars_black_18dp);
63      backRightTyre.setImageResource(R.drawable.ic_stars_black_18dp);
64
65      message = ("Front Left Light Faulty");
66      icon = R.drawable.star_on_circle_final;
67      sendMessageFrontLeftLightImage.setImageResource(icon);
68      Toast.makeText(sendMessage.this, "Front Left Light selected", Toast.LENGTH_SHORT).show();
69  });
```

Figure 4.6 - Button Listener Methods: (One for each image button on the form)

For each image button that is used on the SendMessage activity, an onclick listener method had to be created for each because they needed to be manipulated by the user. Figure 4.6 is an example of the listener that provides the detail in which the user can select what fault is wrong with the car they are reporting about. When one of these images is clicked, the image resources of all the other images are returned to their default image state. This default image state is defined in the XML file imageStates.xml. The message that is sent to the intended recipient is set on the listener, as seen on line 65 in figure 4.6.

These listener methods were difficult to implement because I had numerous issues when trying to set the image states on each button without the other image buttons being in the wrong state. For example, during implementation if one image was clicked, then it turned to the pressed state only for a split second before returning to the original default state. I overcame this issue by introducing the code above with the integer "icon" object that could be manipulated to the pressed image state when it was clicked by a user. Having done that, I had to implement the reset for all the other buttons to ensure that they were not displaying the same image state as the pressed button after a new button was selected. I used code from StackOverflow [\[14\]](#) to help develop the image states file.



Figure 4.7 – The send message activity implemented

Security Activity Code:

This activity is an extra feature that I implemented to show the kinds of encryption and decryption methods that could be implemented in future versions of the application. This activity has been added to show the type of encryption that would be carried out when adding sensitive data or retrieving sensitive data from the database. There needed to be some demonstration of the techniques that would be carried out to meet the objective of Aim4 discussed in the approach.

Encryption Function:

```
30     public static byte[] encryptMsg(String message, SecretKey secret)
31         throws Exception
32     {
33         /* Encrypt the message. */
34         Cipher cipher = null;
35         cipher = Cipher.getInstance("AES");
36         cipher.init(Cipher.ENCRYPT_MODE, secret);
37         byte[] cipherText = cipher.doFinal(message.getBytes("UTF-8"));
38         return cipherText;
39     }
```

Figure 4.8 – Encryption Function

The function shown in Figure 4.8 is the encryption method that has been used to encrypt a message inputted by a user into a text box. The method takes a string and a secret key that is generated as parameters and encrypts the string that is input by the user. The method uses the built-in Android Cipher class and its functions to create an AES encryption standard that can be used to encrypt the message together with the secret key provided in figure 4.7. A byte array is created to store the message in bytes rather than as a string because the Android Cipher decryption method takes a byte as its input parameter, meaning it is important to return a byte rather than a string. The string is converted into bytes using the Cipher.doFinal method.

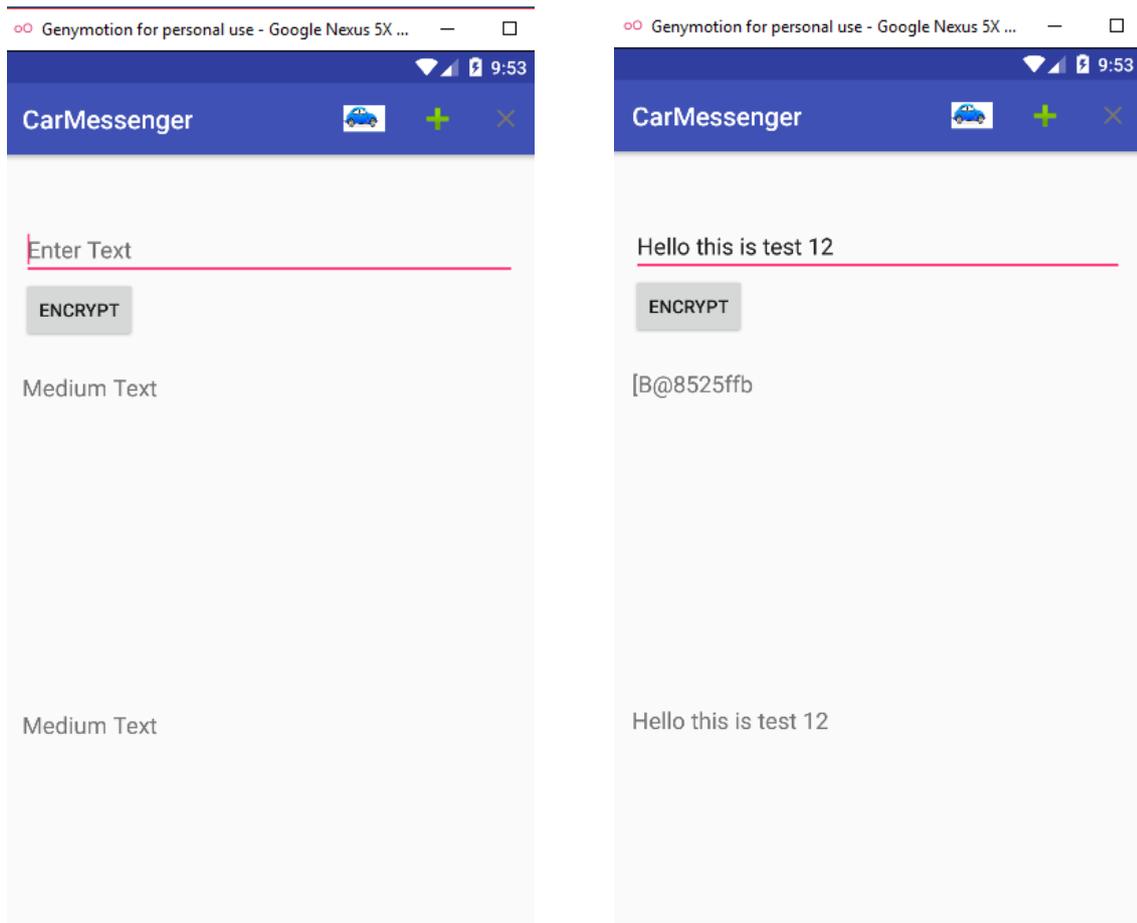


Figure 4.9 – The encryption and decryption functions that have been implemented.

```
41 public static String decryptMsg(byte[] cipherText, SecretKey secret)
42     throws Exception
43 {
44     /* Decrypt the message. */
45     Cipher cipher = null;
46     cipher = Cipher.getInstance("AES");
47     cipher.init(Cipher.DECRYPT_MODE, secret);
48     String decryptString = new String(cipher.doFinal(cipherText), "UTF-8");
49     return decryptString;
50 }
```

Figure 4.10 – Decryption Function

The decryption function shown in Figure 4.10 works by taking a byte that is returned from the encryptMsg method, and a secret key that is generated in the generateKey method and returns the decrypted version of the encrypted message. Again, this function relies on the Android Cipher class and uses its Cipher.DECRYPT_MODE to decrypt the bytes from the input parameter.

Secret Key Generator:

```
21     public static SecretKey generateKey()
22         throws Exception
23     {
24         KeyGenerator kg = KeyGenerator.getInstance("AES");
25         kg.generateKey();
26         sks = new SecretKeySpec(kg.generateKey().getEncoded(), "AES");
27         return sks;
28     }
```

Figure 4.11 – Secret Key Generator Function

generateKey() method uses a KeyGenerator object from the Android KeyGenerator class. It takes an AES encryption method to produce the key. The SecretKeySpec object is then generated to get the encoded version of the key which is then used for encryption and decryption. The key is used in both the encryption and decryption methods as the input parameter to either encrypt or decrypt the message that is input by the user.

Inbox Activity

```
110     private void populateListView() {
111         regNo = db.getRegNo(name);
112         Cursor cursor = db.getAllRows(regNo);
113         if(cursor != null){
114             final String[] fromFieldNames = new String[]{DBAdapter.KEY_MESSAGE, DBAdapter.KEY_DATE, DBAdapter.KEY_REGNO};
115             int[] toViewIDs = new int[]{R.id.txtMessage, R.id.txtDate, R.id.txtReg};
116             final SimpleCursorAdapter myCursorAdapter;
117             myCursorAdapter = new SimpleCursorAdapter(getApplicationContext(), R.layout.custom_resource_layout, cursor,
118                 fromFieldNames, toViewIDs, 0);
119             myList = (ListView) findViewById(R.id.inboxMyList);
120             myList.setAdapter(myCursorAdapter);
121         }
```

Figure 4.12 – Inbox Activity Function

The method displayed in Figure 4.12 is the main functionality of the Inbox activity. This is where the DB Adapter directly interacts with the activity to retrieve the data from the database and display it for the user. This was difficult to implement because it makes use of a list view object, which can be manipulated with a custom layout, which is what has been done here, in the layout.custom_resource_layout file. It was challenging because the code here makes use of lots of different components, there are CursorAdapters, Cursors, String arrays and list views. Each of these components had to be working together to achieve the functionality wanted for the activity. These components can seem complex at first sight but after reading the documentation and tutorials, the methodology behind them seemed easier to understand. Once the understanding of these components was there, they could be adapted to fit the requirements of the application.

```

136     public void showDialog(){
137         feedbackDialog = new Dialog(this);
138         feedbackDialog.setContentView(R.layout.custom_dialog_star_rating);
139         feedbackDialog.show();
140     }
141
142     public void onActionDialog(View view){
143         respondToClick = (Button)feedbackDialog.findViewById(R.id.customDialogFeedbackButton);
144         final RatingBar starsFeedback = (RatingBar)feedbackDialog.findViewById(R.id.customDialogRatingBar);
145         final EditText feedbackComments = (EditText)feedbackDialog.findViewById(R.id.customDialogFurtherComments);
146
147         respondToClick.setOnClickListener(new View.OnClickListener() {
148             @Override
149             public void onClick(View view) {
150                 float numStars = starsFeedback.getRating();
151                 String feedback = feedbackComments.getText().toString();
152
153                 Toast.makeText(Inbox.this, numStars + feedback, Toast.LENGTH_SHORT).show();
154             }
155         });
156     }
157 }

```

Figure 4.13 – Inbox Dialog function

The code displayed in Figure 4.13 shows the dialog box that is used specifically for the custom feedback on the messages in the inbox activity. The difficult part of implementing this feature was that a custom dialog interface(custom_dialog_star_rating.xml) needed to be created to display the star rating and the comment section that was needed rather than using the default dialog objects that can be created in Android. The custom dialog was a new aspect to coding that was not yet implemented within the project. Learning and researching about dialog interfaces online helped to understand how they worked, and how custom dialogs were different to the default dialog boxes that you could use. The custom layouts provide a customisable design that can include an array of features rather than just the basic ones that can be generated in Android.

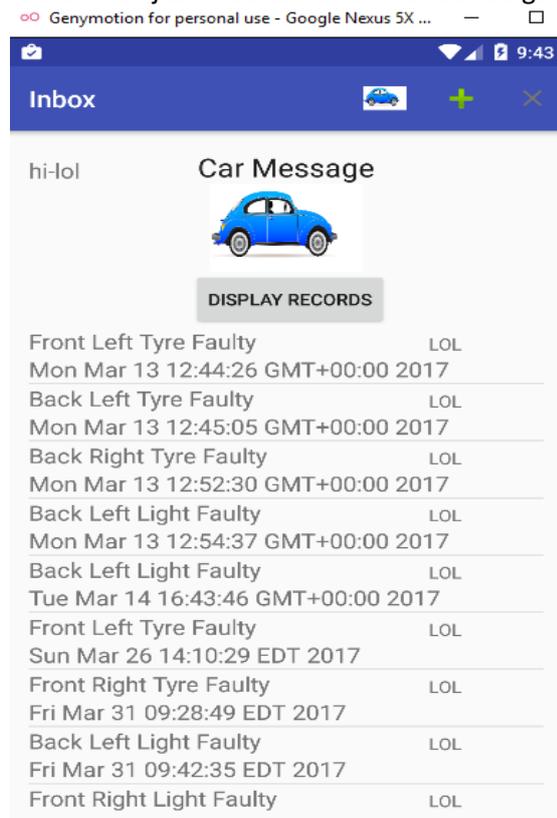


Figure 4.14 – The inbox activity implementation

HomeScreenActivity

```
67     public void loginButton(View view){
68         uniqueID = (EditText) findViewById(R.id.HomeScreenLoginIdentifier);
69         password = (EditText) findViewById(R.id.homeScreenLoginPassword);
70         Button loginButton = (Button) findViewById(R.id.HomeScreenLoginButton);
71         loginButton.setOnClickListener(new View.OnClickListener(){
72             public void onClick(View view){
73                 //get unique id and password from DB
74                 String uniqueIDEntered = uniqueID.getText().toString();
75                 String passEntered = password.getText().toString();
76
77                 //fetch unique id from database
78                 String storedUniqueID = db.getSingleEntry(uniqueIDEntered);
79                 String storedPass = null;
80                 storedPass = db.getPass(passEntered);
81                 //check id stored matches id entered
82                 if(uniqueIDEntered.equals(storedUniqueID) && passEntered.equals(storedPass)){
83                     SharedPreferences.Editor editor = sharedPreferences.edit();
84                     editor.putString(username, uniqueIDEntered);
85                     editor.commit();
86                     Toast.makeText(HomeScreen.this, "Login Successful", Toast.LENGTH_LONG).show();
87                     Intent mainMenuIntent = new Intent(HomeScreen.this, MainMenu.class);
88                     mainMenuIntent.putExtra(unique_ID, uniqueIDEntered);
89                     startActivity(mainMenuIntent);
90                     finish();

```

Figure 4.15 – HomeScreen loginButton function

The code in Figure 4.15 has many different features that had to be moulded together. It made use of the onClick listener methods, database interactions and the shared preferences methods. The shared preferences method was used because it allowed for storing the user's uniqueID in their preferences file which is stored on the phone itself. This method was like using sessions in PHP which is a more familiar process. Being able to adapt this approach to keep track of the user logged in, was similar to that of using sessions in PHP, which helped create the functionality. The shared preferences function allowed the use of the uniqueID across the entire application for that user, which helped a lot in the case of retrieving a user's messages as the system knew exactly who was logged into the application through the preferences file.

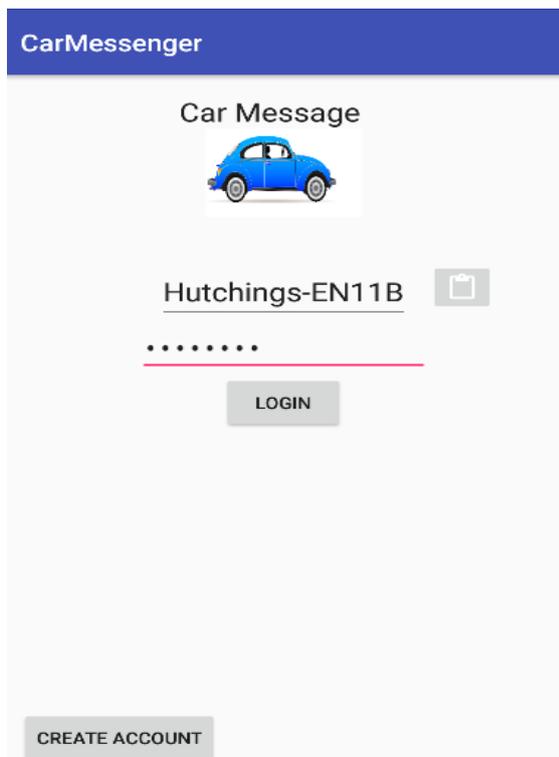


Figure 4.16 – The Home activity implementation

Traffic Information Activity

```
115     public void populateAccidents(){
116         List<MyMarkerObj> m = db.getMarkers();
117         for(int i = 0; i < m.size(); i++) {
118             String lat = m.get(i).getLat();
119             String lng = m.get(i).getLng();
120             LatLng latitudeAndLongitude = new LatLng(Double.valueOf(lat), Double.valueOf(lng));
121             String street = m.get(i).getStreetName();
122             String time = m.get(i).getAccidentTime();
123             mMap.addMarker(new MarkerOptions()
124                 .position(latitudeAndLongitude)
125                 .title(street + " " + time)
126                 .draggable(false)
127             );
```

Figure 4.17 – Traffic Information Populate Accidents function

The populateAccidents method shown in Figure 4.17, is important for displaying the markers from the database onto the TrafficInformation activity demonstrating that other users can then see the new markers, which “simulate” accidents that have occurred. It is important to note that this method makes use of the getMarkers function in the DBAdapter class. This is important because it retrieves the markers from the database table allowing them to be used to populate the map displayed in the TrafficInformation activity. The method here was complex to create because it makes use of a Marker Object class created for this solution, which contains setters and getters to make an object, “Marker”, which contained the necessary information which could correlate with the database table that houses the data about each marker.

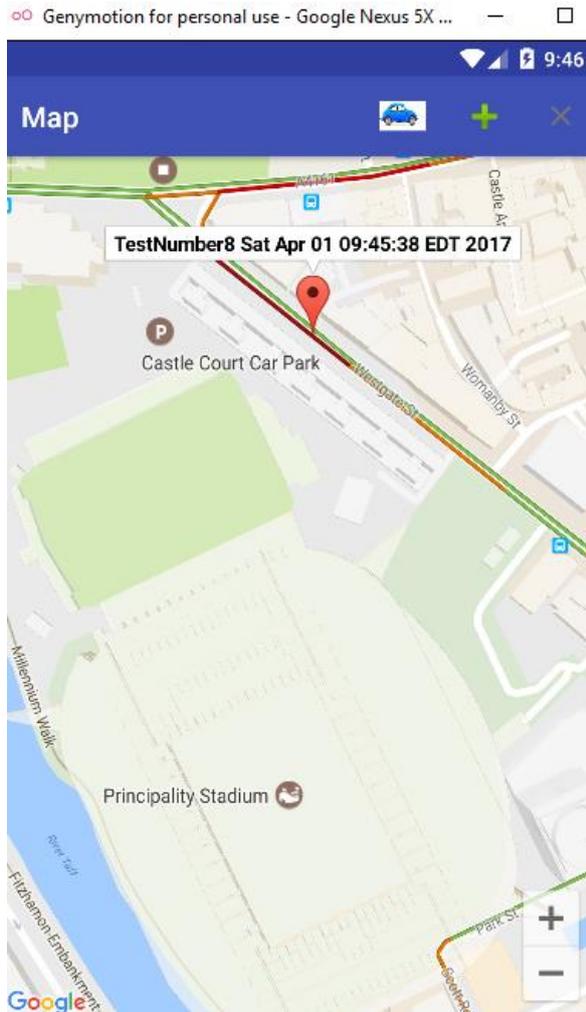


Figure 4.18 – Traffic information activity implementation

Report Accident Activity

```
121 @Override
122 public void onMapReady(final GoogleMap googleMap) {
123     checkPermission();
124     requestPermission();
125     mMap = googleMap;
126
127     streetName = (EditText)findViewById(R.id.enterStreetNameActivity);
128     if (ContextCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_FINE_LOCATION) ==
129         PackageManager.PERMISSION_GRANTED &&
130         ContextCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_COARSE_LOCATION) ==
131             PackageManager.PERMISSION_GRANTED) {
132         mMap.setMyLocationEnabled(true);
133         mMap.getUiSettings().setMyLocationButtonEnabled(true);
134         mMap.getUiSettings().setZoomControlsEnabled(true);
135         mMap.setMapType(1);
136     }
137     else {
138         ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.ACCESS_FINE_LOCATION}, PERMISSION_REQUEST_CODE);
139     }
140     Location currentLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
141     if (currentLocation != null) {
142         mLatitude = currentLocation.getLatitude();
143         mLongitude = currentLocation.getLongitude();
144
145         float zoomLevel = (float) 16.00;
146         LatLng latLng = new LatLng(mLatitude, mLongitude);
147         mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, zoomLevel));
148         mMap.addMarker(new MarkerOptions().position(new LatLng(mLatitude, mLongitude)).title("You are here!").snippet
149             ("Consider yourself located"));
150     }
```

Figure 4.19 – Report Accident Permission and map loader functions

Figure 4.19 demonstrates the Report Accident activity's method that checks to see whether a user has enabled their location to be used by the application. The code here makes use of the Google Map API and the Android Manifest file within the application. Installing Google Play services on the Genymotion emulator devices was quite difficult as the default emulators that are created do not come with pre-installed versions of Google Play Store and its other dependencies. To install that on the devices I wanted to use, I had to transfer zip files from my laptop to the emulator devices, set up a Google account and then update the play store on the device itself.

The method shown in Figure 4.19, prepares the activity for the user and is dependent on whether they have selected for the application to use their location or not. The activity carried out is the same task regardless of whether the location is provided or not, if the location is provided the user is directed to that location, otherwise, the user is directed to a pre-determined location. The location permissions must be given in the Application manifest file to enable the request permission method to work. This is an important aspect of the application because it integrates the Google API with the functionality of reporting accidents within the application. The activity gives users the opportunity to report accidents directly on to a Google Map which stores the markers that they put down into the database, so the other users can see them.

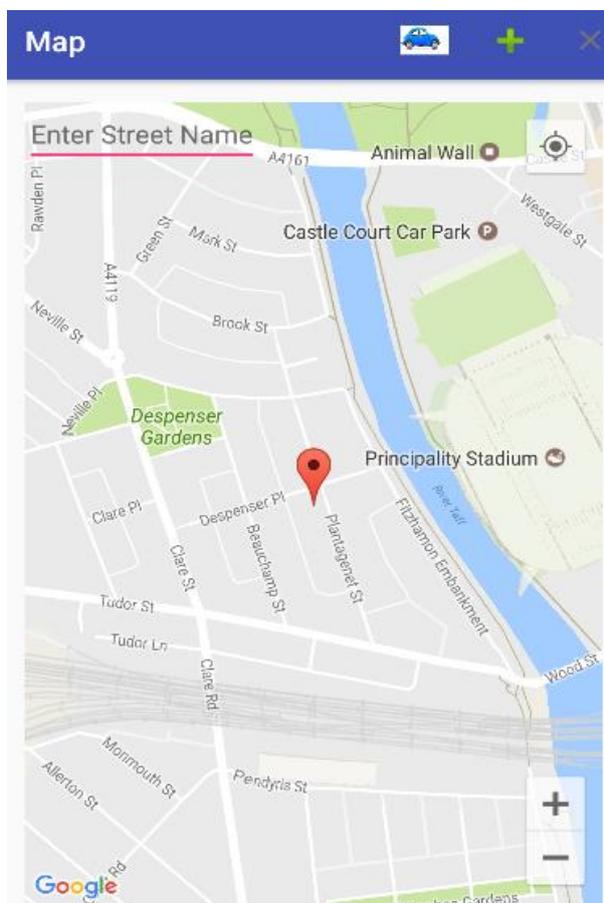


Figure 4.2 – The Report Accident activity implementation

Action Menu XML and Java Code

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto" >
4      <item android:id="@+id/action_homeButton"
5          android:title="@string/action_home_button"
6          android:icon="@drawable/carlogodesign"
7          app:showAsAction="always">
8
9
10     </item>
11     <item
12         android:id="@+id/action_new_message"
13         android:title="@string/action_new_message"
14         android:icon="@android:drawable/ic_input_add"
15         app:showAsAction="always">
16
17     </item>
18     <item
19         android:id="@+id/action_logout"
20         android:title="@string/action_logout"
21         android:icon="@drawable/places_ic_clear"
22         app:showAsAction="ifRoom">
23
24 </item>
</menu>

```

```

82  @Override
83  public boolean onOptionsItemSelected(MenuItem item) {
84      switch (item.getItemId()) {
85          case R.id.action_homeButton:
86              Intent mainMenuIntent = new Intent(this, MainMenu.class);
87              startActivity(mainMenuIntent);
88              finish();
89              return true;
90
91          case R.id.action_new_message:
92              Intent sendMessageIntent = new Intent(this, SendMessage.class);
93              startActivity(sendMessageIntent);
94              finish();
95              return true;
96
97          case R.id.action_logout:
98              Intent logoutIntent = new Intent(this, HomeScreen.class);
99              startActivity(logoutIntent);
100             finish();
101             return true;
102
103         default:
104             // If we got here, the user's action was not recognized.
105             // Invoke the superclass to handle it.
106             return super.onOptionsItemSelected(item);

```

Figure 4.21 – Action Bar functions and XML

The code displayed in Figure 4.21 presents the action menu bar that was created for this application. There are three actions that a user can take from the icons displayed in the menu, Send Message, Home or Logout. These are the main areas of the application that a user will need access to, referring to the design heuristic regarding user control and freedom, providing a user with free-flowing movement around the application as well as increasing ease of use. Having the action bar provides users with the opportunity to get out of any activity that they did not intend to click on, or if they wanted to jump straight to writing a new message from another activity, they have that option as well. It was not difficult to implement but it is an important feature of the application that corresponds directly to the design heuristics.



Figure 4.22 – The action bar implementation

4.3 Implementation concerns

During the implementation phase of the project there were many hurdles that had to be overcome. Firstly, the choice of what to implement. At the first stage of implementation I lost sight of the original aims and objectives of the project and began creating a fully functional In-Car system with lots of functionality, rather than what I originally set out to do. I created the “sending a message” aspect of the project quite early on, but neglected to fully complete this functionality by making it device to device communication rather than sending a message to the database from one user and then checking the inbox for that message on another user’s application. This oversight, led to the assumption that the database could be used for multiple devices rather than just a local file stored on the single device which, aforementioned, caused changes to the scope of the project and the implementation schedule. Having not fully implemented device to device functionality yet, I needed to focus on that aspect rather than making changes to the database. During the final weeks of implementation, the major functionalities of the application were working and minor improvements were being made throughout the system, for example debugging and fine-tuning. The socket programming technique used to create device-to-device communication functionality was an alternative method to the original method planned. Regardless of the technique being an alternative method, it was a success and achieved the objective set out.

A challenge experienced while implementing the solution was Android Studio itself. At the beginning of the project the version of Android Studio used was 2.1.2, and the most recent version today is 2.3.1. This created some issues because a lot of the online tutorials or developer help was using updated API versions and the new features. For example, when creating the security code and looking for possible ways to encrypt or decrypt the data using built-in Android methods, the latest solutions all used methods that were not available in the version of Android Studio being used, methods had been changed slightly and the new updated code was not available in the libraries in the project. Updating Android Studio to the latest version of the software to try and include some of these methods caused the entire application to stop building correctly. This led to the manifest files becoming corrupted which led to reverting to the older version to continue working on the solution. I ran into problems with the built-in emulator on Android Studio as well. Not only was it very slow at starting up and running the emulated programs, it stopped working during the seventh week of implementation. As expected, this caused some delays in the implementation cycle as I then had to diagnose the issue with the emulator software and when I couldn't find any legitimate reason for the crashes and the timeouts that were occurring I decided to switch to the Genymotion software emulators which were a lot faster and more user-friendly than the previously used emulator. The only difficulty with Genymotion was as I mentioned already, the setting up of the Google Services on the device.

Emulators were used to simulate the application due to the high-level API of the android devices available. The only device available was a Samsung Galaxy S7 which has the latest operating system and Android Developer API level, (API level 24) which was not compatible with the version of Android Studio being used and although the application runs on the Samsung Galaxy S7, I could not root the device to pass the database file allowing access to the features of the application. This was another problem of using SQLite instead of using another database program, because if I had an online database that was available like phpMyAdmin then the S7 would be able to communicate with it via the internet rather than the file stored locally on the device. Coming back to Android Studio, when I created my project and the first application version, I selected the minimum Software Development Kit(SDK) level of 15 and the maximum at 23, which was the current version at the time of creation. The SDK level stated that 95% of devices in the Android market would run the application if I used those SDK levels, but if I changed it to the latest version SDK only, then only 3% of the devices would run it, therefore, the majority of devices could install and run the application.

5. Evaluation

The evaluation of the project focusses on how well the solution meets the overall aims and objectives of the problem and if the application created “solves” the problem that it was built to solve. As well as next steps that would be taken to improve the solution and how different approaches could have been taken. This section will also detail the testing strategy and testing that was carried out for the application.

5.1 Testing

Test Strategy

Throughout development of the solution, I have carried out numerous testing and debugging sessions to fix bugs and errors that have been prevalent within the application. The ongoing testing of the solution was important to ensure that the original aims and objectives were met and that if they were not, then a valid reason has been provided. This section details the test cases that I have created to test the final implementation of the solution against the aims and objectives to decide how well the implemented system stands up to the system designed. The results show how successful the implementation has been. The test cases themselves have been chosen to demonstrate the main functionality of the system and the performance on a mobile device.

Test Cases

Test	Pre—conditions	Expected Outcome	Actual Outcome
1: Can a user create an account	None	A user will be able to create an account by filling in the form on the application. Their details will be stored in the database.	The user created an account with the credentials: Hutchings, EN11BXH, password. These details were stored successfully.
2: Log into the system.	Must have completed test 1, or use previously stored credentials to log in.	User enters their uniqueID and their password and can log in to the system. A message saying that they have been logged in should appear.	The user Hutchings-EN11BXH with the password, “password” was successfully logged into the system.
3: Log into the system with details that are not stored in the database.	None	There will be an error message saying that incorrect credentials have been entered and the user will not be able to log in.	The user was not able to log in using incorrect credentials and an error message was shown saying. “incorrect uniqueID”.
4: Create an account with an invalid registration plate.	None	User will not be able to create an account with a number plate that does not follow the format XX11XXX.	The user could not create an account as the registration number was invalid and a message was displayed to the user saying this.
5: Send a message	Must be logged in to the application.	User will send a message selecting the	The message was sent successfully and a

		front left tyre as the problem, and the registration number as EN11BXH. Should be sent successfully and the user will be told this. Message should also be stored in the database.	message told the user "message sent successfully". The message was also stored in the database.
6: Send a message without the registration number entered.	Must be logged in to the application	The message will not be sent successfully and a message will tell the user they need to input a registration number.	The message was not sent and a popup message told the user to enter a registration number.
7: Check user messages within the application.	Must be logged in to the application	The user's messages are displayed in a list-view on the form. Displaying all the available messages that have been sent to that user.	The user's messages were displayed on the form and they were able to select a message to give feedback on them.
8: Report an accident with the street name (location) written as TestNumber8 and the marker click on West Gate Street in Cardiff City Centre.	Must be logged in to the application Must give permission for the application to use your location.	User enters a location name where the accident takes place and then adds a marker to the map.	A marker was added on the map where the user selected the accident took place, visually showing the marker on the map.
9: Check that the accident reported on WestGate Street in test 8 is displayed in the Traffic Information activity.	Must be logged in to the application Must have completed Test 8.	All markers from the database are displayed on the map. When the user clicks on the marker, the title and street name should appear.	The marker for WestGate Street was shown on the map where the accident was reported.
10: Give feedback on a message received in a user's inbox.	Must be logged in to the application. Must have messages in the user's inbox.	Select a message in the user's inbox and a dialog box should appear. Select 3 stars and write a comment. Click submit when done. Feedback submitted should be displayed.	When the user clicked submit feedback, a message was displayed with the number of stars given and the comment that they wrote.
11: Give feedback on an accident reported.	Must be logged in to the application. Must be at least one accident reported.	Clicking on a marker will display a dialog box. The dialog box allows users to enter a comment and submit a star rating. Feedback	User was able to successfully add feedback from the form. A message box was displayed with

		successfully submitted message should be displayed.	the star rating and the comment.
12: Encrypt a message	Must be logged in to the application	In the EditText box on the security form, enter the phrase, "Hello, this is test 12". The message is then encrypted and displayed in the textbox below.	The message "Hello this is test 12" was encrypted to the following: [B@8525ffb.
13: Decrypt a message that has been written by the user.	Must be logged in to the application Must have entered a phrase or message to be encrypted, like in Test12.	User is shown the decrypted version of the message.	The message "Hello this is test 13" was decrypted successfully.
14: Measure CPU performance of application on the emulator	Using the Android Studio monitoring system while the application is running.	To see some activities that are more intense than others running at higher CPU rates.	There were many "spikes" shown in the screenshots in the appendix. The spikes were mainly when the CPU had to carry out an intense task like loading the Google API libraries for the report accident activity.
15: Measure Memory usage of application on the emulated phone	Using the Android Studio monitoring system while the application is running.	To see some activities that are more intense than others running at higher memory usage.	The memory usage was consistent throughout the application usage.
16: Send a message using your own registration number.	Must be logged in to the application.	User enters their own registration number and selects back left tyre as the issue. Add successful should be displayed.	The message was sent successfully and that was displayed to the user by saying, "message sent successfully". In the inbox, the message is displayed as well.
17: Logout of the system	Must be logged in to the application	User is logged out of the system and taken back to the HomeScreen activity.	User is taken back to the HomeScreen activity and pressing back on the android device doesn't take them back to the previous activity.
18: Test action bar new message button	Must be logged in to the application	Clicking the + icon in the action bar will take a user to the new message form.	The user was taken to the send message form where they could send a new message.

19: Test action bar home button.	Must be logged in to the application	Clicking the button on the action bar will take users to the main menu form.	The user was taken back to the main menu form.
20: Allow the application to use your location on the Report Accident activity	Must be logged in to the application. Must be the first time accessing the Report Accident page in that session.	The user's location is turned on and the map is zoomed in to their location.	The camera does not find the user location, and pans to the pre-determined location, that was set up in the code.
21: Deny the application use of your location data.	Must be logged in to the application.	The user's location will not be turned on and they will not have the find location button enabled. They should still be able to report accidents.	The user is shown a message saying that the application cannot use their location data. The find location button is also disabled. Users are still able to report accidents.
22: Click the find location button on the Report Accident activity.	Must be logged in to the application.	The device should reposition to the user's location and find them on the map.	The location button does not find the user on the map because the current user location cannot be found by the application, so it defaults to the hardcoded pre-set location that I have added.
23: Send a Message via the client-server relationship	Must be logged into the application. Must be running the application on two devices. Must know the devices IP address	The user follows the send message activity by entering a fault and a registration number. They are then taken to the new screen where they can connect to the device they want to send that message to, where they are able to send their message to the user.	The user sends the message which is stored in the local database for reference. The user is taken to the client page, where they input the server address for the phone they want to message and that phone receives the new message from the client device.
24: Set up a client-server connection	Must be logged into the application. Must have two devices running the application. Must know the two devices IP addresses.	The client phone connects to the server phone by inputting the server phones IP address. The server phone is listening for	The client sets up the connection to the server phone successfully by inputting the IP address: 192.168.0.27 which is the address

		commands sent by the client phone.	for the device connected to my home network.
--	--	------------------------------------	--

Evidence of test case scenarios being implemented and tested can be found in Appendix section 9.2.

5.2 Evaluation of the Implementation

The implemented solution that I have created needs to be evaluated against the aims and objectives that are described in the approach section of this report.

Provide a complete application that provides users with a fully functional car messaging system.

This objective has been completed to a high standard because of the features that have been implemented. Users can communicate with other drivers via their devices, they can view messages in their inbox, check for accidents along their route and report any accidents that they see on their journey. These features give the application good overall functionality and allow users to have a complete application rather than a single featured one. A way this objective could be improved is to extend the database scope from local to global, ensuring that all the features including Reporting Accidents are available across devices rather than only on the local device. An alternative database approach would extend the capabilities of the application across multiple devices, rather than to a single device. The functionality works in the same way as it would with an extension of the scope, allowing transmission of data through the internet rather than local storage, meaning the features are still programmatically correct and working as intended, only the Report Accident is affected by the locality of the database.

Allow users to send messages from one device to another device regarding the status of their respective vehicles.

This objective has been achieved, by using the technique of socket programming to communicate messages between devices. Evidence of this can be seen in Figure 5.1, which demonstrates a message being sent from one device to another. These messages are sent from one client device to a server device, which is actively listening for messages. The client device can form a connection to the server device, by inputting the IP address of said device. A connection was formed which was used to transmit the message between devices, therefore, displaying successful evidence of this part of the project. The messages that can be sent contain the fault with the respective vehicle in question, and their registration number, so that the user receiving the message knows the fault with their vehicle.

Building on this objective, a static IP address could be implemented to remove the ground work for the users. The static IP address would be connected to a server device that does not belong to another user, but is used to forward messages between the devices rather than having to input an IP address manually.

Figure 5.1 shows the implemented solution to the objective of communication between devices. As aforementioned, the user must input the IP address of the server device which must be listening for commands from a user. The message that gets sent from the client device gets stored in the server device's database so that user can then see that message in their inbox. This is important for the communication between devices aspect, as drivers do not have to check their messages immediately because they are able to check their inbox at a more convenient time.

Alex Hutchings 1412529: Car Messaging System.

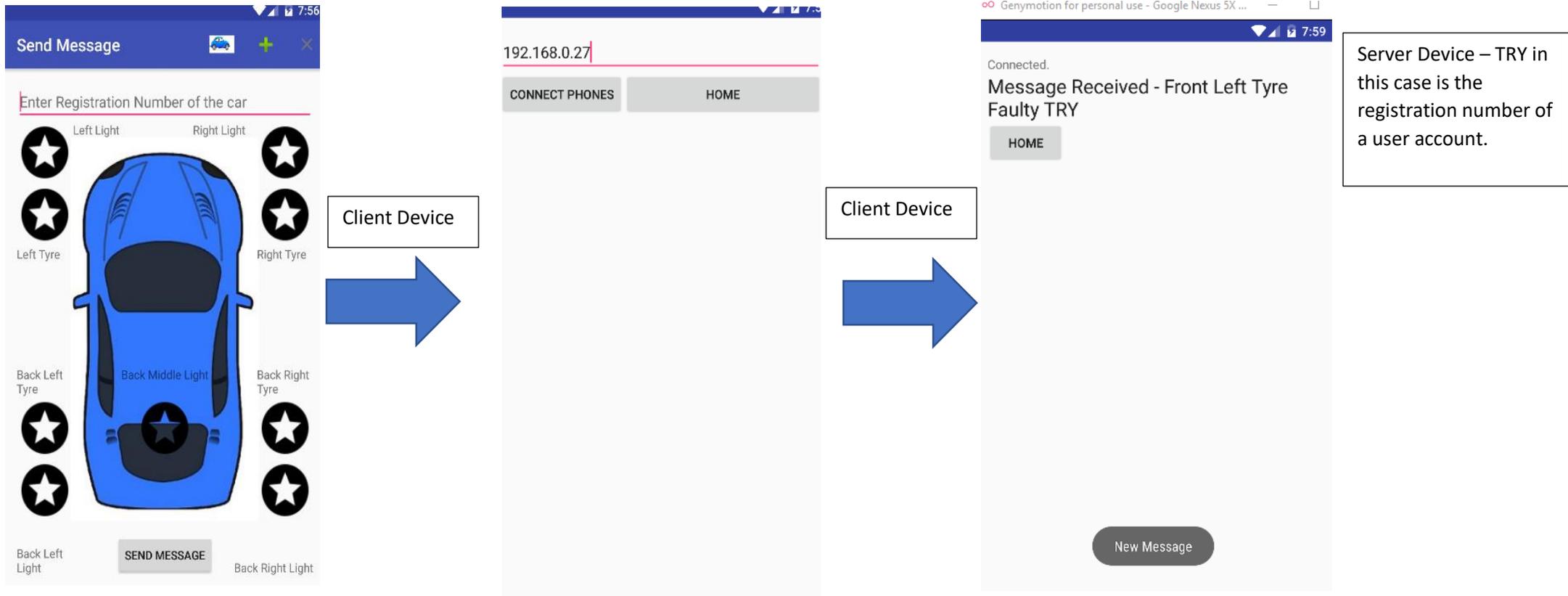


Figure 5.1 – Flow of events for sending a message in the implemented solution

The Left image shows the send message screen where a user will input a registration number and then select a fault with the car and click send message. From that screen, the user is taken to the second image, where they must enter the server device's IP address. From there they connect to the server device, like that in the third image. The third image shows the message being sent from the client device to the server device and how that message is displayed once received.

Provide a simple and easy interface for drivers to communicate with each other.

The user interface that has been created is simple and provides the user with an aesthetically pleasing interface within the application. The following heuristics I mentioned in the approach, have been achieved and examples of that have been provided in the implementation section of this report:

- User control and Freedom
 - An action menu that allows users to return to the main menu from any form they are on. The same menu also allows users to log out and go directly to the SendMessage activity.
- Help users recover, diagnose and recover from errors
 - An error handler activity that diagnoses errors in a separate activity rather than crashing the system to the device's home. The error handler posts a user-appropriate message rather than the stack trace callback to find the error.
 - On-screen pop-up messages that detail when a user may have forgotten to input text into textboxes, or if they are missing some component of the activity to continue. For example, when a user adds an incorrect registration number on the Create Account activity, they are told this.
- Aesthetic and minimalist design
 - The minimum for each activity is detailed on the screens. Allowing users to carry out their task without facing cluttered screens.
 - Hints and labels are all implemented with the user in mind, detailing what is expected in each place and an error message if the wrong type of input is entered.
- Consistency and standards
 - The same style text, buttons, logos used throughout the application.
- Match between the system and the real world.
 - Registration numbers are expected to be in the real UK fashion e.g. AB11 CDE
 - Logical ordering of elements on the activity screen has been followed to ensure users can follow them in a natural sequence.

Having carried out these heuristics throughout the application, I can say with some confidence that this objective has been achieved, and any future development should follow in the same direction to maintain those standards.

Store all data used within the application securely and in a way that does not compromise the performance of the application itself.

I have mentioned in the implementation section of this report that I was unable to implement security features in a way that maintained the security of the user's data when sending and receiving messages or when setting up their accounts. I did implement a security activity that demonstrates the functionality that I hoped to create through the application, however I faced major challenges trying to combine the different elements of the security within Android programming to work in the way that I wanted. The following diagram details, the intended security protection I hoped to implement:

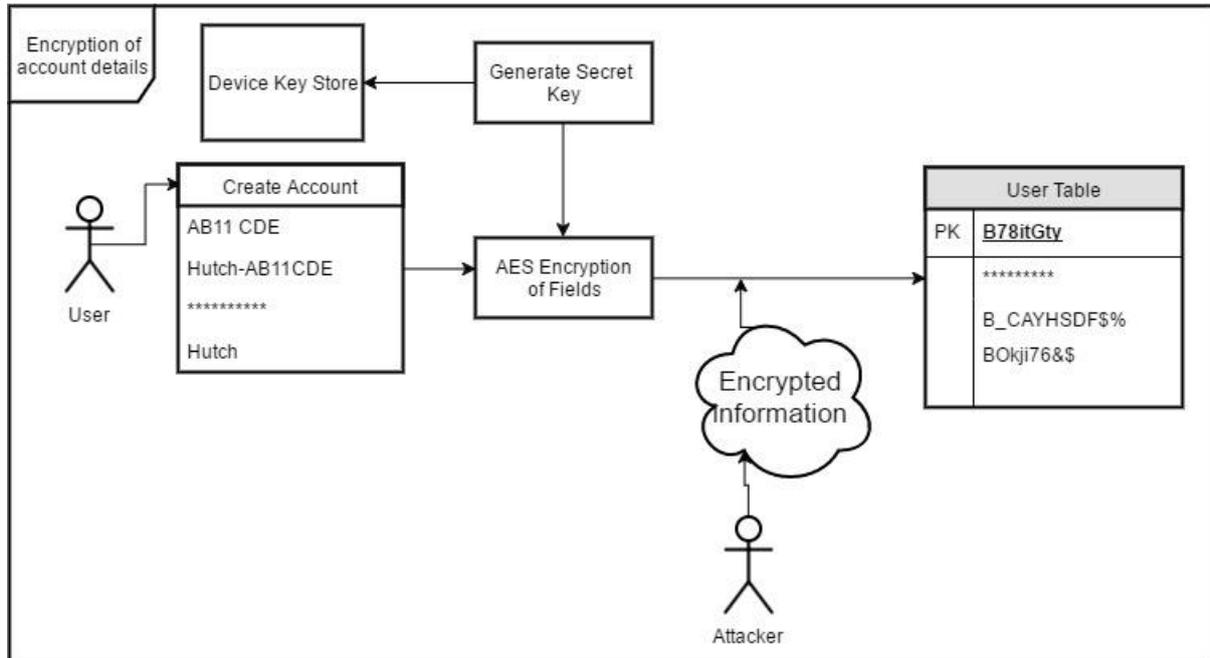


Figure 5.2 – Future Encryption use case

The way that the encryption should have been carried out in the application can be seen from Figure 5.2. It demonstrates an AES encryption method that would use Android’s Key Store class to securely hold the keys generated with the key generator method used in the security activity. From the Figure, even if an attacker was listening in on the transmission from the user’s application to the database, then they would only see encrypted traffic. The only key that would be able to decrypt that information would be stored on the user’s device, protecting their data stored. The key would be secure within that device only. The implemented solution does contain some workings of this type of encryption and is demonstrated in the implementation section of this report in Figure 4.5.

As demonstrated in Figure 5.3, the user would be able to log in with the same username and password that they provided in the create account activity even though the credentials stored in the database would be encrypted. The same key that was used to create an account would be taken from the user’s device to decrypt the data received from the database and to ensure that the user can successfully log in to the application. The two methods shown in Figures 5.2 and 5.3 would also be carried out in the send message activity where users transmit another user’s registration number as that can be classed as sensitive information.

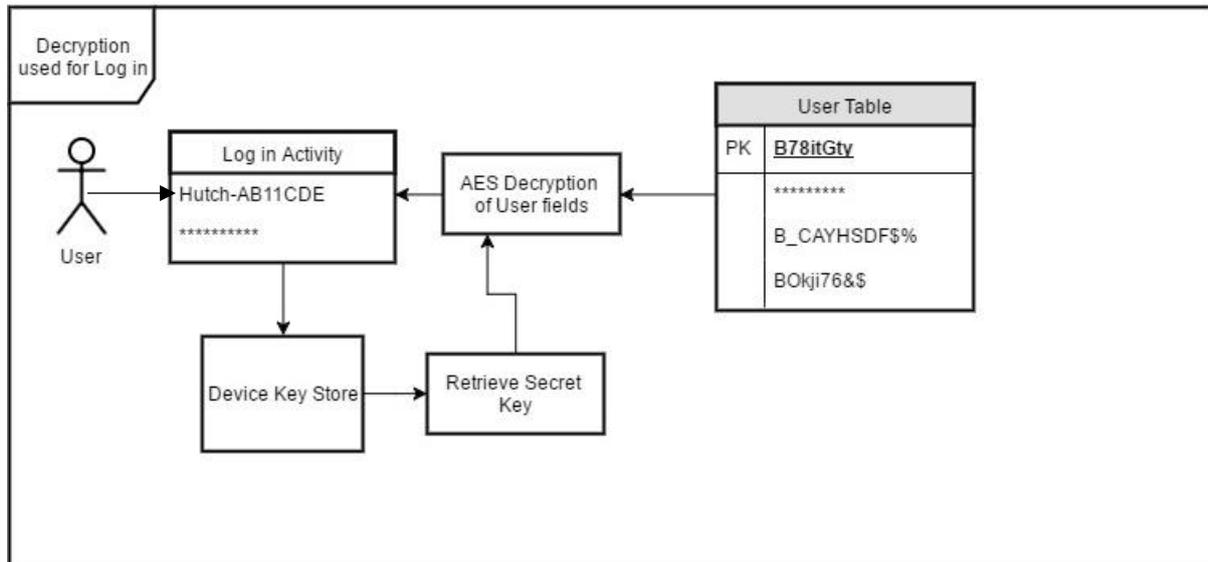


Figure 5.3 – Future Decryption use case diagram

The decryption would work by retrieving the key stored in the device’s key store and then combining that with the AES decryption function within Android to decipher the encrypted data retrieved from the database.

Overall the solution solves the problem identified at the start of the project. The solution demonstrates that driver-to-driver communication can be achieved via the method of socket programming. The additional functionality of the application provides drivers with a complete software package that enables them to carry out extra tasks within one application rather than multiple. The solution provides an effective way of communicating between drivers and in the identification of faults within their respective vehicles.

5.3 Next Steps

A more suitable software package that would allow for more than local storage on the user’s device would be a good change that would benefit the application. This is the main implementation feature that I would change to make the application more user-compatible. The reason for doing this has been mentioned throughout this report, and it would allow me to ensure that the application features are available across multiple devices from different users in synchronisation, rather than just for users using the same device and switching between users.

Another feature I would like to continue working on that has been discussed during this report, are the security aspects of the solution that were not fully implemented. I would like to implement functionality shown in Figures 5.2 and 5.3 as that would be the ideal scenario where user data is secure and protected from any unauthorised users. Effectively completing that aim of the project to ensure that user data is protected and secure.

A feature I would like to adapt in future versions of the development cycle would be the socket programming technique used to carry out device to device communication. Adding Wi-Fi direct capabilities to the application would benefit the users as more devices become readily equipped with Wi-Fi direct functionality. This would ensure the application keeps up to date with current market standards and provides the best way to communicate between drivers.

Another addition to make would be to add notifications because it adds extra functionality to the application. Notifications would add a new dynamic to the application and allow for drivers to be made aware that they have new messages which in turn would make them aware of any problems with their

vehicle, regardless if they were running the application or not. It would also present an opportunity to alert drivers to new accidents that may have occurred on the course of their journey. For example, using their location data to track their journey and then alerting them to an accident on the same road that they are currently driving on, so that they can alter their course.

5.4 Device Performance

The application is not resource heavy for the devices used. The device that I ran these tests on was a Google Nexus 5X model, which uses a 1.8GHz hexa-core 64 bit Adreno 418 GPU and has 2GB RAM, specifications which had no issues running the application smoothly, showing no signs of struggle. Figure 5.4 shows a large spike in the network usage of the application when it loads a Google Map, as it is the first time the application runs any activity that requires internet access and it has to load a map pointing to the location of the user. Similarly, in Figure 5.4 there is a slight increase in the CPU usage of the application because of the increase in activity of loading the markers from the database, and loading the Google Map from the library itself.

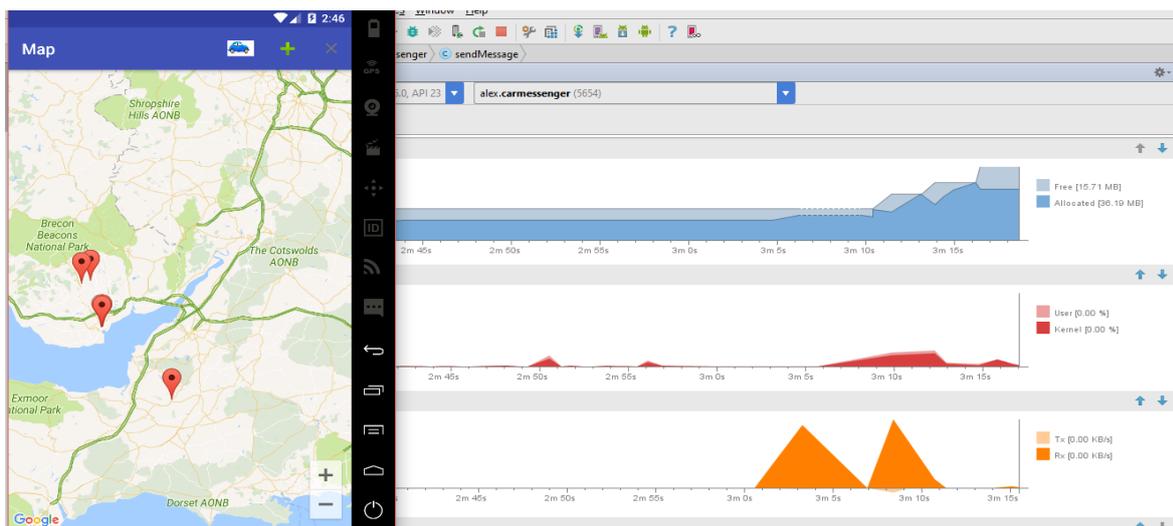


Figure 5.4 – Traffic Information Activity and its monitor levels

Figure 5.5 is the most intense level that the CPU spiked to during the running of the application. This is mostly down to the loading of the Google Map, and finding the user location with the device. The CPU usage at this point did not slow the application down and it ran smoothly during the spike in the usage. There is also very little network activity during this activity, because it only had to briefly load the map from a very zoomed in and precise location, whereas, compared with Figure 5.4 the map level is very far out and requires more of the UK to be loaded in, which would explain the network usage.

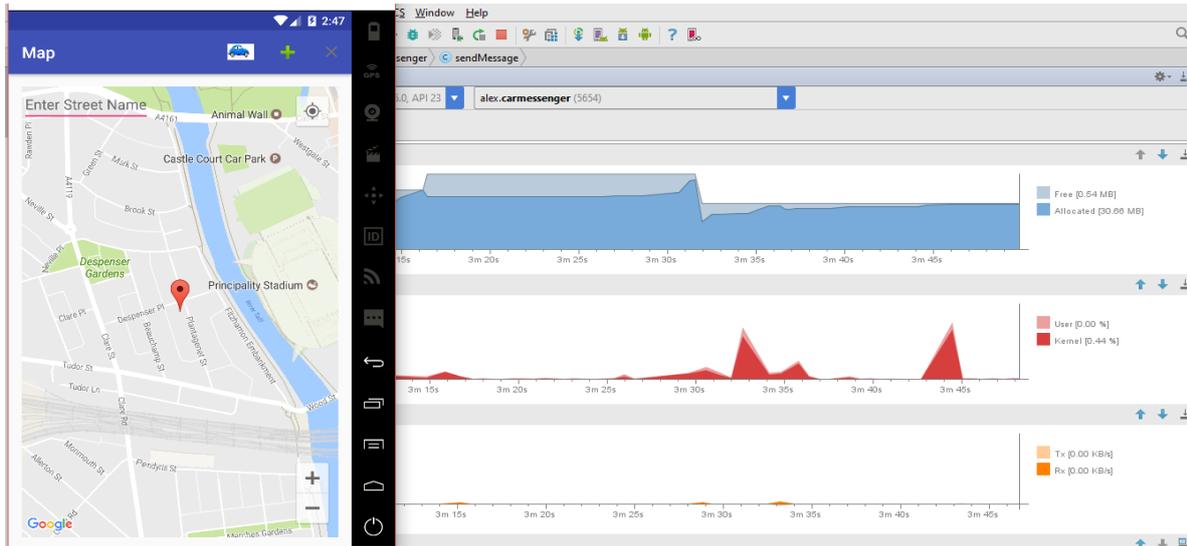


Figure 5.5 – Report Accident Activity and its monitor levels.

Evidence of further tests carried out on the applications use of resources can be found in the Appendix, in section 9.2. There are no strenuous features that are carried out within the application apart from the two activities that load Google Maps. These activities require the most computing power to load all the necessary components from the Google Libraries within the application which are not stored locally. As this project uses a laptop to simulate the connection between two drivers that are communicating, the resources used by the emulators on the laptop may be down to processing power of the laptop rather than the actual application. Further testing could be carried out on a mobile phone to gain further insight of the resources used by the application.

6. Reflection

This project has been a very good test of my skills and has proven that the original problem that I set out to achieve is very much achievable, in the fashion of the application that has been created to solve this problem. The project was a very good test of character, because it is a large project that has required lots of problem-solving techniques, to overcome challenges along the way, it has required good time management and project management skills to ensure that I stuck to a schedule to provide adequate functionality and features for the application within the time frame provided. These skills have given me an opportunity to complete this project in a way that I feel is successful and has met the goals that I set out to achieve.

The project itself was marred with issues mostly concerning Android Studio and its emulators. As I have mentioned there was a period in the implementation cycle where my project was updated but all the files became corrupted and filled with errors so I had to revert to the previous versions. I had trouble with the program itself, constantly crashing if the emulator didn't finish updating the latest gradle builds, not to mention that the program ran very slowly while the emulators were running as well so it was frustrating at times trying to debug a program with a very limited amount of processing power and RAM available. However, I think my choice for picking Android Studio to carry out the development of the application outweighs the problems I faced, and if I was to re-create the entire application, I would opt to use Android Studio again but instead of using emulators to test and demonstrate the working application, I would use real devices instead. This would allow for less reliance on the laptop that I was developing on, it would be more realistic for using devices that were available in every-day situations rather than devices that may not be as common as others.

For a period of the project I was very focussed on developing an application that had a multitude of features rather than maintaining a clear vision of the main aims and objectives of the project which were to create an application with driver-to-driver communication functionality. I was focussing on producing a complete application with lots of important features for drivers, and during this time I lost sight of the critical objective of achieving driver to driver communication, which caused me to change direction, realigning my focus on the original aim of the project.

Throughout the project, I called upon knowledge from my studies at university. I used my understanding of user interfaces that was taught during the Human Computer Interaction module, where I gained knowledge of using heuristics to assess an interface based on certain criteria, which would ensure a balanced and user-friendly interface. This knowledge helped me when it came to assessing the applications design and coming up with the heuristics that I could use, to ensure a good interface was developed. I made use of knowledge from the Database Systems module studied at university as well. This knowledge was used in writing queries for the database and interacting with the database in the code that I developed. Specifically, writing SQL queries for inserting and selecting data within the database.

The application that has been developed achieves the overall aims of the project and I would consider it to be a success. This is based on the evidence I have provided throughout the report, like the test cases and the implementation section of the report that detail the objective specifics of the application that have been accomplished. There are still some considerations that mean an application like this could be challenging to implement as you would have to use mobile phones while driving which brings into question the safety of the road-user. Full voice control and ability to follow instructions based on natural language techniques or with a virtual assistant that could capture user communications to

relay these messages, could be a technique of deploying this application without the use of a mobile phone.

6.1 Self-Evaluation

I am pleased with the outcome of the project as I have created an application that could be very useful in the future of communication between drivers. My time-management during the project has been effective and allowed me to maintain the schedule proposed at the start of the project. By setting weekly targets and goals with deliverables I was able to track the progress I was making over the course of the project. The application created achieves the problem that I wanted to solve in an effective manner, that I would like to continue developing.

Looking back at the project and the implemented solution that I have created, there are aspects to my own learning and methods that I would change for next time. For example, when learning the Android programming language, I jumped straight into it without looking through sample applications provided in Android Studio or looking through some books to get a basic understanding of the fundamentals of the language. Next time I would go about learning the language in a more methodical approach, trying to learn the basics of the language by following some tutorials to understand the layouts and objects to use in certain situations. A prime example of this was the action bar menu that I included in the activities of the application. I implemented this feature very late on during implementation because I had assumed it was not an important aspect to the application because users could use the emulators back buttons to return to the previous screen. Having looked through some other applications and tutorials online trying to fix another issue, I discovered that the action bar would be very useful from a user perspective to help navigate successfully around the application. Not only that, it would help with the user preferences that I had used to retrieve the user's inbox messages based on who was logged in to the application on that device. I feel that if I had looked at basic sample applications first, then I would have noticed how fundamental the action bar would be in my application and implement that sooner within development because it is so useful when navigating around the program.

Another aspect of my learning throughout the project was that I faced some challenges understanding the Android language. Although it is like Java, which I have used previously. I haven't had to use Java in the same capacity as I did for Android and that is what I found difficult. Learning and understanding all the different aspects of the language to be able to make use of the complex parts of Android was challenging because they are slightly different to the Java versions of the language, or they might use the same methods yet return different parameters, so each time I was using new methods I had to ensure that I checked for any changes in the developer's guide online to keep in touch with what I was doing.

I think I handled big decisions well throughout the project. I had to make key decisions during the development of the application, for example, if I would use a server or not, and whether it would be easier to use a local web server or an online server. I came to the decision that I would use a database because it would be the best way of making sure that messages reached user's inboxes and that all the accidents reported were available to be seen by other users. It was good to learn a new aspect of programming for a different audience than I have done in the past. I enjoyed learning about each component of the language and how mobile developers have to take other factors into consideration like resources and what permissions are needed in order to access content or part of the user's device. All these aspects were new to me as when developing programs within any programming language I have experience in, I have never had to consider the implications of why I would access a user's location or even need to add that feature. Within this project, I was required to access the location to locate the user for the Google Maps activity, making it necessary to request that user's permission

prior to accessing it. An interesting aspect of the Android language which I found different to other languages is that, you cannot implement certain methods if there is no corresponding link in the application's manifest file explicitly stating a certain permission has been used. For example, the permissions aspect of the application must be explicitly stated in the manifest file for you to be able to implement the methods that utilise a user's location or other permission related methodology within the application.

I made some assumptions when I was developing the application, which I had overlooked during this stage of the project. For example, when it came to implementing the client-to-server communication between devices. I spent a couple of weeks trying different methodology until I found the socket programming technique which was successful. Overall I think these assumptions cost me some time and resources while developing, and that next time I would ensure that the main functionality was implemented fully before trying to create the entire application as then I could focus on additional features with whatever time was left to spare, rather than working on the main functionality right up until I had no time left to continue development.

7. Conclusion

To conclude, this report has described in detail the methodology and approach I have taken to solve the problem of drivers not being able to communicate with each other, regarding their vehicles. The problem has been solved by the implementation of an application that has the functionality to carry out communication between drivers among a host of other features that are considered useful to a driver, such as traffic and accident information.

The application created is a success because of its overall functionalities and its ability to solve the problem of communicating between drivers. The communication aspect is an achievement because it takes the project goals slightly further than what I intended to initially implement, and it is a big step in the industry as it looks at the possibilities of driver-to-driver messaging services that could be implemented. If I can make an application that takes this idea to a good standard and does allow drivers to communicate in a respectful manner, then car manufacturers with multi-million dollar industries can consider developing something similar on a much larger scale and within an environment that allows safe testing and development.

The projects achievements are that the application allows drivers to communicate with each other regarding the status of their vehicles. This is important in regard to maintaining a fully law abiding vehicle that has no issues that could give law enforcement agencies an excuse for giving out a fine or penalty. The application has the option for users to report accidents which is a useful feature for those who inevitably end up in a traffic jam that causes delays. Reporting accidents is easy to do within the application and can help other drivers a lot as they reduce the chance of getting caught up in the traffic as well. Alongside the reporting accidents feature, there is the traffic information functionality that lets drivers see the accidents reported and other important traffic information. These features have created a successful application that could be used by drivers around the country to enhance their road experience.

There are some issues with the application I have developed, but it has been a successful project overall and has taught me many things about the language and its components, that I can take into any other development or projects I am part of in the future.

8. References

1. Department for Transport, Statistics Relating to licensed vehicles and new vehicle registrations for 2015, 14/04/2016, <https://www.gov.uk/government/statistics/vehicle-licensing-statistics-2015> - [Accessed 30/01/2017]
2. Cory Schmidt, What is Android? Here is a Complete Guide for Beginners, July 2016, <https://www.androidpit.com/what-is-android> – [Accessed 10/04/2017]
3. Jeff Dunn, There’s no hope of anyone catching up to Android and IOS, 22/08/2016 <http://uk.businessinsider.com/smartphone-market-share-android-ios-windows-blackberry-2016-8?r=US&IR=T> -[Accessed 10/04/2017]
4. Liam Turner, Mobile_Android Slide 27, Learning Central, Emerging Technologies, Cardiff University- Accessed [11/04/2017]
5. <http://www.bump.com/> -[Accessed 30/01/2017]
6. *Co-operative Road Traffic: Foresight, Safety, and Comfort*: Car2carConsortium: Newsletter, November 2015, <https://www.car-2-car.org/index.php?id=5> – [Accessed 31/01/2017]
7. Reis et al.2014: Deploying Roadside Units in Sparse Vehicular Networks: What Really Works and What Does Not. IEEE Transactions on Vehicular Technology, Vol.63, No.6, July 2014 [Accessed 31/01/17]
8. Reis et al.2014: Deploying Roadside Units in Sparse Vehicular Networks: What Really Works and What Does Not. IEEE Transactions on Vehicular Technology, Vol.63, No.6, July 2014 [Accessed 31/01/17]
9. Pete Bigelow, Feds want V2V Communication in New Cars Starting in 2021, 15/12/2016, <http://blog.caranddriver.com/feds-want-v2v-communication-in-new-cars-starting-in-2021/> [Accessed 01/02/2017]
10. Ford, <http://www.ford.co.uk/Cars/Focus/Features> - [Accessed 01/02/2017]
11. Rosa Yanez Gomez, Daniel Cascado Cabellero, and Jose-Luis Sevillano, Heuristic Evaluation on Mobile Interfaces: A New Checklist, The Scientific World Journal, 01/06/2014, <https://www.hindawi.com/journals/tswj/2014/434326/> [Accessed 11/04/2017]
12. Jason Wei, Incorporating Socket Programming into your Applications, 27/03/2010, <https://thinkandroid.wordpress.com/2010/03/27/incorporating-socket-programming-into-your-applications/> Accessed [01/04/2017]
13. Jason Wei, Incorporating Socket Programming into your Applications, 27/03/2010, <https://thinkandroid.wordpress.com/2010/03/27/incorporating-socket-programming-into-your-applications/> Accessed [01/04/2017]
14. Change Button Background on Touch, 12/2012, <http://stackoverflow.com/questions/8132384/change-button-background-on-touch/8132500#8132500> Accessed [10/03/2017]

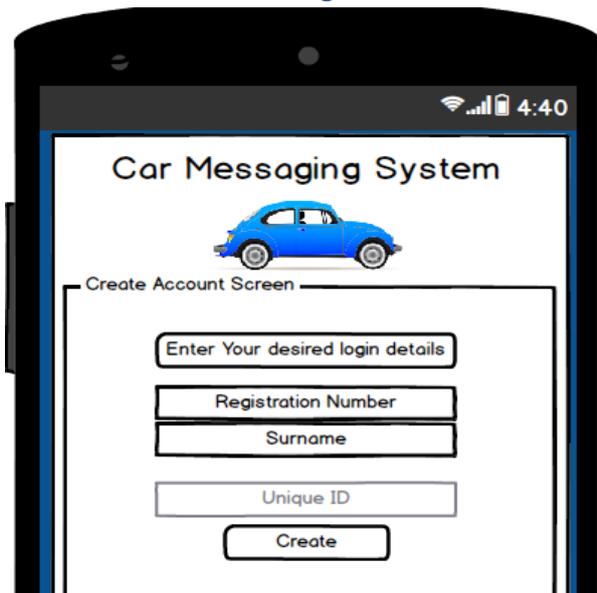
The following references are all code related. Either I have used them in some capacity in order to create my own code or have used the methods provided in their API package.

15. <https://developer.android.com/reference/java/security/package-summary.html> [Accessed 30/01/2017]
16. Java Security, <https://developer.android.com/guide/topics/connectivity/wifip2p.html> [Accessed 31/07/2017]
17. Dark Blue Racing Car (Top View) Image, Created by user qubodup, 20/01/2014, <https://openclipart.org/detail/190177/dark-blue-racing-car-top-view> car image used in my SendMessage Activity - [Accessed 06/02/17]

18. Hardik Trivedi, How to Avoid Force Close Error in Android, 20/08/2011, <https://trivedihardik.wordpress.com/2011/08/20/how-to-avoid-force-close-error-in-android/> - used for my error handler activity, [Accessed 20/03/2017]
19. Daniel Bradley, Regular Expression to Validate UK Number Plates, <https://gist.github.com/danielrbradley/7567269> - [Accessed 20/03/2017]
20. Ravi Tamada, Android Working with Action Bar, 11/2013, <http://www.androidhive.info/2013/11/android-working-with-action-bar/> - Helped me to develop my own action bar menu, [Accessed 25/03/2017]
21. Smruti Ranjan, Android Login Screen with SQLite Database Example, 13/03/2013, <http://techblogon.com/android-login-registration-screen-with-sqlite-database-example/> - [Accessed 10/02/2017]
22. Thread With Handlers, Android Review, http://androidexample.com/Thread_With_Handlers_-_Android_Example/index.php?view=article_discription&aid=58 - [Accessed 23/04/2017]

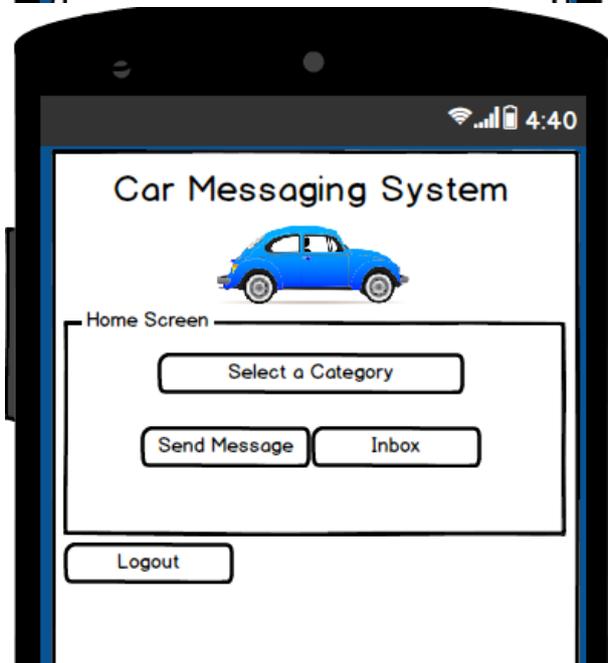
9. Appendix

9.1 User interface designs



This alternative method of creating an account works in parallel with the ApplicationLogin2 screen that requires the uniqueID to be entered upon logging into the system.

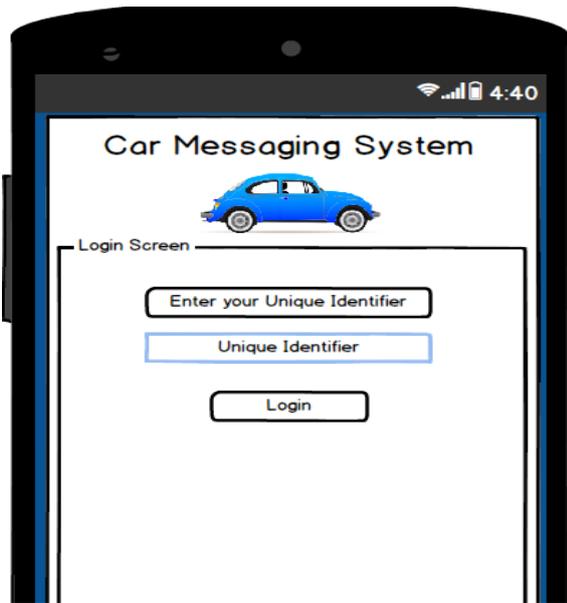
This approach is simple and still secure. The UniqueID lets you log into the system once and then will be used to authenticate the messages sent by that user by recognising the ID sent with the message itself.



This is the home screen that would be displayed before users, after going through the login process. This is the basic menu screen that would be available to users.

The functions available are sending a message or accessing the inbox to see if their own car has any problems.

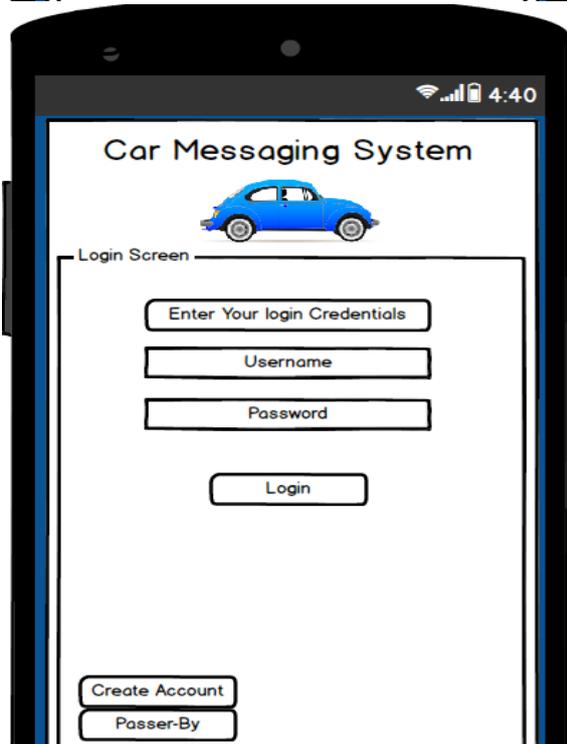
This is only the basic menu form as these are the main requirements of my system.



This is the alternative approach to the login screen. Users get a unique identifier based on creation of their account. When they first use the system they will be taken to the create an account form. This will require their surname and registration plate to be entered. From that data a unique identifier will be formed based on the data entered.

This unique id will then be what they use to login to the application. This approach would look at storing the access token so that from the same device, the user only has to log in once until they actively log out of the system.

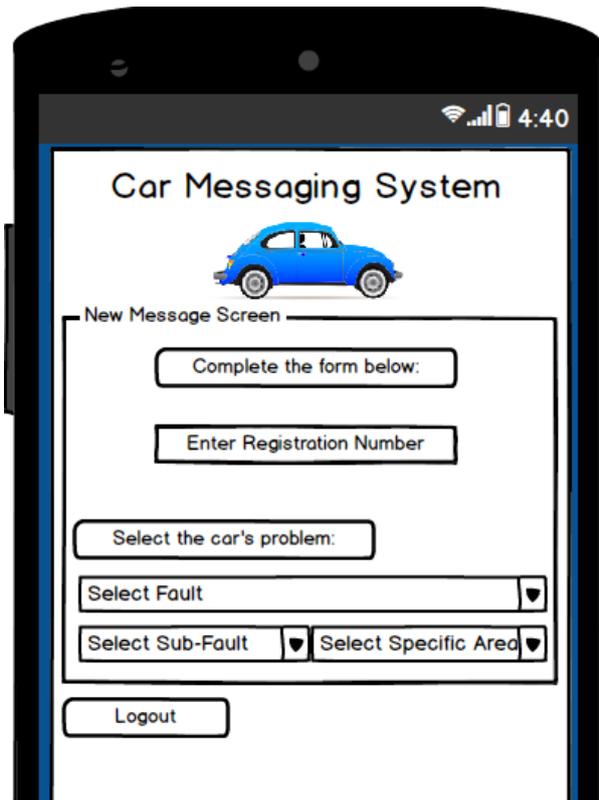
Storing the access token would be a lot easier for the user rather than having to log in each time. It is also faster and will allow the user to record the registration plate of the target vehicle more quickly.



Another approach to the login-screen that will allow people who do not own vehicles to use the application to report problems. The way they would do this is by clicking on the passer-by function rather than creating an account and logging into the system.

This approach is a possibility if I have time to implement the additional functionality at the end of the implementation stage. My target audience for the application is vehicle owners because I have a way of authorising them, by using their registration numbers.

If there is no vehicle to register with and random users are then manipulating the system by sending bogus reports and issues, this could cause irritation among the other vehicle users. So I will need to consider options when implementing these features.



The New Message Form will be the main function of the system. It will provide users with the ability to send messages reporting any issues with other driver's vehicles.

The form itself is quite straight forward but is not very easy on the eye. The idea is that users will enter a registration number of the car that they have identified, and then follow the combo boxes in an attempt to diagnose the problem.

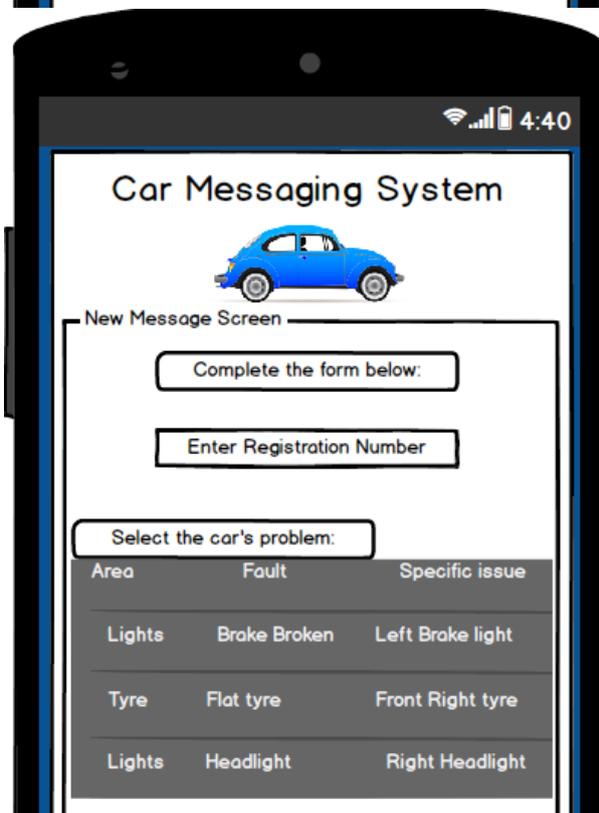
E.g. First combo box provides all the possible classes they may be reporting, i.e. a car fault, dangerous driving etc. The sub-fault is the secondary class related to the first combo box and lastly the third combo box is then again related to the selected thing in the sub-fault category.

Example:

Select Fault - Problem with Car

Select Sub-Fault - Brake lights are not working

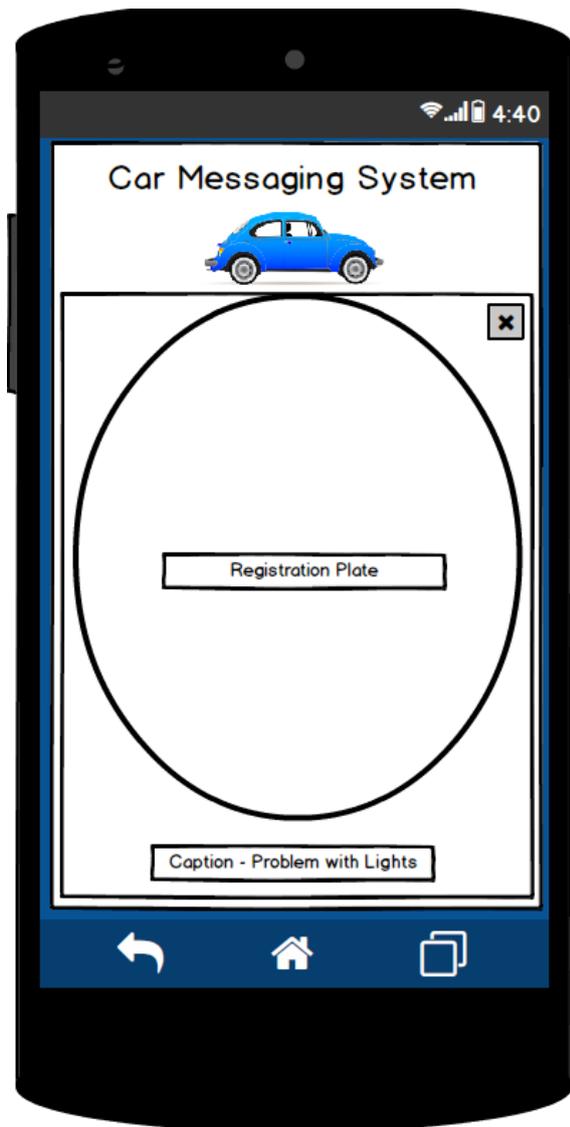
Select Specific Area - Left rear brake light is not working.



The Problem area for the table in the screen would be touch screen. User would only have to select which problem they wanted to report

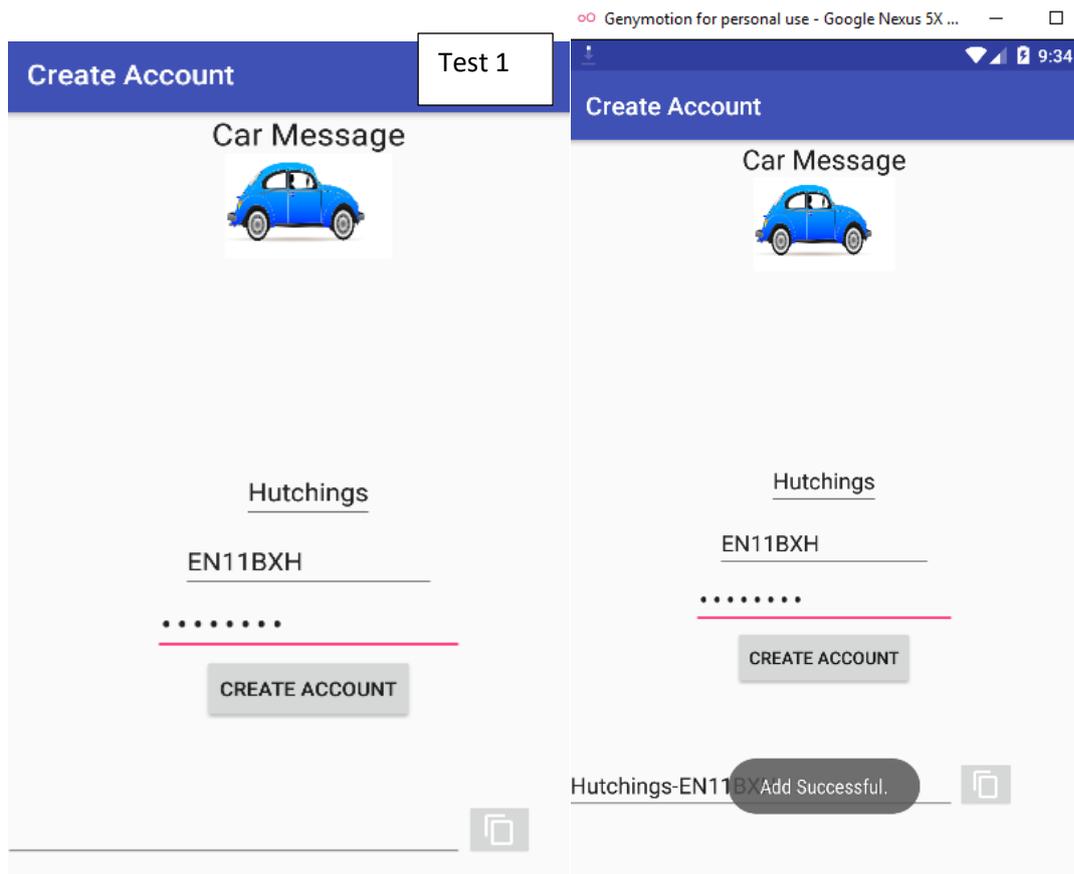
A much simpler approach than the previous form. This time I would display a pre-determined set of issues that could be selected from a table of faults.

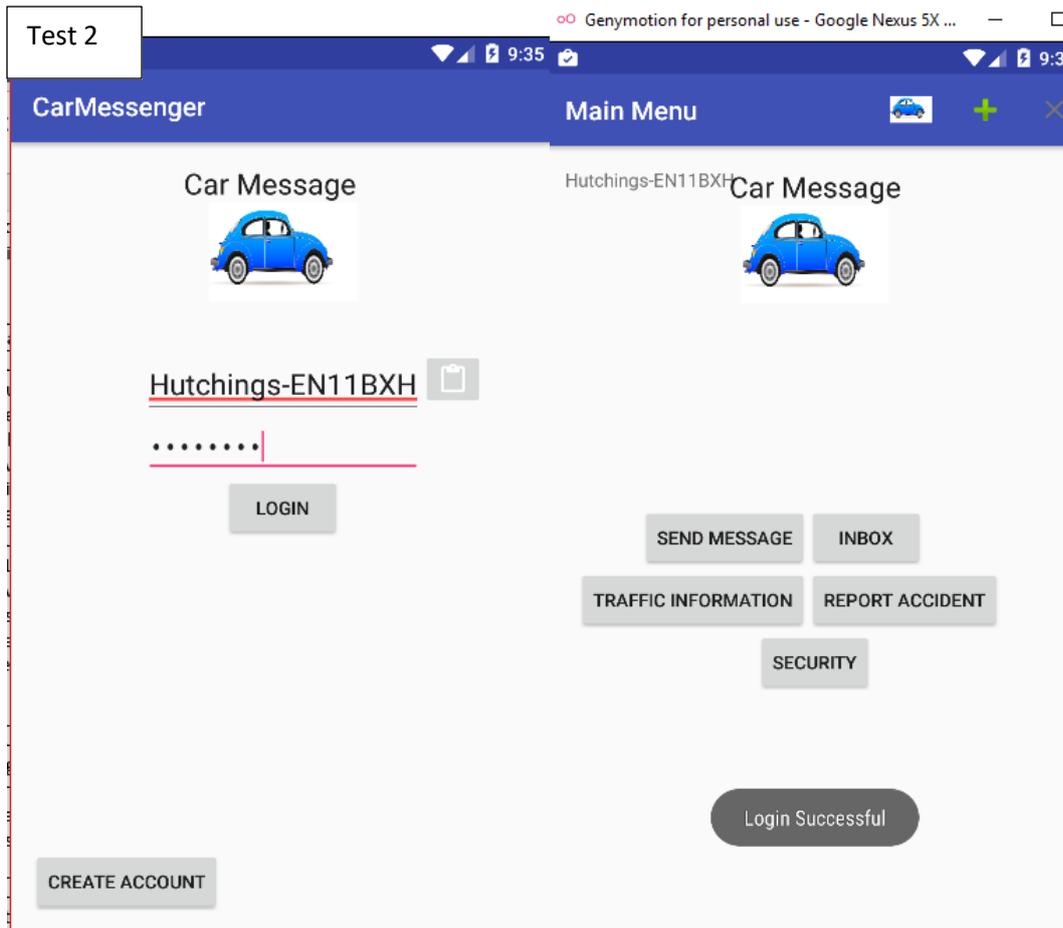
Simple, effective and gets the main functionality across.

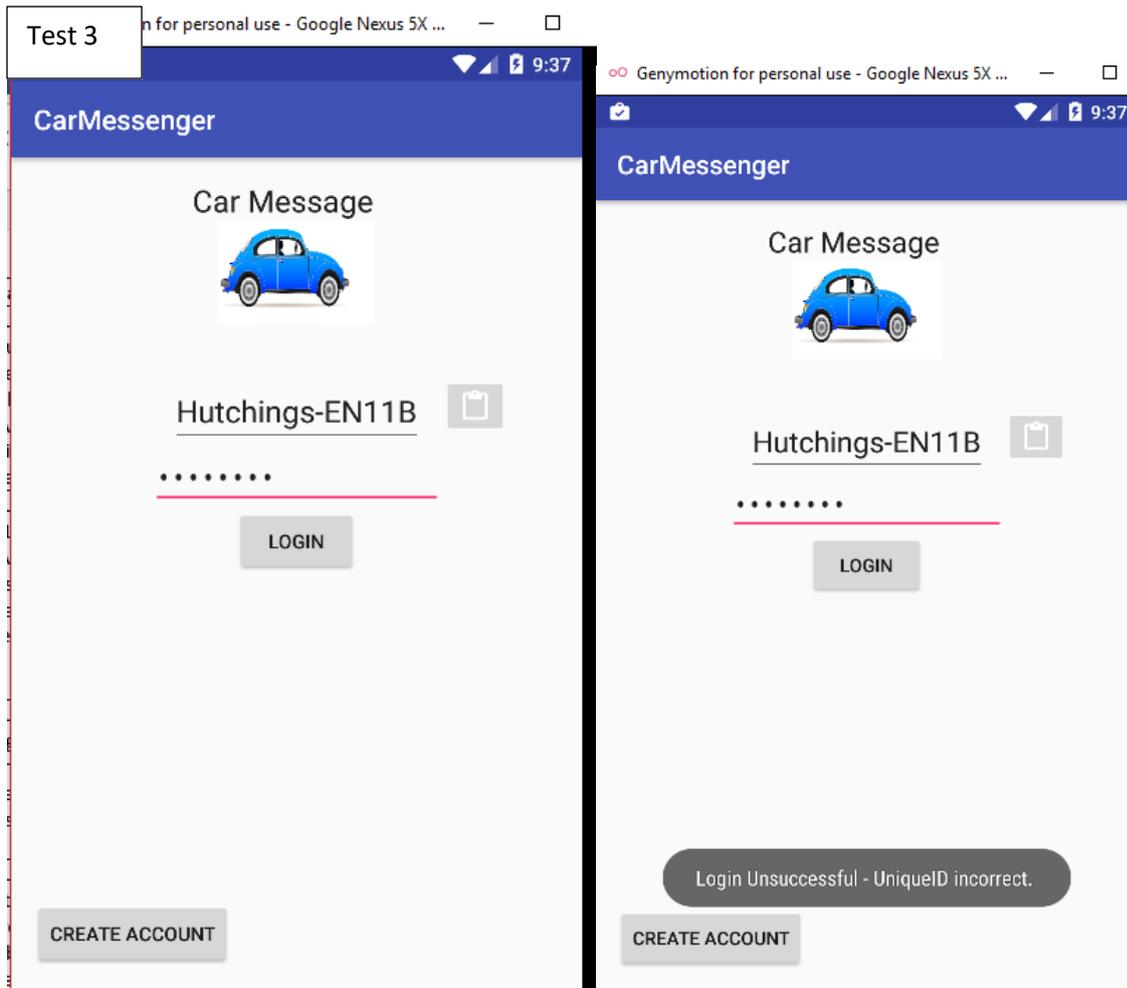


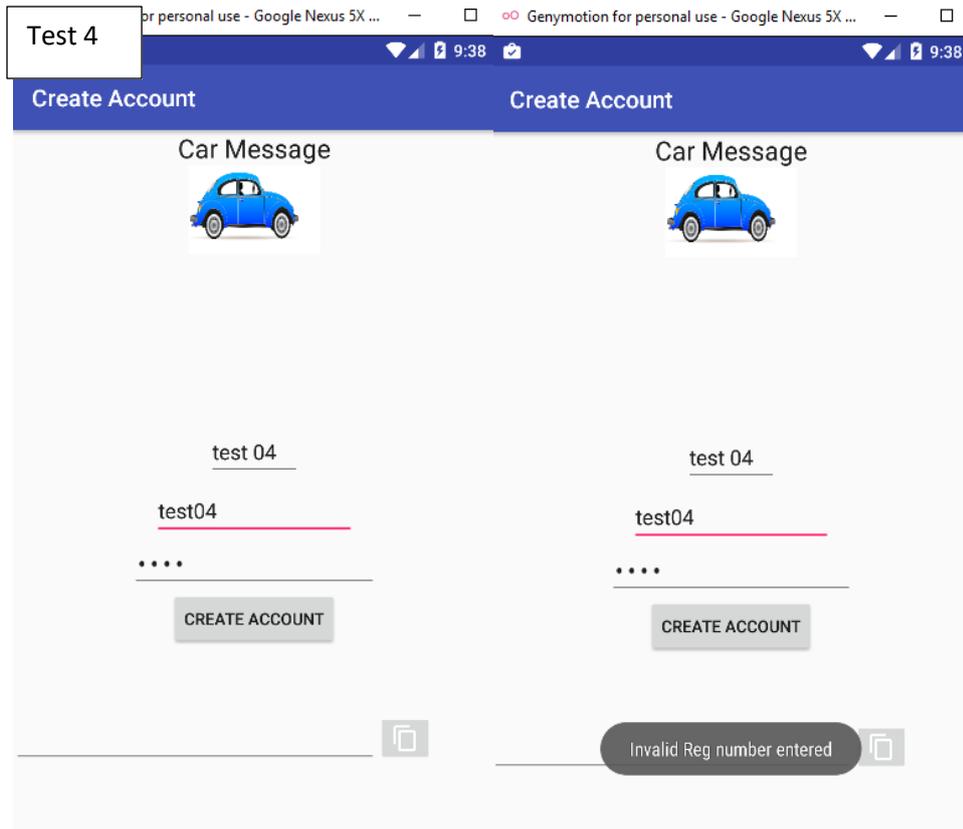
The Passer-by screen is an additional feature that I am hoping to implement if I have time. The main function of this feature is to access the camera of the user's phone and allow them to take a picture of the number plate, and then giving them a caption with which they can write a sentence about what the fault of the car they have spotted is. The idea very much follows social media applications like WhatsApp when taking photos using their camera feature.

9.2 Test Case Screenshots

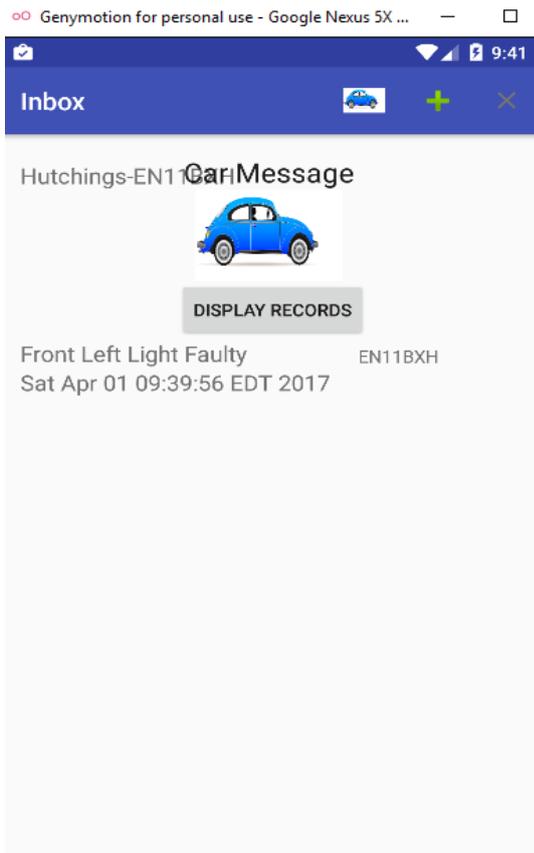
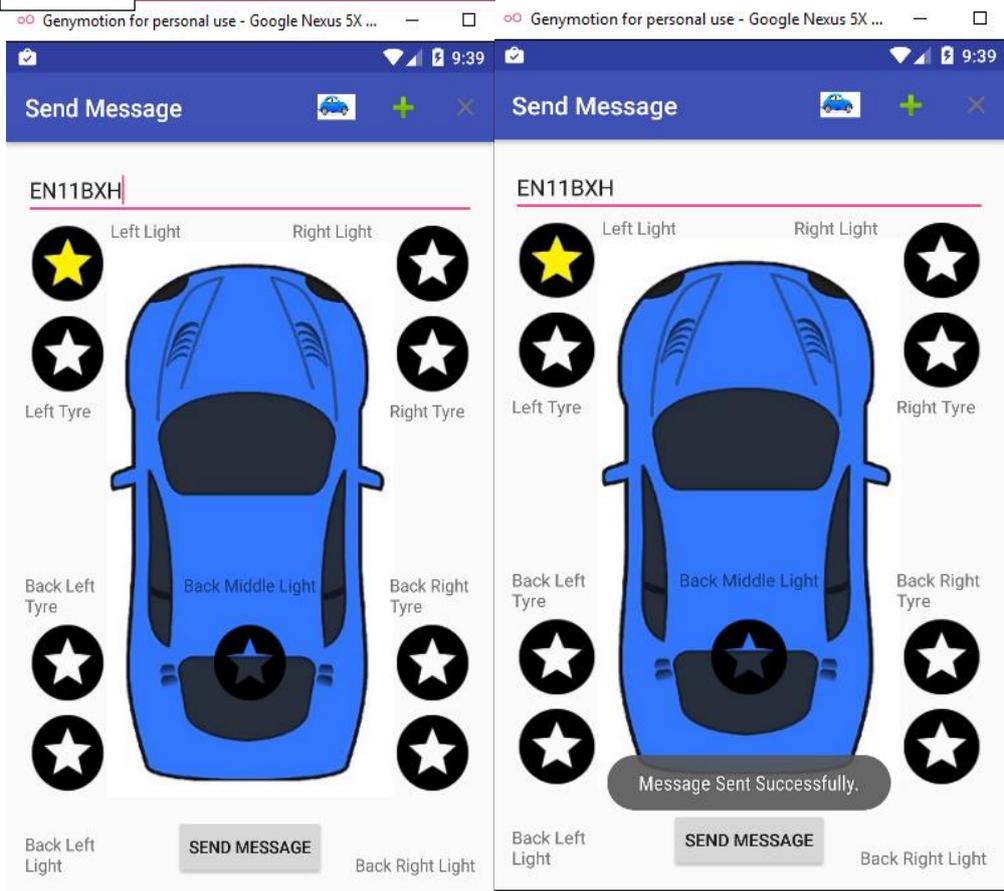


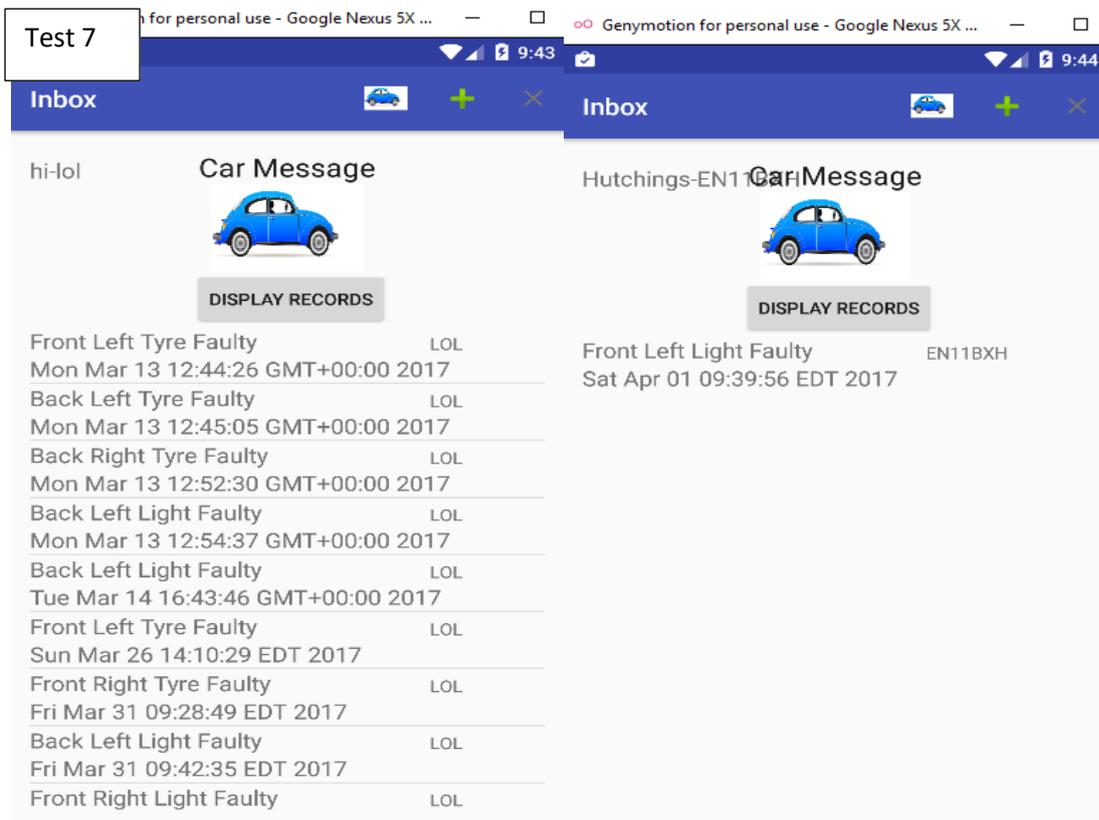
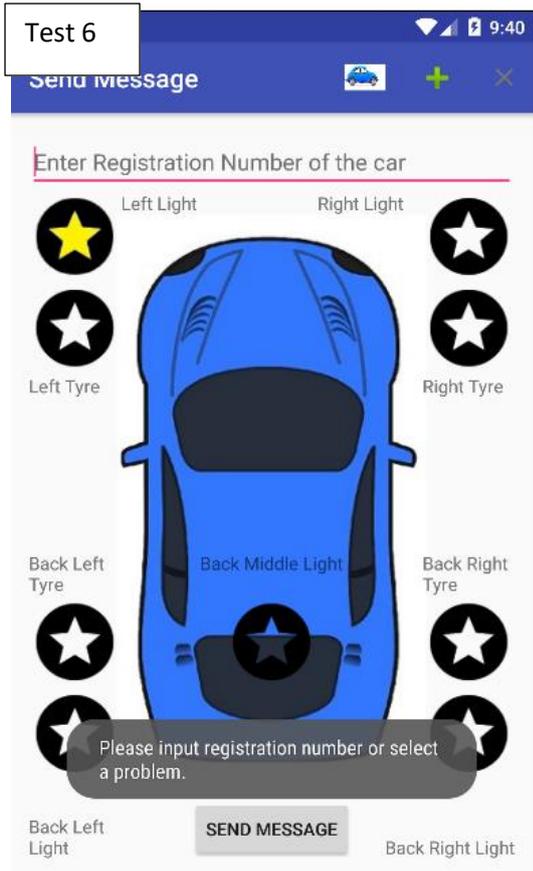


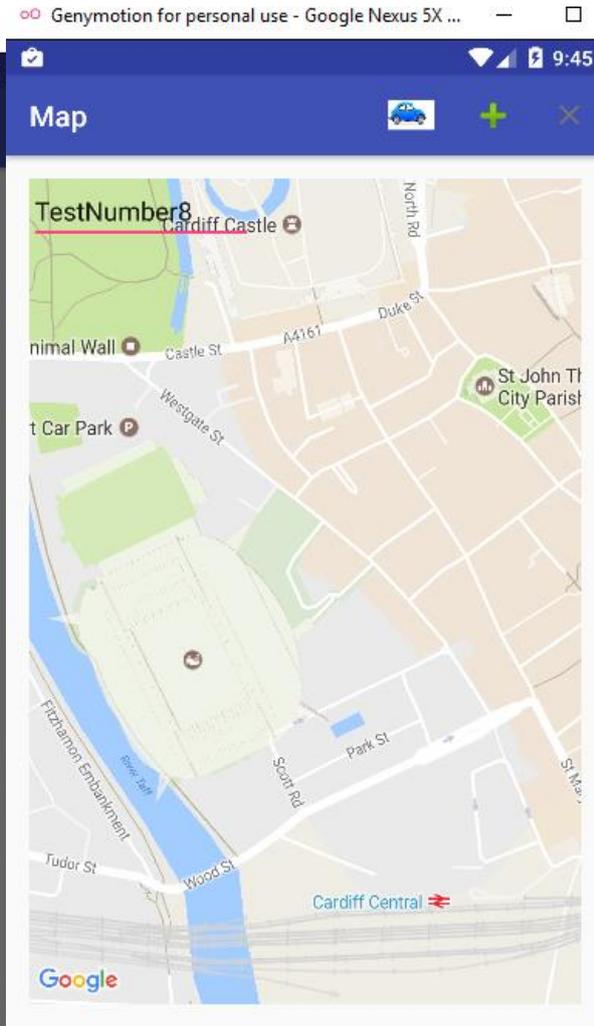
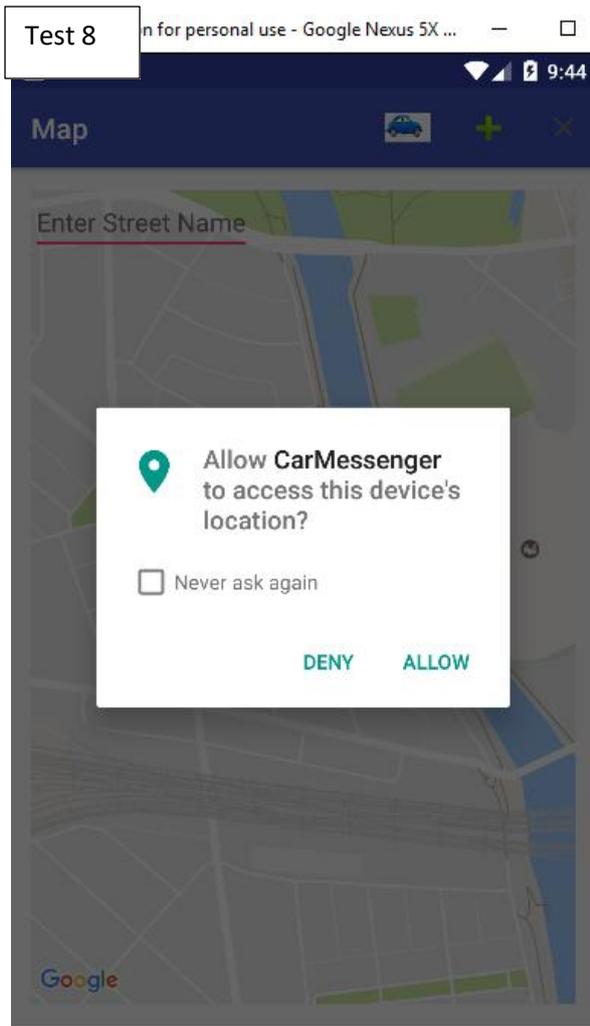




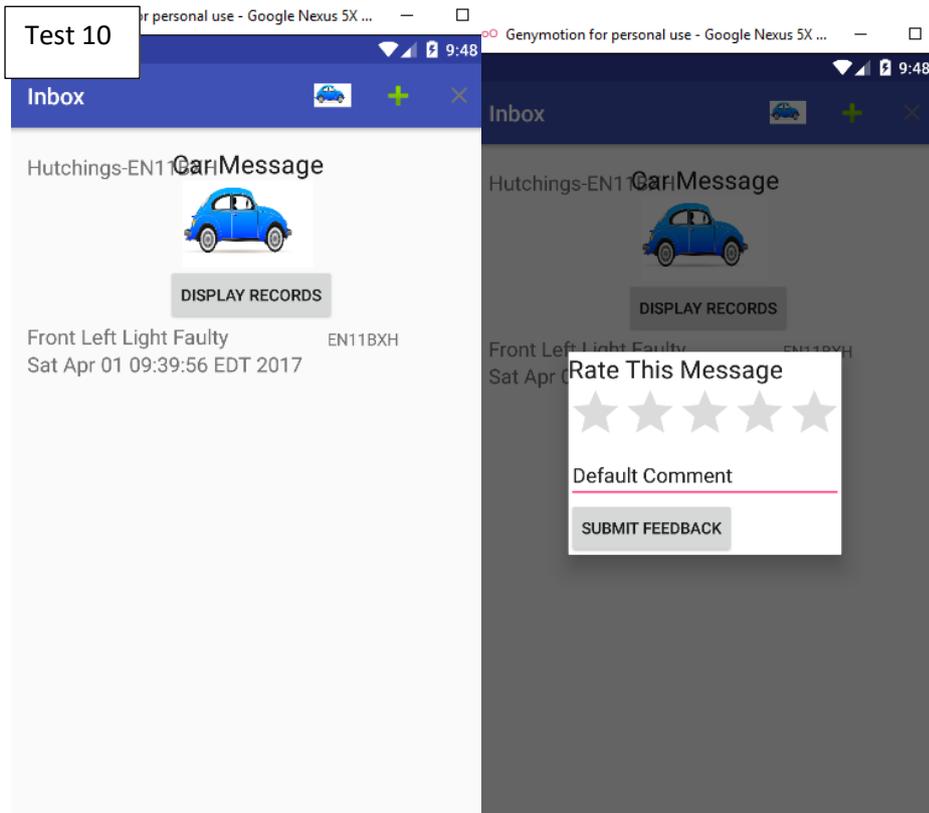
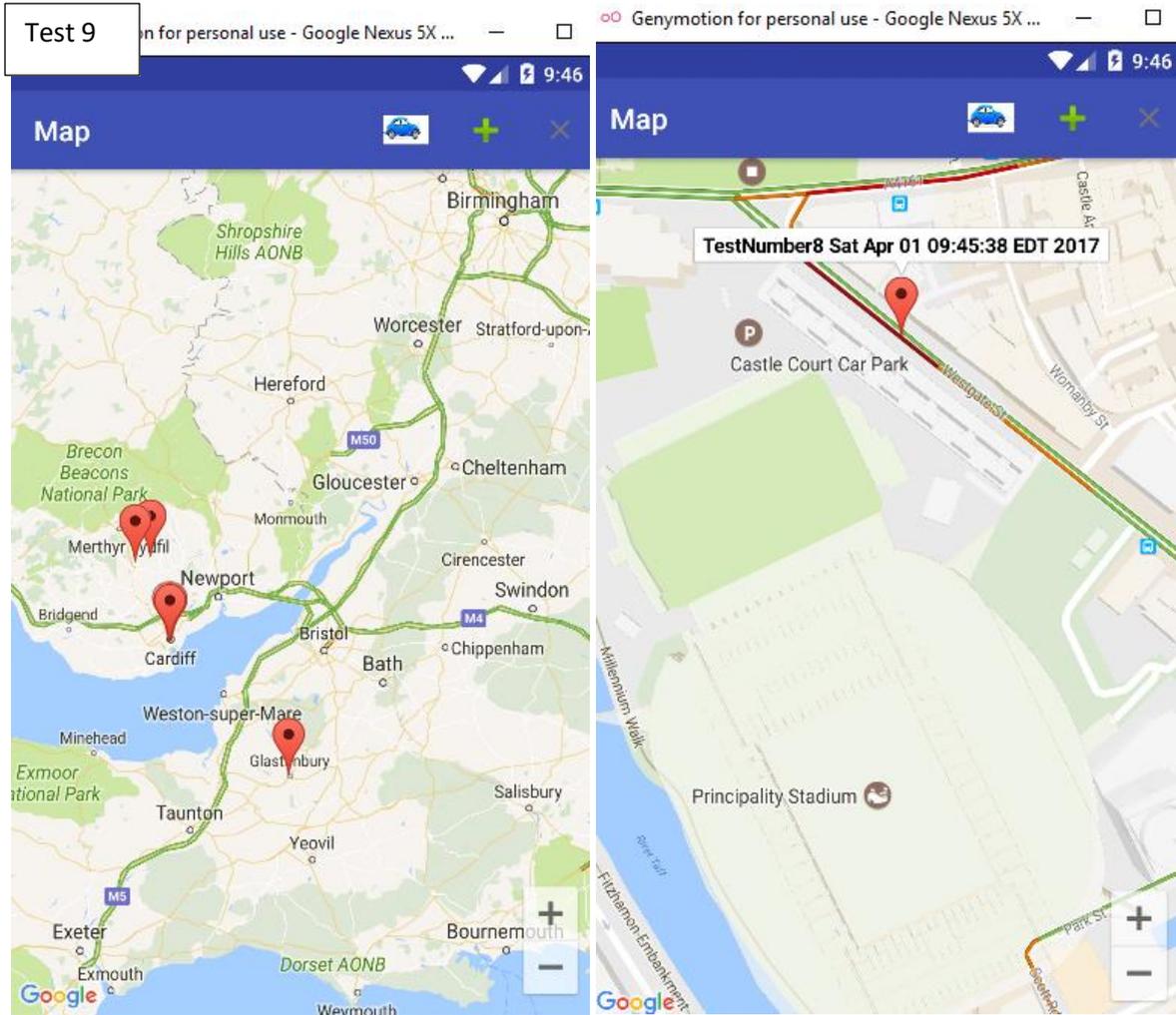
Test 5

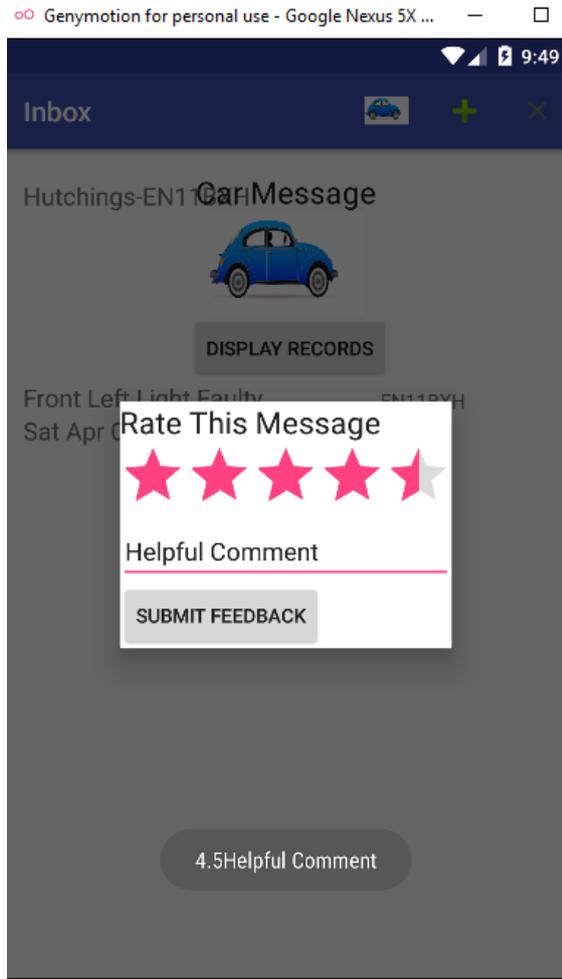


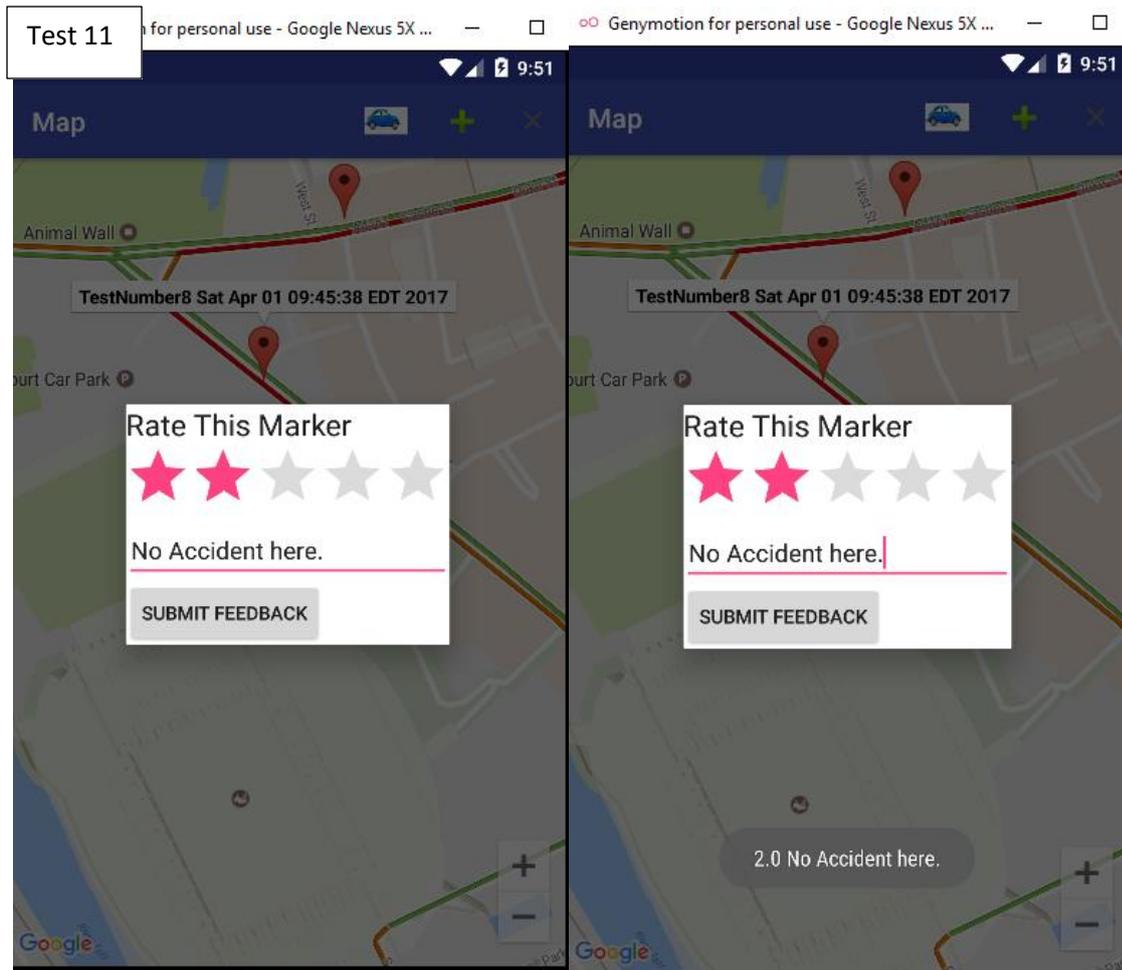




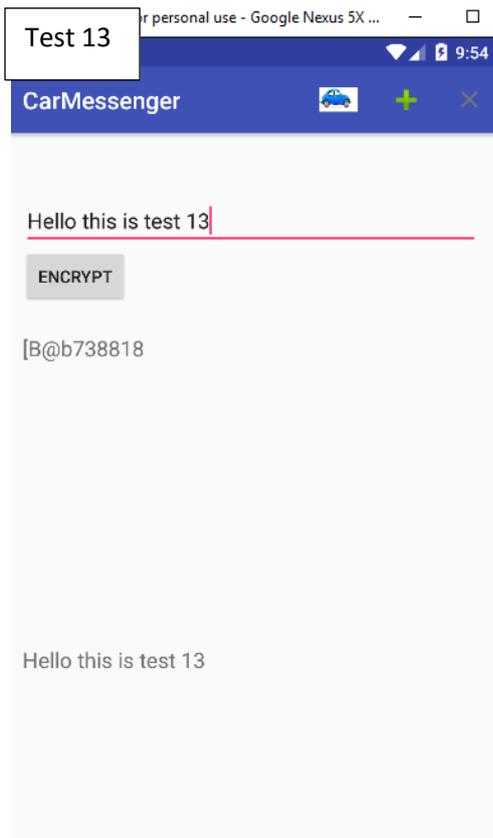
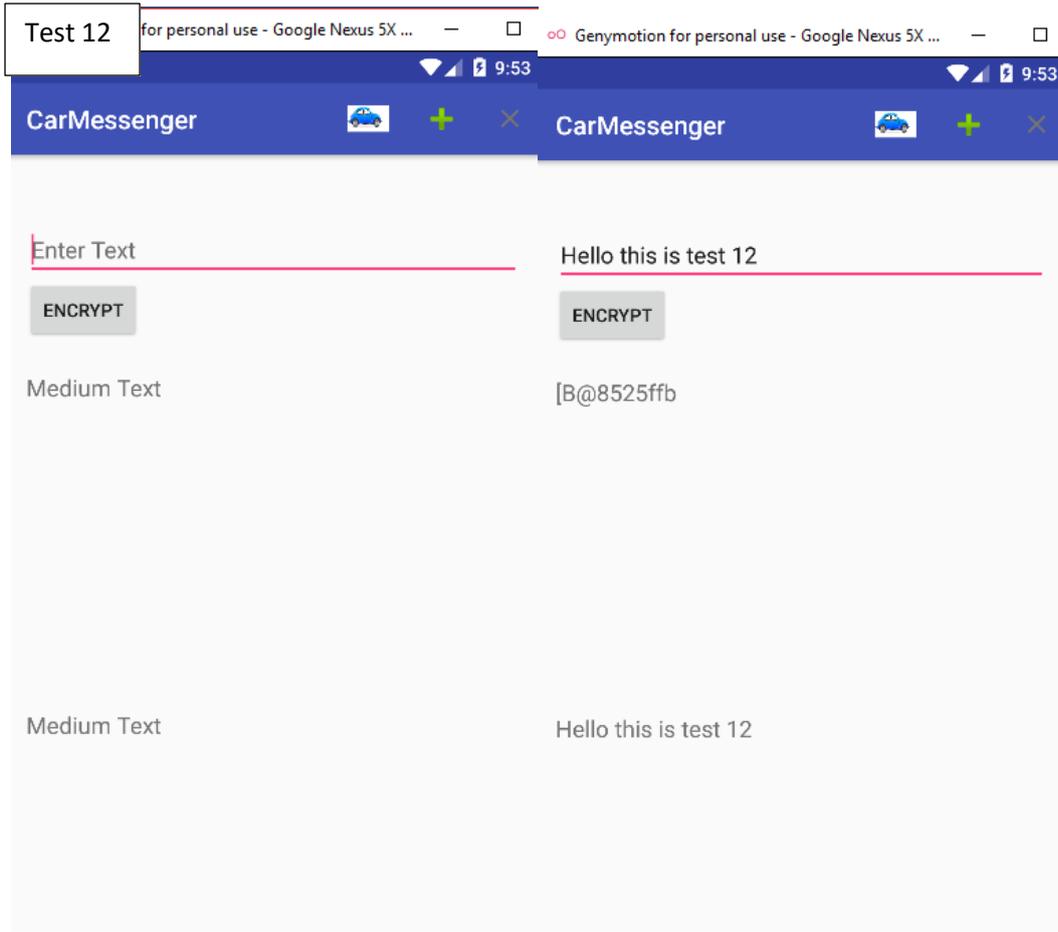
Alex Hutchings 1412529: Car Messaging System.







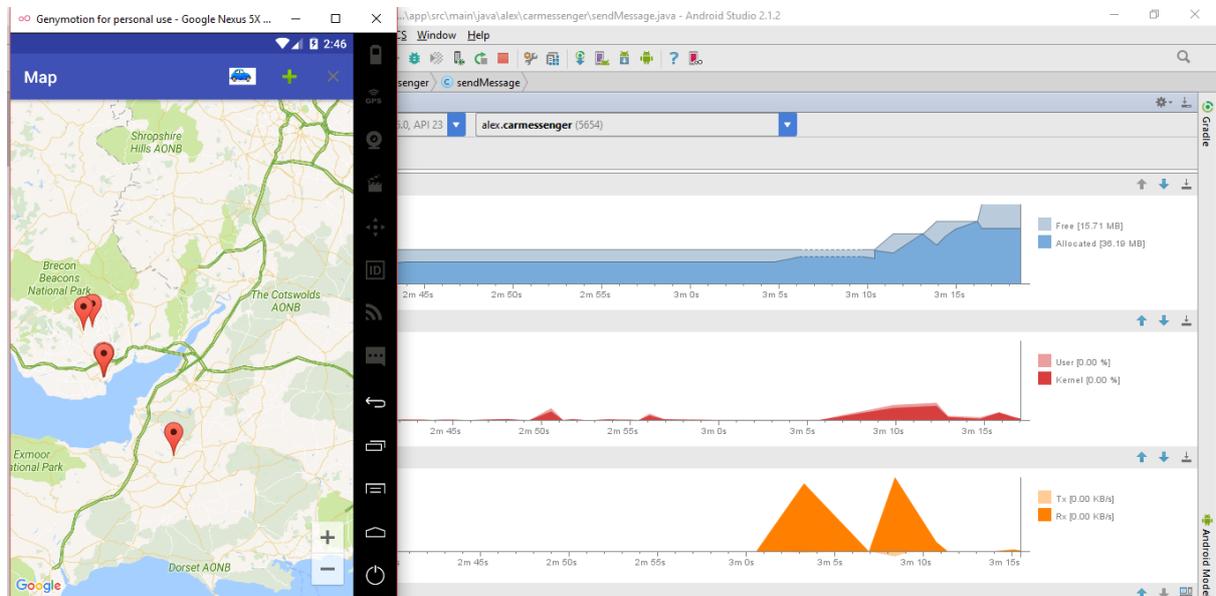
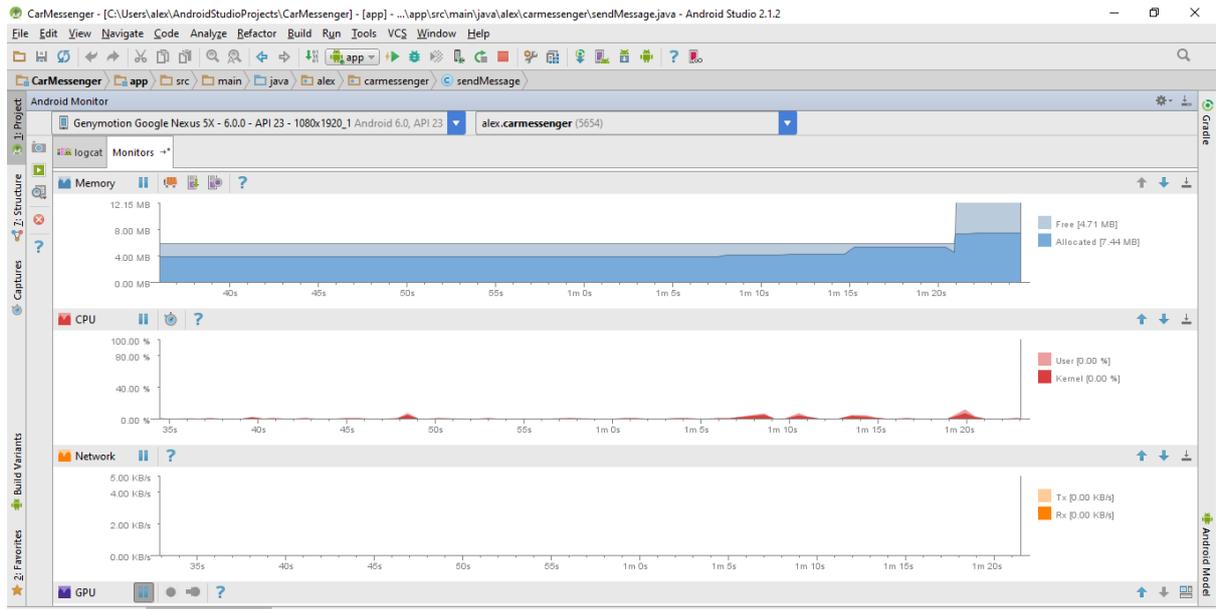
Alex Hutchings 1412529: Car Messaging System.

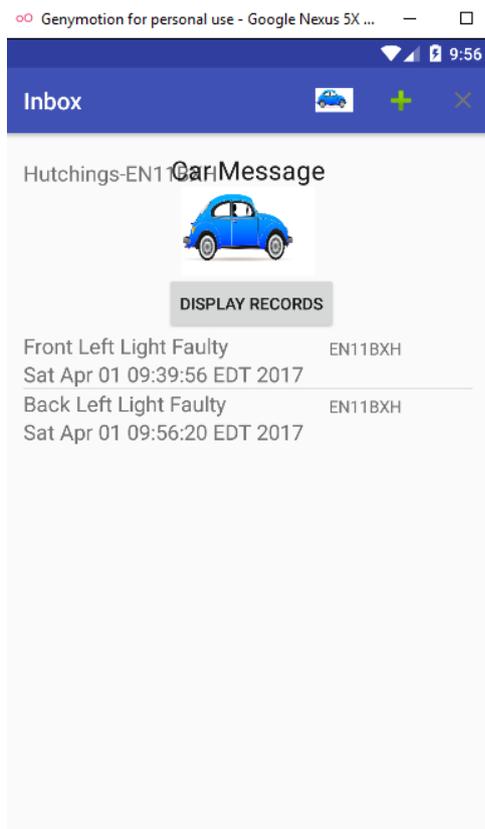
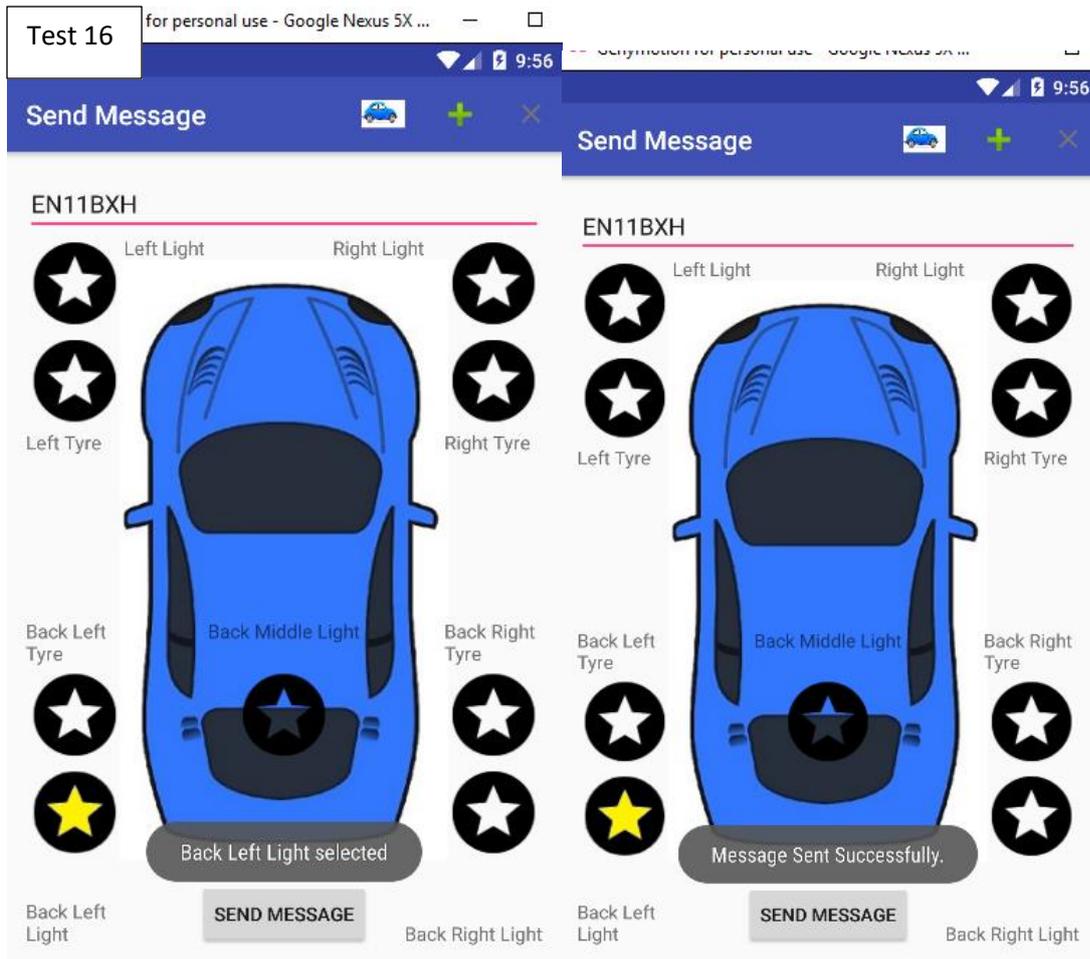


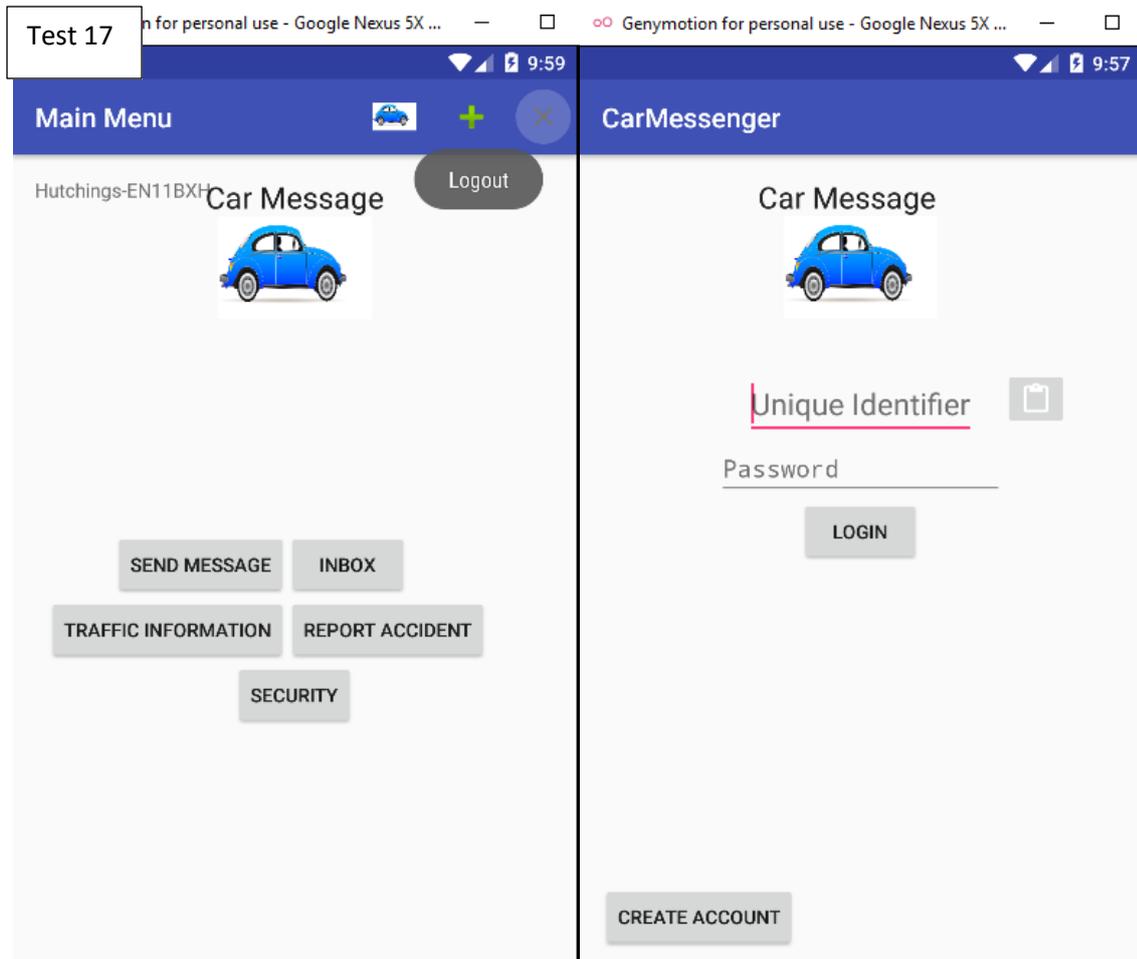
Test Cases 14 and 15

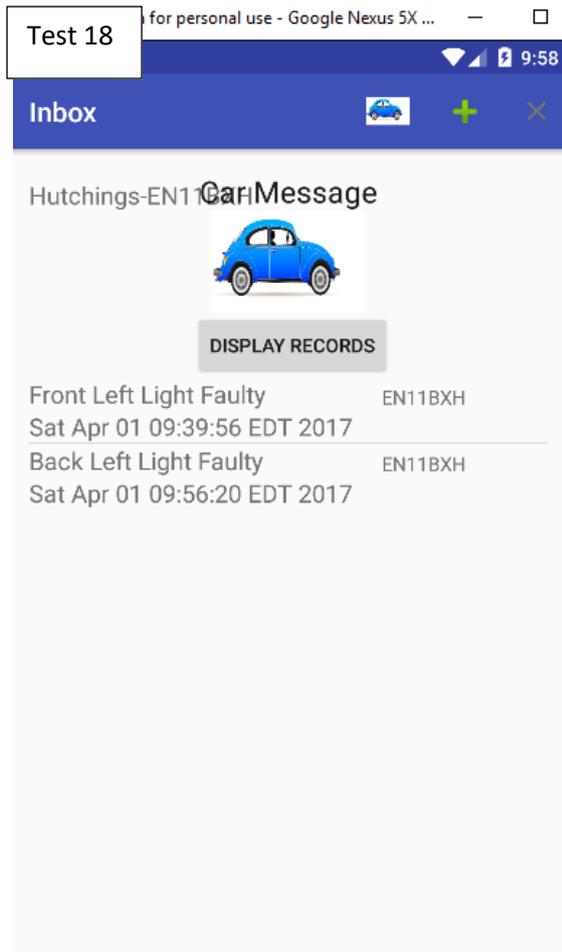
The image displays three screenshots related to the Car Messaging System. The top screenshot shows the login screen of the CarMessenger app, featuring a blue car icon, a 'Unique Identifier' field, a 'Password' field, and 'LOGIN' and 'CREATE ACCOUNT' buttons. The middle screenshot shows the Android Monitor interface, displaying performance metrics such as Memory (Free: 3.37 MB, Allocated: 8.29 MB), CPU usage (User: 0.00%, Kernel: 0.00%), and Network activity (Tx: 0.00 KB/s, Rx: 0.00 KB/s). A log entry shows the message: 'message = ("Back Right Tyre Faulty");'. The bottom screenshot shows the 'Send Message' screen, which includes a car diagram with buttons for 'CONNECT PHONES', 'HOME', 'Left Light', 'Right Light', 'Left Tyre', 'Right Tyre', 'Back Left Tyre', 'Back Middle Light', and 'Back Right Tyre'. A 'SEND MESSAGE' button is at the bottom, and a notification at the bottom of the screen reads 'Message Sent Successfully.'.

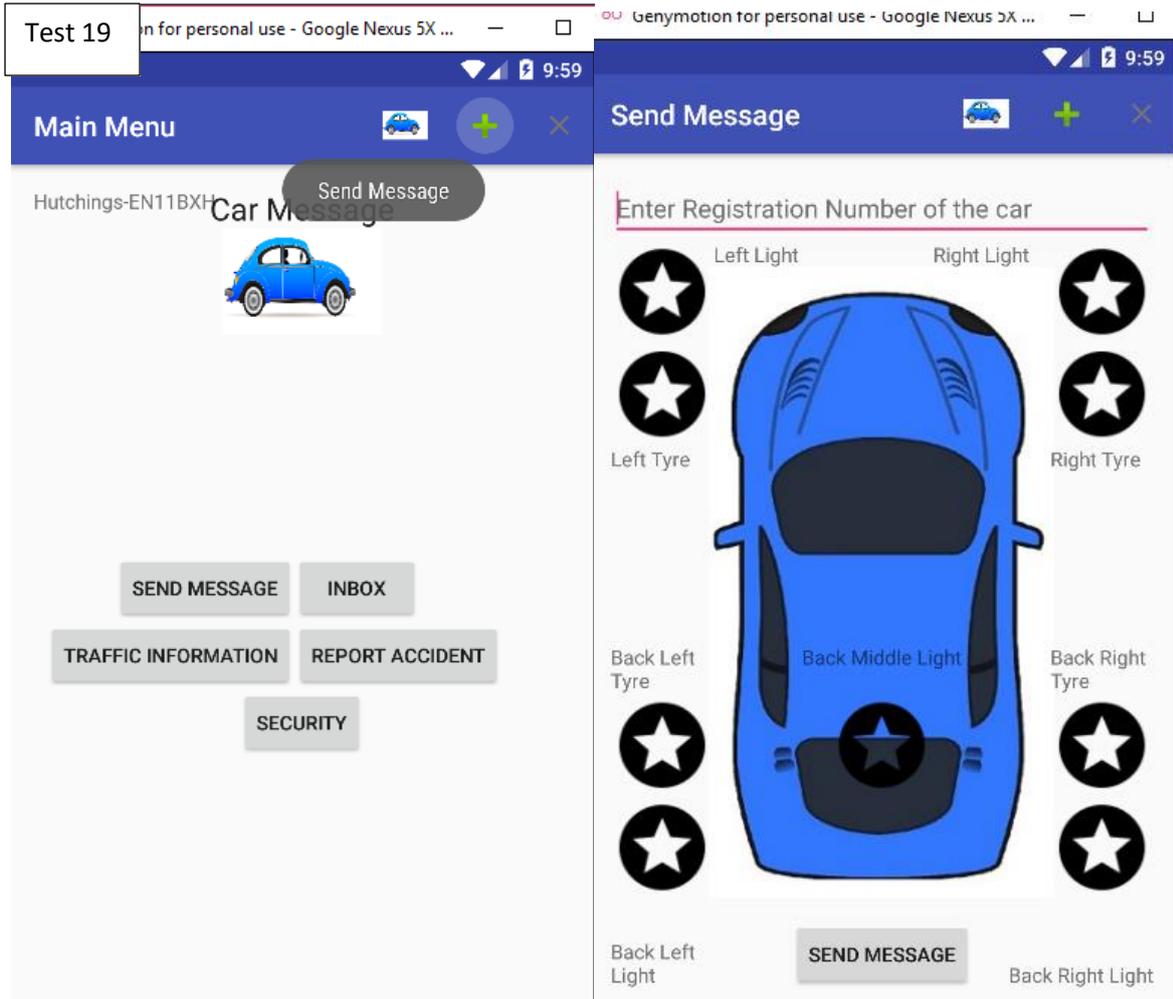
Alex Hutchings 1412529: Car Messaging System.



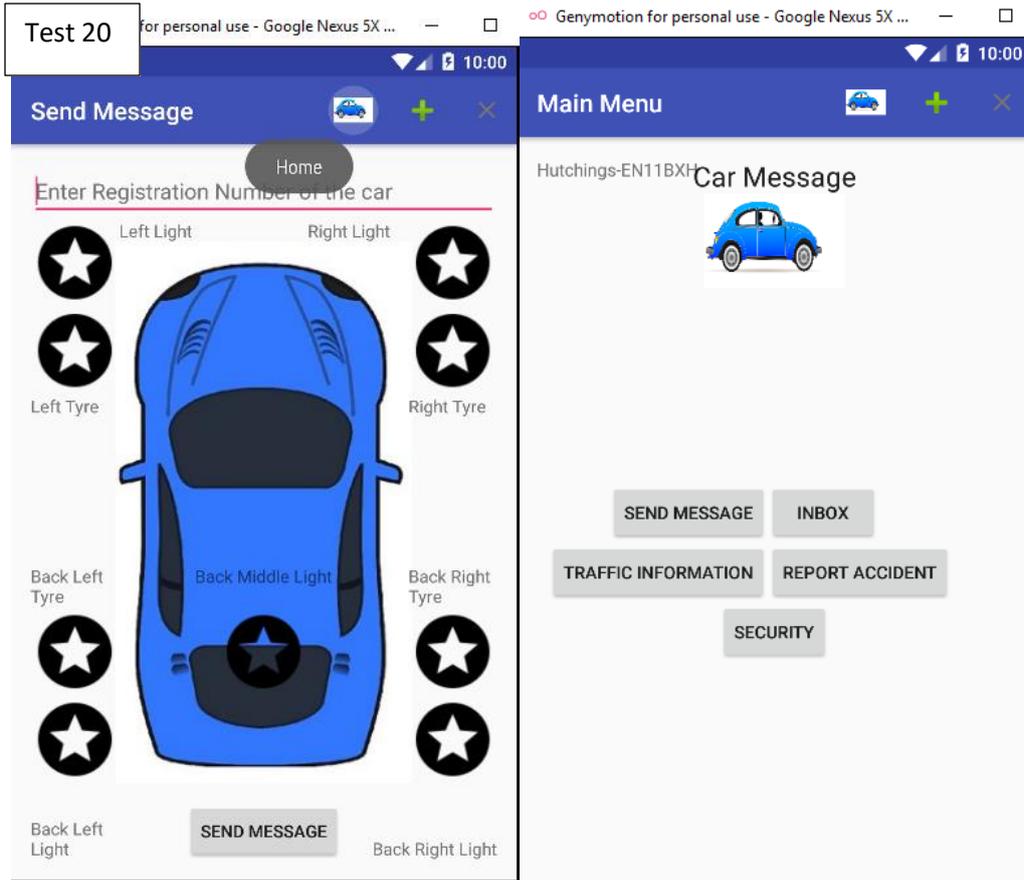




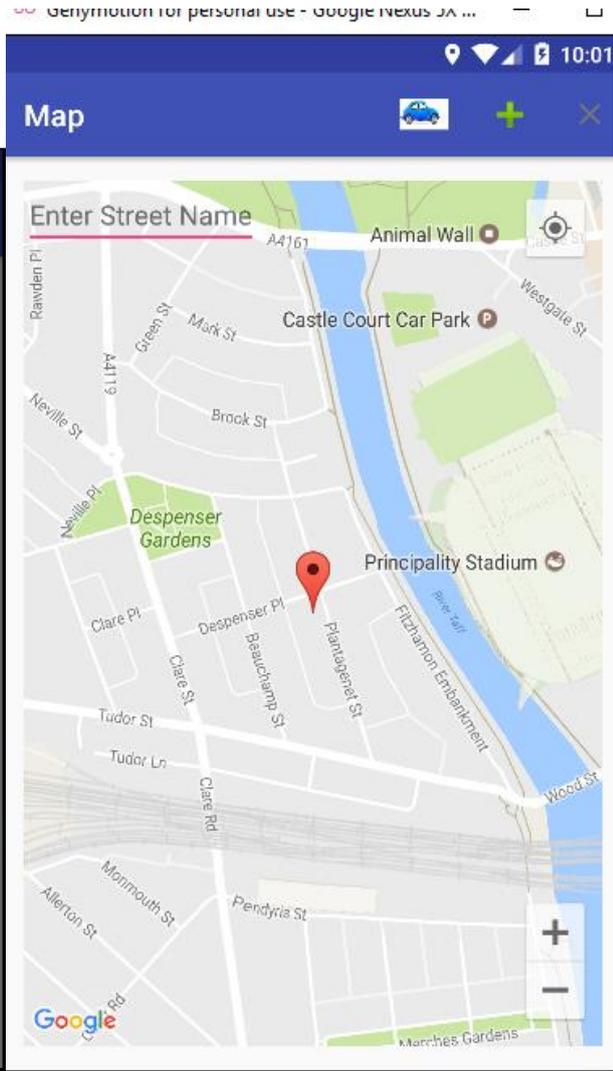
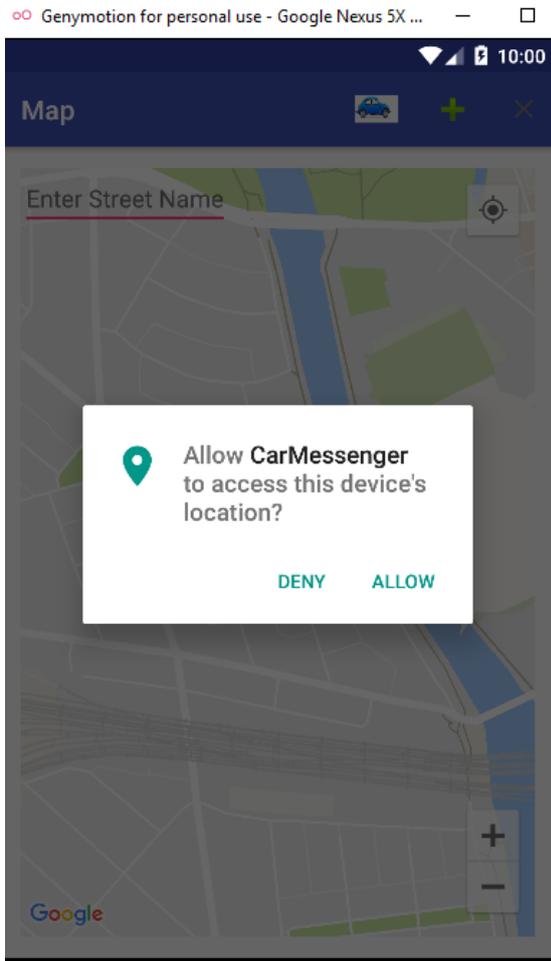




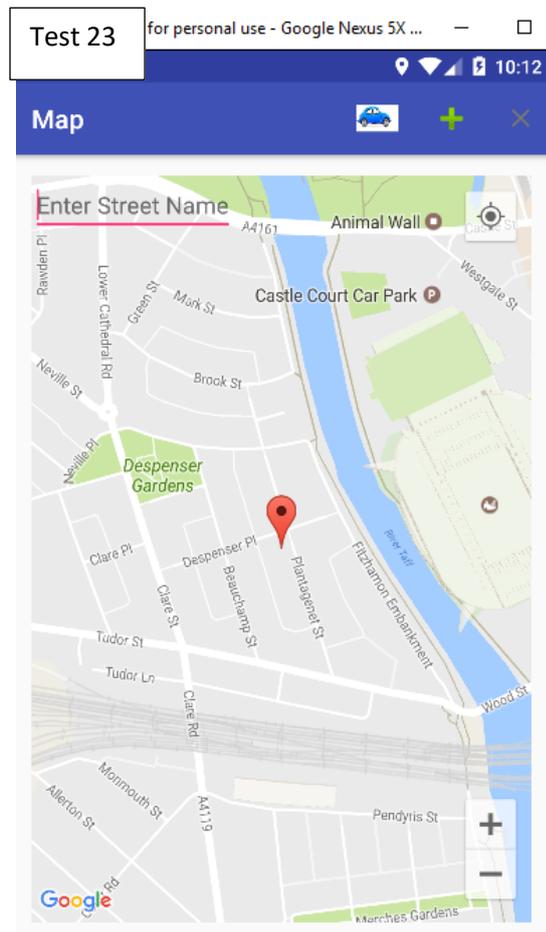
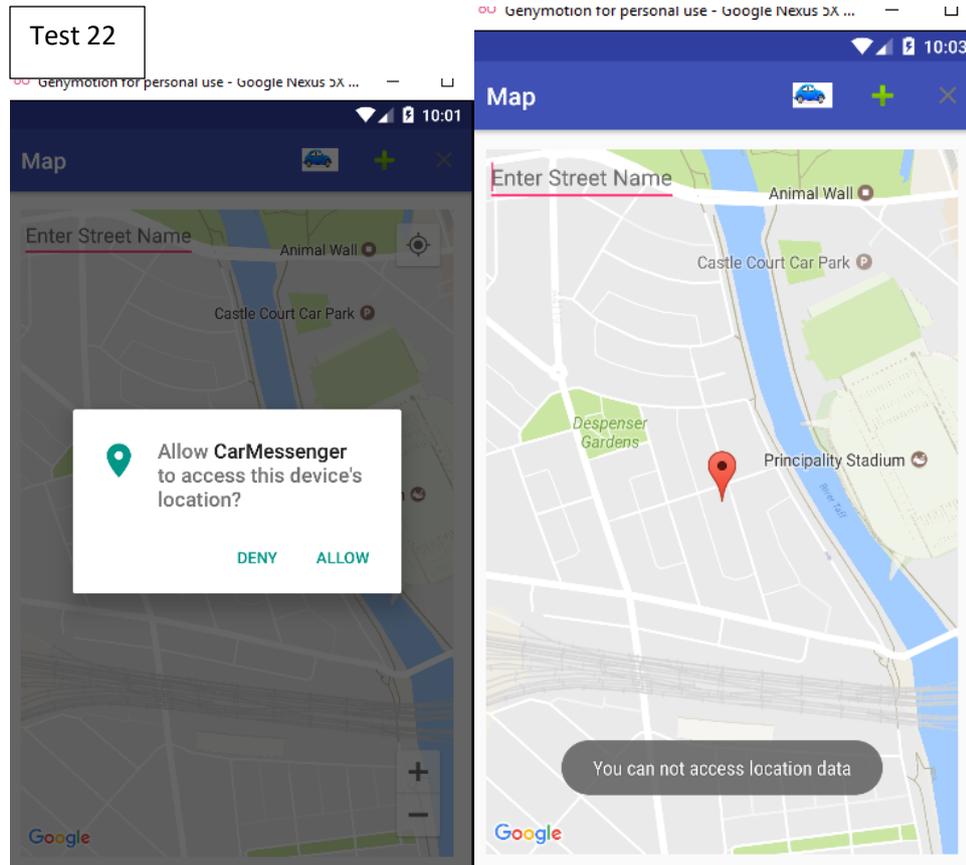
Alex Hutchings 1412529: Car Messaging System.



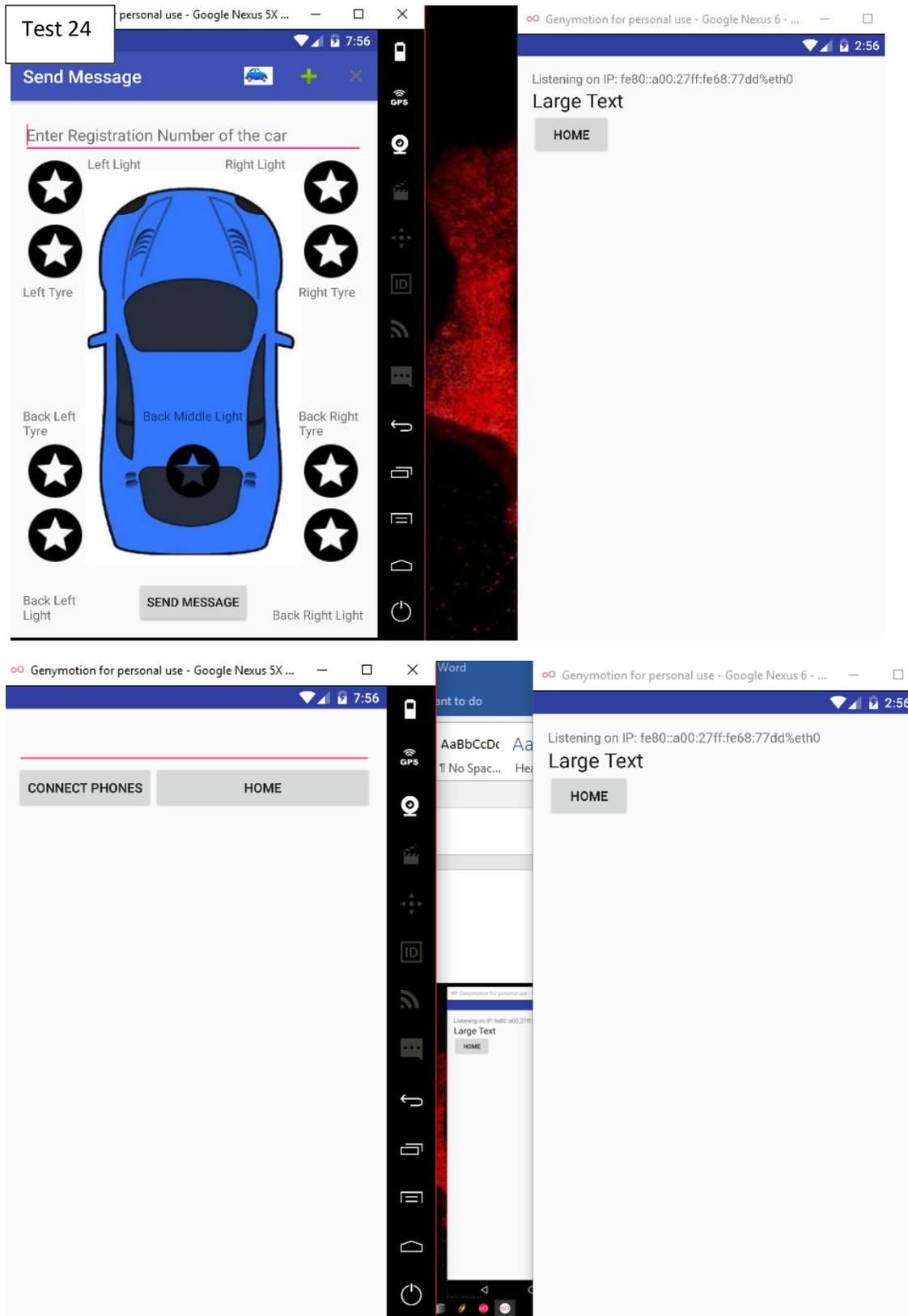
Test 21



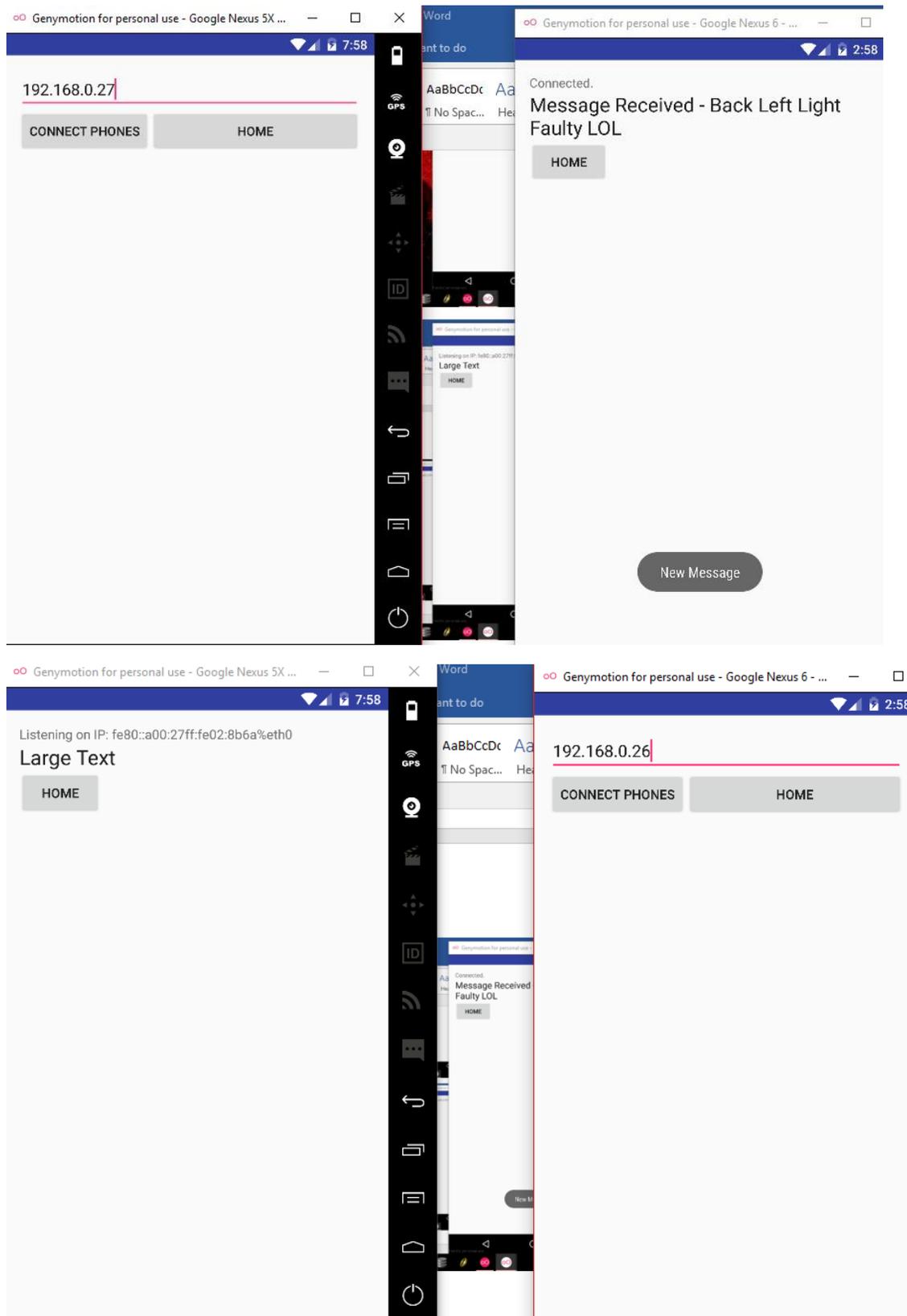
Alex Hutchings 1412529: Car Messaging System.



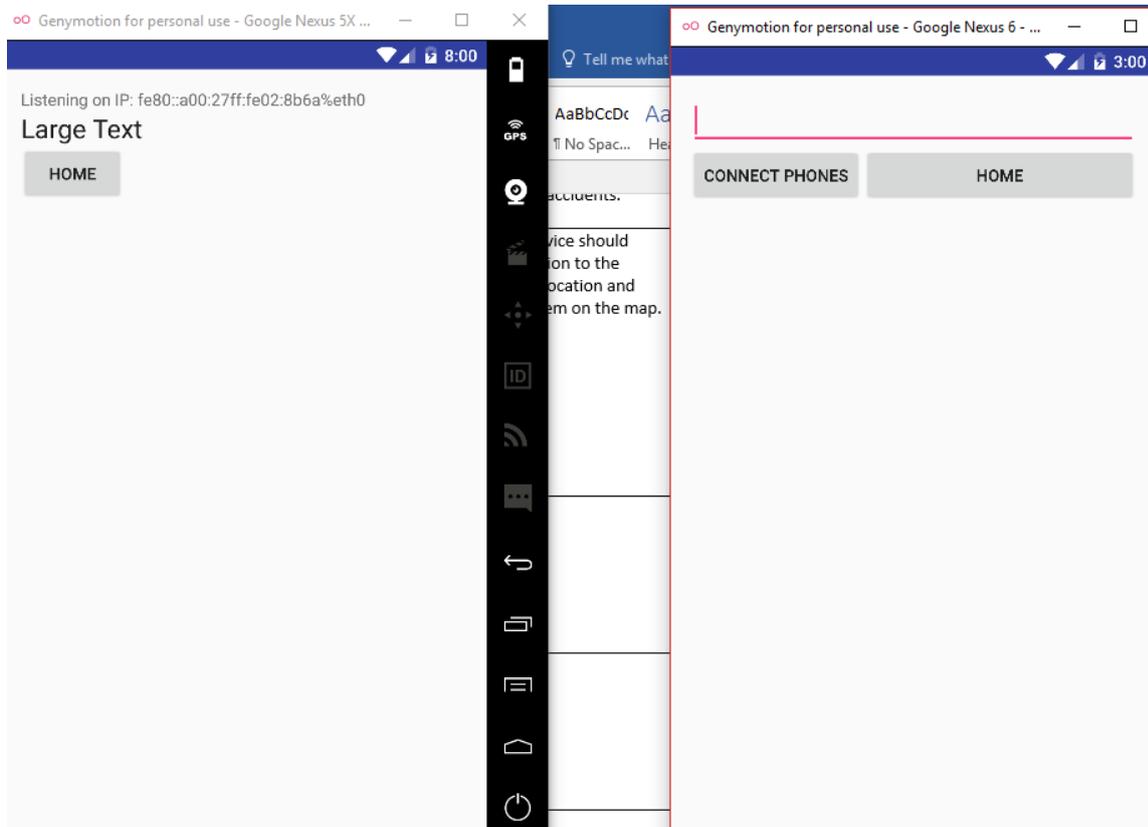
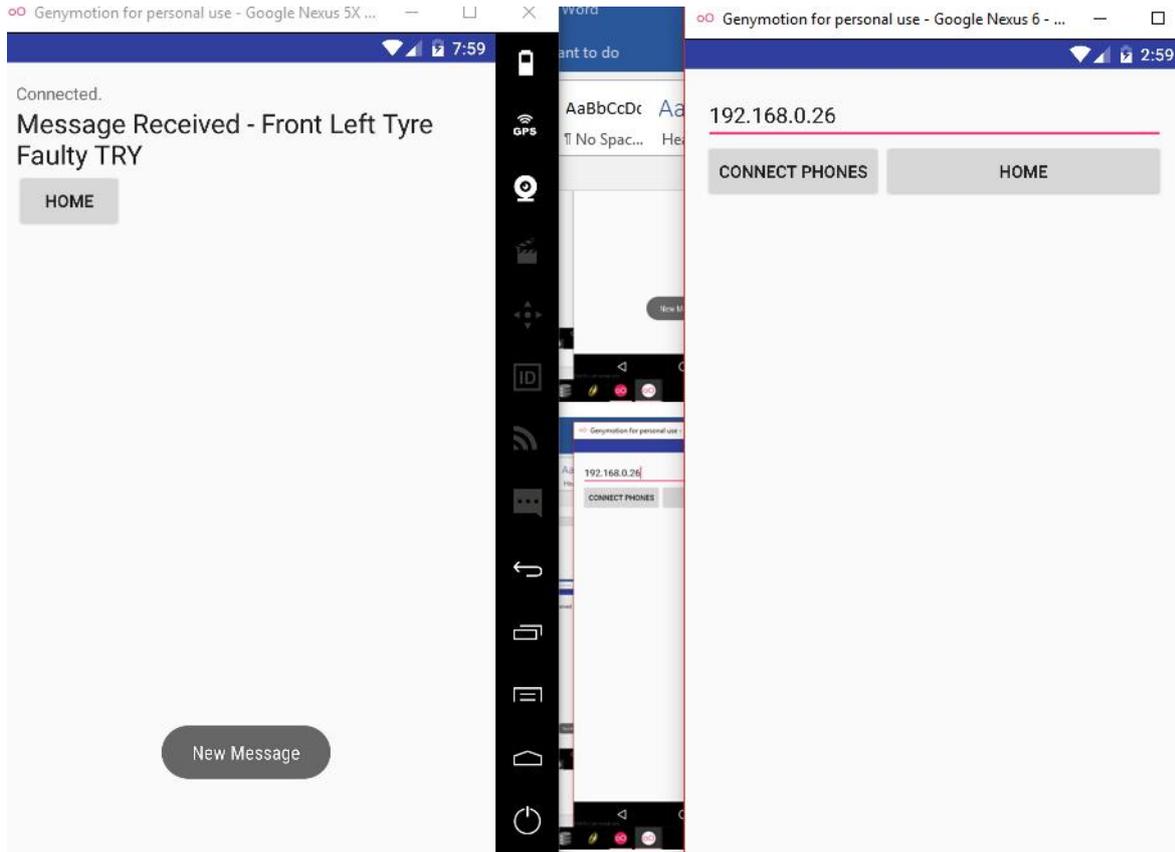
Alex Hutchings 1412529: Car Messaging System.



Alex Hutchings 1412529: Car Messaging System.



Alex Hutchings 1412529: Car Messaging System.



Alex Hutchings 1412529: Car Messaging System.

