National College of Ireland

BSc (Honours) in Computing

Software Development

2020/2021

Alex Eyre

18108008

X18108008@student.ncirl.ie

FixIt

Technical Report

Contents

Executive Summary

FixIt is an Android application designed to aid County Councils in Dublin to manage traffic light maintenance. FixIt acts as a mobile platform for an online management system where users can create inspection reports for all the traffic lights located in Dublin. These reports would be created on a monthly basis. After an inspection report has been created, on open maintenance report will be created, this report will remain open until maintenance has been performed on the reported traffic light. All inspections created by each individual user will be tracked with metrics showing the total number of reports created.

1.0 Introduction

1.1. Background

I decided to undertake this project because I have been working with Java and Android projects for the last three years. I have always had a keen interest in working with android technology but I was never content with the projects submitted over the last three years. This fourth-year final project was an opportunity for myself to use all of the knowledge I have gained not just from using android, but a total accumulation of my Java skills over the last five years. I also wanted to push myself in order to see what I could achieve and learning new technologies that could be implemented into my application to stand out from my past projects.

1.2. Aims

The aim for this project is to develop an android application that would be used by council workers or contracts working on behalf of the council. This application can aid in improving work carried out by these employees when it comes to the maintenance and repairment of traffic lights throughout Dublin. After research was conducted, it was discovered that no such application like FixIt were being used for council work. This project is a niche application designed to benefit workers in the maintenance sector. The main goals for the project, involved the creation of inspection reports for traffic lights, in which an open maintenance report would be created. These maintenance report would be closed off when the required maintenance has been performed. Admins and employees should be able to keep track of how many reports they have created through the use of metrics. These metrics should give admins, stakeholders, owners, and employees the information needed in order to improve working conditions, work ethic and how efficient the current system of maintain traffic lights maintenance performs.

1.3. Technology

The technology used in this project consists of Java for the programming language and Android studio for the IDE to develop the application. Firebase is used as the real-time database that handles all data from user and admin authentication. A JSON list that includes

all 900 traffic lights in Dublin, including latitude-longitude coordinates, location names have been imported into the database. Data relating to inspection and maintenance reports as well as data storage for images/photos have been implemented into the database. The project makes use of Google APIs that will allow a user to access each of the traffic light locations visually by using the latitude and longitude coordinates.

1.4. Structure

Section one covers the introduction of the project. The background, aims and technologies used are discussed here. In section two, the functional requirements and use cases for the project are discussed. There is a total of nine requirement use cases mentioned on this report. Next on the report follows the design & architecture of this project. The implementation of the project comes next with images containing code and algorithms. Towards the end of section two, the GUI is mentioned which includes screenshots of the application alongside text explaining the contents of the screenshots. Finally, a section for the testing implementation carried out during the life cycle of the application and an evaluation of the project as a whole. Section three begins to discuss the conclusions of the project. Section four goes over any further development that could be taken with the project given additional resources or time allocation. Section five includes the references used through-out the project and report followed by section six which contains the appendix. In section six a project plan, Gantt chart, ethics form, and monthly journal reports have been included.

2.0 System

2.1. Requirements

All requirements should be verifiable. For example, experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

2.1.1. Functional Requirements

The main function requirement that must first take place and has the highest priority is the Admin Login as opposed to regular logging in by employees. The next highest priority on the list would be the ability for admins to create and assign jobs to users/employees so that other requirements of the application can function. There will be a requirement for the user/employee to login to access their account and jobs portal but this requirement is expected in everyday technology so it does not contain a high priority as compared to other requirements for the app to function. The next requirement is a high priority for the users as it would allow the users/employees to view their assigned jobs and interact with them. Following this function requirement, the user must be able to fill in a job report of the assigned task had hand, possibly through a checklist, pictures and signatures as well as the option to submitted the report to the database.

2.1.1.1. Use Case Diagram

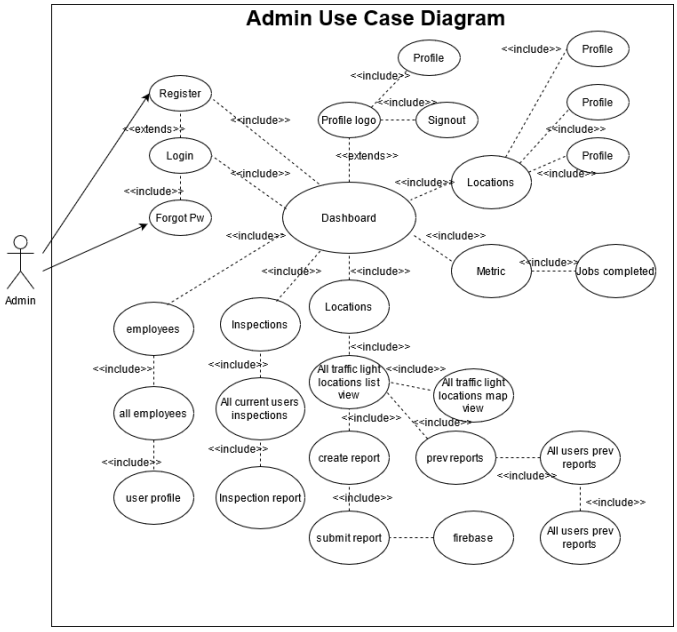

Figure 1.1 Admin Use Case Diagram

In the above figure 1.1, we see the use case diagram for the admin. This figure displays the complete functionality of the application. The figure starts with authentication, the admin must login or register and also has the option to reset their password if forgotten. Once the admin has logged in or registered successfully, they will be redirected to the dashboard. From this dashboard the admin has six features to choose from, this being user profile, employees, inspections, locations, settings or metrics. Each of these activities have further features. The admin has the privileges to view the employee's activity, however, regular users cannot see this feature.

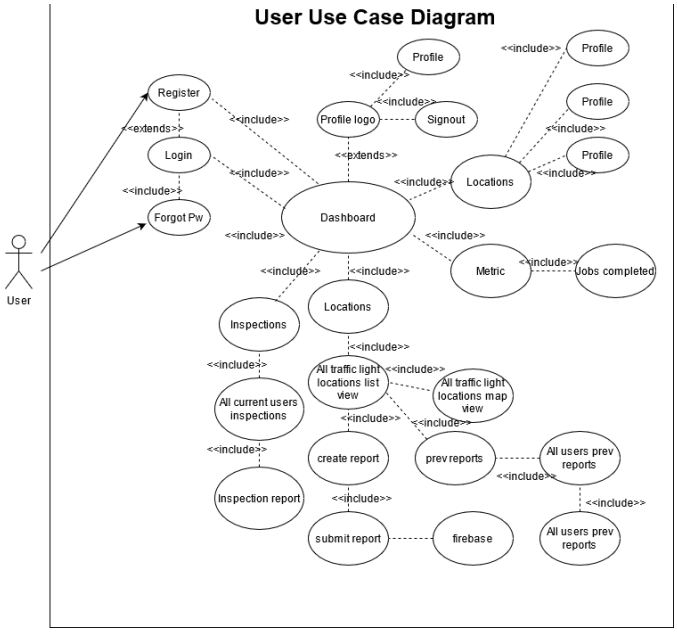

Figure 1.2 User Use Case Diagram

In the above figure 1.2, we see the use case diagram for a regular user. This use case follows the exact same pattern as seen in figure 1.1, however, the user cannot view the employee's activity as they have not been granted admin privileges.

2.1.1.2. Requirement 1 (Employee/Admin Authentication)

2.1.1.2.1. Description & Priority

This requirement is the process that the user must follow in order to proceed to the dashboard of the application. The User must login if they have an account or register if they do not have an account. Both of these requirements hold the same priority which can be seen as a high priority since its necessary to gain access to the application.

2.1.1.2.2. Use Case

Authentication 1.0

Scope

The scope of this requirement allows the user to authenticate themselves and access the system, this can be accomplished by login or registration.

Description

This use case allows the employee to gain access to the dashboard by logging into their account. The employee may register an account if they do not currently have an account on the system. The user may also use the forgot password feature if they have a registered account but cannot remember their password.

Use Case Diagram

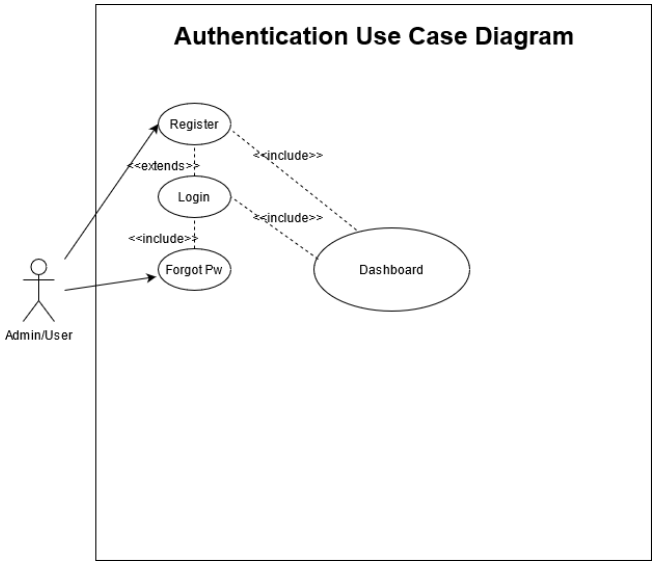

Figure 2.1

Flow Description

Precondition

The user must be given access to the APK download for the application.

Activation

This use case starts when an employee downloads the application and registers and account for the first time or is a reoccurring user looking to login again.

Main flow

1. The user downloads the APK file which gives them access to the application.
2. The employee registers an account for the first time.
3. The system registers the account to the database.
4. The user must verify their email to gain access to the system.
5. The user now has access to the application and the dashboard launches.

Alternate flow

1. Account registered already.
2. The user already has an account and logs into their account.
3. The system validates the login.
4. If the user has not validated the email, they must do so before gaining access to the system.
5.

Exceptional flow

1. User attempts to logs in.
2. The user has already registered an account and verified their email.
3. The use case continues at position five of the main flow.

Termination

This use case terminates when the user has registered or logged in successfully.

Post condition

The system processes the login or register and directs the user to the dashboard if successful, otherwise the system throws an error message stating what should be done to continue.

2.1.1.3. Requirement 2 (Creating inspection reports)

2.1.1.3.1. Description & Priority

This requirement holds the highest-ranking priority assuming the user has logged into the application. FixIt is designed around creating inspection reports that will provide the maintenance team with the information they need to know in order to perform repairs if any are requested. This is the core functionality of the application and without this requirement the application cannot function.

2.1.1.3.2. Use Case

Creating inspection report 1.0

Scope

The scope of this requirement is to allow a user to create an inspection report to be passed onto the maintenance team and give them the information needed to complete their job.

Description

This application is based around creating inspections in the form of a report, this report contains information detailing the problems of a traffic light if any are present. The report uses check boxes, text fields and image fields in order to create a detailed report to be passed onto the maintenance team. When a user has created a reported, it will be submitted to the database, from here the inspection report will be stored and can be viewed by anyone else on the application, at the same time a new report will be created for the maintenance employees and will act as an outstanding maintenance task.

Use Case Diagram

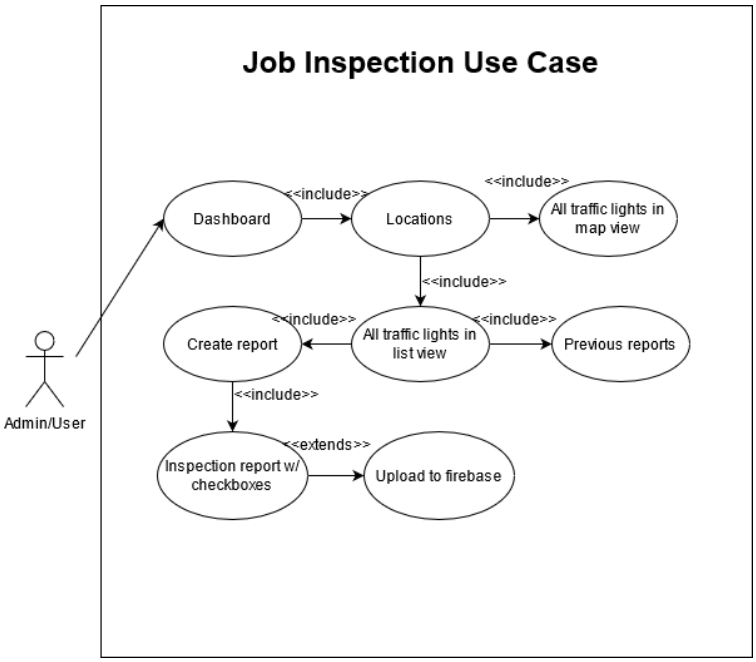

Figure 2.2

Flow Description

Precondition

The user needs to choose one of the 900 traffic light locations from the list and create a job inspection report.

Activation

This use case starts when a user has chosen one of the 900 locations from the list of traffic lights to perform a job inspection on.

Main flow

1. The user chooses a traffic light location to perform a job inspection
2. The employee fills out checkboxes that represent that current state of the traffic light being inspected
3. The user makes any extra notes that may help the next employee that performs maintenance on the traffic light
4. The user captures an image of the traffic light in order to have picture evidence of any present issues
5. The user signs off on the job inspection report with a digital signature as proof.
6. The user submits the job inspection database, which also proceeds to create an awaiting maintenance job.

Alternate flow

1. The use case continues after position three of the main flow.
2. The user does not add a signature or capture and image of the job.
3. The employee tries to submit the report to the database.
4. The employee is prompted with an error message letting them know that a signature and image must be added to the job inspection report before submitting it to the database.
5. The use case continues at position four of the main flow.

Exceptional Flow

Exceptional flow follows the same suit as the main flow.

Termination

This use case terminates when the job inspection report has been submitted.

Post Condition

When the job report has been submitted to the database, the use can view job inspections reports that only they have created or choose to view job inspections reports created by everyone on the application.

2.1.1.4. Requirement 3 (View all previously created inspections)

2.1.1.4.1. Description & Priority

Requirement three follows suit in ranking of priority just after requirement two. Requirement two must be completed for this requirement can be used. This requirement allows the user to view all previously created inspection reports.

2.1.1.4.2. Use Case

View previous inspection reports 1.0

Scope

The scope of this use case is to be able to view previously created inspection reports by all users.

Description

This use case allows a user to view all previously created inspection reports, this use case differs from the outstanding maintenance reports. This use case serves to only display previous report information only, this can be used by any employee if they wanted to check the history of a traffic light and see what maintenance might have been done in the past. This use case will contain the necessary information in order to produce metrics if used.

Use Case Diagram

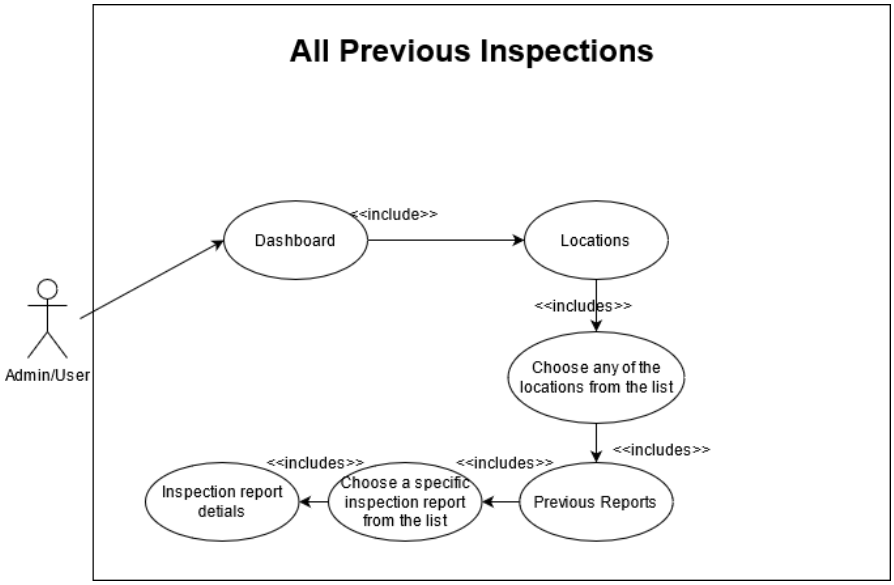

Figure 2.3 All previous inspections

Flow Description

Precondition

The user must have previously submitted an inspection report to the system in order to produce an inspection report that can be viewed by any other user inside this activity.

Activation

This use case starts when a user has pressed the submit button on an inspection report and has made sure to fill in the requirements such as the signature and captured image of the job as evidence, if the user has not checked any check boxes or filled in the notes section, they will be set to defaults which indicate there are no issues with the inspected location.

Main flow

1. The user opens the list of the 900 traffic light locations contained within the locations list activity.
2. The user chooses one from the list.
3. The user has two options, create a report or view previous inspections.
4. The user chooses to view previous inspections.
5. The user must choose from a list of previous inspection reports that have been created for that specific traffic light.

6. The user chooses one from the list and they can view all the details regarding that previous inspection report from who it was created by, when it was created and what were the details of the report etc.

Alternate flow

1. The admin wants to check a previous inspection report for a traffic light with the ID 333.
2. The admin opens the traffic light locations list and locates the location with the ID 333.
3. The admin wants to check for a specific user that has created inspection reports for this particular location.
4. The admin uses a search filter to locate all specific inspections created by a particular user.
5. The use case continues at position five of the main flow

Exception flow

This flow follows the exact same flow as taken in the main flow.

Termination

The use case terminates when the user is content after viewing the details from the previously created inspection report.

Post condition

These previously created inspections can be very useful for tracking employee progress, creating metrics and understanding how often an inspection should be created for a traffic light location, giving an employer the necessary information needed to know how many employees need to be hired for a job etc.

2.1.1.5. Requirement 4 (View all of the current users created inspections)

2.1.1.5.1. Description & Priority

Requirement four follows suit in ranking of priority as requirement four. Requirement two must be completed for this requirement can be used. This requirement allows the user to view all of their own previously created inspection reports.

2.1.1.5.2. Use Case

View all current user's inspections 1.0

Scope

The scope of this use case is for a particular user to view all of their own previously created inspection reports.

Description

This requirement acts in the exact same way as requirement three, however this details that are shown for this requirement are shown in a different section of the application. This requirement allows a user to track all of their own inspections that they have previously

created. This can act as a personal indicator on how well they are performing and how productive they are.

Use Case Diagram

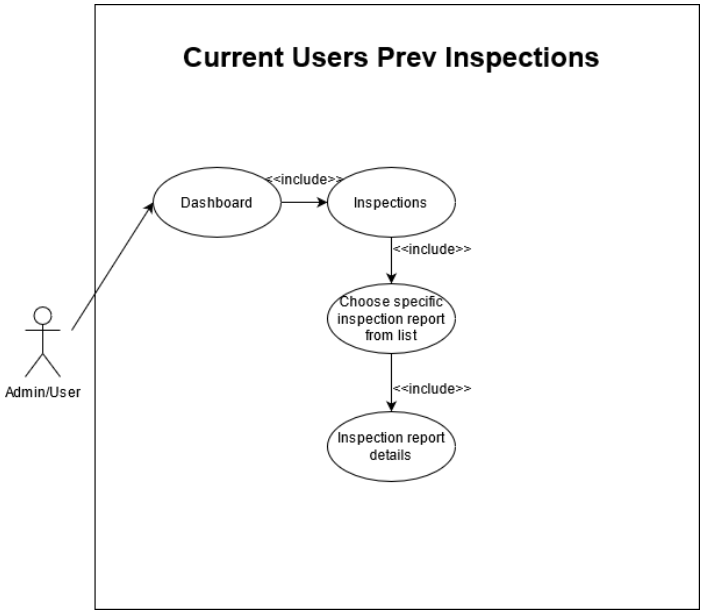

Figure 2.4 Current users' previous inspections

Flow Description

Precondition

The user must have previously submitted an inspection report to the system in order to produce an inspection report so they can view any previous inspection reports that they have created.

Activation

This use case starts when a user has chosen from the list of their previously created inspection reports inside the Inspections activity.

Main flow

1. The user wants to view the details on one of the previously created inspection reports.
2. The user opens the Inspections activity.
3. The user locates the inspection report that they want to view by location, timestamp etc.
4. The user opens the inspection report to view the details of this particular report.

Alternate flow

1. The user wants to view a particular previously created inspection to see what the details were.

2. The user opens the inspection activity.
3. The user searches through the list using a search feature to find the particular inspection by time.
4. The use case continues from position four of the main flow.

Exceptional flow

This flow follows the exact same flow as taken in the main flow.

Termination

The use case terminates when the user is content after viewing the details from the previously created inspection report.

Post condition

The user must have previously submitted an inspection report to the system in order to produce an inspection report so they can view their previously created inspection reports.

2.1.1.6. Requirement 5 (Locations)

2.1.1.6.1. Description & Priority

This requirement does not have a high priority for app functionality. This requirement serves as a very useful tool that may assist employees for finding job locations and using google maps to create directions during travel. The user may choose if they want to use this or not.

2.1.1.6.2. Use Case

Locations 1.0

Scope

The scope of this requirement allows the user to access all traffic light locations on a google map.

Description

This use case allows the user to view all 900 of the locations of traffic lights on a google map. Each one of these traffic lights are possible job locations. The user will be able to view what traffic light location they want to see, click on the marker of that traffic light and then receive directions using google maps. This will ensure the user can easily identify where and how to get to their job location.

Use Case Diagram

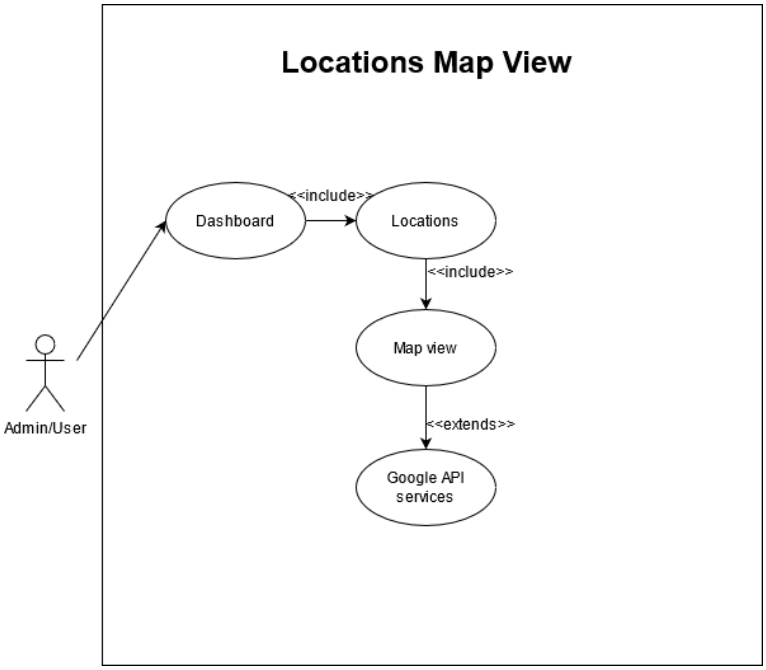

Figure 2.5 Locations map view

Flow Description

Precondition

The user must be logged into the system to view the locations.

Activation

This use case starts when a user opens the locations activity.

Main flow

1. The user opens the locations activity.
2. The user chooses a location marker.
3. The user loads the directions to that marker.

Alternate flow

1. The user wants to create a job inspection report for that specific traffic light.
2. The user must return to the locations list view, and choose the traffic light from that list in order to create a job inspection.

Exceptional flow

Exceptional flow follows the same suit as the main flow.

Termination

This use case terminates when the user leaves the locations activity and is brought to google maps through the applications API services.

Post condition

When the user wants to get directions to the job they have been assigned, they can click on the marker for that particular traffic light and load up directions on google maps.

2.1.1.7. Requirement 6 (View all employee's)

2.1.1.7.1. Description & Priority

This requirement does not hold much priority compared to the previously mentioned requirements, this requirement acts as a feature to allow other users or admins to view what employees are currently registered with the application.

2.1.1.7.2. Use Case

View all employees 1.0

Scope

The scope of this use case allows a user or admin to view all employees registered with the application.

Description

This use case will allow users or admins to search for particular users in the application. The user profiles will be able to be viewed when searching for a particular user, displaying the users profile image, username, email and phone number. This information could be useful someone wanted to contact the user.

Use Case Diagram

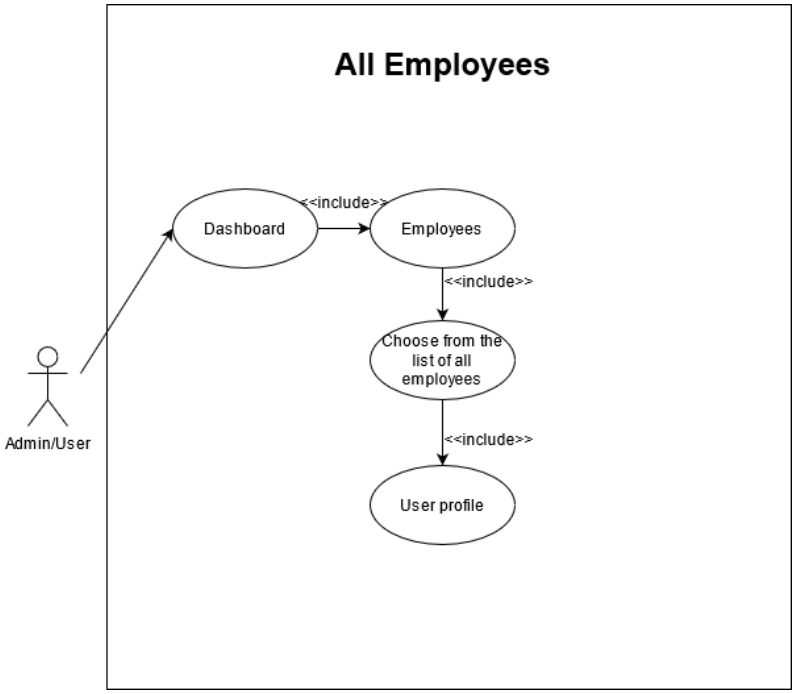

Figure 2.6 All employees

Flow Description

Precondition

There must be registered users on the system to search through the list of employees. The current user searching the list will appear but if no one else has registered, nobody else will appear.

Activation

The use case starts when an admin or use wants to view all employees on the application and view their profile containing contact/profile information.

Main flow

1. A user wants to view their colleague's profile.
2. The user navigates to the employee's activity.
3. The user searches through the list of all currently registered employees and navigates to the profile in the list they want to view.
4. The user clicks into the profile to view any information displayed.

Alternate flow

1. The admin wants to contact a particular user about a job they need done/ or to check progress on an outstanding job.
2. The admin navigates to the employee's activity.
3. The user searches through the list of all currently registered employees and navigates to the profile in the list they want to view.
4. The user clicks into the profile to view any information displayed.
5. The admin uses the employee contact information to ring the employee.

Exception flow

Exceptional flow follows the same suit as the main flow.

Termination

This use case terminates when the user/admin is happy with the information they have seen/retrieved any information about the particular user they searched for and have

Post condition

This feature can be handy for contacting employees about any outstanding jobs or for whatever other reason. Having this information at ease makes it very convenient. This can also be used in the future to implement a currently online feature.

2.1.1.8. Requirement 7 (Edit User Profile)

2.1.1.8.1. Description & Priority

Requirement seven does not have a high-ranking priority when it comes to application functionality, which is why it is towards the end of the list. However, there are a few things to keep in mind with this requirement in terms of the application. When a user creates an account, they input their name, phone number, email. When a user wants to view all

employees on the application, if the user has not edited the profile, they won't have a profile picture beside their account name. This requirement allows a user to update their information such as their password and name, but also have the option to add a profile photo will which be useful when an admin wants to look for a particular user on the application.

2.1.1.8.2. Use Case

Edit User Profile 1.0

Scope

The scope of this use case is to allow a user to edit their information on their user profile

Description

This use case allows a user to go to their user profile and edit the information on it by updating it which will proceed by updating on the database. A user will have the option to add a profile image, update their name or password. The user will not be able to change their password as this will have to be done through the admin privileges that have access to the database.

Use Case Diagram

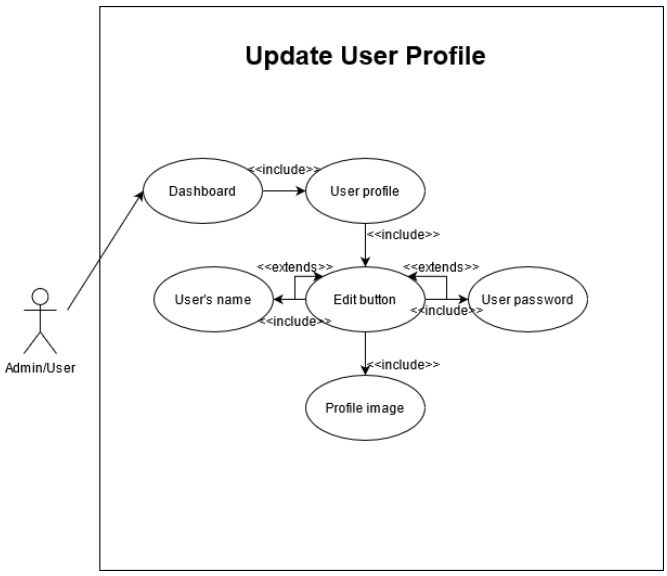

Figure 2.7 Edit user profile

Flow Description

Precondition

The user must have a registered account in order to edit their profile

Activation

This use case starts when the user presses the edit button in the form of a pen to begin the feature to updating user profile information.

Main flow

1. The user wants to add a profile image.
2. The user presses the round image icon in the application header next to their name to access their profile.
3. The user presses the pen icon to begin editing their profile.
4. The user clicks the placeholder image to add a photo from the gallery.
5. The user then clicks the pen icon again to save their changes and update the user profile.

Alternate flow

1. The user wants to update their password without clicking the icon image in the application header on the dashboard.
2. The user clicks into the employee's activity.
3. The user locates their own name and account and clicks into it.
4. The user is taken to their profile.
5. The user clicks the pen to open the editing feature.
6. The user updates their password.
7. The use case continues at position five of the main flow.

Exception flow

Exceptional flow follows the same suit as the main flow.

Termination

The use case terminates when the user has successfully updated their user profile.

Post condition

When the user has updated their profile, the application will respond differently depending on what was updated. If the user has updated their profile picture, all place holder icons in the dashboard, employee's activity and profile activity for that particular user will be updated. This can be said if the user updates their name, any name space for that particular user will be updated. When the user chooses to update their password, the user will have to enter their new password the next time the attempt to login to the application, however, they will not be logged out of the application when they have updated their password.

2.1.1.9. Requirement 8 (Settings)

2.1.1.9.1. Description & Priority

This requirement does not hold much in terms of priority for the functional requirements. It is towards the end of the list as it only contains some features that may be useful in certain scenarios. The settings feature holds multiple features and provides the user a medium to contact the application developers.

2.1.1.9.2. Use Case

Settings 1.0

Scope

The scope of this use case is to give the user setting options to help improve usability and be more user friendly through the use of language support and dark mode whilst providing information about the application.

Description

The settings feature has a few features that can be useful to the user, there are options to change the language of the application and to use dark mode, this can enhance user experience. For the user's that want to know a little bit more about the application there are settings that provide that. There are also options where the user can get in contact with the app developers via email to provide feedback out the application, user experience or to report any bugs that have been found by the user. This can be important for the developers as the users that use that application on a daily basis will be able to find any bugs quickly if any are present. The settings also show the terms and conditions and policies of the application for anyone looking for them.

Use Case Diagram

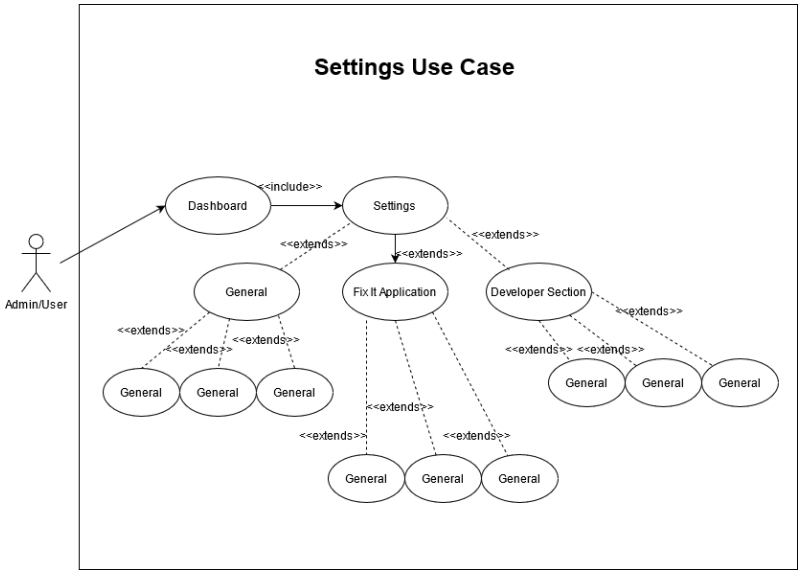

Figure 2.8 Settings

Flow Description

Precondition

The user must have an account on the application and be currently logged in to access the settings page.

Activation

The use case starts when the user wants to access the settings page and clicks the settings activity from the dashboard.

Main flow

1. The user wants to report a bug to the development team.
2. The user navigates to the settings activity from the dashboard.
3. The user navigates to the report a bug feature under the Fixit application heading.
4. The user clicks the button to report a bug and is met with a popup to choose how to contact the developers about the bug.
5. The user navigates to their email that they use to send a message to the developers.
6. The developer email automatically populates the send to option.

Alternate flow

1. The user wants to use dark mode to improve user experience on the application.
2. The user navigates to the settings activity from the dashboard.
3. The user toggles the dark mode switch found under the general sections of the settings activity.
4. The application now uses darker colours throughout the application which is easier on the eye for the user.

Exception flow

Exceptional flow follows the same suit as the main flow.

Termination

This use case terminates when the user is content with the information they have seen in the settings or have successfully used one of the features in the settings.

Post condition

The settings can be used for many features, when the user activates dark mode, all colours should follow a darker theme than default. Same goes for the language, when the user chooses which language they want to use, all text throughout the application should match the language requested by the user. The settings activity can be scaled and more features.

2.1.1.10. Requirement 9 (Sign-out)

2.1.1.10.1. Description & Priority

This requirement has a very low priority, the only purpose of this feature is to sign the user out of their account.

2.1.1.10.2. Use Case

Sign-out 1.0

Scope

The scope of this use case is to sign the user out of their account.

Description

This feature is located inside the user profile page, this feature should not be used at all or often, but it provides the user with the opportunity to sign out of their account for whatever reason. It could be useful if an employee has forgotten their phone on the day of the job

and asks to sign into their account on their colleagues' phone, that colleague would need to sign out first.

Use Case Diagram

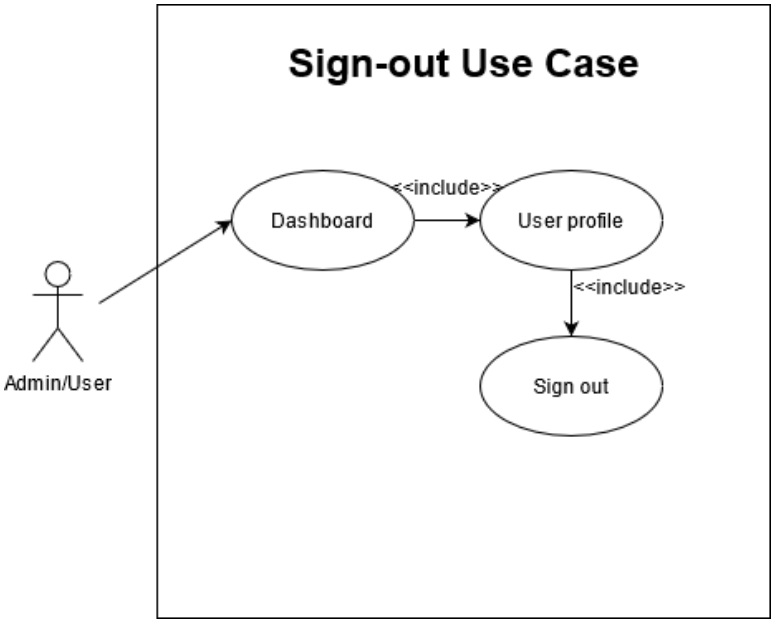

Figure 2.9 Sign-out

Flow Description

Precondition

The user must be currently logged in before the can have the option to sign out of their account.

Activation

This use case begins when the user decides to press the sign-out button located inside the user profile.

Main flow

1. The user wants to sign-out of their account as they do so after they are finished using the application every day.
2. The user navigates to their user profile.
3. The user clicks the submit button.
4. The user is prompted if they are sure they want to sign-out and they will have to login again if they proceed to do so.
5. The user presses yes and is re-directed to the login activity

Alternate flow

1. The user is asked by a colleague can they sign into their own account on their phone as he has forgotten his phone at home.
2. The user agrees and the use case continues from position 2 of the main flow.

Exception flow

Exceptional flow follows the same suit as the main flow.

Termination

This use case ends when the user has successfully signed out and is greeted by the login page again.

Post condition

When the user has signed out, they must re-login to access their dashboard again. This feature should not be used that often and may only come in handy if a colleague wants to login to their account.

2.1.2. Data Requirements

2.1.2.1. Requirement 1 (Database)

One of the main data requirements for this application is the database. The FixIt application is linked with the Firebase real-time database and most of the algorithms used through-out the application require the database to be linked with the application in order to perform tasks such as creating traffic light inspection reports, viewing previously created inspection reports, viewing all employees, and user authentication. This application will not work without the database so it is one of the most important data requirements.

2.1.2.2. Requirement 2 (JSON Traffic light data)

Another important data requirement is the use of JSON data which contains information for all 900 traffic lights in Dublin. This information contains latitude and longitude coordinates as well as traffic light location names. This data has been imported into the Firebase real-time database so that it can be used by the algorithms within the code base of the application. It is important that this data requirement is present as without the JSON data, much of the application core features cannot work.

2.1.3. User Requirements

2.1.3.1. Requirement 1 (Internet Connection)

It is important that the user has a connection to the internet when using this application, whether the user is connected to WIFI or data using 4G does not matter. The application will not be able to function properly without the use of an internet connection. The app will load and try to launch the home page of the application, however, if the user is not connected to the internet the homepage cannot load correctly and the application cannot access or retrieve data from the database.

2.1.3.2. Requirement 2 (Location Permissions)

Having location services turned on while using the application is a requirement needed for much of the functionality of the application. It is important to give location permissions with the FixIt application in order for the application to use the google map API services on the device. This will ensure that any functionality that the user can access a google map location

will work without any issues. By not enabling the location services permission the application might display blank activities and not work as intended.

2.1.3.3. Requirement 3 (Camera Permission)

It is important that the user gives the FixIt application full permissions to use the camera services. This will ensure that when a user is creating an inspection report, they will be able to access the camera to capture photos. If the user has not granted camera permissions the user will not be able to use the core functionality of the application in creating inspection reports.

2.1.3.4. Requirement 4 (Storage Permission)

As said previously, it is also important to give the FixIt application full permissions for the storage. By allowing the device and application to use the storage permissions and the camera permissions the user will be able to successfully create an inspection report. The storage permission is important as the user must add a digital signature to the inspection report in order to submit it to the database. This digital signature is saved within the device's gallery, meaning it is a requirement that the storage permission is accepted. Another feature that requires the storage permission to be accepted is the feature to update the user profile and add a profile image. This profile image is taken from the device's gallery as well. If the user fails to allow these permissions the application won't work as intended.

2.1.4. Environmental Requirements

N/A

2.1.5. Usability Requirements

When designing the application, it was important to ensure usability was implemented. During development stage many testing techniques were performed such as questionnaires, surveys and interviews with focus groups and users of different backgrounds. By providing these users and focus groups with a beta application, data and feedback could be gathered and analysed in order to improve upon usability. Some usability techniques implemented consisted of time to use, user friendliness, error tolerance and more.

Requirement 1: Time to use

The whole app has been designed with the user in mind. The UI has been kept as simple and easy to use to that the user can easily recognise all the features in the application. The app has been designed in a way that a first-time user of the application will take up to three to four minutes to complete any task and use any feature, however, for an experienced user, it will only take one to two minutes to navigate any activity in the application from login to registering to creating jobs inspection reports.

Requirement 2: User Friendliness

It was important when creating the application to determine the focus group the app would be designed for. It is designed for employees that work on behalf of the council or contractors. Working conditions within construction/ maintenance are not generally technology based and there are a lot of people over the age of 40 in the industry. Keeping this in mind, it was important to ensure that the application was simple to use and cater for

users of all age brackets. This was accomplished by using some very clean and nice colour patterns, using large buttons with large icons, and easy to read fonts. The easy-to-read fonts and large buttons were important because during testing and receiving feedback from focus group interviews, one of the reoccurring things were that the application was easy to navigate for users with larger fingers or bad eye-sight. The simple steps taken when developing the application have gone a long way in terms of usability.

Requirement 3: Error tolerance

One of the most important things to keep in mind when creating any type of software, whether it be a website, a software tool or android application is that error tolerance is vital for the longevity of an application/service. For this application it was important to ensure that if a user encountered an error, it should be made clear to the user what has happened and what they can do to overcome such an error. This feature of error tolerance is present in the application when a user attempts to submit an inspection report without adding a digital signature and image to the report. If the user tries to do so, they will be prompted by a pop-up message asking the user to add a digital signature and photo before continuing. Error tolerance can also be seen when the user has pressed the button to sign out. The user will be prompted by a pop-up asking the user if they are sure they want to sign-out. This gives the user a second chance to decide whether they want to sign-out or not, in case it was pressed by accident.

2.2. Design & Architecture

The design architecture for this application is very simple. Following the design architecture diagram, we can see how the design flows from user to the android application and then to the database.

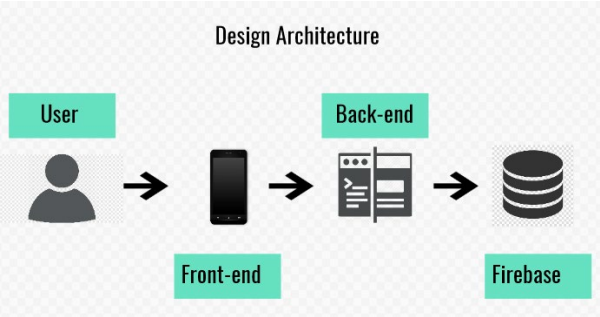

Figure 3.0 System design architecture

In *figure 3.0* we can see the flow of the system. The architecture begins with the user whether that be a regular employee or an admin. When the user launches the application for the first time, they will be greeted with the FixIt login/register pages. This consists of both front-end which implements the GUI and back-end which contains the algorithms for the user authentication that links up with the real-time database. This is one of the required algorithms in order to allow the app to function but is not necessarily the main algorithm. The algorithm for user authentication is simple yet very effective. A user must follow strict rules in order to register/sign in. This consists of using passwords over 6 characters in length, using correct email formatting, using a phone number that is 10 digits in length etc. When the user attempts to sign in, if they have successfully authenticated the user will have

access to the application. An example of user authentication algorithms can be seen in the figure below.

```
private Boolean validateEmail() {
    String email = mEmail.getEditText().getText().toString();

    if (email.isEmpty()) {
        mEmail.setError("Email is required");
        mEmail.requestFocus();
        return false;
    }
    if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
        mEmail.setError("Please enter a valid email");
        mEmail.requestFocus();
        return false;
    } else {
        mEmail.setError(null);
        mEmail.setErrorEnabled(false);
        return true;
    }
}
```

Figure 3.1 Valid email algorithm

When the user has access to the dashboard, they will be greeted by a very simple, elegant front-end design with large buttons. The main application algorithm is located within the location's activity on the dashboard, from here the user will have to choose one traffic light location from a list of 900. When the user has made their choice, they will be able to create an inspection report for the traffic light location they have just chosen. It is here in the application where one of the main applications algorithms can be found. The algorithm used here uses Firebase algorithms in order to function. These algorithms are used to reference my real-time database in order to access data to be used for recycler views, card views, text views and image views. An example of using an algorithm to access data from the real-time database to be used on a card view/recycle view in the application can be seen in the figure below.

```
//reference the database path for the coordinates
FirebaseDatabase.getInstance().getReference().child(Constants.COORDINATES).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        mInspectionTemplateList = new ArrayList<>();

        for (DataSnapshot trafficLightDataSnapshot : snapshot.getChildren()) {
            try{
                if (trafficLightDataSnapshot != null && trafficLightDataSnapshot.hasChildren()) {
                    TrafficLightModel trafficLightModel = trafficLightDataSnapshot.getValue(TrafficLightModel.class);
                    trafficLightModel.setkey(trafficLightDataSnapshot.getKey());//We can assume all data is present
                    mInspectionTemplateList.add(trafficLightModel);


                }
            }catch (Exception e){

            }

        }
        locationsListAdapter = new LocationsListAdapter( mContext: LocationsListActivity.this, mInspectionTemplateList);
        mRecyclerView.setAdapter(locationsListAdapter);
```

Figure 3.2 Traffic light data algorithm

2.3. Implementation

The finished product of this application has featured a lot of back-end and front-end implementation. The goal for this application was to be able to work on both ends of the application in a seamless way where the application flows smoothly and follows important usability factors whilst providing the user with functional back-end features. From top to bottom this application has had much thought and work done. The application has very elegant and professional looking animations that mimic the most popular applications on the market. The splash screen has a very smooth animation where text and the app logo slide from the top and bottom of the screen until they meet in the middle. This animation takes about three seconds to complete before fading and transitioning to the login/register activity or the dashboard depending on the situation. When the dashboard activity launches, the application features a dashboard of controls that redirect the user to different activities. This can be seen in the figure below.

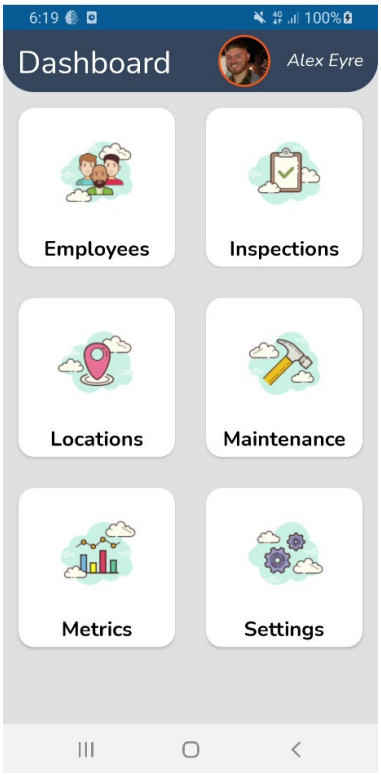

Figure 4.0 Dashboard Controls

When developing the controls, it was important to implement them using efficient code which resulted in the research of how code could be reused through-out the application to reduce development time and bloated code. The result of this research involved the use of adapters and model classes. A template was created for one of the buttons seen in *figure 4.1,* using this template, multiple controls were created. The algorithm used for these controls can be seen in the figure below.

```
private void setupControls() {
    controls.clear(); //Clear previous controls

    //Admin ONLY
    if (currentUser.getadmin() != null && currentUser.getadmin()) {
        controls.add(new HomePageWidgetModel(R.drawable.icon_employees,  title: "Employees", EmployeeListActivity.class));
    }

    //Added for everyone
    controls.add(new HomePageWidgetModel(R.drawable.icon_inspections,  title: "Inspections", InspectionsActivity.class));
    controls.add(new HomePageWidgetModel(R.drawable.icon_chart,  title: "Metrics", MetricsActivity.class));
    controls.add(new HomePageWidgetModel(R.drawable.icon_locations,  title: "Locations", LocationsListActivity.class));
    controls.add(new HomePageWidgetModel(R.drawable.icon_settings,  title: "Settings", SettingsActivity.class));

    setAdapter(controls);
}
```

Figure 4.1 Controls for the dashboard

Th algorithm in *figure 4.1* shows the controls being added to an adapter, for each of the buttons an instance is called from the model class to add the icons, text and path to which the button links to. This method helps reduce many lines of code, whilst also implementing admin/user privileges as seen in *figure 4.1* where the admin has access to the employee's activity but regular users do not.

For the activities such as Inspections, Locations and Employees, similar algorithms were used on the main dashboard activity. Adapters were created for each of these activities and recycler views for each of the adapters. By doing so templates were created for each of these activities which can be seen in the figure below.

Figure 4.2 All inspections with template

In *figure 4.2* a list of items can be seen. The information for each of these list items are displayed on a card view template. These templates are created using adapters and recycler views to display information from the database on to that template. These card views are used through-out the application from Employees, Inspections and the Locations activity, however, they all have slightly different colour patterns to differentiate from one another.

Each one of the card views use separate adapters and recycler views in order to display different information. These algorithms are identical other than what information has been displayed which can be seen in the figure below.

```
@Override
public void onBindViewHolder(@NonNull ReportListViewHolder reportListViewHolder, int position) {

    //set the template lists with data using ID and Location from TrafficLightModel
    reportListViewHolder.id_tv.setText(String.format(Locale.ENGLISH,  format: "ID: %s", trafficLightModels.get(position).getkey()).trim());
    reportListViewHolder.location_tv.setText(String.format(Locale.ENGLISH,  format: "Location: %s", trafficLightModels.get(position).getname()).trim());

    //set the template lists with data when the inspection was reported from TrafficLightInspectionModel
    //if there has been no inspections previously reported, print out N/A

    if (trafficLightModels.get(position) == null) {
        reportListViewHolder.reported_on_tv.setText(String.format(Locale.ENGLISH,  format: "Reported on: N/A"));
    } else {
        reportListViewHolder.reported_on_tv.setText(String.format(Locale.ENGLISH,  format: "Reported on: %s", trafficLightModels.get(position).gettimestamp()).trim());
    }

    reportListViewHolder.created_by.setText(String.format(Locale.ENGLISH,  format: "Created by: %s", trafficLightModels.get(position).getinspection_by()).trim());
```

Figure 4.3 Adapter algorithm for all inspections

Figure 4.3 presents the algorithm used to display the information onto the card view shown in *figure 4.2.* This algorithm uses data from the real-time database with the help of model classes in order to access and retrieve the relevant data. An example of one of these model classes that contains the database data for the above card view can be seen in the figure below.

```
public class InspectionReceiptModel implements Serializable {

    private String timestamp;
    private String path;
    private String created_by;
    private String id;
    private String location;

    public InspectionReceiptModel() {
    }

    public String gettimestamp() { return timestamp; }

    public void settimestamp(String timestamp) { this.timestamp = timestamp; }

    public String getpath() { return path; }

    public void setpath(String path) { this.path = path; }

    public String getcreated_by() { return created_by; }

    public void setcreated_by(String created_by) { this.created_by = created_by; }

    public String getid() { return id; }

    public void setid(String id) { this.id = id; }

    public String getlocation() { return location; }

    public void setlocation(String location) { this.location = location; }
}
```

Figure 4.4 Model class for inspections card view

Figure 4.4 above contains the getters and setters used in order to access certain data from the database, however, this application contains the use of many model classes, each one of

them extend to other model classes as a way of inheriting data from multiple model classes. This was implemented in order to combine to be used together for the likes of a single card view template.

One of the main algorithms found in this application, involves the ability to create an inspection report and submit it to the database. The implementation of this algorithm allows a user to create a report that contains checkboxes, text fields, and image fields where the user must add a captured photo from the camera and a digital signature. This report then be submitted to the database to appear in the All-inspections activity that can be found in the above *figure 4.2*. This implementation was the hardest and contains the most extensive algorithms. The algorithms included here involve the ability to access the camera, take a picture and convert the image to a string in order to be submitted to the database and later used to display on previous created reports. Part of this algorithm can be seen in the figure below.

```
private void uploadImage(String name, Uri contentUri) {
    StorageReference imageRef = storageReference.child("inspection_images/").child(key).child(myCurrentDateTime).child(name);
    imageRef.putFile(contentUri).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
            imageRef.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
                @Override
                public void onSuccess(Uri uri) {
                    Picasso.get().load(uri).into(btnImage);
                    downloadURL = uri.toString();
                }
            });
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText( context: ReportActivity.this,  text: "Failed to upload image", Toast.LENGTH_SHORT).show();
        }
    });
```

Figure 4.5 Camera URI image to string algorithm

When a camera photo is taken is needs to be converted from a URI which acts as the image address to a string to be uploaded to the database. The above algorithm shows the URI being converted to a string in order to be set to an image view an submitted to the database.

Just like the camera photo that needs converting for this report, when dealing with digital signatures, a conversion from Bitmap to string was needed in order to set to an image view and submit to the database. When a user adds a digital signature, the result produces a Bitmap. The following image shows the algorithm uses in order to convert that Bitmap to a string.

```
@Override
public void applyBitmap(Bitmap bitmap) {
    bitmapclone = bitmap;
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    bitmapclone.compress(Bitmap.CompressFormat.PNG,  quality: 90, baos);
    byte[] data = baos.toByteArray();
    StorageReference bitmapRef = storageReference.child("signature_images/").child(key).child(myCurrentDateTime);
    UploadTask uploadTask = bitmapRef.putBytes(data);
    uploadTask.addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
            taskSnapshot.getStorage().getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
                @Override
                public void onSuccess(Uri uri) {
                    contentSignatureUri = uri;
                    signatureURL = uri.toString();
                    Picasso.get().load(signatureURL).placeholder(R.drawable.progress_animation).into(((ImageView) findViewById(R.id.add_signature_btn)));
                }
            });
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText( context: ReportActivity.this,  text: "Failed to get signature url", Toast.LENGTH_SHORT).show();
        }
    });
```

Figure 4.6 Signature Bitmap to string algorithm

After the implementation of the algorithm for capturing a camera photo and digital signature, an algorithm was needed to capture user input from the checkboxes to be submitted to the database. This involved, opening an inspection report and choosing to check each check box in relation to the task at hand. Checking the box or not would result in a yes or no answer. At first when this feature was implemented, the code consisted of regular if/else statements to handle the check boxes. This method of implementing the checkboxes resulted in a lot of repetitive and bulk code, in order to combat this, research was performed in order to find smarter ways to implement the algorithm. When researching ways to reduce code, turnary operators were discovered which greatly cut down the amount of code originally used. This algorithm can be seen in *figure 4.7*.

```
//turnary operator for checklists
trafficLightReportModel.setphysical_issues(cb1.isChecked() ? "Yes" : "No");
trafficLightReportModel.setelectrical_issues(cb2.isChecked() ? "Yes" : "No");
trafficLightReportModel.setlight_issues(cb3.isChecked() ? "Yes" : "No");
trafficLightReportModel.setbutton_issues(cb4.isChecked() ? "Yes" : "No");
trafficLightReportModel.setsound_issues(cb5.isChecked() ? "Yes" : "No");
trafficLightReportModel.setsequence_issues(cb6.isChecked() ? "Yes" : "No");
trafficLightReportModel.setrepairs_needed(cb7.isChecked() ? "Yes" : "No");
trafficLightReportModel.setcreated_by(UserSingletonModel.getInstance().getuser_name());

//for writing notes to the database
trafficLightReportModel.setnotes(notes.getText().toString());//Returns "" if nothing in the input field
```

Figure 4.7 Algorithm for check boxes and notes text field

Figure 4.7 shows the implementation of the turnary operator. This reduced the code significantly as it replaced the need for if or else statements. This algorithm was fired after pressing the submit button on the report. The algorithm would look to see if the user had

checked the check box or not and set the setters in the model in which was then used to be submitted to the database.

A useful feature used when dealing with algorithms was the use of design patterns. Design patterns represent the best practices used for object-oriented solutions. After doing some research it was discovered that a singleton could be used in this project in order to reduce code within the algorithms and to develop more efficient code. As seen in the figure below, the “UserSingletonModel” has been created in order to get access to the variable for the current user signed. This variable that represents the current user signed in can be treated as a global variable which can be access in any class by calling upon the “UserSingletonModel”. This singletons model can be seen in the figure below.

```
public class UserSingletonModel {
    private static UserSingletonModel instance = null;

    // variable of type String
    public String user_name;
    public String user_uid;

    public UserSingletonModel() {
    }

    public static UserSingletonModel getInstance() {
        if (instance == null)
            instance = new UserSingletonModel();

        return instance;
    }

    public static UserSingletonModel getinstance() {
        return instance;
    }

    public static void setinstance(UserSingletonModel instance) {
        UserSingletonModel.instance = instance;
    }

    public String getuser_name() { return user_name; }

    public void setuser_name(String user_name) { this.user_name = user_name; }

    public String getuser_uid() { return user_uid; }

    public void setuser_uid(String user_uid) { this.user_uid = user_uid; }
}
```

Figure 4.8 User singleton model

This model shows how the instance of the singleton was declared. The singleton is a design pattern that allows me to create a global variable as such to be accessed from any class. In order to use this singleton, I had to implement it in a section of my application where the current users full name is used throughout the application. On the main activity dashboard, the current users full name is displayed at the top right of the activity next to the user’s profile picture. Using this information, the singleton model was able to be used to set the current users name to the model as seen in the figure below.

```
//Get current User Data
FirebaseDatabase.getInstance().getReference().child("users").child(FirebaseAuth.getInstance().getCurrentUser().getUid()).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        if (snapshot.getValue() != null && snapshot.hasChildren()) {
            currentUser = snapshot.getValue(UserProfileModel.class);
            if (currentUser != null) {

                //Setup User Singleton
                UserSingletonModel.getInstance().setuser_name(currentUser.getname());
                UserSingletonModel.getInstance().setuser_uid(currentUser.getuid());

                setupControls();
                setupUI();
            }
```

Figure 4.9 Singleton model set to current user

After this singleton had been set, I was able to include it as part of the algorithm for setting the traffic light report model in order to submit the report to the database as seen in *figure 4.7.*

For this application I have imported a JSON list into my real-time database which contains all 900 traffic light locations latitude, longitude and location names. This data is required by the application in order for any other feature to work as intended. The structure of this data inside the real-time database can be seen from the figure below.

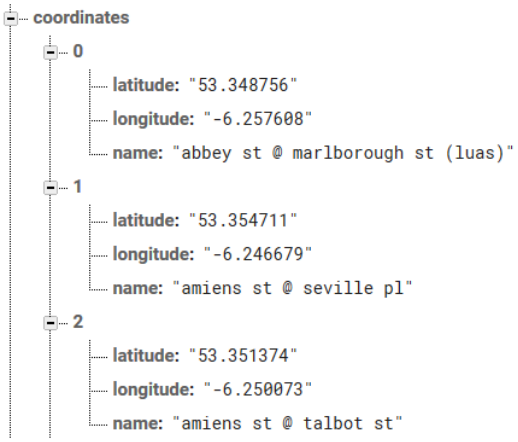

Figure 4.10 Firebase traffic light data structure

The data from the above photo has made it possible to implement all of the features and algorithms that I have previously discussed. This data is used in an algorithm in two ways in the application. The first implementation of this data can be seen on the location's activity. A list of 900 traffic lights will be shown using the same card view template as previously discuss in *figure 4.2.* The algorithm used to put this data into a list view format can be seen in the figure below.

```
//reference the database path for the coordinates
FirebaseDatabase.getInstance().getReference().child(Constants.COORDINATES).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        mInspectionTemplateList = new ArrayList<>();

        for (DataSnapshot trafficLightDataSnapshot : snapshot.getChildren()) {
            try{
                if (trafficLightDataSnapshot != null && trafficLightDataSnapshot.hasChildren()) {
                    TrafficLightModel trafficLightModel = trafficLightDataSnapshot.getValue(TrafficLightModel.class);
                    trafficLightModel.setkey(trafficLightDataSnapshot.getKey());//We can assume all data is present
                    mInspectionTemplateList.add(trafficLightModel);


                }
            }catch (Exception e){

            }

        }
        locationsListAdapter = new LocationsListAdapter( mContext: LocationsListActivity.this, mInspectionTemplateList);
        mRecyclerView.setAdapter(locationsListAdapter);
```

Figure 4.11 Algorithm for 900 traffic light locations list

Figure 4.11 shows the algorithm for accessing all of the locations found in the database, each location has been added to a temporary array list which will be used to add the data to an adapter. The adapter is used to create a template for 900 of the locations which can be seen in the locations list view.

Another use of the data found in *figure 4.11,* instead of creating a list view of all of the 900 hundred locations, a google map marker has been implemented for each of the traffic light coordinates using the latitude and longitude. This algorithm can be found in the figure below.

```
private void createMapMarkers() {

    try {
        if (mMap != null) {
            mMap.clear(); // Clear any old markers
            for (TrafficLightModel trafficLightModel : allTrafficLightsList) {
                mMap.addMarker(new MarkerOptions().position(new LatLng(Double.parseDouble(trafficLightModel.getlatitude()),
                        Double.parseDouble(trafficLightModel.getlongitude()))).title(trafficLightModel.getname()).icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_CYAN)));
            }


        }
    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

Figure 4.12 Creating map markers using location coordinates

Figure 4.12 shows the algorithm used to create map makers for each location given the coordinate and location name data has been provided. This algorithm then populates a google map with all of the 900 traffic light locations and their markers. This can be seen below in *figure 5.11.*

2.4. Graphical User Interface (GUI)

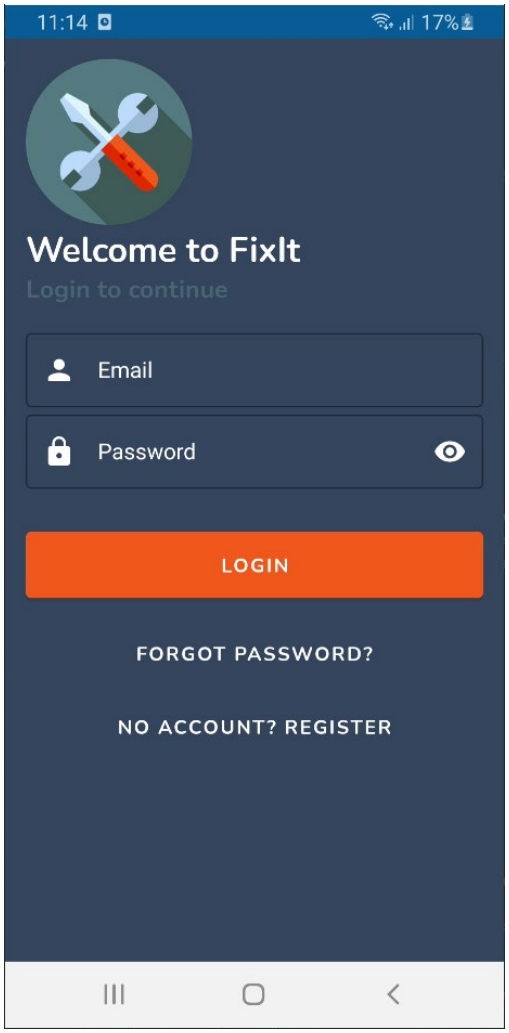

Figure 5.0 Login GUI

Figure 5.0 shows the GUI for the login page. It has a simple and elegant design, with only two user inputs for email and password. Both of these user inputs follow strict input validation. The user has the option to hide their password or show it with the eye symbol to the right of the password text field. This activity is the first activity the user will see after the splash screen has finished its animation, however, if the user has logged in previously and has not signed out, they will bypass the login GUI every time the launch the application. This login GUI also provides options to the user to register an account or to reset a password if they cannot remember it.

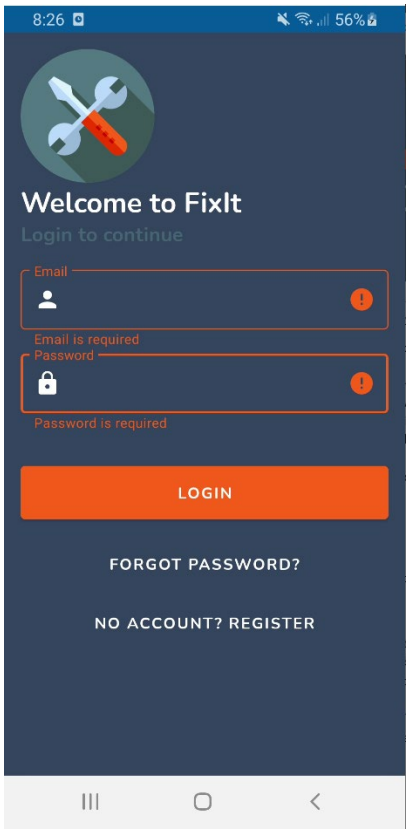

Figure 5.1 Input validation for Login GUI

Figure 5.1 shows the login gui of the user has input the incorrect email or password. This gui is to show the error tolerance that has been implemented into the application.

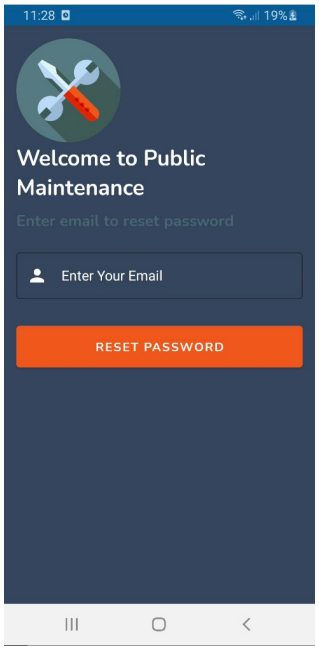

Figure 5.2 Forgot password GUI

When a user clicks the forgot password button in *figure 5.1*, they are brought to the activity found in *figure 5.3*. Here the user has to enter their email. After clicking the reset the

password button, the user will receive an email from Firebase with a link on how to reset their password.

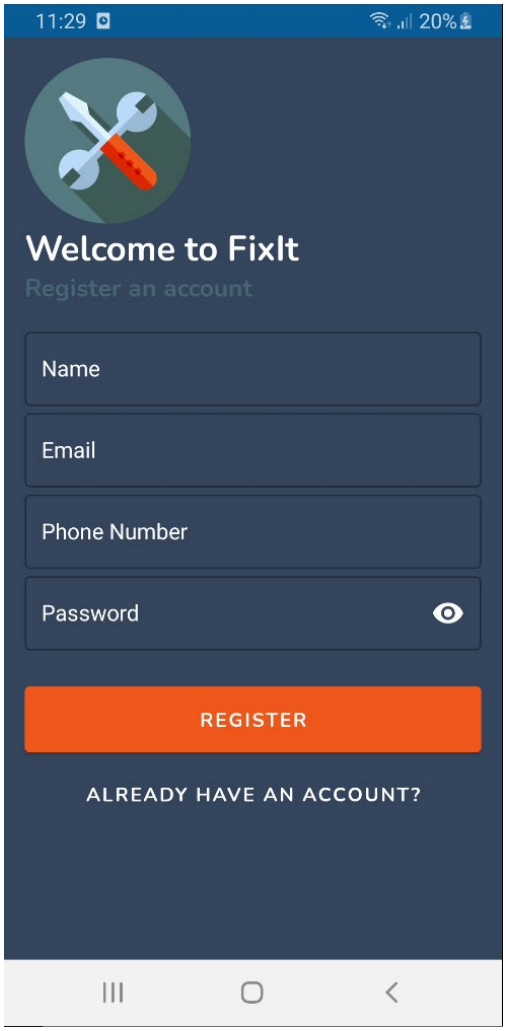

Figure 5.3 Register account GUI

When the user clicks the No Account? Sign-up button in *figure 5.1,* they are taken to this register page. This GUI is for when a user does not already have an account and would like to register. Here the user submits their name, email, phone number and password. These input fields are handled with input validation to ensure correct formatting for emails and correct password length etc. This information will be stored on the database when they register and will be used to login. The user has the option to return to the login page from the register screen, in case they already have an account.

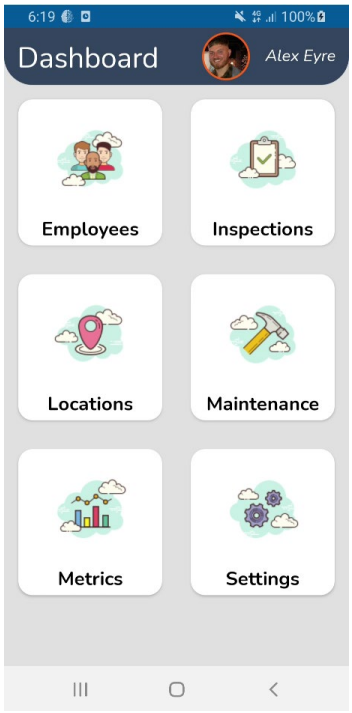

Figure 5.4 Dashboard GUI

Figure 5.4 shows the GUI for the main activity of the application. Here the user will find all everything they need in order for the app to function. In the top right, the name for the currently logged in user will be displayed as well as the user's login picture, if they have not updated their profile picture previously this profile picture will display a placeholder image instead. This GUI shows the admins perspective of the application as regular users cannot see the Employees activity. Employee's activity is to show all the current registered users on the application and their profiles. Inspections is used to show the all the inspections reports previously created by the currently signed in user. Locations, is where a user can find all traffic light locations in list view or map view, as well as the option to create an inspection report and view all inspection reports for a particular traffic light location from all users. Metrics shows metrics on traffic light information and settings has options to contact the developers etc.

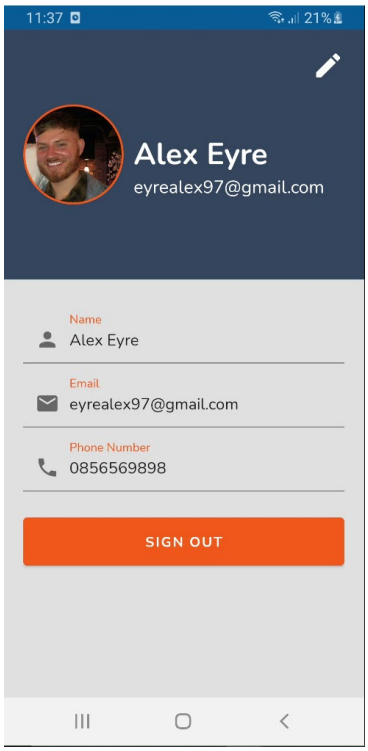

Figure 5.5 User profile GUI

When the user clicks the profile icon at the top right corner of the application in *figure 5.4,* they will be taken to their user profile, here a user can edit their profile picture, name, phone number and also sign out of the application.

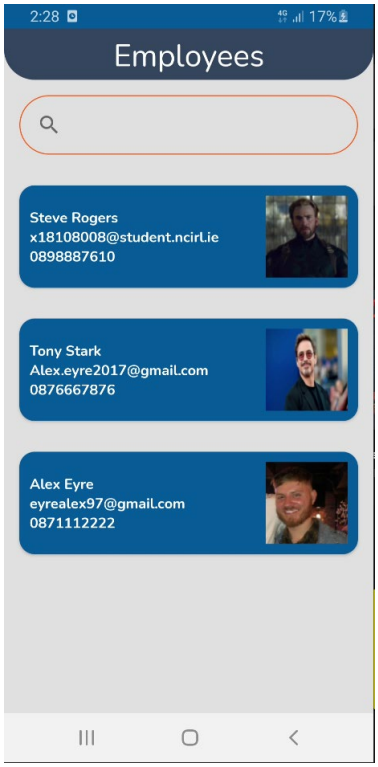

Figure 5.6 Employee Activity GUI

Figure 5.6 shows all employee's registered on the application when the user has clicked the Employee's button in *figure 5.4.* Each user in the list is clickable which will take you to their user profile as seen in *figure 5.5,* however, only the currently signed in user can edit their own account.

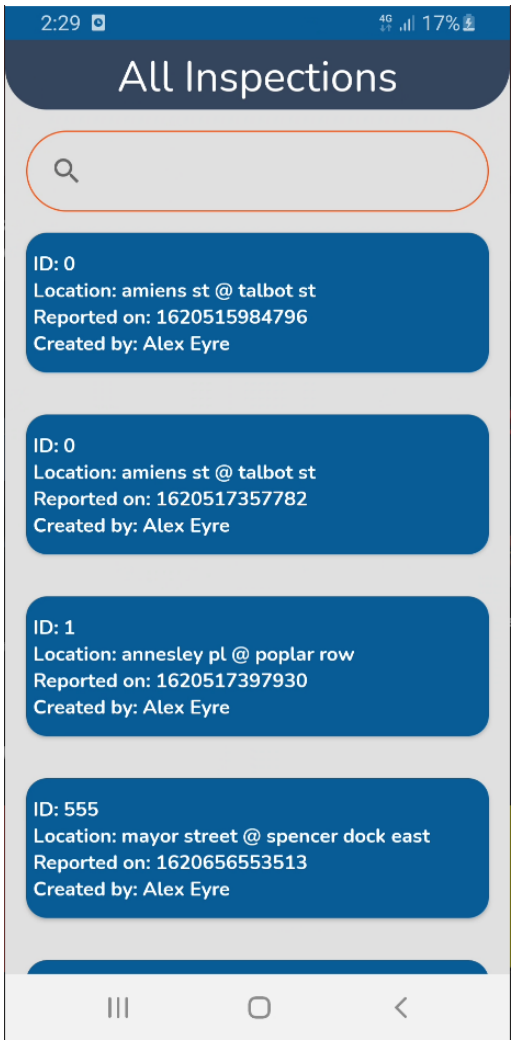

Figure 5.7 All inspections GUI

Figure 5.9 shows all inspections created by the currently signed in user. This activity can be found when the user clicks the Inspections activity found in *figure 5.4.* Each inspection contains a detail that were reported when the inspection report was first created. This GUI gives the user information such as the ID of the traffic light, the location, the date it was reported (Currently in time in Millis, needs to be converted to a user-friendly readable date) and the user that created the report.

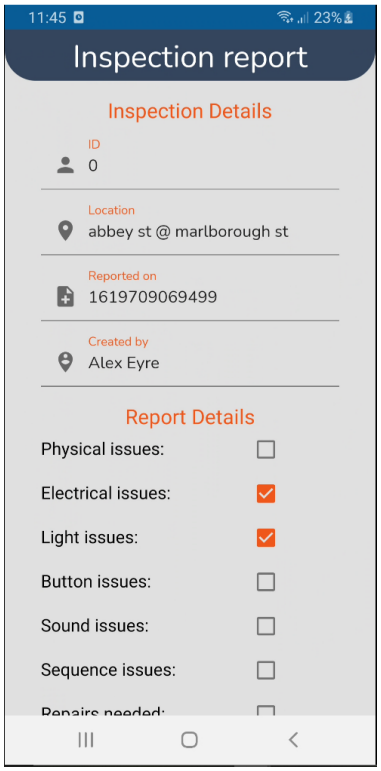

Figure 5.8 Inspection report details GUI

When one of the items from the list in *figure 5.7* has been clicked, it will bring the user to the details report that was previously reported as shown in *figure 5.8*. The same information from the previous activity has been passed over to this activity in order to ensure the user knows exactly what they are looking at. The user can also see each check box that was checked or not when previously reported, indicating what was wrong with the traffic light during that particular inspection. More details to be shown in the figure below.

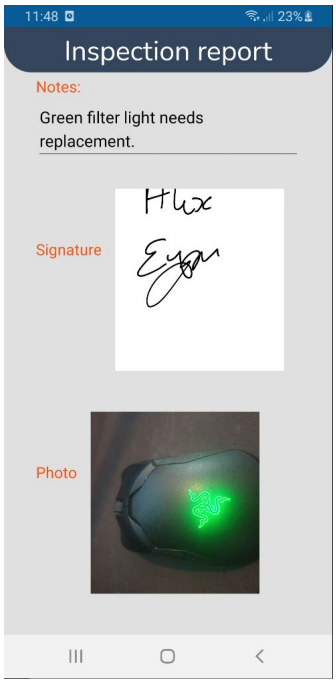

Figure 5.9 Inspection report details GUI (continuation)

Figure 5.9 shows the continued detail report found in *figure 5.8.* After the user has scrolled past the checkbox details, they can see the notes left by the user when this report had been created, as well as that user's signature and the photo they captured of the job on the day.

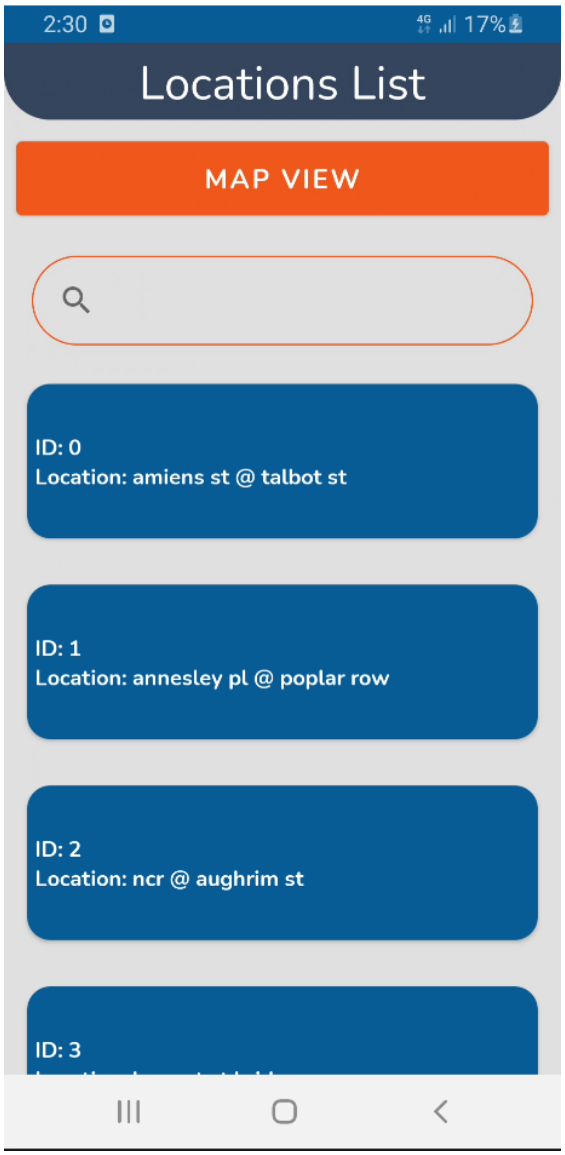

Figure 5.10 Locations list GUI

Figure 5.10 shows the activity for Locations when the Locations button is clicked in *figure 5.4.* This activity displays the list of all traffic light locations in list view. Each item in the list shows the ID of the traffic light and the locations name. This GUI also has a button to change the view from list view to map view for all traffic light locations.

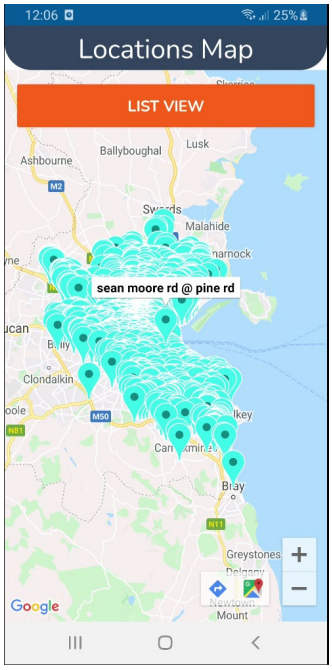

Figure 5.11 Map view GUI

Figure 5.11 shows all 900 of the traffic light locations but in a map view. Each one of these markers are clickable which shows the location name inside a text box. There is an option to get directions for each of these locations which may be useful for an employee that does not know the directions to a particular location. The directions can be accessed by clicking either of the two icon buttons at the bottom right-hand corner of the GUI, to the left of the plus and minus symbols.

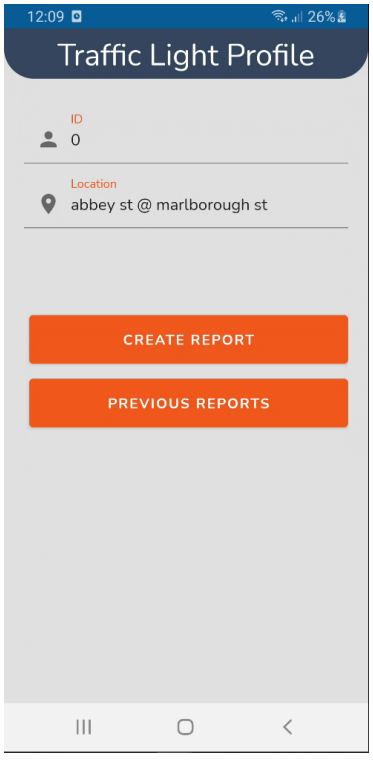

Figure 5.12 Traffic light profile GUI

The image above shows the traffic light profile activity. This page can be accessed by clicking one of the locations from the list view found in *Image 37.* The information is passed from the item list to this activity to ensure the user knows where they are in the application. From here the user has two options, they can create an inspection report for this particular traffic light or view all previously created inspection reports for this particular traffic light.

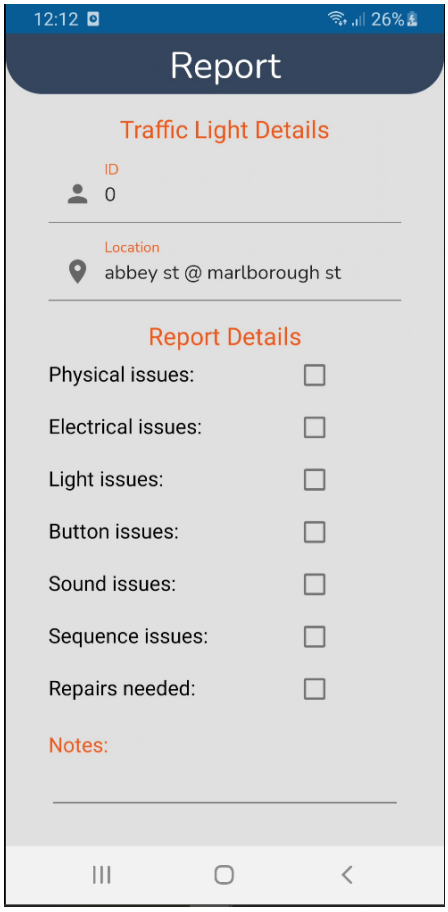

Figure 5.13 Create an inspection report GUI

Figure 5.13 shows the activity that displays when a user clicks the create report button found in *figure 5.12.* This activity is similar to the one found in *figure 5.8,* however, this activity is the creation of an inspection report so all of the checkboxes are set to default "no", the text field is empty and the report is awaiting a signature and photo taken from the camera.

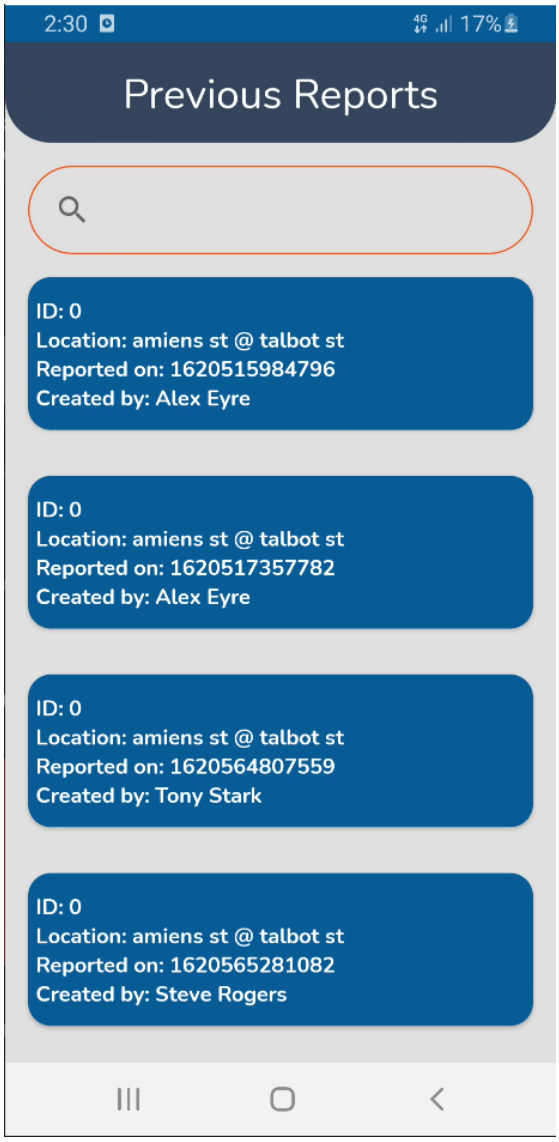

Figure 5.14 Previous inspection reports by all user's GUI

Figure 5.14 shows the activity for all created inspection reports by all users. This activity is displayed when the user clicked the previous reports button found in *figure 5.12.* This list similar to the list found in *Image 34,* however, with this activity, the list shows all inspection reports created by all users for this particular traffic light. The only information that differs is the created by field. Each item brings the user to the details report as seen previously.

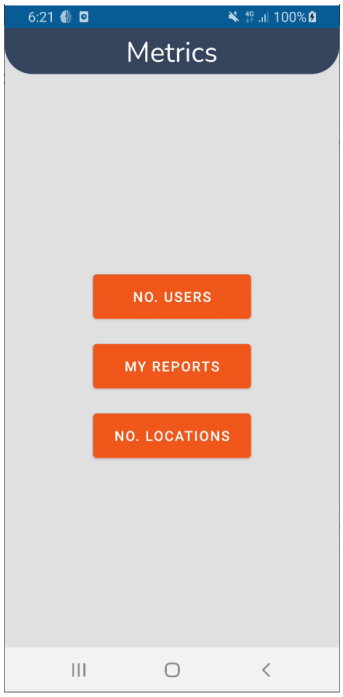

Figure 5.15 Metrics activity

Figure 5.15 shows the metrics activity. This activity has buttons that link to three types of metrics. These metrics involve the number of users registered to the system. The number of reports created by the currently signed in user and metrics that show the number off traffic lights in Dublin.

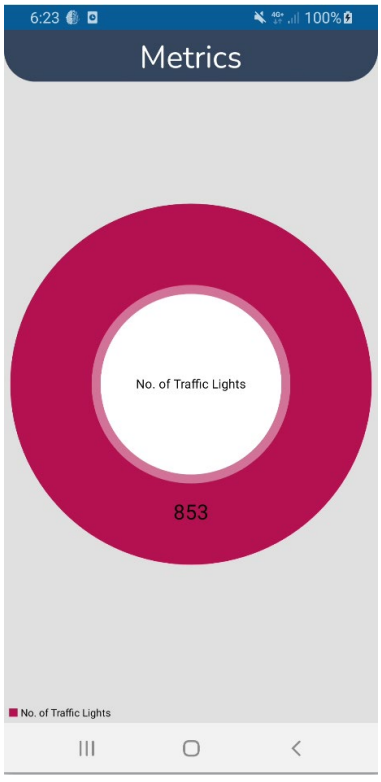

Figure 5.16 No. of Locations metric

Figure 5.16 shows one example of the metrics. The metrics shown in this figure display the total amount of traffic lights in Dublin. These metrics are dynamic and update in real time. The metrics are the same for all other metric locations except for the data that's shown.

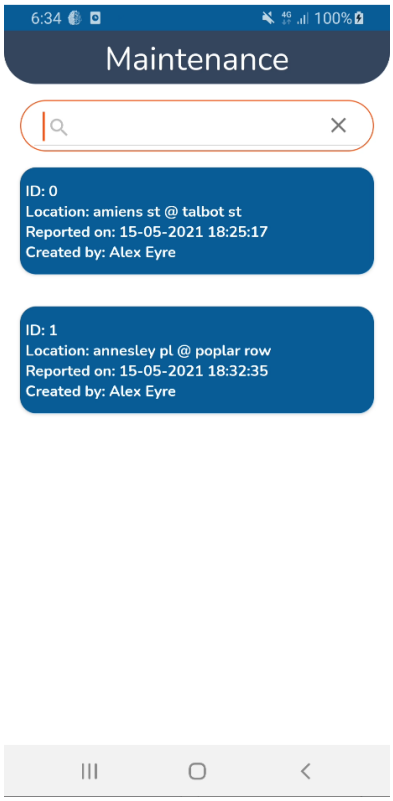

Figure 5.17 Maintenance

Figure 5.17 shows the open maintenance reports that are ready to be completed. These open reports are created as soon as a report is submitted in *figure 5.13.* These open maintenance reports can be seen by all users. This system acts like a ticketing system, where any available maintenance staff can come in, choose a ticket and complete the job. The user can also search through all open maintenance reports.

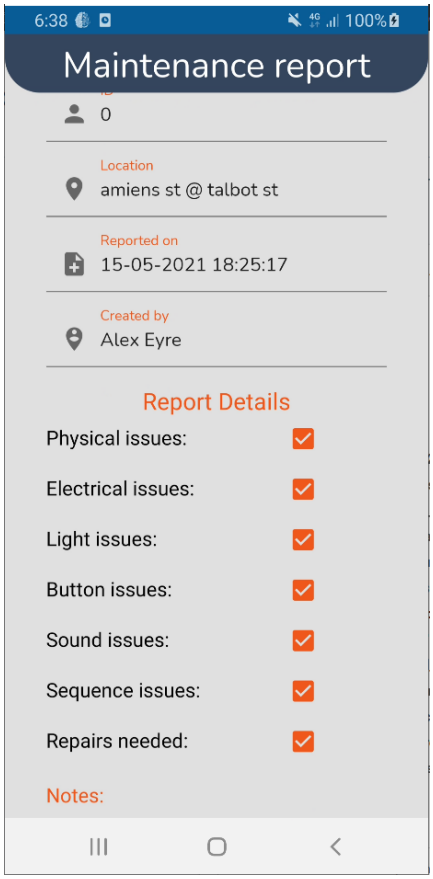

Figure 5.18 Maintenance report details

Figure 5.18 shows the details of the maintenance report shown in *figure 5.17.* These details show the maintenance staff what exactly needs to be checked and what needs to be repaired. The maintenance staff will be able to see who opened this report and when, what the location is and the id of the traffic light. The users report details, the user who open the reports signature and a photo of evidence for the traffic light issues.

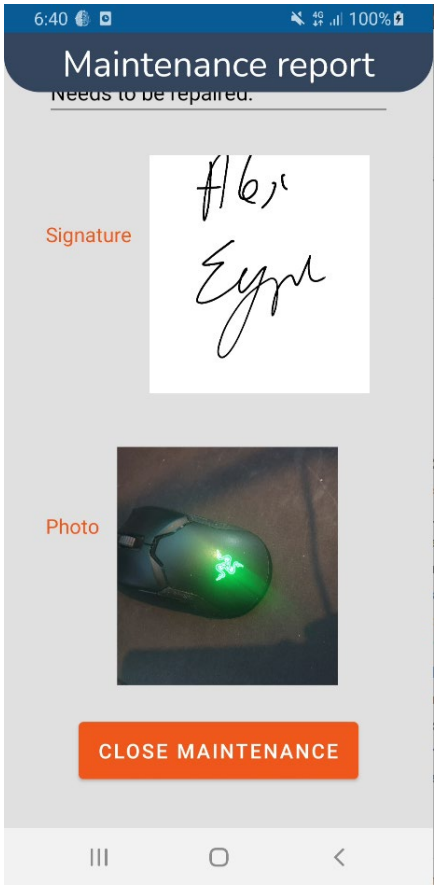

Figure 5.19 Maintenance report details (Continuation)

Figure 5.19 is a continuation from *figure 5.18.* This figure shows a button that allows the maintenance staff to close the maintenance. When this button is pressed, it signals that the current job has been completed and the open maintenance report seen in *figure 5.17* will be removed from the list. This button has error handling in a form of a dialogue ensuring the user has fully completed the job, this is to ensure the maintenance staff does not delete the report by accident.

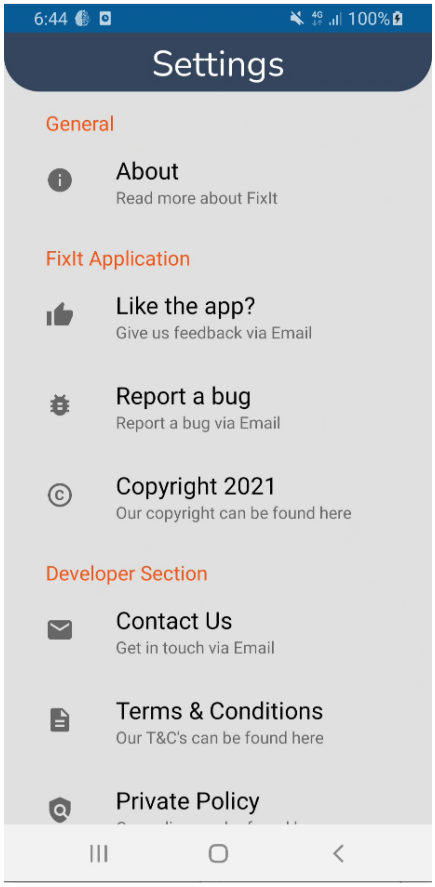

Figure 5.20 Settings GUI

Figure 5.20 shows the settings activity, when the settings button is clicked on the dashboard found on *figure 5.3.* This activity has options to use dark mode, change the language of the application or contact the developers about any bugs, feedback etc.

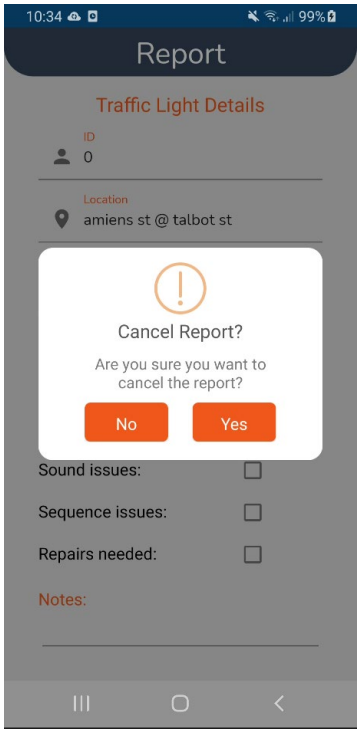

Figure 5.21 Error handling

Figure 5.21 shows error handling that will appear to the user for certain tasks. In this figure the user can be seen pressing the back button on the create a report activity. This dialogue box makes it clear to the user that they will be terminating the report. This dialogue appears to the user when they attempt to delete anything from the system, sign-out, terminate reports or create reports.

2.5. Testing

When performing testing for this project, two forms of testing for conducted. This testing involved unit testing for the testing of Model classes using jUnit tests and Instrumentation & UI testing which involved the use of expresso tests.

The unit testing carried out was performed on five model classes. The unit test involved navigating to one of my model classes and created a jUnit test for that particular model class. In this test model class, an object of the original model class being tested was needed in order to test the getters and setters. An example of a test model class performing jUnit tests can be seen in the figure below.

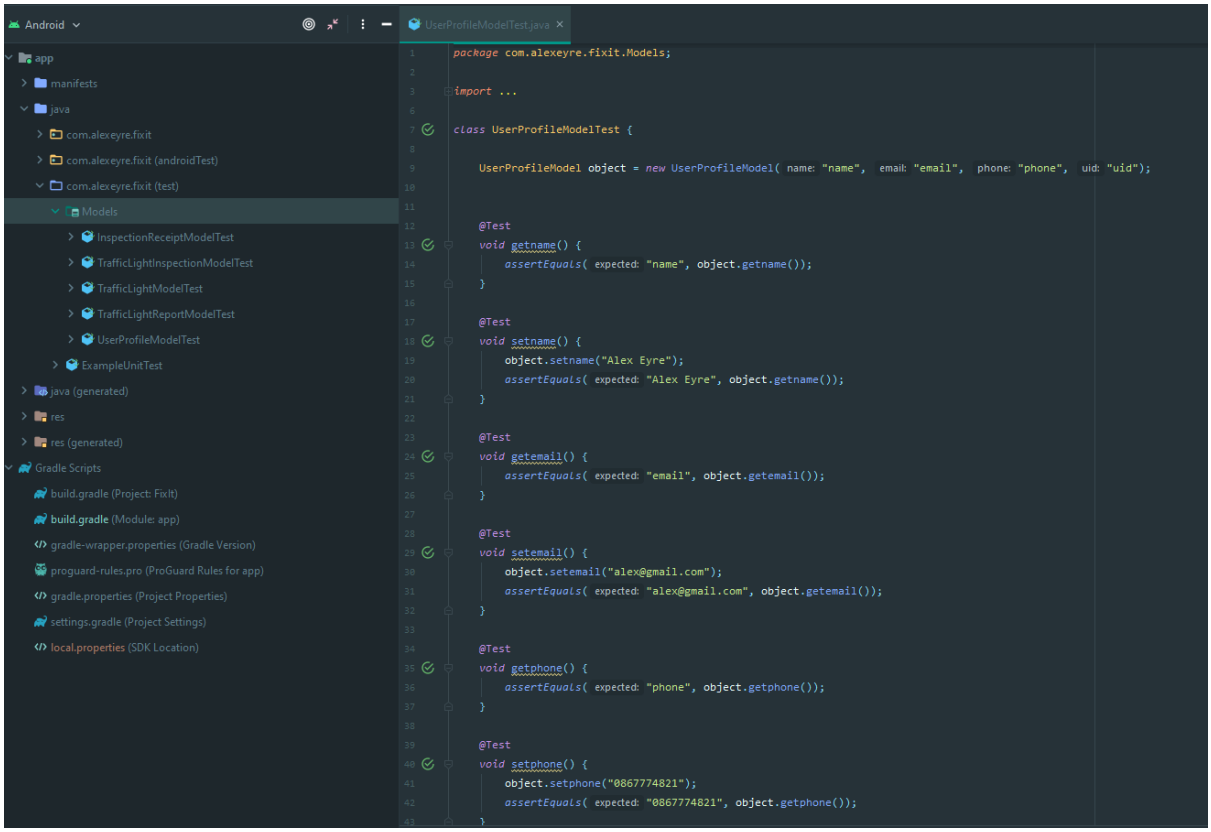

Figure 6.0 User Profile Model Test

Figure 6.0 shows the getter and setters being testing for the user profile model. An object has been created called 'object' which is used to access the variables within the model class. Tests are then performed on all of the getter and setters using the object. The test for getters uses the assert equals method to access the variable passed into the object. For testing the setters, the object is used to set a new variable, the assert equals method is then called to test the new variable against the object. The green tick inside the green circle to the left of the tests displays that each of the tests were successful. This testing has been replicated across all other model classes. Another example of one of these unit tests can be seen in the figure below.

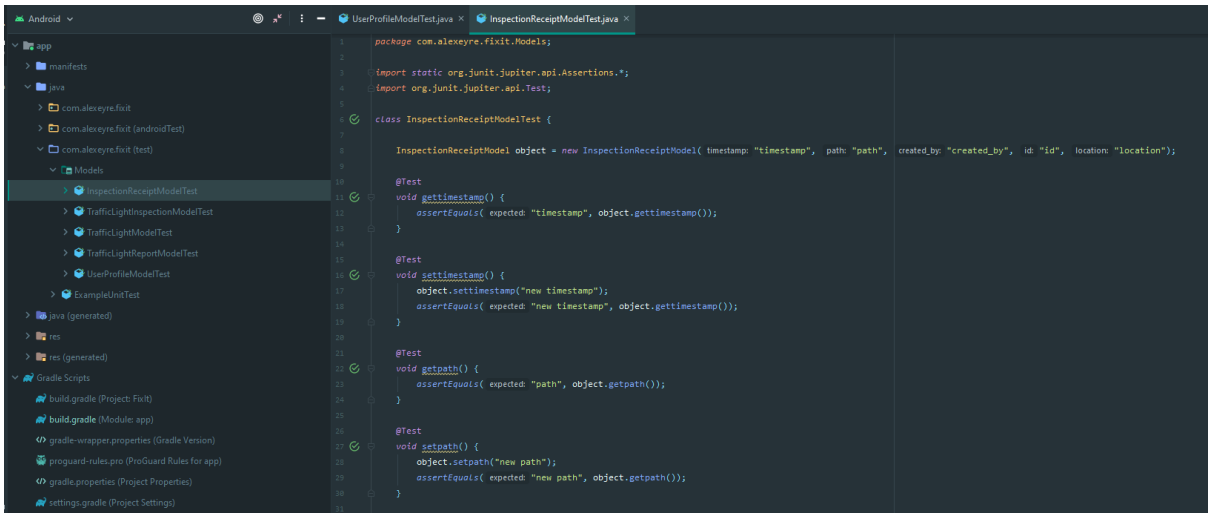

Figure 6.1 Inspection Receipt Model Test

Figure 6.1 is just another example of unit testing that can also be seen in *figure 6.0.*

Instrumentation & UI testing was carried out by performing expresso tests. When performing an expresso test, the application would run in debug mode and record all inputs made when using the application. The expresso recording was then used to create a test class. This test class when first created awaits execution in order to perform testing, where the system would mimic all inputs made by the user in order to test the UI. In the figure below we can see an example of an expresso test.

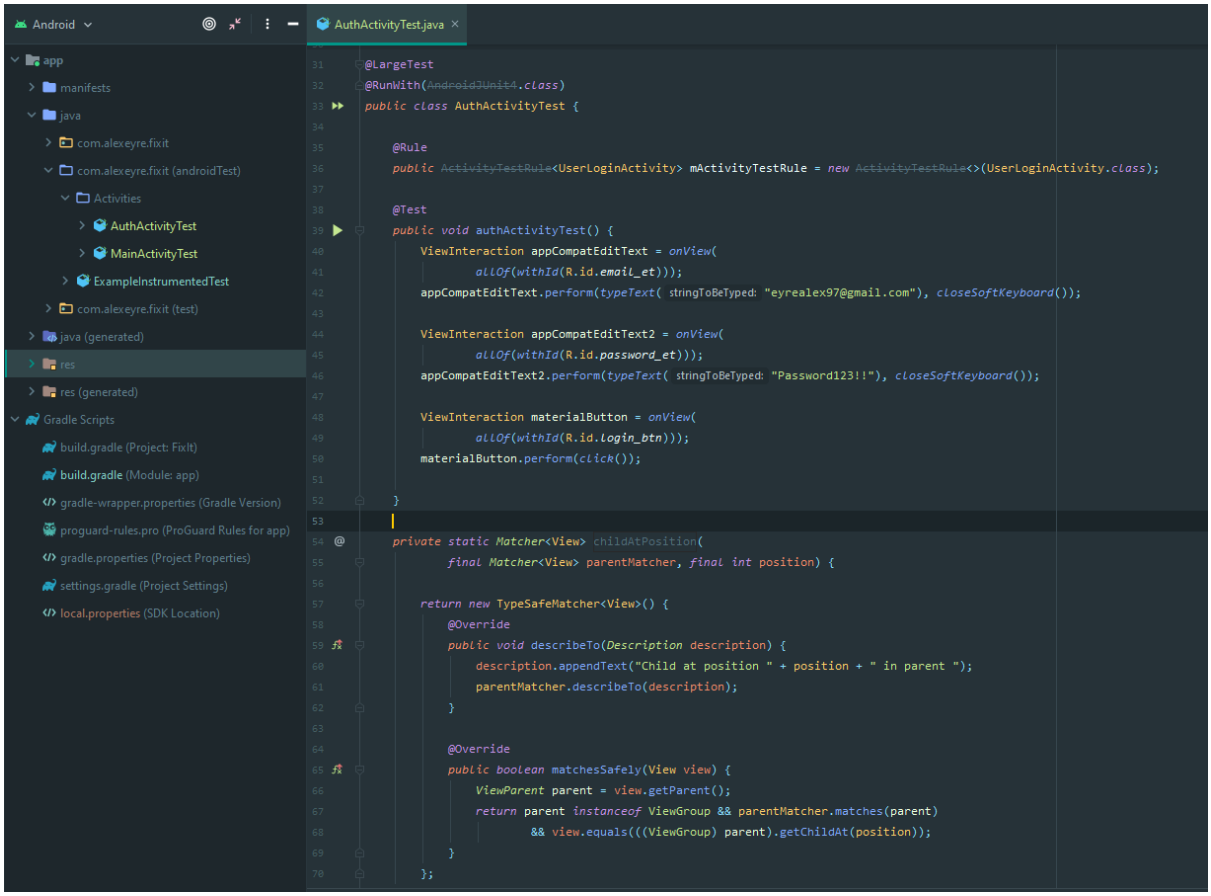

Figure 6.2 Authentication Activity Test

Figure 6.2 shows the test class needed in order the test the UI for the login activity. The test has been provided the correct credentials for one of the users on the system. This allows the system to authenticate itself when it is testing the UI during execution. This expresso test passed successfully.

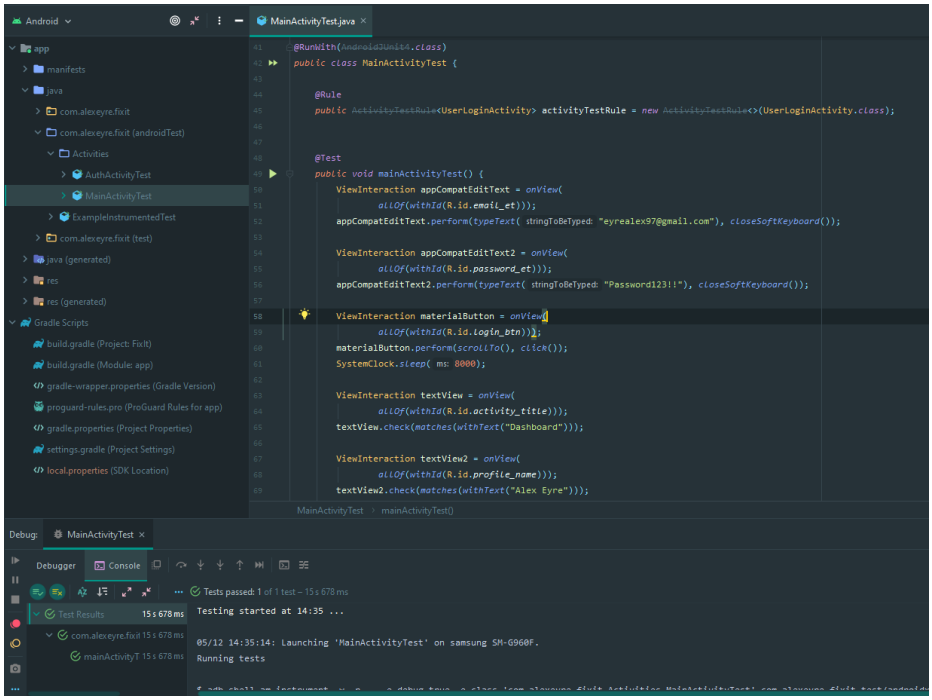

Figure 6.3 Main Activity Test

Figure 6.3 shows the testing that was performed on the main activity using the expresso test recorder. The test algorithms shown in *figure 6.3* show that after the application had been authenticated the system had to sleep for 8000 milliseconds in order for the main activity to load data from the database. After the test algorithms could check for assertions on the main activity in order to check the UI was working as intended, these assertions test for text fields present on the main activity such as activity title and also profile name, which is loaded from the database. All tests passed successfully.

2.6. Evaluation

In terms of evaluating the project, I think that I have reached many of the goals that I had initially set out. The development process from start to finish followed a strict schedule on implementing features, algorithms, designing the UI and testing. During many stages through-out the development phase, I had produced major deliverables to my supervisor for the year showing the progress made every one to two months. These meetings gave me the feedback needed to progress to the next development stage.

I think the finished product of my application has reached expectations on what I had initially set out and I have put a lot of work in to reach other users expectations through the use of surveys. This application, however, is not perfect but it is scalable. There are many features that can be implemented into this application that would surpass expectations if given the time to implement them.

3.0 Conclusions

FixIt serves as a niche project designed for the focus groups withing the maintenance and labour industry. This app has some very useful features to increase productivity, and reduce workload for employees on the job. The application has been designed to scale, giving it a

lot of potential to develop into an application that is necessary for the job given more time for development.

The main strengths found in this application revolve around the user/ employee having a digital platform in order to complete tasks and help with their workload. It is designed as an all-in-one application where users can get directions for job, create inspections and report on all details regarding the task at hand, followed by uploading the report to a database. This is one of the biggest strengths of this application, as users with a mobile phone or tablet have access to a platform which aids in improving work ethic and efficiency. The major strength of this application can be seen to be accountability and efficiency in user tracking. Any user that creates a job inspection, needs to fill out a report on the job and sign off on the inspection. These details are stored in a database and also recycled through the application so that other users can follow up and view these past inspections. This allows for metrics to be created which can display user performance and to track patterns in order to maximize work ethic and minimize wasted resources. The app has been designed to be used by any user of any age category, this is a strength as employees in the labour industry fall into various age categories. In order to a provide a pleasant user experience for all of these users, many testing techniques and prototypes were used in order to gather as much feedback to develop a UI best suited for the majority of people's needs. This application for the mean time is developed for traffic light maintenance, however, another strength in this application is the scalability that is had been developed with. The real-time database, data sources and activities within the application can be altered in order to include other possible maintenance required by the council such as drain maintenance, road maintenance etc.

The main disadvantages of this application relate to resources such as time and manpower. One of the biggest disadvantages was taking this application on as a final year project. This was not because the goals set out for the application could not be achieved but rather that there were many features and functions that could be implemented given extra resources such as time and man power. This application has the potential for scalability as there are many ideas and directions this application could take given more time.

4.0 Further Development or Research

If this project was given additional time and resources many features could be implemented. For the finished project, the goals set out from the beginning were achieved, however, it would have great to implement more features given more time. The metrics activity needs a lot more work and has the potential to scale very well. This application would be very useful to the users and admins if the metrics feature was fully flushed out in terms of functionality. The metrics could include features such as inspection report tracking and open/closed maintenance tracking, as well as tracking user performance in order to improve work ethic.

Other metrics could track how many traffic lights were checked within a certain period of time and if that value met the quota required, for example there are 900 traffic light locations, each of these locations need to be inspected when the application goes live. Given an employee work force of six to eight employee's, if eight employees were to inspect

at minimum four traffic light locations per day, that value adds up to 32. Multiple 32 by five working days in a week results in 160 traffic lights inspected in a week by 8 employees. Multiple this value by four to get the result for a month, which results in 640 traffic lights inspected in a week. Given this information metrics could tell the system admins, how often each traffic light should be inspected per year, how many employees are needed to get the job done. A nice feature I would like to implement in the future alongside these metrics would be to let the system admin know how efficient their inspections are and the quality of their traffic lights etc. This would be done by showing the amount of traffic lights that have problems per inspection report. It is important to know that out of the 900 traffic lights, it is quite possible that more than 80% of the time the inspection will result in a success and no maintenance will be required, however, if the percentage of failures grows and continues to grow per inspection period, this can provide data to the system admins and councils that maybe they need to source materials, traffic lights etc from different suppliers. I believe the metrics has a lot of potential to really display the power of this application and given more time it could be implemented.

Besides from implementing extra metric features, I would have like to structure my code more efficiently. I had research of a model-view-view-model architecture or better known as MVVM. I would have liked to implement this as it would greatly cut down on repetitive code and would allow the app to run faster as its more code efficient. I had research a little bit of MVVM and tried to implement it but stopped in order to finish my project and task at hand. But with extra time and resources this and the metrics would be the most important things to work on.

5.0 References

1. Firebase. 2021. *Read and Write Data on Android | Firebase Realtime Database*. [online] Available at: <https://firebase.google.com/docs/database/android/read-and-write> [Accessed 21 November 2020].
2. Firebase. 2021. *Get Started with Firebase Authentication on Android*. [online] Available at: <https://firebase.google.com/docs/auth/android/start> [Accessed 24 November 2020].
3. Firebase. 2021. *Upload files with Cloud Storage on Android | Firebase*. [online] Available at: <https://firebase.google.com/docs/storage/android/upload-files> [Accessed 5 January 2021].
4. Square.github.io. 2021. *Picasso*. [online] Available at: <https://square.github.io/picasso/> [Accessed 9 January 2021].
5. Google Developers. 2021. *Maps SDK for Android overview | Google Developers*. [online] Available at: <https://developers.google.com/maps/documentation/android-sdk/overview> [Accessed 15 January 2021].
6. GitHub. 2021. *pedant/sweet-alert-dialog*. [online] Available at: <https://github.com/pedant/sweet-alert-dialog> [Accessed 10 February 2021].
7. GitHub. 2021. *zahid-ali-shah/SignatureView*. [online] Available at: <https://github.com/zahid-ali-shah/SignatureView> [Accessed 5 April 2021].

8. Code Envato Tuts+. 2021. *Android Design Patterns: The Singleton Pattern*. [online] Available at: <https://code.tutsplus.com/tutorials/android-design-patterns-the-singleton-pattern--cms-29153> [Accessed 9 April 2021].

6.0 Appendices

6.1. Project Plan/Proposal

6.1.1. Objective

The objective that I have set out for my project is to create an application that can be used for employees working on behalf of a county council in Ireland or as a contractor. The idea for my app is to create a job packet application that will allow a supervisor to assign the workers underneath them tasks on a weekly/monthly basis. Employees should be able to see the jobs they have been assigned, see where the job is located and what needs to be done for the task for that week/month. The employee will have to fill in documentation that comes with the application such as checking boxes from the work that has been fulfilled, providing evidence by the means of a photo and also a signature. This report will then be stored in the database that can be pulled up by head office to be manually inspected to ensure jobs have been completed before signing the job off. My objective for this application is to make the process of the councils more efficient and digitalised as a phone application is easily accessible by everyone, cutting down on paper waste whilst also ensuring that jobs can be seen and altered on the go. In terms of what I will be using as a template for what work I will be dealing with; I shall be incorporating public data from Gov.ie that I will be able to manipulate to help me create these job packets that I hope to be implementing.

6.1.2. Background

The reason why I have gone with the idea that I have, is that I know someone that is working within the council and they have said that a lot of their work is done through the word of mouth or plans that are drawn up on pieces of paper. There is also ticketing systems that can be used such as Jira where tasks can be distributed. I think the convenience of an android application will greatly improve upon allocating jobs and submitting the reports that are necessary. I think the idea of having an application such as this can provide a lot of potential and many features can be added at request of people within the council using the app or contractors using the app. These features can be implemented at any given time that will further upon make the job easier for people.

6.1.3. Technical Approach

The approach I will be taking with this application is drawing up all the function and ideas that I can think of that should be implement into this application, focus on the most necessary features and implement more once the previous features have been tested appropriately. I've done some of my own research to see how work would be allocated for these public workers and I can't find anything that would tell me what I don't already know. There seems to be no current applications that the councils can use to allocate job packets. There seems to be no other way of an employee knowing what needs to be done other than to have training provided to them and then been told from word of mouth what needs to be

done. Of course, training is a must but to have an app at your disposal with very clear guidelines and tasks set out for you will improve in the accuracy and make working on public assets more efficient. I will need to create a system that will allow for admin and employee access, this will allow me to create features that can only be seen by the admin such as allocating jobs etc. I will need to link this application with a database to store the information of all the admins, and users that will have this application, as well as images taken onsite, signatures and to store reports.

In terms of implementation of my approach to this project, I will be following the agile methodology. I will be taking this approach because I think it will be the most flexible in what I am looking to achieve with this project. By taking the agile methodology approach, I will be able to plan my objectives, create my designs, develop my code, test my code and then evaluate on my work to see what needs to be done for my next iteration of my project. I will also be outlining my objectives that need to be done for the day, every day as if I was in a scrum meeting to ensure that I am well aware what needs to be done and by following timelines, will help me reach an end product that is shippable by May.

6.1.4. Gantt Charts

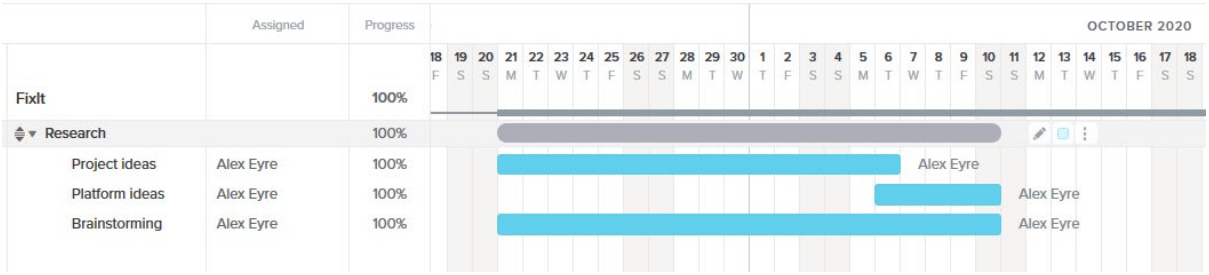

Figure 7.0 Researching

The image above shows the timeline that I had originally set out at the start of the year. The timeline started on September 21st and finished up on October 10th 2020. This time line was giving sufficient time to brain storm ideas in depth in order to produce a project pitch that would meet the criteria of the final year project. These criteria consisted of complexity, innovation, achievable given to final year project deadline and most importantly, an idea that I was content in developing. All of these factors were thought out thoroughly before I started my development phase.

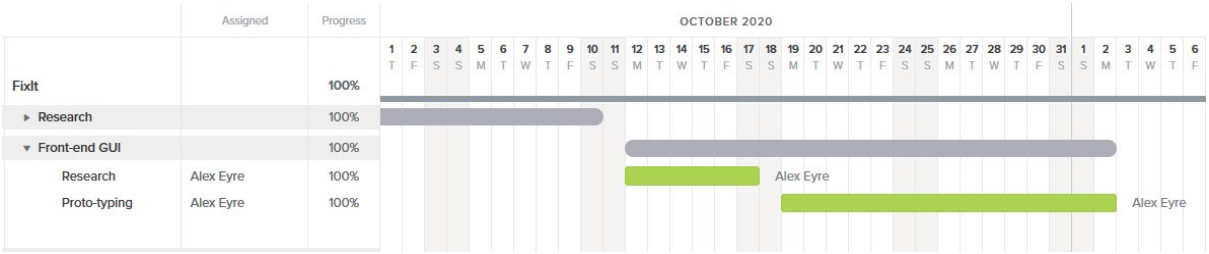

Figure 7.1 Front-end GUI

The image above shows the timeline that I followed when developing the front-end gui. This phase began after the researching phase. This phase started on October 12th and lasted until

November 2nd 2020. I spend the most of a week researching ideas on how I wanted my GUI to look for the application. I looked at many other applications through the play store, viewed images on google and gathered a vast amount of knowledge that I could use for my own project. After this research I started creating prototypes and began working on the GUI inside Android studio. I also used some proto-typing tools such as Invision to aid me with the look and feel of the application. When I was content with having a good prototype, I started development. I would later come back to the GUI phase and refine it at a later date when I had most of my functionality working.

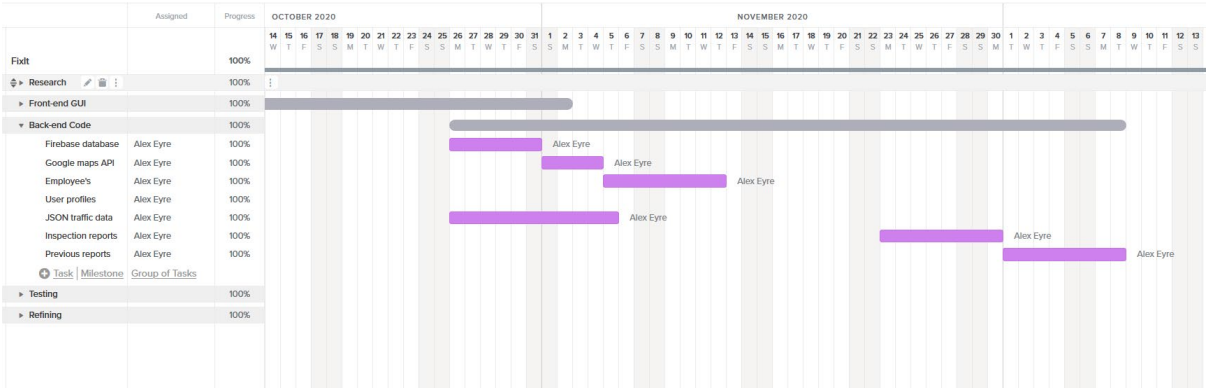

Figure 7.2 Back-end code

The image above shows the schedule that I had created for the back-end development phase of my application. From the image we can see that back-end development began on October 26th and finished around December 8th 2020. The workload that I had scheduled for this period included implementing the firebase real-time database and working with user authentication. Using the database, I also had to import JSON data which was used to create my locations list and inspection reports. I set time to work on most of the core features of the application such as the google maps API, employee implementation, user profiles, traffic light location lists, inspections and previous reports. The plan was to have most of this work finished by December; however, I was not able to implement the inspections and reports at the time displayed on my Gantt chart. Much of this work was postponed to a later date, around April 2021, due to other projects and modules having submission dates around January/February 2021. That being said, although I did not reach my timeframe goals that I had set out, I implemented all the features that I had initially set out, just at a later date.

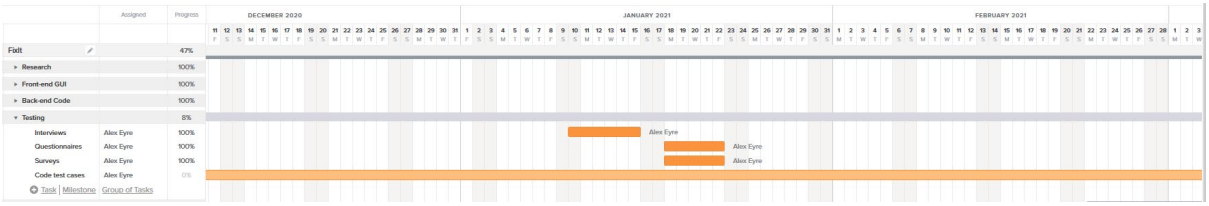

Figure 7.3 Testing

The image above shows the testing that was involved throughout the whole process of my application. The plan that I had scheduled was to perform testing on my application in terms of back-end code testing from the beginning of the development phase. This would allow

the project to follow a strict test-driven development. Testing was to begin on October 26th 2020 and finish up around May 10th 2021. This was to ensure every implementation of code could be tested. This however, was not always the case when put into practice, the testing schedule did not follow what was set out in the Gantt chart, there were months when no code or development was done because of priorities with other modules, submissions, Testing was not as smooth as I had hoped it to be. I had implemented many features in my application without following a test-driven approach, however, testing after multiple features rather than after every feature implemented. Other than code-based testing, I had performed testing in terms of usability, gui and user experience of the beta applications produced. This testing consisted of focus group interviews, questionnaires and surveys completed by users of different career backgrounds in order to gather enough information and feedback on their user experience. This data was recorded which was then used to improve on features, GUI and the overall performance of the application in order to develop and application that catered to as much users as possible. These tests took place around the middle of January 2021, over a period of two weeks, this gave me enough time to implement all the changes and feedback before the project submission date in May.

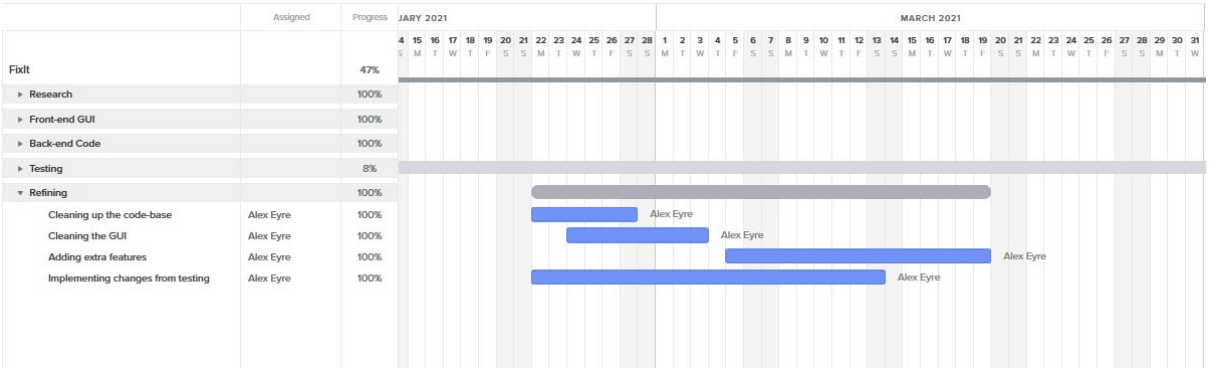

Figure 7.4 Refining project

The image above shows the scheduled workload that was set out as the final steps for the project. These steps included implementing any final back-end functionality, front-end gui, any extra details or features and the final testing necessary from this phase. The schedule was due to start towards the end of February 2021 and finish around March 19th 2021, however this was not the case with the project. Due to time allocation conflicts from other modules and projects, the schedule for this phase was extended all the way up until May 2021, the month of submission.

6.2. Ethics Approval Application (only if required)

N/A

6.3. Reflective Journals

6.3.1. Reflective Journal 1: October

For the first month of the final year, most of my time spent involved brain storming and prototyping ideas for what I was going to do for my project. Some of these ideas consisted of whether I would build a website, an android application, or technology that involves the

use of a raspberry pi. In the end, I decided to choose a technology that I was very confident with which was android development. Another key factor that made me decide on an android app was the mobile development module. I began researching ideas for an android application that would be complex and exciting to develop for a final year project. The idea that came to mind involved a mobile platform that allowed construction workers to report jobs. I began drawing out templates and ideas for this idea. It was also used for my project pitch; however, no feedback was received whether this pitch had been accepted, but given the pitch and complexity of an application like this I think it was okay to continue working on the idea, and application. When the pitch has been accepted, I can put all my resources and time into the in order to build a usable prototype to be shown for the midpoint presentation in December.

6.3.2. Reflective Journal 2: November

For the month of November, I have made a lot of progress in terms of the user interface for the application and some of the features that I wanted to show off in the mid-point presentation come December. Currently the application has a login and a fully functional dashboard. I have been working on mainly back-end with minimal but still elegant looking front-end UI. The reasoning for this was to make sure that my application can actually function for the mid-point presentation. The app currently shows all the traffic light locations in Dublin in list format. I am hoping to have a map view of all traffic light locations in Dublin as well which would look very nice for the mid-point presentation. Overall, I am content with the amount of work on my final year project and also the other modules.

6.3.3. Reflective Journal 3: December

The third month for this report has been a very difficult month when it comes to working on my final year project, currently I have been overwhelmed with many projects across five modules. I have had very little time to work on my final year project. Given the limited time I have had to work, I have designed a work flow plan that I would like to follow to get me up to speed with my project, I need to add a login portal for an admin and a user on the application. I have done some research on how I will implement this and I hope to get as much done when I have submitted some of my other module projects that are due very soon.

6.3.4. Reflective Journal 4: January

The month of January involved getting back into the swing of things after the long Christmas break. I continued to work on my modules during the Christmas break but I took some days off to spend time with my family and have a break. When lectures started again mid-January, I was fully focused and began working hard again. My final year project is in a very good position right now but I still have a lot of work to do. The last features implemented from the last time include the user profiles and database structuring to be used for the next feature to be implemented. Other module submissions have been going well up to this point in time and I am not currently under any stress.

6.3.5. Reflective Journal 5: February

February has been a good month in terms of progress for the college year. I have implemented features in my final year application such as showing all users on the application and their user profiles. I have also been working on the start of the inspection reports which are the core

functionality of the application. There are a lot of project submissions for the ned of March, so this month I have mainly been focusing on other modules, with little work done on my final year project, just so I can stay on top of all the other modules.

6.3.6. Reflective Journal 6: March

March has been a very busy month; I have had a lot of submissions. It has been very stressful as the modules have been tough to stay on top of. I have found the workload for the distributed systems to be very difficult, however, I managed to overcome any issues and was very content with the work submitted for the distributed systems project. I have been working a lot on my cloud application project this month as well as the submission for that is due towards the end of March. I am content with the progress made this month and I am on track for having all work completed before submission. In terms of my final year project. I have been working on the inspection reports but have not been able to get as much work completed as hoped due to the distributed project, however, next month ill have more time to focus on my final year project as the cloud application project is not far off completion and the usability module which won't take long to complete as the workload is split between a colleague and I.

6.3.7. Reflective Journal 7: April

The month of April has been very smooth in terms of workload. I have submitted both the cloud application development project and the usability project. The submissions were very smooth and I am quite happy with the work submitted. I have had a lot of time to work on my final year project this month and have implemented many features. I have completed the inspection reports feature for creation and view previous reports. I need to perform some extra testing and clean up the UI for the next month in order to be in a reasonably good state by the time the project is due for submission. The project report has been going very well, I have received a lot of feedback from my supervisor, so the month of May will consist of adding the final touches to the project and report.

6.4. Other materials used