National College of Ireland
BSc in Computing
2016/2017

Conor Keenan
X13343806
Conor.keenan@student.ncirl.ie

In Time

Technical Report

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name:
Conor Keenan
Student ID:
x13343806
Supervisor:
Adriana Chis

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: _____ Date: _____

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- the requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

Technical Report

Revision History

Date	Version	Action	Contributor
28/11/2016	1	Created	Conor Keenan
08/12/2016	2	Updated for Mid-Point Presentation	Conor Keenan
03/01/2017	3	Updated after Mid-Point Presentation	Conor Keenan
06/05/2017	4	Prepared for Final Upload	Conor Keenan

Distribution List

Name	Title/Role
Eamon Nolan	Lecturer
Adriana Chis	Project Supervisor

Table of Contents

Executive Summary

According to Brooks (2015), traffic is the most common reason why employees are late for work. This is an issue that is hard to avoid for even the most punctual people.

A proposed solution to this is “In Time”, an Android alarm application that is designed to reduce/overcome this problem. In Time checks current and predicted traffic conditions before one wake up in order to adjust the alarm activation time if needed. This means that if there is particularly bad traffic one morning, the alarm will activate early to reduce the chances of one being late.

This functionality works for both public and private travelling methods and is aimed at commuters living in the greater Dublin region. The system works by combining the transport waiting times from the Dublin Real-Time Passenger Information(RTPI) Application Programming Interfaces (API) to the direction provided by Google’s Directions API.

This mobile application is built in conjunction with a NodeJS based web application that performs the logical calculations and expose them in the form of a RESTful API. This allows for separations of concerns between the mobile application and the journey time update calculations.

This separation also allows for statistics to be gathered anonymously from the user. These statistics can then be used later to potentially increase the accuracy of the system. As of 5th May 2017 the system has an overall accuracy of 87%, meaning all travel time estimates are within 13% of the actual travel time.

1 Introduction

It is important to arrive to work on time for various reason including appointments, meetings, contractual agreements, etc. Some of the cause that could potentially cause one to be late is traffic, public transport scheduling issues among many other factors.

In Time is a smart alarm system, designed to solve this issue. It uses real time public transport information and traffic conditions to adjust the alarm time. For this project a system has been developed that helps users arrive at work on time, particularly during periods of unforeseen traffic conditions.

The target market for this application is people living and working/studying in Dublin, Ireland. It will however be usable by all people requiring an alarm, but some of its functions may be limited. For example, the public transport information will be restricted to Dublin due to limitations of the RTPI Api. Other functions such as basic alarm and smart alarm with private transport should be fully functional.

1.1 Background

There are several reasons why I want to develop this application. The first being that there is a need for this application. After doing some research I found that *"traffic is the most common cause for employee lateness" (Brooks, 2015).* According to Mercer (2012), in the UK there are nearly 600,000 workers late for work on a daily basis, costing the British economy approximately £9 million. Although this is a study done in the UK, Ireland has a very similar culture and would produce similar results. Statistically this means that roughly 1% of People are late every day making it a target market of roughly 46,000 people. This could potentially be greater if you consider that fact that it would not be the same people being late every day.

Another reason is that I am a very punctual person and tend to be on time, if not early. During workplace I noticed that certain days of the week or months of the

year seemed to cause differences in travel conditions, this often made it hard to predict what time to leave at. Another factor, that was even less predictable, was car accidents. Depending where these happened they seemed to cause a near standstill in traffic.

In addition to these points, this project gives me the chance to develop upon my existing Android development knowledge, which is currently at a basic to intermediate level. I have previously worked with Android application during my second year project and have since developed an interest in the area. Recently, I have been researching about mobile development and look forward to applying the knowledge that I have gained.

Finally, the complexity of this project gives me a chance to apply the web development knowledge that I gained during my work placement. Although I do not plan on using the same web stack that I worked with in AOL, I still have knowledge of concepts and good practices that I will apply, while also learning a new technology.

1.2 Aim

The aim of this project is to develop a mobile application that will help commuters arrive at work/college on time. This application will be essentially a smart alarm that extends the features of the stock android alarm. It will use current, historical and predicted traffic conditions to calculate commuting time and change the time of the alarm accordingly.

The target market for the app is anyone living in Dublin and travels to work and/or college. Although it is aimed to be used in Dublin it will be designed to work internationally for private travel, such as a car, but may have limitation for public commuting.

The project will consist of two main parts, a mobile application and a web based application. The mobile application will be used to set any alarms and locations, while the web based application will perform all the logic for traffic predictions and

expose the results in the form of a RESTful API. This web based application will also provide the user with the ability to view statistics gathered by the system.

1.3 Market Research

While researching my idea I found three other similar applications that would be in competition with In Time. They are as follows:

1.3.1 Never Late

Never Late is an alarm application that claims to adjust its time depending on traffic and weather conditions. When trying out this application I found that I was unable to add in my location as it seemed to only pick up American addresses, making it unusable in Ireland. I also did not find the UI very pleasing to the eye and I intend to improve this using Androids material design. (Never Late Smart Alarm, 2015).

1.3.2 wUp Traffic Alarm Clock

wUp also uses traffic to predict the time for the alarm. In this case however you enter the time u wish to arrive, opposed to a base alarm time. The issue with this app is that it does not include public transport and only works when driving to work/college. (wUp, 2015)

1.3.3 Puzzle Alarm Clock

This alarm would not be in direct competition as it does not change time but instead offers the ability to add puzzles when deactivating an alarm. I am noting this here for two reasons. The main one being that it has an appealing UI that is similar to how In Time looks. The second is that the puzzle feature is interesting and could be a stretch goal or evolutionary feature that could be added to In Time. (Wro Claw Studio, 2016)

1.4 Technologies

Grouped into four main categories the technologies that are used to develop this system are as follows:

1.4.1 Mobile Application

Apple IOS and Android were both considered when choosing a mobile platform for this project. Android was chosen as it has the largest market share. In the third quarter of 2015 it had a market share of 83.4% versus 13.4% that IOS had. (IDC, 2016)

The android application was developed using the base Android stack which includes Gradle, Java and Xml. To help with development of this app I have also used some dependencies, these are listed below.

- Butter Knife: Used to for view injection in Android. (Wharton, 2013)
- OkHttp: Used to make HTTPS requests to the server. (Square.github.io, 2016)
- Gson: Used for conversions between Java objects and JSON. (GitHub, 2008)

1.4.2 Web Application

Many Web frame works were considered for this project such as Ruby on Rails, Django, asp.Net, etc. I chose to use ExpressJS + NodeJS as it is a lightweight framework that does not require a lot of boilerplate code. The web application is primarily designed to be an API meaning, it does not require the creation of many elaborate front end elements. The front end of the web app will be designed for two main purposes, firstly as a means to advertise the mobile app and secondly to allow visualization of the anonymous data collected from the users. NodeJS still allows for more complex and elaborate front end elements if the application evolves over time.

NodeJS is a JavaScript based environment that is designed to be able to run on server side. I have deployed this Web Application to Digital Ocean and it is

currently hosted at InTimeAlarm.com. This Digital Ocean droplet is set up using Ubuntu 14.04 as its Operating System and using Nginx as its Http Server. NodeJs uses Npm to manage its dependencies. The dependencies/libraries I have used are as follows:

- Express: This is a web framework for NodeJs. (Express, 2016)
- Google/Maps: This is used for making requests to the Maps Api. (Google Developers, n.d.)
- Moment: Used for converting timestamps. (Moment, n.d.)
- Request: Used to make network requests. (GitHub, 2004)
- Yaml-config: Used to store global variable in a YAML file. (Jyo, 2011)
- Bower: Used as a package manager for front end libraries. (Bower.io, 2012)
- MongoDB: A NoSQL database used to store trip information for evaluation. (MongoDB, 2017)
- Pug: Used for html templating. (GitHub, n.d.)
- Mocha: Testing framework for NodeJS. (Mocha, 2017)
- Supertest: Library for testing express server APIs. (Holowaychuk, 2014)
- Gulp: Manages builds during development. (Gulp, 2013)

1.4.3 Testing

Testing is done on the web application using the Mocha and Supertest node dependencies that are listed above. The mobile written using JUnit and Mockito to mock native java classes.

1.4.4 APIs

The APIs used in this system to get information about travel times are Google Map's Directions API and the real-time passenger information (RTPI) API. The project also implements its own custom API for the android application to make requests to.

2 System

In Time is a complex system that features both a mobile and web application. This section discusses in detail the requirements, design, architecture, implementation and testing of the system. These are the main stages of the software development lifecycle (Sommerville, 2016).

2.1 Requirements

2.1.1 Functional requirements

- Create alarm, it could be basic or smart.
- Perform logic on external APIs to calculate new time.
- Make web service request from Android device.
- Stop Alarm.
- Add a location.

2.1.1.1 Use Case Diagram

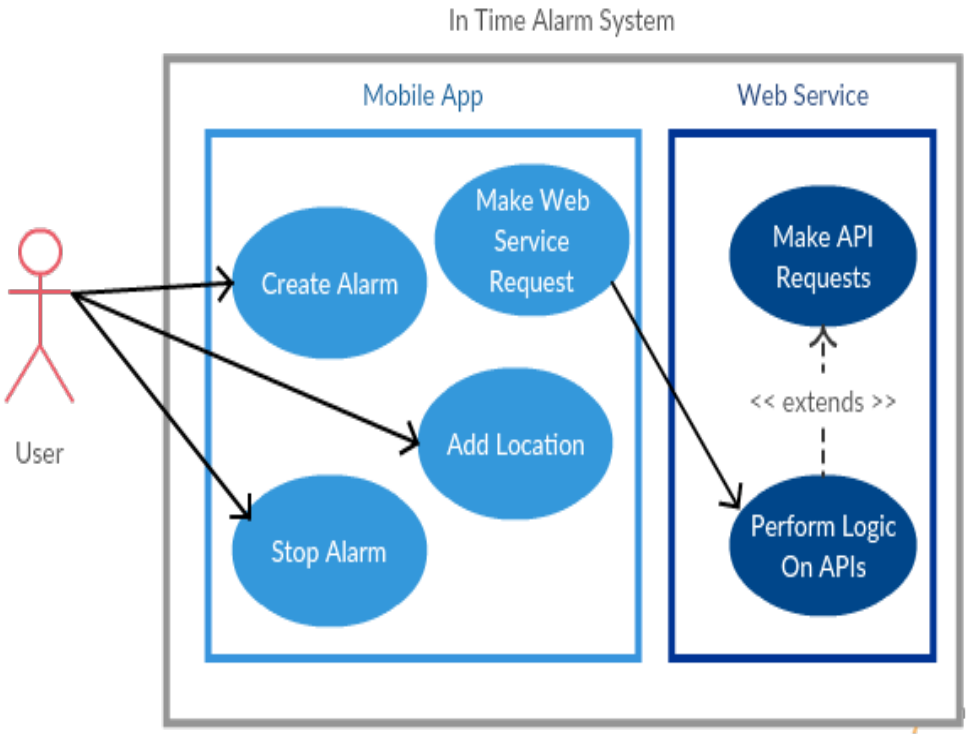


Figure 1: Use Case Diagram

Figure 1 illustrates a high level of the main use cases for this project. It shows the main functions that a user can perform. Namely these are to create an alarm, add a location and stop an alarm. It also shows the use case of the application making a request to the service to get an updated alarm time.

2.1.1.2 Requirement 1: Create Alarm

Description

This is the main function for the user. It allows them to create an alarm on their smartphone, which will activate at a specific time.

Use Case

Scope

The scope of this use case is for the creation of an alarm. This could be either a basic alarm or smart alarm.

Description

This use case describes a user making an alarm on their device.

Flow Description

Precondition

The application is open on the android device.

Activation

This use case starts when a user clicks the “create alarm” button.

Main flow

1. The user enters a time.
2. The user selects days for the alarm to go off.
3. The user selects the smart alarm radio button (See A1)
4. The user chooses a start location (See E1)
5. The user chooses a target location.
6. The user enters usual time to get ready.
7. The user enters their due time at the target location.
8. The user clicks Save.

Alternate flow

A1: User is creating Basic alarm

1. The user does not select smart alarm box.
2. The use case continues at position 8 of the main flow.

Exceptional flow

E1: User has no locations to choose

1. The user clicks add location option.
2. The user performs the Add a location use case.

Termination

The System returns to main home screen.

Post condition

There is a new alarm in the android system waiting to activate.

2.1.1.3 Requirement 2: Preform Logic on APIs

Description

This is the use case that performs an algorithm, that is run by the web application to calculate the new time for the alarm. It is essentially what makes the alarm smart.

Use Case

Scope

The scope of this use case is to calculate the best possible time for the alarm to activate in order for the user to arrive at work on time.

Description

This use case describes the web service calculating the new alarm time.

Flow Description

Precondition

The web service is in a waiting state and there is a pending alarm set on the user's device.

Activation

This use case start before the activation of the alarm. The android device sends a request to the service to get updated information.

Main flow

1. The user's device sends a GET request to the web service.
2. The web service requests information from Google Directions API. (See A1)
3. The web service parses these to extract relevant data.
4. The web service uses this data to calculate a new alarm time.
5. The web service returns this information in JSON format.

Alternate flow

A1: User is traveling by public transport.

1. The web service requests information from Google and RTPI APIs.
2. The use case continues at position 3 of the main flow.

Termination

The web service returns to a waiting state.

Post condition

The alarm on the user's device is updated accordingly.

2.1.1.4 Requirement 3: Make Web Service Request

Description

This is the use case of the android application sending a request to the web service to get an updated alarm time.

Use Case

Scope

The scope of this use case is to send a request to the webserver containing all the relevant information needed to calculate a new alarm time.

Description

This use case describes the android app updating the alarm time.

Flow Description

Precondition

The android application has a pending smart alarm set.

Activation

This use case start before the activation of the alarm.

Main flow

1. The user's device wakes before the pending alarm is due to activate.
2. The device sends a request to the server containing initial time, mode of transport, end location and start location.
3. The device checks response is error free. (See E1)
4. The device updates the alarm according to the response.
5. Alarm activates at new set time.

Exceptional flow

E1: Response contains error code.

1. Alarm activates at original time.
2. User is informed about a problem updating Alarm time.

Termination

The application waits for next pending smart alarm.

Post condition

The alarms time is updated and activates at new time.

2.1.1.5 Requirement 4: Stop Alarm

Description

This is the use case of a user stopping an activated alarm.

Use Case

Scope

The scope of this use case is the user stopping an alarm that has been activated.

Description

This use case describes a user stopping an alarm.

Flow Description

Precondition

The android device must have a pending alarm.

Activation

This use case starts when the system time on the device is equivalent to that of the pending alarm.

Main flow

1. The user's device begins to play an alarm tone.
2. It displays a button to the user to stop the alarm.
3. The user clicks the button.

Termination

The application waits for next pending smart alarm.

Post condition

The alarm has successfully lived its lifecycle.

2.1.1.6 Requirement 5: Add a Location

Description

This is the use case of a user adding a new location to their app. The location can be used for both the start and end of an alarm and it is chosen at time of creating an alarm.

Use Case

Scope

The scope of this use case is the user adding a location to their list of saved destinations.

Description

This use case describes a user adding a location.

Flow Description

Precondition

The android application must be active.

Activation

This use case starts when the user clicks the “Add Location” button.

Main flow

1. The user search for the desired address. (See A1)
2. The user selects their address from the autocomplete list.
3. The user confirms their address by viewing on a map and a textual representation of the full address.
4. The user clicks save.

Alternate flow

A1: User chooses their address using the map.

1. The user uses the embedded google map to choose a location

2. The use case continues at position 3 of the main flow.

Termination

The application returns to the screen that initiated the request.

Post condition

A location has successfully been added to the users list of saved locations.

2.1.2 Non-Functional Requirements

2.1.2.1 Data requirements

All the data sent between the application and the service is done using HTTPS in order to secure the location information.

2.1.2.2 User requirements

In order to use this application the user is required to have an android device running a minimum SDK version of 21, which equates to Android 5.0 (Lollipop).

They must also be reasonable technologically literate in order to be able to understand and use its functions.

2.1.2.3 Usability requirements

Although the user is required to have some technological literacy, the application will be designed to be as simple to use as possible. It will feature reasonable sized buttons and will not over-crowd the screen with information.

2.1.2.4 Reliability requirement

The Android application should be reliable to the user. If an alarm is set the app should provide the user with confidence that the alarm will activate at that time or possible earlier. The device should also automatically re-enable alarms once the device has been restarted

2.1.2.5 Efficiency requirement

The application should be efficient with the mobile resources. In particularly power and network usage. The efficiency for network usage will be implanted by

only performing one network call to the web service. The web service will then make all other necessary API calls and perform the logic on the results. This reduction in network calls will also help reduce power consumption. Power consumption will also be saved by developing the alarm using android best practices. This will stop it constantly running in the background until the set time is reached.

2.1.2.6 Availability requirement

The alarm application should be available to the user regardless of network connection. If the user's device does not have internet connection during the period before alarm activation, it will not be able to update the time. In this case the alarm will activate at its original set time.

2.1.2.7 Accuracy requirement

The algorithms run by the web service to calculate a new alarm time should produce an accurate result. This will be achieved by continuously testing the algorithm and evolving it if any inconsistencies arise.

2.1.2.8 Security requirement

The requests made to the web service by the application should be done over HTTPS. This will encrypt the network traffic so that the user's location will not be revealed. To do this a SSL certificate is installed on the server. This certificate is provided by the certification authority Let's Encrypt.

The API key for all third party APIs will be stored on the web service. This will protect them if someone tries the reverse engineer application apk file.

2.1.2.9 Maintainability requirement

The code should be written in a maintainable way so that if bugs are found or features are added it is easy to do so. Both the web and mobile applications will be built while conforming to their relevant coding best practices. The android developer website (Android Depevelors, 2017) for information on Android best practises. The web application will also be built by modularising its components to all easy code maintainability

2.2 *Design and Architecture*

The following presents high level architectural diagrams that illustrate the communication between the system components.

2.2.1 Architecture Level 0

Figure 2 illustrates at a high level how the system works. It consists of two main components, an application installed on a user's android device and a web application hosted under the domain of InTimeAlarm.com. When the application is nearly due to activate it sends a request to the server, which responds with an approximate travel time to the user's location. The web application calculates this time by combining data from third party APIs, namely Google Directions (Google Developers, n.d.) and RTPI (Data.gov.ie, 2014). The system uses Google Directions to get information about the route including travel time between stops. It then uses RTPI to calculate how long the user will have to wait once s/he reaches a stop. By adding these durations together one will receive an accurate time that the trip will take.

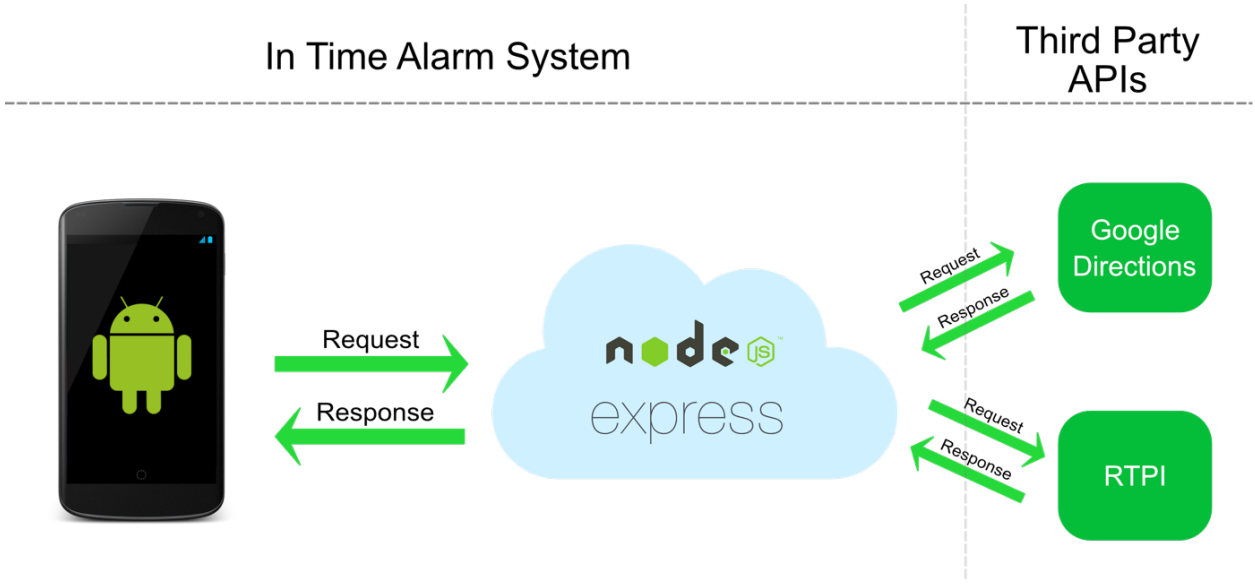


Figure 2: Level 0 Architecture

2.2.2 Architecture Level 1: Android App

Figure 3 illustrates the architecture of the mobile application. The application is composed of several Activities, Services and Adapters among other Java classes that allow the application to function.

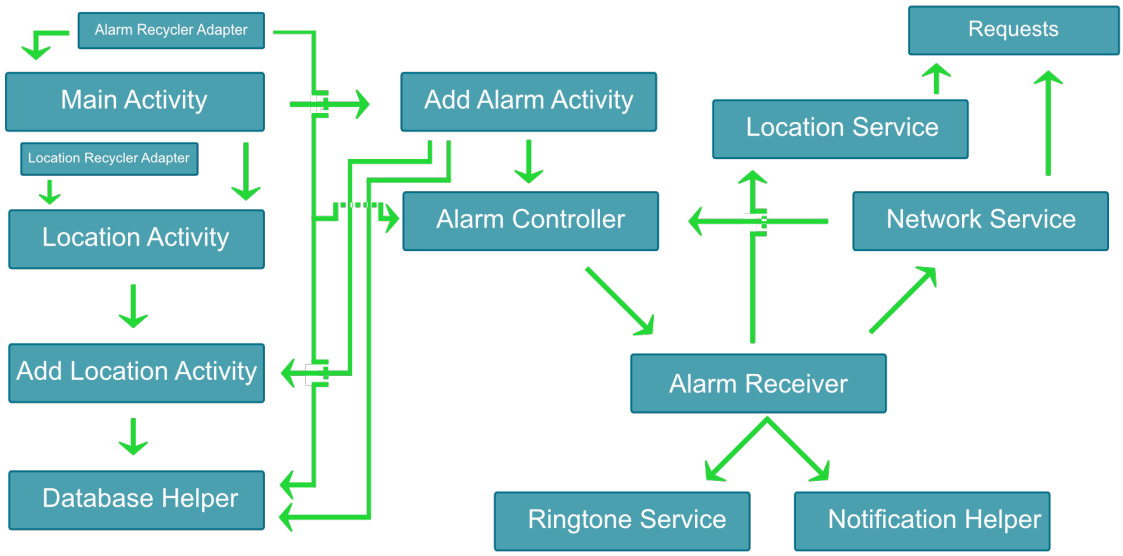


Figure 3: Level 1 App Architecture

The Main Activity (Home Screen) is the first activity which the user interacts with. This activity presents the user with a list of their current alarms both active and inactive. It provides the user with the ability to interact with these alarms by enabling/disabling them, deleting them and opening them up to edit. The Alarm Recycler Adapter is used to populate the list of alarms and communicates any changes to the alarms state to the Database Helper. The Main Activity also provides means for the user to navigate to the Add Alarm Activity and the Locations Activity.

The Location Activity is a simple recycler view that allow the user to add, view and delete saved locations. The Location Recycler Adapter is used to populate this list and communicate between the Database Helper and the Location Activity. When adding a new location, the app uses the Add Location Activity which uses Google Places Api. This provides both the user with a standard user

interface that they are familiar with, and the app with formatted location data. These locations are then saved to the SQLite3 database, using the Database Helper class.

The Database Helper class provides an interface for all the CRUD functionality to the database. It provides methods such as “getAllAlarms()” and “addAlarm(Alarm a)”. The Alarm Recycler Adapter and Add Alarm Activity also communicates with the Database Helper in order to add a new alarm and to update current alarms.

The Add Alarm Activity allows the user to both create new alarms and edit existing ones. It provides a user interface for users to choose the activation time, if it is a repeating alarm, if it is smart or basic etc. Once the user clicks save the alarm, the information is either added or updated in the database and it is set to active.

The Alarm Controller provides all the functionalities for setting and stopping alarms. When setting an alarm, the controller categorizes it into one of four categories:

1. **Single Alarm:** This is a basic alarm that will activate at the user’s chosen time and then deactivate.
2. **Single No Disable:** This is a single alarm that will remain flagged as active in the database after activation. This type of alarm is set by the smart repeating alarm.
3. **Repeating Alarm:** This is a basic alarm that will activate at the user’s chosen time and day. This alarm will repeat on the relevant days until it is manually disabled by the user.
4. **Smart Single Alarm:** This alarm will wake the device up silently, before the activation time, to make a request to the server. From the server’s response, the device will then set a single alarm that will activate at or before the user’s chosen time. After activation of the single alarm the alarm is deactivated.
5. **Smart Repeating Alarm:** This alarm works in a similar way to the smart single alarm but repeats for the user’s chosen days. However, unlike the

smart single alarm, when the response is received by the server this alarm sets a single no disable alarm to prevent the system from disabling the smart repeating alarm.

6. **Statistics Alarm:** This alarm is only ever set if the user agrees to provide anonymous information when they install the app. If they do agree, this is set in conjunction with the smart alarms. This alarm activates at the time when the user should be at their destinations. It then checks their location with the Location Service.

The Alarm Receiver class extends the Android broadcast receiver and is activated when an alarm (any type) activates and when the device is turned on. By default, when an Android device is turned off it disables all system alarms. The Alarm Receiver class is responsible for resetting all the active alarms when a device is turned back on. It is also responsible for performing the appropriate alarm that activates. For instance, if a single alarm activates it will notify the Ringtone Service and Notification Helper. If it is a smart alarm that is received, then the receiver sends it to the Network Service to make a request to the server before setting a new alarm.

The Network Service is responsible for making and parsing requests to the server. This is done on a background thread to prevent it from blocking the user interface. It is also highly likely that the user will not be using their device when this request is being made as it happens before the activation time. This means no user interface is need for it.

The Requests class contains methods that build the relevant OkHttp requests for the given parameters. This class is used by the network service.

The Ringtone Service and Notification Helper are both called when an alarm is activated to notify the user. The Ringtone Service plays the devices default alarm tone and the Notification Helper displays a notification that gives the user the ability to disable the alarm and stop the ringtone.

The Location Service is responsible for getting the devices current location using Google Services. Once it gets the location it compares it to where the device should be at that particular time. It creates a statistic from the result of this comparison. This object is then converted to JSON using Google's GSON library and sent to the web application using the Request class.

2.2.3 Architecture Level 1: Web Application

Figure 4 provides a high level view of the architecture of In Time's web service. All requests made to intimealarm.com begin at index.js. This class sets up the express server to listen for requests and send them to their corresponding route class. For instance, if a request is sent to "/api/*" it will be sent to API.js.

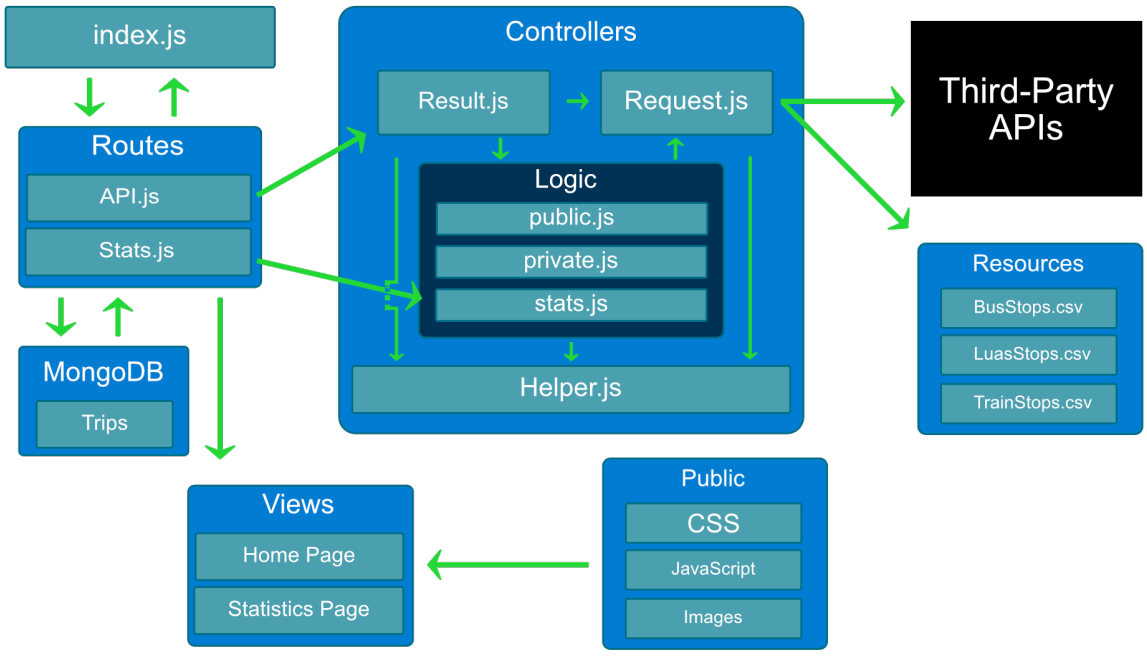


Figure 4: Level 1 Web Architecture

The routes folder contains two classes, API.js and Stats.js. The Stats.js provide routing for requests for the adding and view the statistics of In Time. If a POST request is sent to "/stats", the JSON body is saved in the MongoDB database and the user is redirected to GET "/stats". This GET request makes a request to the MongoDB to retrieve all the trip records.

These records are then sent to the stats.js class, located under the controller's folder. This class contains a method to calculate the accuracy of the trips which is returned to the routing file. This information is then displayed using the view files, with the relevant styling and JavaScript from the Public folder. The view files for this web site are written using Pug, a templating engine for HTML.

The second class in the routes folder, API.js, provides the ability to get the calculated travel time between two destinations, via a GET request to "/api/updatetime/*". It gets the parameters from the request and checks that the departure time is not in the past. If it is a successful request, it returns the result from the Result.js controller. Otherwise, it returns an error message.

The Results.js controller provides two main functions. First it uses the Request.js controller to make a request to the Google Directions API. It then passes the result of this request to the corresponding controller, Public.js for public transport and Private.js for private transport. Private.js checks for error in the response and formats it appropriately. Public.js is a more complex class. It uses the Google response to find out which forms or transport are used during the journey. The supported forms are walking, Dublin Bus, Luas and the Dart. For each stop in the journey the system uses the Request.js controller to get the waiting time if any. If there are no waiting times available, the system defaults to 0 minutes waiting. It then calculates a new waiting time by adding the relative waiting times and travel times together.

The Request.js controller contains all the methods necessary to access both third-party APIs and the static resources that are stored in the resources folder.

The Helper.js controller provides many utility functions that are used in all the other controllers. These include functions such as converting a string duration to a timestamp in milliseconds. E.g. "10 mins" = 60000.

2.3 Implementation

2.3.1 Calculate New Travel Time

Once the web app receives a request for a public journey travel time it makes a request to Google Directions in order to find out what route will be taken to the destination. It then makes requests to RTPI for each step of the journey that requires public transport. The system is then able to use this data to calculate the waiting times for each stop, relative to when the user arrives at them. This can then produce a new total travel time by including all waiting times.

Figure 5 shows the “addWaitingTime” function that performs a crucial part of this process. Before this function is ran, a request has already been made to Google Directions and the “data” object has been populated with important information such as journey steps, departure time and public transport stop numbers.

Lines 86-89 declare and initialize variables that will be used throughout the function. Most notable from these are the “minTilActivation” and “travel_time”. The “travel_time” variable begins at 0 and each waiting time and travel time between steps will be added to it, resulting in the total travel time of the trip. The same process happens to the “minTilActivation” variable, however this starts at the length of time, in minutes, between when the server received the request and the departure time sent by the mobile app. For instance, if the user is meant to depart at 9:30am and a request is made at 9:00am the value of “minTilActivation” would be 30. This is done as the RTPI API produces the arrival times of public transport in real-time and the calculated waiting time needs to be relative to when the user arrives at the stop.

Lines 91-111 preform the actual logic of this function. The function uses the map function provided by the async dependency (McMahon, 2010) This method is used to map each step of the journey to the “getWaitingTime” method which returns the data from RTPI or -1. The results for each are added to a result array and passed to the callback function. This allows the asynchronous requests to be made to RTPI while ensuring that the results are processed in the order of travel.

The callback function loops through each of the objects in the results array and adds the duration of the step to “travel_time” and “minTilActivation”. If a step is public transport a helper function is used to calculate the waiting time, relative to when the user arrives at that stop, and adds it to the “travel_time” and “minTilActivation” variables.

If there have been no errors, then the “data” object is updated with the new travel time. If there has been an error the data is updated via the “setErrorJson” method. The data is then returned to the user by using the “sendResponse” function.

```
85  function addWaitingTime(res, data) {
86      var now = new Date();
87      var minTilActivation = Math.round((data.timeStamp - now.getTime()) / 1000 / 60);
88      var steps = data.steps;
89      var travel_time = 0;
90
91      async.map(steps, getWaitingTime, function(err, results) {
92          if (!err) {
93              for (var i = 0; i < results.length; i++) {
94                  var duration = helper.convertDuration(steps[i].duration) / 1000 / 60
95                  if (results[i] == -1) {
96                      travel_time += duration;
97                      minTilActivation += duration;
98                  } else {
99                      var waiting_time = helper.calcWaitingTime(results[i], minTilActivation);
100                     travel_time += (Number(waiting_time) + duration);
101                     minTilActivation += (Number(waiting_time) + duration);
102                     data.steps[i].transit_details['waiting_time'] = waiting_time + " mins";
103                 }
104             }
105             data = helper.updateDataTime(data, travel_time);
106         } else {
107             console.log(results);
108             data = helper.setErrorJson(err, results);
109         }
110         sendResponse(res, data);
111     });
112 }
```

Figure 5: Add Waiting Time Method

2.3.2 Android Smart Alarm

When a smart alarm activates, it uses the Network Service class to make requests and parse the response from the web application. The code for this can be seen in Figure 6. The Network Service makes an instance of the Network Tread class that implements Runnable to perform its tasks on a background thread. The “run” method of this class, seen on lines 66 -74, extracts the relevant information from the database and creates a request to the web application. When a response is received it is parsed, lines 76-87, using an instance of the JSON Parser class. During the parsing, the status of the request is checked and the arrival time updated if the request was successful. Otherwise, the arrival time is set to -1. Once the arrival time variable is set it is sent to the Alarm Controller class, line 85, to calculate the time for the alarm activation.

```
61      class NetworkThread implements Runnable {
62          Alarm a;
63          NetworkThread(Alarm a) {
64              this.a = a;
65          }
66          public void run() {
67              String to = db.getLoc(a.getTo()).getAddress();
68              String from = db.getLoc(a.getFrom()).getAddress();
69              String departureTime = help.departureTimePrintOut(a);
70              String response = request.getInTimeResponse(a, departureTime, to, from);
71
72              parseResponse(response);
73
74          }
75
76          private void parseResponse(String jsonString){
77              JsonParser parser = new JsonParser();
78              JsonObject jObj = parser.parse(jsonString).getAsJsonObject();
79              String status = jObj.get("status").getAsString();
80              long arrival_time = -1;
81              if (status.equals("OK")){
82                  arrival_time = jObj.get("newTimeStamp").getAsLong();
83              }
84
85              aController.setAlarm(a, arrival_time, type);
86
87          }
88      }
```

Figure 6: Smart Alarm Request

Figure 7 shows the process of setting a new alarm based on the information received from the server. Lines 36-37 shows the method checking if the arrival time has been set to -1. If it has been set than the method skips the logical calculation and sets the alarm according to the original activation time. If the arrival time is not equal to -1, the method compares it to the alarm due time. If the arrival time is greater than the due time it means that the user would arrive late. In this case it gets the difference between the times. This difference is subtracted from the original activation time, resulting in the alarm activating early. As a precaution the system also checks that the new activation time is not older than the current time as this would result in the alarm being set in the past. This process can be seen on lines 45-52. Lines 60-65 set the new alarm and applies the relevant alarm type depending on if it is a single or repeating alarm.

```
35      public void setAlarm(Alarm a, long arrival_time, int type) {
36          if (arrival_time == -1){
37              Log.d(Constants.TAG_ALARM_CONTROLLER, "setAlarm: arrival = -1");
38          } else{
39              Calendar alarmTime = help.getTimeFromString(a.getTime());
40              Calendar originalTime = help.getTimeFromString(a.getDueTime());
41              Calendar arrivalTime = Calendar.getInstance();
42              Calendar now = Calendar.getInstance();
43              arrivalTime.setTimeInMillis(arrival_time);
44
45              if (arrivalTime.getTimeInMillis() > originalTime.getTimeInMillis()){
46                  long timeDiff = arrivalTime.getTimeInMillis() - originalTime.getTimeInMillis();
47                  long newTime = alarmTime.getTimeInMillis() - timeDiff;
48                  if(newTime > now.getTimeInMillis()) {
49                      alarmTime.setTimeInMillis(newTime);
50                      a.setTime(help.calToShortString(alarmTime));
51                  }
52              }
53
54              Log.d(Constants.TAG_ALARM_CONTROLLER, "setAlarm: Due Time: "+ help.timePrintOut(originalTime));
55              Log.d(Constants.TAG_ALARM_CONTROLLER, "setAlarm: Arrival Time: "+ help.timePrintOut(arrivalTime));
56              Log.d(Constants.TAG_ALARM_CONTROLLER, "setAlarm: Alarm Time: "+ help.timePrintOut(alarmTime));
57
58          }
59
60          if (type == Constants.SMART_SINGLE_ALARM){
61              singleAlarm(a, Constants.SINGLE_ALARM);
62          }
63          else if (type == Constants.SMART_REPEATING_ALARM){
64              singleAlarm(a, Constants.SINGLE_NO_DISABLE);
65          }
66
67      }
```

Figure 7: Set New Alarm

2.3.3 Statistics

This system implements the ability to collect anonymous data from users. This data can be used to assess the reliability of the travel predictions by viewing the statistics of previously made trips. In addition, this data could be used in the future to increase accuracy and view any trends in user journeys. In order not to intrude users' privacy, when installing the mobile application, the user is asked if they wish to supply this data. If they choose to supply this data, then every time a smart alarm activates it will enable a statistics alarm. This alarm is set to activate at the user's due time, i.e. when they are meant to be at their desired location. The code for this can be seen in Figure 8. Lines 122 – 132 show the app checking the user's preference of whether they wish to supply statistics and setting the alarm accordingly.

```
118     private void SmartAlarm(Intent intent) {
119         final Alarm a = (Alarm) intent.getSerializableExtra(Constants.EXTRA_ALARM);
120         Log.d(Constants.TAG_ALARM_RECIEVER, "ACTIVATED Smart: "+help.timePrintOut(Calendar.getInstance()));
121
122         new Thread(new Runnable() {
123             @Override
124             public void run() {
125                 SharedPreferences prefs = context.getSharedPreferences(Constants.PREFS_FILE, MODE_PRIVATE);
126                 boolean provideStats = prefs.getBoolean(Constants.SHARED_STATS, false);
127
128                 if (provideStats){
129                     aController.setStatsAlarm(a);
130                 }
131             }
132         }).start();
133
134         Intent start = new Intent(context, NetworkService.class);
135         start.putExtra(Constants.EXTRA_ALARM, a);
136         start.putExtra(Constants.EXTRA_ALARM_TYPE, type);
137         context.startService(start);
138
139     }
```

Figure 8: Alarm Receiver - Smart Alarm

When a statistics alarm activates it starts the Location Service class which gets the devices current location using the Google Service Fused Location Provider. It then compares this location to the destination which the user is traveling to. If they are in within a radius of approx. 200 meters of their location then the journey is seen as a success, otherwise the journey is a failure. The Location Service class then creates a statistic object that contains the to and from location as well as if it is public transport that if the trip was successful. The process of checking this location is seen in Figure 9.

```
98      private void checkLocation(android.location.Location loc) {
99          double deviceLat = loc.getLatitude();
100         double deviceLng = loc.getLongitude();
101
102         double latLower = deviceLat - Constants.RADIUS;
103         double latUpper = deviceLat + Constants.RADIUS;
104         double lngLower = deviceLng - Constants.RADIUS;
105         double lngUpper = deviceLng + Constants.RADIUS;
106
107         boolean hasArrived = false;
108
109         if ((deviceLat >= latLower && deviceLat <= latUpper) && (deviceLng >= lngLower && deviceLng <= lngUpper)){
110             hasArrived = true;
111         }
112
113         Statistic stat = new Statistic(origin.getAddress(), destination.getAddress(), alarm.getPublic(), hasArrived);
114
115         Log.d(Constants.TAG_L_SERVICE, "Sending Request");
116         new Requests().sendAutomatedStatistic(stat);
117
118     }
```

Figure 9: Check Location

Figure 10 illustrates the application using OkHttp to send a post request to the web application containing the newly created statistic object, in JSON format. This process runs on a background thread in order to prevent the request from blocking the user's UI thread.

```
100     public void sendAutomatedStatistic(final Statistic s){
101         new Thread(new Runnable() {
102             @Override
103             public void run() {
104                 OkHttpClient client = new OkHttpClient();
105                 MediaType mediaType = MediaType.parse("application/json");
106                 RequestBody body = RequestBody.create(mediaType, new Gson().toJson(s));
107 
108                 HttpUrl url = new HttpUrl.Builder()
109                         .scheme("https")
110                         .host("intimealarm.com")
111                         .addPathSegment("stats")
112                         .build();
113 
114                 Request request = new Request.Builder()
115                         .url(url)
116                         .post(body)
117                         .addHeader("content-type", "application/json")
118                         .build();
119 
120                 try {
121                     client.newCall(request).execute();
122                 } catch (IOException e) {
123                     e.printStackTrace();
124                 }
125             }
126         }).start();
127     }
```

Figure 10: OkHttp Statistics POST

2.3.4 Web Application Build Process

In order to increase the efficiency of developing features for this application Gulp was incorporated to automate the build process. It allows many mundane tasks to be ran automatically using a single command. The tasks that this project uses Gulp for are as follows:

- Minifying front end JavaScript files.
- Compiling SCSS and LESS files.
- Watching files to recompile/minify if any changes are made.
- Running Nodemon when developing to restart the local server automatically after changes are made.

These functions can be run individually by using the “gulp” command followed by their respective method name, or they can all be ran using “gulp” on its own.

2.3.5 Host Environment Setup

In Time is hosted on a Virtual Private Server (VPS) provided by Digital Ocean. This VPS is using Ubuntu 14.04 as its Operating System and Nginx as its web server. NodeJS, NPM, Git, Bower and PM2 were installed manage different aspects of the deployment.

- NodeJs is used to run the web application code on the server.
- NPM is used to manage the NodeJs packages.
- Git is used to pull the latest code changes onto the server.
- Bower is used to manage front end libraries used by the application.
- PM2 is a process manager for NodeJs that allows the application to always remain running.

The web application is HTTPS enabled using a SSL certificate from Let’s Encrypt (2017). A Let’s Encrypt Certificate is only ever valid for 90 days; however, they can be renewed using their provided Certbot Client. This client is set to run on a chron job to ensure the certificate always stays in date.

2.3.6 Application Programming Interfaces (API)

This application uses several third party APIs as well as its own custom API.

2.3.6.1 Third Party APIs

- **Google Places API** – This is used to give a user the ability to search for locations. These locations are then being saved and used by the smart alarm.
- **Google Maps Directions API** – This is used by the web service to calculate the travel time for the journey between the start and end location.
- **RTPI API** – This is used in conjunction with the directions API to accurately calculate the travel time when using public transport in Dublin.

2.3.6.2 Custom API

This mobile application required the development of a custom API. The main purpose for this API is for updating an alarm call, the app will make a request to the API, which in turn will make requests to third party APIs. All the logic to calculate a new alarm time is performed by the web service and the result returned to the device in JSON format. This means that the mobile device will save on network activity by only making a single request. It is also responsible for communication with the MongoDB as well as to view the front end elements of the web application. The web application is hosted at intimealarm.com and has several URL routes listed in Tables 1 to 7.

Table 1: Route - In Time

API Name	In Time
Description:	This route returns the web apps homepage.
URI:	/
HTTP verb	GET
Parameters:	n/a
Resource Contents:	Index (HTML)
Pre-Conditions:	n/a
Post-Conditions:	Homepage is displayed

Table 2: Route - In Time Statistics

API Name	In Time Statistics
Description:	This route returns the web apps statistics page.
URI:	/stats
HTTP verb	GET
Parameters:	n/a
Resource Contents:	Statistics (HTML)
Pre-Conditions:	n/a
Post-Conditions:	Statistics is displayed to the user.

Table 3: Route - Insert In Time Statistics

API Name	Insert In Time Statistics
Description:	This route inserts the anonymous data gathered from the user into the database.
URI:	/stats
HTTP verb	POST
Parameters:	n/a

POST Body	{
	arrivedOnTime: Boolean,
	from: String,
	isPublic: Boolean,
	to: String
	}

Resource Contents:	Redirects to Statistics Page
Pre-Conditions:	n/a
Post-Conditions:	An new Trip object is inserted into the database.

Table 4: Route - In Time Evaluation Statistics (Accuracy)

API Name	In Time Evaluation Statistics (Accuracy)
Description:	This route is used when a user is redirected by nginx after making a POST to /eval over HTTP
URI:	/stats/eval
HTTP verb	GET
Parameters:	n/a
Resource Contents:	Redirects to Statistics Page
Pre-Conditions:	n/a
Post-Conditions:	Statistics is displayed to the user.

Table 5: Route - Insert In Time Evaluation Statistics (Accuracy)

API Name	Inset In Time Evaluation Statistics (Accuracy)
Description:	This route inserts a trip accuracy object into the database
URI:	/stats/eval
HTTP verb	POST
Parameters:	n/a

POST Body	{
	actual: String,
	estimated: String,
	from: String,
	isPublic: Boolean,
	time: String,
	to: String
	}

Resource Contents:	Redirects to Statistics Page
Pre-Conditions:	n/a
Post-Conditions:	An new trip accuracy object is inserted into the database.

Table 6: Route - Get In Time Statistics (ajax)

API Name	Get In Time Statistics (ajax)
Description:	This route returns the web apps statistics that is automatically collected from if they agree to provide anonymous data.
URI:	/stats/ajax
HTTP verb	GET
Parameters:	to (String, Optional), from(String, Optional)
Resource Contents:	A JSON array of all the trip objects. If the 'to' and/or 'from' parameters are specified it returns only objects where there is a full or partial match with their respective fields. The format of the response is as follows: [{ _id: String arrivedOnTime: Boolean, from: String, isPublic: Boolean, to: String },…]
Pre-Conditions:	There are records that have a full and/or partial match to the 'to' and/or 'from' parameters
Post-Conditions:	A JSON array of trip objects is returned to the user.

Table 7: Route - Update Alarm

API Name	Update Alarm
Description:	This route is used to get the calculated length of time that a trip should take.
URI:	/api/updatetime/api/:transport (Public or Private)
HTTP verb	GET
Parameters:	time (String, Required), to (String, Required), from(String, Required), debug (Boolean, Optional), mock (Boolean, Optional), tmodes (JSON String Array, Optional)
Resource Contents:	If debug is true the raw Google Directions result is returned. If mock is true the calculations are performed on a mocked

	Google Directions result.
	A JSON object of the calculations, with the format:
	{
	status: String
	origin: {
	address: String
	lat: Double,
	lng: Double
	},
	destination: {
	address: String
	lat: Double,
	lng: Double
	},
	distance: String
	duration: String
	calculated_duration: String
	original_time: String
	timeStamp: Long,
	approx_arrival: String
	newTimeStamp: Long,
	type: String
	}
	if it is public transport it also contains an array containing information about each step of the journey.
	If there are any errors the follow in returned:
	{
	status: String,
	error_code: Integer,
	external_status: String,
	message: String
	}

Pre-Conditions:	n/a
Post-Conditions:	A JSON object is returned to the user.

2.4 Testing

Automated tests were written and ran on both the mobile application and the web application. These tests ensure that the functionality that is crucial to the system, is working as required.

2.4.1 Web Application Testing

Testing of the web service consisted of automated tests written using Mocha and Supertest. There is a total of 125 tests ran and a report printed to show passed and failed cases. Tables 1 and 2 show the summaries of the tests that ran on 17th April 2017.

There are two type of tests run on the web application, API tests and unit tests. API tests ran on the services custom API to ensure it is returning a correct response. These work by making a series of requests to a selection of different location and checking the response. The tests check the response for both valid and invalid inputs to ensure that the system will respond appropriately to all inputs. Table 8 shows the summary of these tests. The results show that all tests were successful.

Table 8: Web API Test Summary 17/04/2017

TEST	RESULT (#Pass / **#Total**)
Respond 200	29 / **29**
Respond with JSON	28 / **28**
Respond with status: ERROR	14 / **14**
Respond with status: Ok	14 / **14**
Return a New Timestamp	14 / **14**
Return an Error Code	14 / **14**
TOTAL	113 / **113**

The Unit tests are run on the main helper functions to verify that they are producing the correct result for the given input. For each of the main functions, 4 tests are run to validate its acts appropriately to both expected inputs and their respective edge cases. The summary of these can be seen in Table 9. The results show that all tests were successful.

Table 9: Web Unit Test Summary 17/04/2017

TEST	RESULT (#Pass / **#Total**)
Convert Duration String to Timestamp	4 / **4**
Add Two Timestamps Together	4 / **4**
Drop the Decimals of a Float	4 / **4**
TOTAL	12 / **12**

2.4.2 Android Application Testing

Testing of the Android application was done using JUnit and Mockito. These unit tests were written to test all of the methods in the Helper java class. This class contains methods which are used repeatedly throughout the application and it is imperative that they perform as designed. There are a total of 15 tests, all of which passed successfully. These tests are listed in Table 10.

Table 10: Android Unit Tests 02/05/2017

TEST	RESULT (PASS or FAIL)
Get Hour from Time String	PASS
Get Minute from Time String	PASS
Array to String	PASS
String to Array	PASS

Get Active Days	PASS
Integer ArrayList to String	PASS
Calendar to Short Time String	PASS
Time Printout to Calendar	PASS
Calendar to Time Printout	PASS
Alarm to Time Printout	PASS
Add Leading Zero (True)	PASS
Add Leading Zero (False)	PASS
Check Fields are Completed (True)	PASS
Check Fields are Completed (False)	PASS
Calendar From Short Time String	PASS

2.5 Graphical User Interface (GUI)

There has been GUIs developed for both the web and mobile application. As mentioned previously the web application does not require an elaborate GUI as it is mainly designed as a back end service. However, there are still several reasons why a front end is necessary. One reason is to provide the functionality for the user to be able to visualize the anonymous data collected. Figure 11 shows the statistics page where the user can view the data. The large pie chart in the center shows the overall ratio between trips that arrived on time and trips which did not. This can be filtered using the search bars above which allows one to view the ratio between two specific destinations. These search bars are enabled with auto-complete/suggestions functionality so the user can view the locations which have had data collected. The graph to the right depicts the accuracy of trips in regards to actual travel time versus estimated travel time. This is discussed further in the evaluation section of this report.

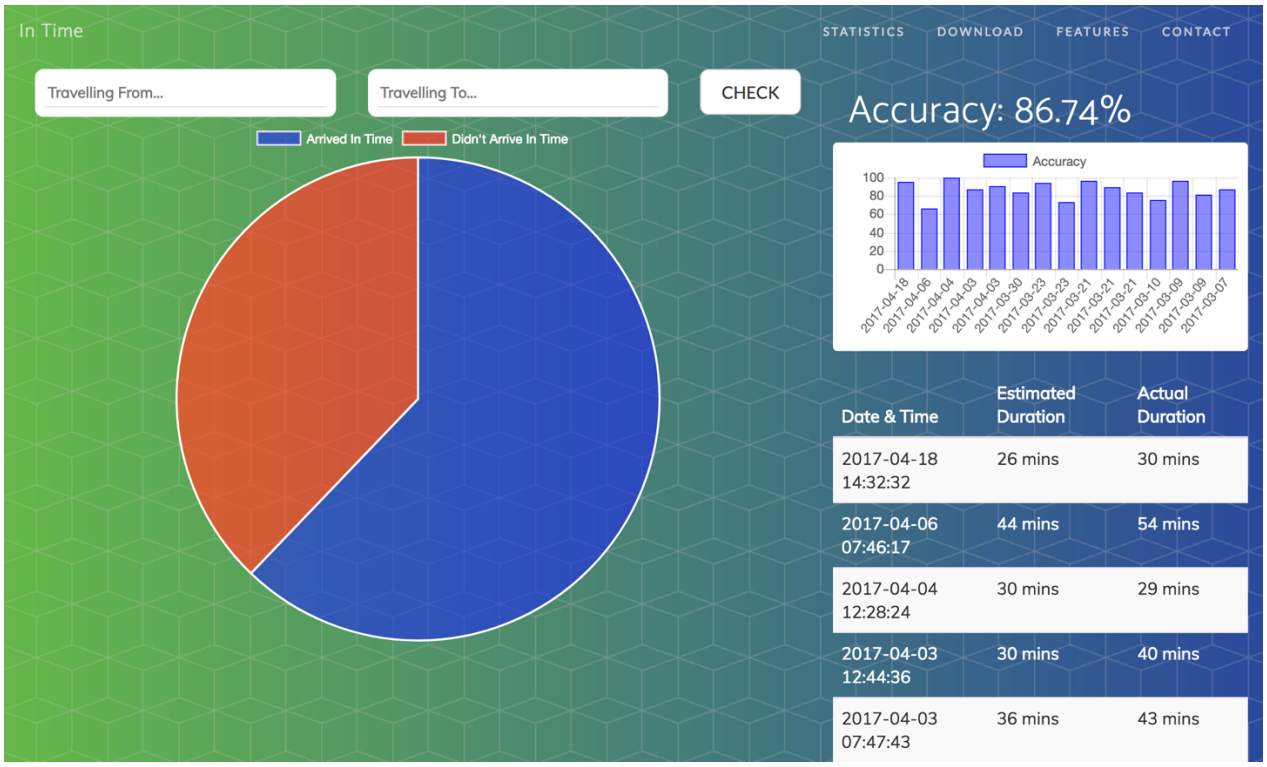


Figure 11: Statistics Page (As of: 02/05/2017)

The Android application is the main point of contact between the user and the system and requires a more interactive GUI than the web application. When designing the user interface (UI) for the mobile app Clifton (2013), it was used as a point of reference for best practices along with the Android Developer's website (Android Developers, 2017).

Figure 12 shows the home screen of the application. It provides a quick and easy way to view all alarms and the main information about them such as their activation time, destinations and whether they are active or not. This screen will be the main screen that the user interacts with. This screen provides the ability for the user to interact with the alarms by means of sliding to delete, clicking to edit and clicking power image to enable/disable. It also provides access to other areas of the application with the floating action button (FAB) for adding alarms and a button to the Locations Activity in the collapsed menu at the top.

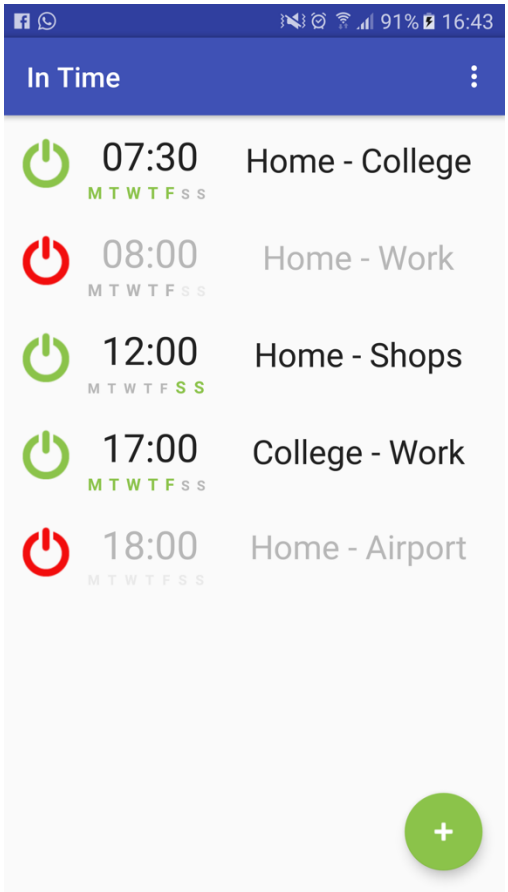


Figure 12: Home Screen

Figure 13 shows the screen that appears when you click on the add alarm FAB. This screen uses the default Android time picker dialog to allow the user to set a time using a 24-hour clock. The default time picker was chosen as it should provide a standard experience that the user is already familiar with.

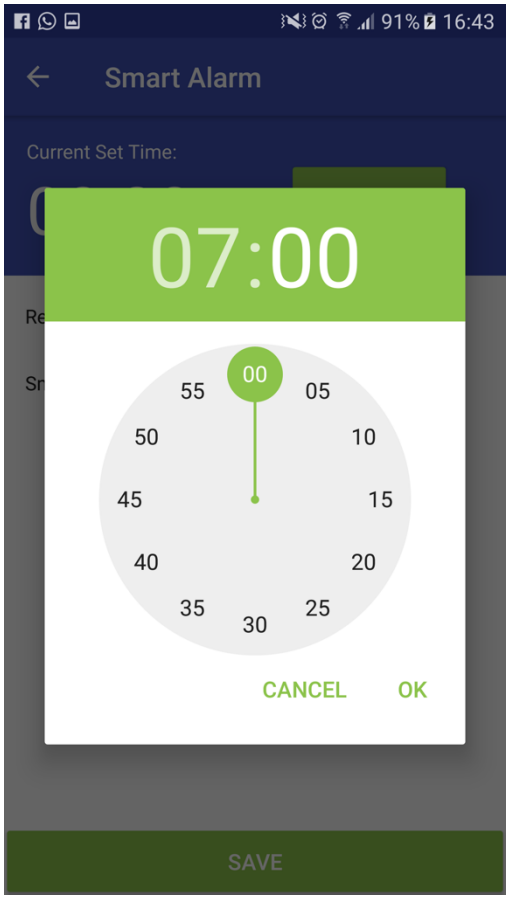


Figure 13: Add Alarm

When a time has been chosen, the user is brought to the smart alarm screen, Figure 14. This screen provides the user with extra options such as setting the alarm to repeat on a particular day and making it smart. Other options that can be set are the desired transport type and the locations traveling to and from. This screen is also used when editing an alarm, the difference being that all the information is prefilled out with the alarms current settings.

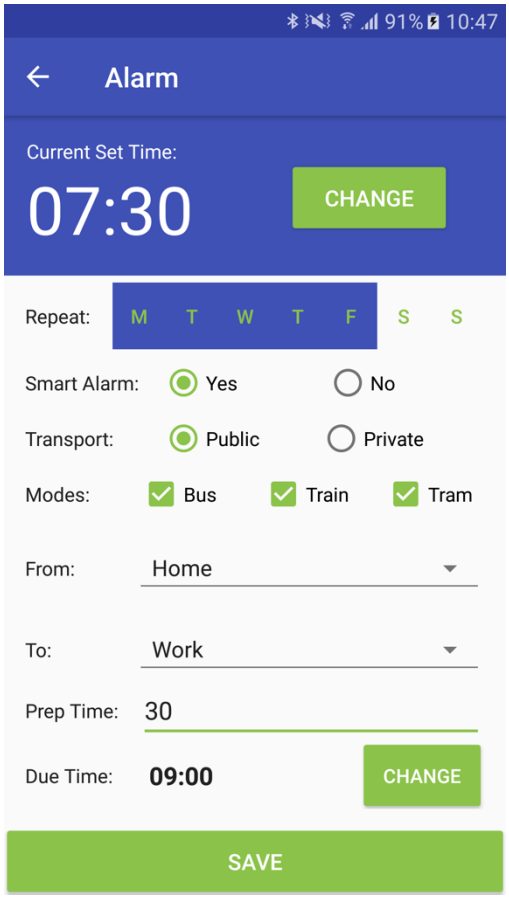


Figure 14: Add Smart Alarm

Figure 15 shows the lock screen notification presented to the user when an alarm has activated. When an alarm activates it plays the phones alarm notification ringtone and displays a notification to the user. Upon clicking the “Stop Alarm” button on the notification, the ringtone is stopped and the alarm is disabled (unless it is a repeating alarm). If the device is not locked when the alarm activates, then the notification will be located in the notification drawer at the top of the devices screen.

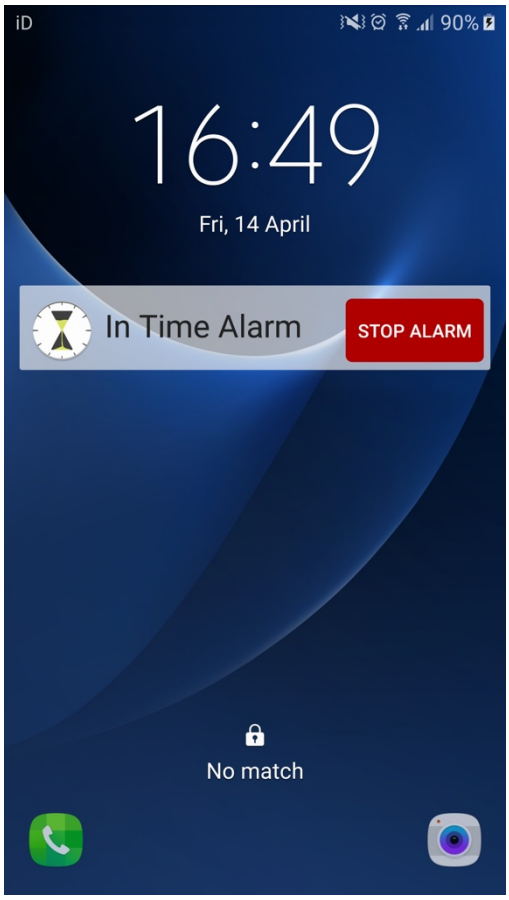


Figure 15: Stop Alarm

2.6 Customer Testing

In order to get an understanding of the usability of this system, ten potential users were given various use cases to perform along with a short survey to answer. Their actions during these use cases were recorded and templates of these documents can be found in the appendix. The use cases that were tested were adding a basic alarm, adding a smart alarm and adding a new location. These use cases were chosen as they are likely to be the most regularly used by all users.

2.6.1 Use Case 1: Add Basic Alarm

Of the total participants 80% of them successfully created a basic alarm with the required parameters. 70% of the total done this without any difficulty while 10% were unsure as to how to correctly use the time picker. This resulted in them clicking back and having to reopen the time picker to reset it.

The remaining 20% set alarms with the incorrect parameters. Most notably from these mistakes were the 10% who assumed the repeating days' button started on Sunday. This caused them to choose Tuesday through Saturday as the week days. This issue could be solved by providing an option of what day the user would like the week to start on.

2.6.2 Use Case 2: Add Smart Alarm

When creating a Smart Alarm 60% of the participants successfully created it without error in the parameters. However, during the process 20% tried to click on the due time text to change it, instead of the provided button. This suggests that the due time should be able to be edited inline. A solution to this issue was implemented by making the text clickable, meaning it acts the same as when the user clicks the designated button.

The other 40% had a mixture of small errors including not setting the alarm to repeat or not setting the due time. This could be caused human error or potentially the user being overpowered by options. The latter of these however,

could be disregarded from the findings of the survey discussed under section 2.6.4.

2.6.3 Use Case 3: Add Location

This use case had the lowest percentage of users performing it in the desired way at only 30%. The remaining 70% added a location but in the process also created a new alarm. It is important to mention that the order in which these test ran could have affected this outcome. This could be because during the other use cases the user seen the “<Add Location>” option, for the smart alarm, in the dropdown menus and assumed this was the only way, when in fact this is a secondary way to add a location. This resulted in them not checking the collapsed options menu at the top of the Main Activity, which contained a link to a designated Locations Activity. If the tests where ran in a different order it could have altered the findings of this test.

A solution to this issue was put in place by only collapsing the menu if there is no room on the screen to show the locations button. This means that on a standard sized phone the button to the Location Activity will always be shown.

2.6.4 Survey

The survey consisted of four questions that assessed the user’s opinion of the mobile and web app’s design. The first two of which were used to get a general opinion as to whether the user liked the overall design. Figure 16 illustrates the general opinion that users have of both the mobile and web app. Both applications have a peak rating of 5, showing that the users have a strong liking towards the design of the system.

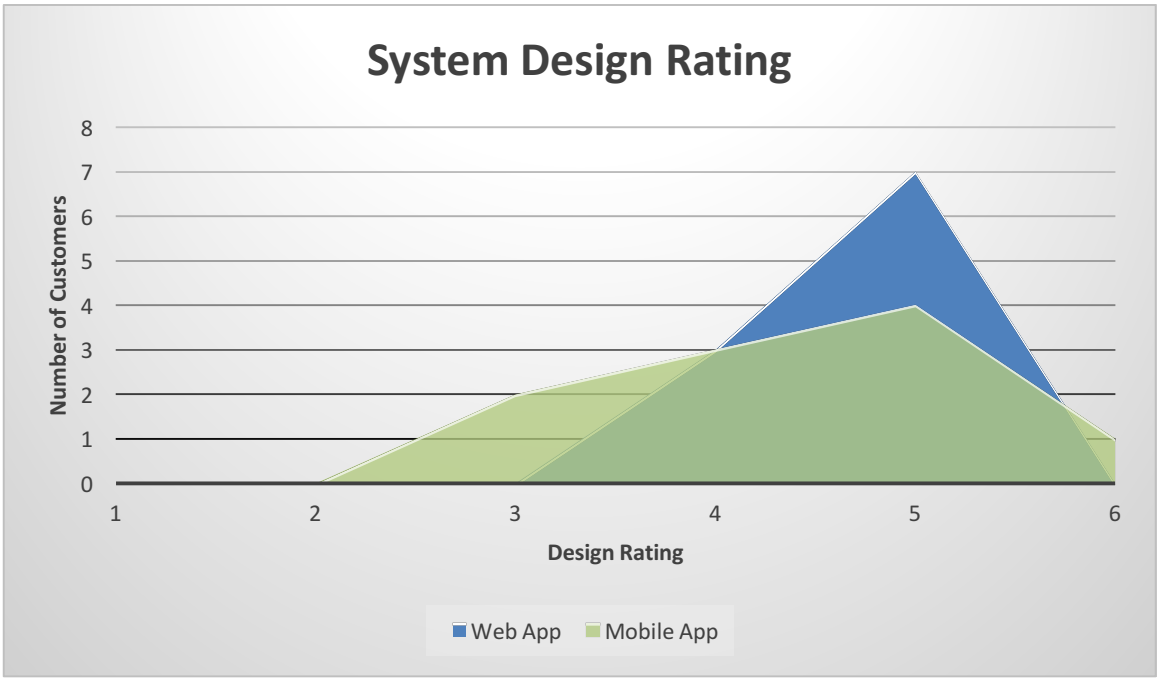


Figure 16: Survey Ratings Graph

When asking the users what their favorite design aspect of the app was, 40% replied with its simplicity and ease of use. This feedback confirmed that we successfully implemented the non-functional requirement of usability.

When asking for their least favorite design aspect, 40% replied with a dislike toward the android time picker, 30% being specifically because it was a 24-hour time picker. This design issue could be resolved by adding in an option to choose between 24/12-hour time formats.

2.7 Evaluation

The system is evaluated based on the accuracy of its estimated travel time. This evaluation was achieved by creating an evaluation activity on the application. The application made a request to the server before making a trip, between any two locations, and then compared the estimated travel time to the actual travel time once the journey was made. The accuracy is represented as a percentage of how close the actual travel time was to the estimated time. For instance, if the actual travel time is 20 minutes with an accuracy of 75% the estimated time will be either 15 or 25 minutes. At the time of writing, the accuracy of the system was 86.74%. Figure 17 shows the breakdown of this accuracy to each individual trip, each bar in the chart represents one trip. The y axis being accuracy percentage and the x being the trip date.

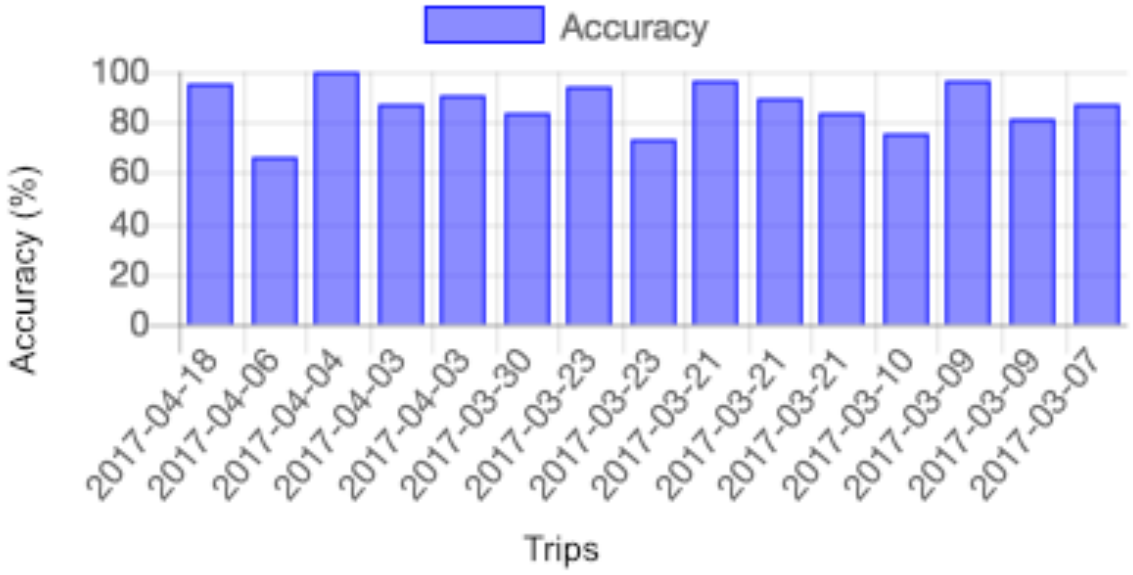


Figure 17: Evaluation Graph

3 Conclusions

This system was designed to solve the common issue of lateness from traffic that affects approximately 46,000 Irish people every day. This means there is potentially a larger target market for this product, as it is unlikely that it will be the same people consistently late every day. The main functionality of this system is to allow a user to set a smart alarm. This smart alarm uses current and predicted traffic conditions to calculate a travel time and wake the user early if s/he would otherwise be late. In addition to this, the system allows to user to view statistics on the reliability of the previously made trips between any two locations.

To provide these users with a sense of reliability and correctness the system was tested extensively to ensure that it functions correctly. A total of 140 automated tests are ran on the system and they can quickly identify any issues that could arise. These tests are run each time a change is made to the system to ensure that it is always working correctly before the update is pushed to GitHub and the Digital Ocean VPN.

Furthermore, the system was evaluated to assess accuracy. The results show that the application has an accuracy of 87% meaning that the travel time estimated will be within 13% of the actual travel time. This provides the user with peace of mind knowing that the system will produce an accurate travel time result and get them to work/college on time.

To conclude, In Time has the potential to reduce lateness by providing an accurate and reliable alarm that will wake up an user early if there are any heavy or unforeseen traffic circumstances.

4 Further development or research

There are many ways that this project could evolve, I will split the possible evolutions into two categories, Web and App.

4.1 Mobile Application

This application could be evolved by adding more intelligent functionalities to it. One of these features could be expanding internationally to support public transport in other countries. The system could also integrate Uber or other taxi services to possible provide pricing of the journey.

Another function that could be added is puzzles to disable alarm upon activation. These Puzzles could be similar to those seen in Puzzle Alarm Clock (Wro Claw Studio, 2016). If a user has multiple devices, they may wish to be able to sync alarms across them. Advertisements could be added in to monetize the application.

4.2 Web Application

One possible way that the web application could evolve is to expand internationally to allow public transport to be supported in various countries. Some other ways are by adding additional APIs to continuously improve travel time predictions, adding functionality to return more detailed information about travel plan, such as exact route and time between stops. Furthermore, this could link in to be used with maps for directions to the location and not just alarm predictions. Finally, the API could be opened up to the public and charge a fee for its usage by third parties.

5 References

Android Developers. (2017). *Android Developers*. [online] Available at: https://developer.android.com/index.html [Accessed 2 Jan. 2017].

Bower.io. (2012). Bower. [online] Available at: https://bower.io/ [Accessed 5 May 2017].

Brooks, C. (2015). *The Real Reasons Employees Are Late for Work*. [online] Business News Daily. Available at: http://www.businessnewsdaily.com/7725-employees-late-for-work.html [Accessed 8 Dec. 2016].

Clifton, I. (2013). *Android user interface design*. 1st ed. Upper Saddle River, New Jersey: Pearson Education, Inc.

Data.gov.ie. (2014). *Real-time Passenger Information (RTPI) for Dublin Bus, Bus Eireann, Luas and Irish rail*. [online] Available at: https://data.gov.ie/dataset/real-time-passenger-information-rtpi-for-dublin-bus-bus-eireann-luas-and-irish-rail [Accessed 10 Jan. 2017].

Express. (2016). Express - Node.js web application framework. [online] Available at: http://expressjs.com/ [Accessed 5 May 2017].

GitHub. (2004). request/request. [online] Available at: https://github.com/request/request [Accessed 5 May 2017].

GitHub. (2008). google/gson. [online] Available at: https://github.com/google/gson [Accessed 5 May 2017].

GitHub. (n.d.). pugjs/pug. [online] Available at: https://github.com/pugjs/pug [Accessed 5 May 2017].

Google Developers. (n.d.). *Google Maps Directions API | Google Developers*. [online] Available at: https://developers.google.com/maps/documentation/directions/ [Accessed 10 Jan. 2017].

Gulp. (2013). gulp.js. [online] Available at: http://gulpjs.com/ [Accessed 5 May 2017].

Holowaychuk, T. (2014). visionmedia/Supertest. [online] GitHub. Available at: https://github.com/visionmedia/supertest [Accessed 5 May 2017].

IDC (2016). *IDC: Smartphone OS Market Share*. [online] Available at: https://www.idc.com/prodserv/smartphone-os-market-share.jsp [Accessed 8 Dec. 2016].

Jyo, R. (2011). Yaml-config. [online] npm. Available at: https://www.npmjs.com/package/yaml-config [Accessed 5 May 2017].

Let's Encrypt. (2017). *Let's Encrypt - Free SSL/TLS Certificates*. [online] Available at: https://letsencrypt.org [Accessed 4 May 2017].

McMahon, C. (2010). *async*. [online] Caolan.github.io. Available at: http://caolan.github.io/async/ [Accessed 5 May 2017].

Mercer, D. (2012). *Staff lateness 'costs the economy £9 billion every year'*. [online] The Independent. Available at: http://www.independent.co.uk/news/business/news/staff-lateness-costs-the-economy-9-billion-every-year-8191289.html [Accessed 8 Dec. 2016].

Mocha. (2017). Mocha - the fun, simple, flexible JavaScript test framework. [online] Available at: https://mochajs.org/ [Accessed 5 May 2017].

Moment. (n.d.). Moment.js | Home. [online] Available at: https://momentjs.com/ [Accessed 5 May 2017].

MongoDB. (2017). MongoDB for GIANT Ideas. [online] Available at: https://www.mongodb.com/ [Accessed 5 May 2017].

Never Late Smart Alarm. (2015). Never Late Smart Alarm. [online] Available at: https://play.google.com/store/apps/details?id=com.project.neverlate&hl=en [Accessed 2 Oct. 2016].

Sommerville, I. (2016). *Software engineering*. 1st ed. Harlow: Pearson Education.

Square.github.io. (2016). OkHttp. [online] Available at: http://square.github.io/okhttp/ [Accessed 5 May 2017].

Wharton, J. (2013). Butter Knife. [online] Jakewharton.github.io. Available at: http://jakewharton.github.io/butterknife/ [Accessed 5 May 2017].

Wro Claw Studio. (2016). Puzzle Alarm Clock. [online] Available at: http://wroclawstudio.com/ [Accessed 2 Oct. 2016].

wUp. (2015). wUp. [online] Available at: http://wupapp.com/ [Accessed 2 Oct. 2016].

6 Appendix

6.1 Project Proposal

6.1.1 Objectives

The main objective of the project is to develop a mobile application that will help commuters arrive at work/college on time. This application will be essentially a smart alarm that extends the features of the stock android alarm. It will use current, historical and predicted traffic conditions to predict commuting time and change the time of the alarm accordingly.

The target market for the app is anyone living in Dublin and travels to work and/or college. Although it is aimed to be used in Dublin it will be designed to work internationally for private travel, such as car, but may have limitation for public commuting.

The project will consist of two main parts, a mobile application and a web based application. The mobile application will be used to set any alarms and locations, while the web application will perform all the logic for traffic predictions and expose the results in the form of a RESTful API.

6.1.2 Background

There are three of reasons why I want to develop this application.

1. I am a very punctual person and tend to be on time, if not early. During workplace I noticed that certain days of the week or months of the year seemed to cause differences in travel conditions, this often made it hard to predict what time to leave at. Another factor, that was even less predictable, was car crashes. Depending where these happened they seemed to cause a near standstill in traffic.
2. This project gives me the chance to develop upon my existing Android development knowledge, which currently are only at a basic to intermediate level. I have previously worked with Android application

during my second year project; and since have developed an interest in the area. Recently, I have been researching about mobile development and look forward to applying the knowledge that I have gained.

3. The complexity of this project gives me a chance to apply the web development knowledge that I gained during my work placement. Although I do not plan on using the same web stack that I worked with in AOL, I still have knowledge of concepts and good practices that I will apply while also learning a new technology.

6.1.3 Technical Approach

To approach this from a technical standpoint I have split it into two main sections, the web application and the mobile application.

Web Application

- Receive request from the mobile application with all the required parameter. E.g. fromLocation, toLocation, alarmTime, etc.
- Send requests to several APIs including Google Maps and RTPI (Real-Time Passenger Information).
- Perform logical operations to combine the results from the APIs and calculate a new time for the alarm to go off.
- Expose all relevant information to the android app, in the form of a RESTful API.

Mobile Application

- Provides the user with the CRUD functions for the alarms.
- Allows the users to create both normal alarms and smart traffic based alarms.
- Sends request to the web application to retrieve data about updating the alarms.
- Incorporates Google Maps to help with location searching and selection.
- Uses system notifications to display useful information to the user.

6.1.4 Special Resources Required

Software

- Android Studio IDE
- Atom Text Editor
- Various Java and Node.js Libraries

Hardware

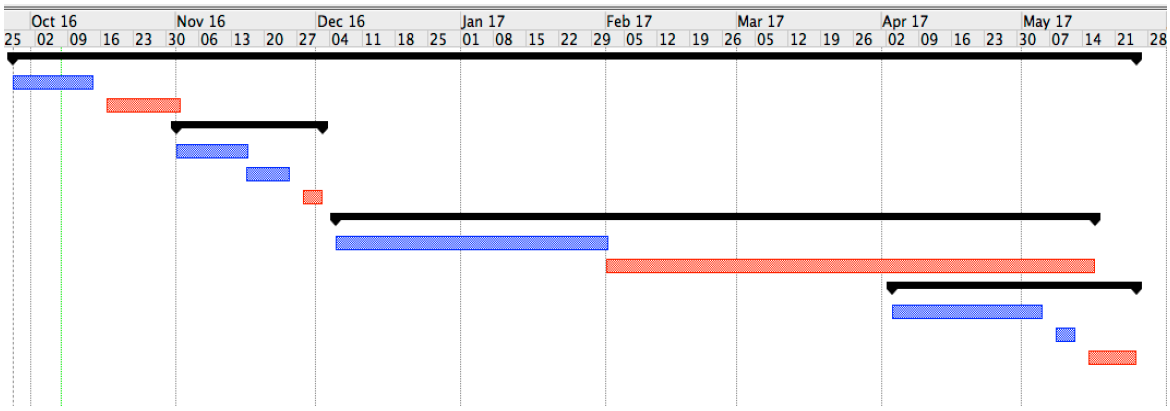
- Web Server (Digital Ocean)
- Android Devices

Other

- Google Play Developer license

6.1.5 Project Plan

⊟Software Project	**173 days?**	**9/27/16 8:00 AM**	**5/25/17 5:00 PM**
Project Proposal	14 days?	9/27/16 8:00 AM	10/14/16 5:00 PM
Requirements Spec	12 days?	10/17/16 9:00 AM	11/2/16 9:00 AM
⊟Project Prototype	**23.875 days?**	**11/1/16 9:00 AM**	**12/2/16 5:00 PM**
Web Proxy Server Prototype	11.875 days?	11/1/16 9:00 AM	11/16/16 5:00 PM
Android Application Prototype	7.875 days?	11/16/16 9:00 AM	11/25/16 5:00 PM
Presentation Prep	4.875 days?	11/28/16 9:00 AM	12/2/16 5:00 PM
⊟Project Development	**116.875 days?**	**12/5/16 9:00 AM**	**5/16/17 5:00 PM**
Web Proxy Server	42.875 days?	12/5/16 9:00 AM	2/1/17 5:00 PM
Android Application	74.875 days?	2/1/17 9:00 AM	5/16/17 5:00 PM
⊟Final Report	**39 days?**	**4/3/17 8:00 AM**	**5/25/17 5:00 PM**
Report	25 days?	4/3/17 8:00 AM	5/5/17 5:00 PM
Show Case Prep	5 days?	5/8/17 8:00 AM	5/12/17 5:00 PM
Presentation Prep	9 days?	5/15/17 8:00 AM	5/25/17 5:00 PM



6.1.6 Technical Details

I plan on using various technologies to create this project. I will list them below but these are subject to change when doing further research and requirements gathering.

Android Application

- Base Android Stack (Gradle, Java, xml, etc.)
- Butter Knife, Android Java library for view Injections

Web Application

- Express, a Node.js web application framework

Testing

- JUnit, Java testing framework
- Mockito, a mocking framework for unit tests in Java
- Mocha, a Behavior-Driven testing framework
- Supertest, Express server testing framework.

APIs

- Google Maps
- RTPI (Real-Time Passenger Information)

6.1.7 Evaluation

To evaluate this project, I intend to write a series of tests throughout the development process. These tests will be written in Mocha and SuperTest.js for the web application and JUnit and Mockito for the android application. They will test that the functions perform as intended and without error.

I also plan to implement a manual testing strategy, to evaluate the accuracy of the application. This will be done by giving friends and family a beta version of my application to use. I will then be able to use this information to analyze predicted vs actual journey

6.2 Project Plan

⊟Software Project	173 days?	**27/09/16 08:00**	**25/05/17 17:00**
Project Proposal	14 days?	27/09/16 08:00	14/10/16 17:00
Requirements Spec	12 days?	17/10/16 09:00	02/11/16 09:00
⊟Project Prototype	12.875 days?	**16/11/16 09:00**	**02/12/16 17:00**
Web Server Prototype	11.875 days?	16/11/16 09:00	01/12/16 17:00
Android Application Prototype	7.875 days?	16/11/16 09:00	25/11/16 17:00
Presentation Prep	4.875 days?	28/11/16 09:00	02/12/16 17:00
⊟Project Development	37.875 days?	**10/02/17 09:00**	**04/04/17 17:00**
⊟Mobile Application	27.875 days?	**10/02/17 09:00**	**21/03/17 17:00**
Ringtone Service	3.875 days?	10/02/17 09:00	15/02/17 17:00
⊟Smart Alarm	8.875 days?	**15/02/17 09:00**	**27/02/17 17:00**
Add location to Alarm	1 day?	15/02/17 09:00	16/02/17 09:00
Update Add Alarm Activity	2.875 days?	16/02/17 09:00	20/02/17 17:00
Network Call	2.875 days?	20/02/17 09:00	22/02/17 17:00
Process Result	3.875 days?	22/02/17 09:00	27/02/17 17:00
Set Alarm on Boot	1.875 days?	27/02/17 09:00	28/02/17 17:00
Beta mode Testing	5 days?	28/02/17 09:00	07/03/17 09:00
Clean Up UI	5.875 days?	07/03/17 09:00	14/03/17 17:00
Unit Testing	5.875 days?	14/03/17 09:00	21/03/17 17:00
⊟Web Server	10.875 days?	**21/03/17 09:00**	**04/04/17 17:00**
Luas Info	1 day?	21/03/17 09:00	22/03/17 09:00
Train Info	2.875 days?	22/03/17 09:00	24/03/17 17:00
Unit Testing	4.875 days?	24/03/17 09:00	30/03/17 17:00
Stats Front End	3 days?	31/03/17 08:00	04/04/17 17:00
⊟Technical Report	38 days?	**04/04/17 08:00**	**25/05/17 17:00**
Final Report	21 days?	04/04/17 08:00	02/05/17 17:00
Presentation Prep	8 days?	10/05/17 08:00	19/05/17 17:00
Showcase	4 days?	22/05/17 08:00	25/05/17 17:00

6.3 Monthly Journals

6.3.1 September

My Achievements

This month I began to research my idea and develop upon my initial plans. For context my idea is a ‘smart’ alarm, called “In Time”, that adjusts depending on traffic conditions. I researched into how alarms work on the android platform and how I would go about implementing the Google maps directions and RTPI APIs.

I have also researched into NodeJS and plan on using the express framework to set up a server to proxy my API calls through. This will be done for three reasons:

1. Hide my API keys from the users’ device.
2. Reduce network activity on the device.
3. Preform majority of the logic on the server to make it easier if there ever needs to be changes made.

I also had my project pitch which went well so I began to write my project proposal document to explain in detail what I intend to develop for my project.

My Reflection

I am happy with the research I have done and have purchased a few books about android development including Android User Interface Design by Ian G. Clifton and Android Development Patterns by Phil Dutson.

I do however feel that I have spent too long in deciding if the project is a good enough idea and could have spent that time in more productive research to develop it, instead of doubting it.

Intended Changes

Next month I will set my first meeting with my supervisor (who has not been assigned yet). I will hopefully finish my project proposal document and begin working on the project requirements. I also intend on creating the initial structure for the project such as a GitHub repo.

6.3.2 October

My Achievements

This month I finished my project proposal and began to work on my requirements documentation. I have done more research for my project and have the bulk of the requirements document completed. The sections I have left to complete are evolution and system architecture. I have a bit more research to do in order to create a system architecture diagram.

I also made progress on my web service. I have set up a GitHub repo and a simple file structure for it. I used NodeJS with various npm packages such as express.js. I also added in the Frisby.js test framework to do some API testing. This framework is based of Jasmine (which I also intend on using). I have also purchased the domain InTimeAlarm.com, which will be used to host the service.

As well as the GitHub repo for the web service I also set up a repo for the Android app. This has been initiated with just the basic Android app skeleton that is built for you by Android Studio. These repos are called InTime_Web and InTime_App.

My Reflection

I am happy with the progress I have made this month and feel that I am ahead of schedule in my project plan. I have also scanned through some of the books I purchased and have seen some good points in them but have not read them fully. I will do this when I begin development.

Intended Changes

Next month I will finalize my requirements document and make any changes to it, if necessary, after my first meeting with my supervisor, Adriana Chis. I will then continue my development in preparation for the prototype due in December.

Supervisor Meetings

Date of Meeting: 28/10/2016

Items discussed: My motivations for the project, functional and non-functional requirements.

Action Items: Email draft of my requirements document to be reviewed.

6.3.3 November

My Achievements

This month I finished my requirements document. I am in the process of combining all the documentation I have done so far and adding it to my technical report, which is to be uploaded by 09/12/16.

This month I have also developed a prototype for my mid-point presentation. I mainly focused on the web server side of the project. I have finished the logic for making requests when traveling by car. I have also set up the public requests however currently this is just returning the raw google response. This server has also been deployed to digital ocean using an Ubuntu OS and using Nginx for the server. It is registered at intimealarm.com.

To complement the web service prototype, I made an activity in my android app to allow me to make requests to the server and print out the response. I also added in a List View to show how I plan to display the alarms to the user.

My Reflection

I am happy with the progress I have made for this project, this month. However, other modules have been more stressful. I feel that, even with several difficulties and bugs I had making this prototype, I am in a good position for my presentation in December.

Intended Changes

Next month I will finalize my technical report and create slides for my presentation. I will also prepare my demo and decide how I am going to go about show all the features that I have developed. I also Intend to enquire about feed back of my report from Adrian as well as asking her opinion of my prototype.

Supervisor Meetings

Date of Meeting: 14/11/2016

Items discussed: Had a group meeting with the other students with the same supervisor and shared all our ideas.

Action Items: Continue to work on my technical report.

6.3.4 December

My Achievements

This month I completed the required sections of my technical report. This required combining all previous documentation into a single report and making alterations and adding in any more relevant information.

I also continued to work on my prototype and was able to add in logic to calculate duration when traveling by public transport. This is currently only implemented for Dublin Bus but will be update at a later stage to work for other method of public transport.

The mid-point presentation took place this month and required me to make presentation slide and present my project in front of my supervisor and another lecturer.

My Reflection

I feel that I have made a lot of progress so far, in particularly with the web server. I done a lot of refactoring to the structure of my web server as I found that the original structure needed duplicate codes and was limiting the potential of the project. This refactor should make it easier to continue development and make it easier to make changes or add functionality.

A lot more work needs to be done to the Android side of the project and that will be my main focus for the upcoming semester.

The pace that I was working on the project has slowed slightly dues to the holidays but I believe that after the exams in January I will get back on track.

Intended Changes

Next month might also be a slow month for progress as my main focus will shift towards the exams. However, any time that I do get to work on the projects will be targeted at the Android App and get its basic functionality working.

Supervisor Meetings

Date of Meeting: 12/12/2016

Items discussed: Got feedback on my technical report and discussed the prototype I was going to demonstrate at the mid-point presentation.

Action Items: Made changes to my technical report and got my prototype ready for demonstration.

6.3.5 January

My Achievements

This month I made developments to both the application and the web service.
On the web service I implemented https by using letsencrypt.org to generate an SSL certificate. I then made changes to the Nginx configuration file to redirect traffic from http on port 80 to https port 443. I also implemented a basic statistics page that calculates the accuracy of the travel time estimation. This supports the POST and GET methods to add all and retrieve the trip statistics respectively. These trips are stored in a mongo database that resides on the same server on port 27000.
On the application I made a lot of advancements, I added in a basic alarm system that the user can set which activates an android notification. The user can set both a single and repeating alarm. I also added in the ability for the user to add and save a location. This implements an intent to the google place picker API. As well as these advancements I have updated the prototype to use a recycler view instead of a list view, allowing better actions in regards to alarm enabling/disabling and deleting alarms.

My Reflection

The semester 1 exams were at the beginning of this month, so all my attention were on them. Once they were over I focused entirely on the project and feel that I have made great progress. The speed of this progress has slowed slightly now that I am back in college and have other assignments but I am still doing it at a maintainable pace and am on track with my plan.

Intended Changes

Next month I intend to finalize the basic alarms by adding in a ringtone service and making the notification aesthetically pleasing. I also intend on adding in a page that will allow me to send trip details to the web service for testing. I will then begin to work on the smart alarm feature.

Supervisor Meetings

Date of Meeting: N/A
Items discussed: N/A
Action Items: I have not met my supervisor since before the semester break but have sent an email to arrange one.

6.3.6 February

My Achievements

This month was a very productive month in terms of making progress on my project.
For the application I have finished the functionality of both the basic alarm and smart alarm. I have also added in the notification and ringtone. The ringtone however is currently commented out, to prevent a constant noise while testing. The smart alarm work by setting an alarm before the time which the user set it for. This is currently set for 1 minute before hand to aid it testing. It then makes a request to the server and set a new alarm based on the results.
For the Web Server I have added in Luas information to increase the accuracy of the travel time. I have also implemented a MongoDB database to store trip data that is used for the evaluation of the accuracy of the project. This evaluation works by the app getting an estimate of the travel time and then reporting back with the actual travel time once the journey is made. I have also added in unit tests to ensure all helper methods work as intended as well as testing the response from numerous requests. Finally, I have also added in a simple front end to the web site using a bootstrap template and charts JS. Gulp has been implemented to manage my builds.

My Reflection

I feel I have made huge progress this month and am ahead of schedule in regards to completing the project. I am however behind in completing the documentation as this has not been updated since the mid-point.

Intended Changes

Next month I have to add unit tests for the application and also clean up the user interface. I also have to add in train data to the webserver, which will be the final source of public transport that will be supported. I also have to begin to update my technical report.

Supervisor Meetings

Date of Meeting: 14th & 28th Feb 2017
Items discussed: Testing, Timeline, General Discussion
Action Items: I have met with my supervisor several times this month and have discussed the project time line and testing.

6.3.7 March

My Achievements

This month I have finished the implementation of the web service Api. This was completed with the addition of train data, specifically the dart. I had originally thought that I would have to use the train specific Api but with research found out that the information is also accessible through the same RTPI endpoint as Dublin Bus.

I have also added in unit tests for the android application, these unit tests cover the Helper class which is used by many other classes throughout the application.

Lastly, I have looked over my technical report from the mid-point presentation and updated areas that no longer match the end product. I have also added in information about testing but this is not yet finished.

My Reflection

The productivity has slowed slightly this month due to various CAs and deliverables for other modules. However, now with the semester officially over and only a hand full of deliverables left I should be able to finish the project in no time.

Intended Changes

Next month, being my final month, I intend to fully finish the project. To do this I have to finish the testing and user interface of the Application. I also have to complete the technical report to accompany the project.

Supervisor Meetings

Date of Meeting: 21st March & 4th May 2017

Items discussed: Testing, Evaluation, General Discussion

Action Items: During the meetings with my supervisor, she has recommended that I add in another feature for complexity after I finish the project.

6.4 Other Material Used

6.4.1 Customer Testing Templates

Task No.	
Task Goal	Create Basic Alarm: 7:30 – Repeat Weekdays
Start Time	
End Time	
Expected Behaviour	1. Click on Floating action button. 2. Select Time 3. Select Repeating Days 4. Click Save
Actual Behaviour	
Notes	

Task No.	
Task Goal	Create Smart Alarm: 7:30 – Repeat Weekdays – From Home – To Work – Due at 9:00
Start Time	
End Time	
Expected Behaviour	1. Click on Floating action button. 2. Select Time 3. Select Repeating Days 4. Select Smart 5. Select Public or Private 6. Set to Location 7. Set from Location 8. Set Due Time 9. Click Save
Actual Behaviour	
Notes	

Task No.	
Task Goal	Add Location – National College of Ireland
Start Time	
End Time	
Expected Behaviour	1. Select My Location from Menu 2. Click Floating Action Button 3. Select Location Confirm **OR** 1. Select Add Alarm 2. Select Smart Alarm 3. Select to or from drop down 4. Select <Add Location> 5. Confirm 6. Save
Actual Behaviour	
Notes	

<table>
<tr><td>Task No.</td><td></td></tr>
<tr><td>Task Goal</td><td>Survey</td></tr>
<tr><td rowspan="4">Questions</td><td>How would you rate the design of the Mobile App?

Very Bad 1 2 3 4 5 6 Very Good</td></tr>
<tr><td>How would you rate the design of the Web App?

Very Bad 1 2 3 4 5 6 Very Good</td></tr>
<tr><td>What is your favourite design aspect of the Mobile App?</td></tr>
<tr><td>What is your least favourite design aspect of the Mobile App?</td></tr>
<tr><td>Notes</td><td></td></tr>
</table>