Tax-E - Android App for Local Taxi Bookings

CS39440 Major Project Report

Author: Rhys Evans (rhe24@aber.ac.uk)
Supervisor: Chris Loftus (cwl@aber.ac.uk)

23rd April 2019
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (G400)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Rhys Evans

Date: 23rd April 2019

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Rhys Evans

Date: 23rd April 2019

Acknowledgements

I would like to thank the following people for their invaluable support and guidance throughout this project:

My project supervisor, Chris Loftus for allowing me to pursue this project and going above and beyond to support me throughout. His organization of the weekly progress meetings was thoroughly helpful, allowing me to reflect on my progress and receive genuine feedback on various aspects of implementation.

My caring partner, Sara for providing me with continuous support, motivation and inspiration. Her honest and constructive feedback on the app's development was invaluable.

My loving family, John, Davina, Carwyn and Delyth for always being there and believing in me.

Abstract

Web and Mobile Technologies have become increasingly ubiquitous in the travel and tourism industry. These technologies, together with a company's ability to successfully adopt them is often pivotal to the company's success within the industry.

This project's aim was to provide a platform for local taxi companies to interact with and provide a service for their customers. The key aspect of the platform is the creation, viewing, and management of bookings. By handling bookings this way, customers and taxi companies can reliably keep track of their past and present bookings in real time.

This project has produced a fit for purpose, fully functional platform. It is split into 3 key systems: a REST API to interact with the backend database and be consumed by other systems within the platform; a web app to be used by controllers within a company to claim, delegate, and manage bookings; and an Android App to be used by customers and drivers to view, create, and manage bookings. The developed product is flexible, allowing taxi companies to choose the level of adaptation that would best fit their business.

Predictably, however, there is room for improvement and future development within the project. Given the scope and time restrictions provided, the produced platform is only intended as a starting point and proof of concept.

This document will provide an insight into the motivation behind platform's development, address weaknesses and key areas for improvement, and discuss the engineering and project management challenges encountered and how they were overcome.

Contents

1 Background & Objectives **1**
- 1.1 Preparation . . . 1
  - 1.1.1 Background Research . . . 1
  - 1.1.2 Functionality Research . . . 4
- 1.2 Analysis . . . 4
  - 1.2.1 Objective . . . 4
  - 1.2.2 Proposed Approaches . . . 5
  - 1.2.3 Deliverables . . . 5
- 1.3 Process . . . 7
  - 1.3.1 Practices . . . 8

2 Design **10**
- 2.1 Overall Architecture . . . 10
- 2.2 REST API Architecture Overview . . . 10
- 2.3 Data Model . . . 11
- 2.4 Android App Architecture Overview . . . 12
- 2.5 Web App Architecture Overview . . . 13
- 2.6 User Interface . . . 14
- 2.7 Technologies Used . . . 14
  - 2.7.1 Implementation . . . 14
  - 2.7.2 Data Persistence . . . 14
  - 2.7.3 Development Environments . . . 15
  - 2.7.4 Project Management . . . 15
  - 2.7.5 Deployment . . . 15

3 Implementation **16**

4 Testing **17**
- 4.1 Overall Approach to Testing . . . 18
- 4.2 Automated Testing . . . 18
  - 4.2.1 Unit Tests . . . 18
  - 4.2.2 User Interface Testing . . . 18
  - 4.2.3 Stress Testing . . . 18
  - 4.2.4 Other types of testing . . . 18
- 4.3 Integration Testing . . . 18
- 4.4 User Testing . . . 18

5 Evaluation **19**

Annotated Bibliography **20**

Appendices **24**

A Third-Party Code and Libraries **26**

B Ethics Submission **27**

C Code Examples **28**
3.1 Random Number Generator 28

List of Figures

List of Tables

Chapter 1

Background & Objectives

The aim of this project was to create a platform to serve as an intermediary between local taxi companies and their customers. The development of such a platform presented many engineering and project management challenges. This chapter aims to discuss the preparations required for the project, together with an analysis of the project's objectives and an in-depth description of the processes used.

1.1 Preparation

1.1.1 Background Research

In order to successfully execute the project, research had to be done into various aspects of the app's development. This includes methodology, technologies, and deployment options.

1.1.1.1 Methodology

Software Development Methodology is a key part of any large project. The effect of the methodology on the product, along with a project's ability to adapt to the methodology should always be considered when making a selection. With this in mind, several options were researched for the project.

The first methodology researched was the Waterfall Model, or 'SDLC' [1]. Having already worked on a project using the Waterfall Model, there was less of a focus on researching the methodology but rather to consider its applicability in this project. As the Waterfall Model is a plan-driven approach it would require a lot of upfront documentation and decision making with regards to requirements, thus potentially making it rather inflexible. Given that this project also involved a significant learning curve with regards to the technologies used, it would perhaps be tricky to accurately gauge the ability to implement various functionalities early on.

The next methodology researched was Feature Driven Development, or 'FDD' [2]. FDD

is an agile methodology created by Jeff Luca, it is an iterative approach with an emphasis on the timely and successful delivery of features. It is broken down into 5 key processes: developing an overall model, building the feature list, planning by feature, designing by feature, and building by feature. It is clear from these processes that the project's development would be done on a 'per-feature' basis, comparable to stories in other agile methodologies. At first impressions, this methodology seemed much more appropriate for the project than a plan based approach.

Extreme Programming or 'XP' [3, 4], was the final methodology to be researched. It was one of the recommended methodologies provided for the project. XP is described as being ideal in projects with dynamically changing requirements, risks from fixed time projects using new technology, and when using technologies that allow for automated tests [4]. The methodology is often identified by its values, which give a great indication to the mission of XP. Reading literature written by one of XP's creators provided a great insight into the application of XP and its key advantages over other approaches.

1.1.1.2 Technologies

Having only experienced Android app development with a local, offline backend; it was essential to the project's success that research was done to explore the tools and technologies available to support a networked mobile app. This research was required for each aspect of the project, including its deployment.

The first and arguably most significant area to research was the API that would support the app(s). It was clear from very early on that there would likely be at least two apps within the platform to consume the API (web app and mobile app). Therefore, an API would need to be developed and a language/framework had to be selected to develop such an API. The researched options were: ASP.NET [5], Rails 5 [6], Node.js & Express.js [7, 8] and Flask (Flask-RESTful) [9]. All of the researched options had fairly similar learning curves (due to the technology they are built on and past experiences), each option was also perfectly fit for purpose to create a REST API to serve a mobile application.

The next technology to research was the database management system (DBMS) that would be used by the REST API for data persistence. The selected DBMS would ideally be able to interface easily with the framework used to create the API itself. The options researched were: PostgreSQL [10], MySQL [11], and MongoDB [12]. The first two options were both traditional relational database management systems, whereas the final option is a NoSQL document-oriented system.

In order to successfully consume the REST API within the Android application, a HTTP Client library had to be chosen. During this selection, two complementary technologies were researched, Retrofit [13] and rxJava [14]. Retrofit is a HTTP Client library for Java and Android that is primarily used to consume REST APIs, rxJava is an asynchronous programming library with observable streams. This allows requests to be made to the REST API from the mobile application and the responses can be stored in a life-cycle aware Observable object, thus fully supporting asynchronous programming.

It became apparent that a relatively small web app would need to be created as part of

the platform to support the Company Administrator's role. Therefore, a front-end web framework to consume the API had to be chosen. The frameworks that were looked into were: Angular [15] and React [16]. Although Vue.js [17] was initially considered, it became apparent that React and Angular were the two most prominent options and would be the most beneficial technologies to learn.

Due to previous experience, it was already decided that Android Studio would be the chosen IDE used to develop the Android application. However, there was still a decision to be made about developing the API and Web Application. The standard code editor 'Atom' had been used for previous web projects. However, upon further research JetBrains's 'WebStorm' IDE was discovered. Therefore research was done into both solutions in order to select the most appropriate.

Lastly, project management, deployment, and development tools had to be researched. In terms of project management, it was already decided that an online Trello board [18] would be used to keep track of the needed tasks. It was also decided early on that GitHub would be the primary version control tool used in the project, due to its familiarity and potential integration with continuous integration tools. In order to fully explore the potential of using an agile development methodology, various continuous integration tools were researched. These included: JetBrains's TeamCity [19], TravisCI [20], and Jenkins [21]. Each CI tool has its strengths, with TravisCI standing out as the most lightweight and easily integrated. TeamCity is developed by JetBrains, several of their products were also considered/used for the project so familiarity would be an advantage. Lastly, Jenkins appeared to be the most widely used tool and therefore would have plenty of documentary and online tutorials. Finally, in order to potentially support any CI tools, the containerization tool 'Docker' [22] was researched for use in the API's development.

1.1.1.3 Deployment Options

Deployment of the Android app itself is made fairly straightforward by Android Studio's build options. Allowing for APK signing and Google Play Store support. However, with regards to the REST API and the Web Application, some research was required to decide on the most appropriate deployment solution. The following solutions were researched and considered as options: Heroku [23], Amazon Web Services (AWS) [24], and self-hosting using Nginx [25]. Each of these solutions had its advantages, deploying via Heroku would be made simple due to its in-built Docker support. Using AWS would allow for very fine grain control over server size, resources, and traffic management; it would also make supporting SSL encryption trivial. Lastly, a custom hosting solution using Nginx would be the simplest and easiest to manage, due to pre-existing server setups. However, this solution could have some security implications.

1.1.2 Functionality Research

1.1.2.1 Related Products

A major part of the functionality research required for this project was to investigate applications already on the market that are targetted towards Taxi booking creation and management. This research would go on to help identify the key functionalities required within the platform and the most effective way to implement them. Having already decided on a name, rough purpose, and short description. Researching similar products on the market is regarded as the next key step in developing an app idea.

The first and perhaps the most prominent product to be investigated was 'Uber' [26]. Uber's mobile app is designed to manage and delegate bookings for registered Uber drivers and not self-contained Taxi companies. However, the interaction and flow of the booking process between the driver and the customer served as a major influence when coming up with the concept for this project. Together with in-app messaging to allow easy communication between customers and drivers, Uber's use of GPS is the ultimate goal for booking creation and tracking. However, it was clear from early on that these well-polished features fell outside of the scope of this project.

Having investigated a commercial solution that did not target Taxi companies directly, the research required a more applicable product. 'My Taxi' [27] is a cross-platform, mobile application that allows customers to book, track and pay for their taxi. The app is incredibly well polished and maintained with an intuitive sign-up flow and booking creation process. The app is supported in many major European cities, including Brighton, Edinburgh, London, Manchester, Nottingham, Oxford, Reading, Derby, and Leicester. In addition to their mobile app, they also have a web client to support taxi business administrators.

Lastly, a primarily web-based service, 'minicabit' [28] was investigated. Minicabit is a UK based service, that allows users to retrieve taxi booking quotes, it is also possible to book a taxi based on the quotes received. The website's user interface was very friendly and intuitive. The look and feel of the quoting system were very similar to other websites within the travel industry. Although this service provides support for booking taxis, it appeared to have minimal support for any booking management or tracking.

1.2 Analysis

1.2.1 Objective

The primary objective of this project was to create a platform that serves as an intermediary between local taxi companies and their customers. The platform would need to support three types of users: Company Admin, Driver and Customer.

It would be intended that a single instance of the platform be used by many taxi companies, each with their own Company Admins and Drivers. The companies would then use the platform as a service in order to support their internal booking mechanisms. How-

ever, it should also have the ability to be adapted to be used by a single Company as their sole booking management mechanism. This would make the platform very flexible, allowing for varying use cases.

The primary motivation for this project was to streamline the process of making taxi bookings in small, local areas. With the hope that a fully mobile, digital solution would lead to a better experience for both the companies and their customers. For example, being able to view a history of all bookings could be extremely helpful for cases of lost property, criminal damage, etc. In addition to this, by viewing the live status of their booking, and communicating with their driver via notes, customers can make sure they are present for their booking at the correct time, hopefully resulting in fewer wasted Taxis.

1.2.2 Proposed Approaches

Having formalized the objectives of this project, it was required that an approach to fulfilling those objectives be selected. Given the initial topic that this project stemmed from ('Android app of your choice'). It was apparent that a native Android App would be the primary solution to complete the project's objectives. However, there was plenty of freedom to decide on an approach for a backend to serve the application; or perhaps a secondary application to aid in the fulfillment of the objective.

The first approach that was conceived involved a single Android App and a server to provide persistence. The single App would support all three user categories by serving content dynamically based on role. Although an entirely valid approach, it was believed that a mobile application may not be suitable for use by a Taxi Controller in an office environment.

Bearing in mind the realizations gained from the 'single app' approach, a desktop application to support Company Admins was considered. This would likely involve a fairly simple desktop application being developed alongside the mobile application to consume a common backend. This approach certainly addresses the concerns raised by the previous approach. However, it would limit the use of the application to an office environment. There are some taxi companies that operate with shared roles, sometimes the controllers also act as drivers.

The final and accepted approach was to support Customers and Drivers with an Android App and to support Company Admins with a Responsive Web Application. Both of which would still be served by a common backend. This was believed to be the most flexible and fitting solution. It would allow the application to be used from an array of devices and potentially allow multiple systems to be used at once in some very specific use-cases.

1.2.3 Deliverables

As discussed in Section 1.2.1, broadly, the primary objective of this project was to create a fully functioning Taxi booking platform. Six tangible deliverables can be drawn from

this objective.

1.2.3.1 Full set of requirements in the form of User Stories

This deliverable would contain a full set of formally written user stories. Each story would contain four parts: title, description of the feature, 'essential' or 'non-essential' flag, and the acceptance criteria. Below is an example of a User Story for this platform:

Customers can cancel a booking at any time (Essential)
As a customer, I want to be able to cancel my booking so that if I no longer need a taxi I don't waste the taxi company's time driving out.

Acceptance Criteria

- If a customer selects to cancel a booking they are prompted for a confirmation.
- If the customer confirms the cancellation their booking is updated.

Together, all of the 'essential' stories would form the minimum viable product for this project. With various systems partially fulfilling the story, until it has been fully completed. To consider the above example, the cancellation of a booking must be supported both within the API and the mobile application.

1.2.3.2 A fully functioning REST API to behave as the 'core' of the system

Given that each application within this platform would consume the API, it would need to partially meet the majority of the requirements laid out by the User Stories. Therefore, this deliverable would involve the creation and deployment of an API that can fully support each requirement of the platform. Considering the importance and use of this deliverable, security will most certainly be a primary concern.

1.2.3.3 User Interface prototypes and style documentation for the user-facing systems within the platform

This deliverable would involve the creation of UI prototypes and a decision about the general look and feel of the platform to be documented. This would allow the development of the user-facing systems to be informed and consistent with the rest of the platform.

1.2.3.4 A user-friendly and intuitive Android Application to consume the API and be used by Customers and Drivers

In order to support all of the requirements and API functionalities concerning Customers and Drivers, a 'front-end' would need to be created to allow these users to interact with

the platform. This would be the primary user-facing application within the platform. It would also require support for allowing the creation of users and enforce access control for unauthenticated users.

1.2.3.5 A mobile-friendly Web Application to consume the API and be used by Company Admins

Similarly to the mobile application, a system would need to exist to allow Company Admins to interact with the platform. This system would need to satisfy all of the requirements relating to Company Admins. In order to do this, it would need to strictly enforce role-based access control to ensure only registered Company Admins can make use of the system.

1.2.3.6 A set of automated and manual tests to verify the correctness of each system and their success at fulfilling the set requirements

As with any project, correctness and accuracy are imperative. Therefore, thorough testing of the platform is undoubtedly a key deliverable. The testing strategy would also ensure accurate acceptance measurements of the requirements as outlined in each user story.

1.3 Process

Having narrowed down the options for development methodology to those discussed in Section 1.1.1.1 and researched each option it was decided that an adapted version of Extreme Programming would be used for the project. This section will discuss the specific practices and methods that will be applied in this project's development methodology.

Given the restrictions that a plan-based approach such as SDLC would have on the project, with specific regard to the requirements it was decided the approach had to be agile. Due to the relatively small scale and timeframe of the project, it was also thought that it would be very beneficial to focus on high quality, working software over the documentation required by SDLC.

Feature Driven Development was another agile approach that was considered. Although with adaptations, it could have been very well fitted for a single person project, it was apparent that FDD was intended for use in a team-based project. This is seen in its emphasis on roles. Therefore, it was clear that XP was the most appropriate agile methodology for the project.

XP was selected in the hope that its use would help mitigate the increased risk associated with a project where there is a lack of experience and a lot of learning to do with the technologies used. Due to this increased risk, it was believed that creating a static requirement list would prove difficult and likely be very flawed. Therefore, given that XP is an Agile Approach and is intended to support changing requirements it fit perfectly.

1.3.1 Practices

The following practices are all considered primary practices within Extreme Programming [3, Ch. 7]. They have been selected due to their relevance to the project. The following inapplicable practices have been omitted: Sit Together, Whole Team, Informative Workspace, Pair Programming, Test First. These practices were aspects of XP targetted towards team-based projects and larger workspaces.

1.3.1.1 Energized Work

Although mainly focused on software development teams to ensure a healthy workplace and work/life balance, this practice also suits individual projects very well. By following this practice, work is undertaken at the right time and location for the right length of time, without burning out. This requires that steps be taken to ensure a focused state can be entered to complete work without distractions.

1.3.1.2 Stories

Stories are a key practice in most agile methodologies, in this project user stories were written for each desired functionality. Each story was labeled as either 'essential' or 'nice to have', this ensured that work was prioritized correctly and essential aspects of the system were built in a timely way. Stories were written on physical, coloured cards and placed on a corkboard in the main work area of the project. They also existed electronically on a Trello [18] board, this allowed progress to be shared with the project supervisor and allowed work to be completed away from the main work station. There was also be a static document created with each high-level story written formally, this can be found in the Appendices.

1.3.1.3 Weekly Cycle

Weekly Cycles are synonymous with iterations in other agile methodologies. In this project weekly cycles ran from Saturday-Friday, meaning each cycle began on a Saturday and ended on a Friday. The first day of the cycle mainly consisted of planning and choosing which stories to tackle; there was also an overall reflection and a look at the project's current progress. The goal was that by the end of the cycle all stories had been realized and are now implemented, integrated and tested features within the system. The final day of the cycle involved writing an entry into the project's weekly blog [29] reflecting on the week's work and documenting the goals for the next week. By using a weekly cycle, software was constantly being produced, allowing for a regular opportunity to analyze and get feedback.

1.3.1.4 Quarterly Cycle

Quarterly Cycles are synonymous with releases. Given the scale of this project (roughly 12 weeks), a quarterly cycle consisted of 3 weekly cycles. This meant that the first quarterly cycle passed just in time for the mid-project demonstration, thus ensuring there was at least a partially working system to be delivered. At the beginning of each quarterly cycle, a plan was drafted for which stories should be completed in that cycle. The quarterly cycle plan was re-visited briefly at the beginning of each weekly cycle.

1.3.1.5 Slack

The purpose of slack is to add low priority or non-essential stories to weekly and quarterly cycles. This allowed tasks or stories to be discarded if time became an issue, thus accounting for inaccurate estimates.

1.3.1.6 Ten-Minute Build

The 10-minute build practice is intended to enforce a quick and easy build process, if a build process is long and arduous it is less likely to be done often. Therefore an easy, automatic build process ensured that builds were done frequently, allowing for more feedback and less time between errors.

1.3.1.7 Continuous Integration

Continuous Integration supports the idea of a fast build process. By immediately testing and integrating code systems into the overall system, integration issues were caught much sooner and with less consequence. Continuous Integration also encouraged and enforced frequent and reliable automatic testing.

1.3.1.8 Incremental Design / Refactoring

Incremental Design suggests that some spike work be done up front to understand how much effort is required and any issues that might arise. This approach reduced risk and allowed for a better estimation for a given story. This practice allowed for informed design decisions to be made when necessary and be made with the most current information available. The practice of incremental design also calls for the use of frequent refactoring to ensure correct design.

Chapter 2

Design

This chapter will discuss the overall architecture and higher level aspects of the platform's design. It is only intended to provide an overview of the design. A more in-depth discussion of specific design decisions will be presented on a per-iteration basis later in the report.

2.1 Overall Architecture

The platform is comprised of 3 key systems: a REST API that interfaces with a MongoDB database; an Android App to consume the API and serve content to Customers and Drivers; a Responsive Web App to consume the API and serve content to Company Admins.

[INSERT OVERALL GRAPHIC / USE-CASE DIAGRAM?]

Both the Android and Web App are categorized as user-facing systems. Meaning the platform's users are never intended to interact with the API directly, but rather through one of these two Applications (depending on their role). This also means that the two user-facing Applications never interact with each other, they exclusively send and receive data from the API. This is depicted in the sequence diagram below:

[INSERT SEQUENCE DIAGRAM]

Data received from the API is in JSON format, which is natively deserialized by Javascript and Typescript. However, Android's GSON library is used to support JSON conversion to and from 'plain old Java objects' (POJOs).

2.2 REST API Architecture Overview

Broadly, the REST API is a modular system split up into the following: models, services, controllers, routes, middlewares, and helpers.

The model module contains all the models for the system. These directly map onto a collection within the MongoDB database. Some models may reference each other using MongoDB ObjectIDs, comparable to SQL's foreign key constraint. Any model file can be referenced from anywhere within the system and used as a 'type' to initialize objects.

The services module is the lowest level of abstraction for interfacing with the backend database. It exposes several necessary public functions to the rest of the system. Typically, it will implement CRUD functions via mongoose. The service files are only intended to be accessible from their respective controllers.

Controllers are the next level up from services in the API's abstraction layers. Controller files are called by the API's router to pass information from the HTTP request to the required service. Then execute the relevant middleware. The functions within a controller file are typically very simple and simply serve as an intermediary between the router and the services themselves.

Routes are used to capture information from the HTTP request and pass it to the controllers so the relevant operations can be completed (via the service). These include retrieving values from request headers, the request body, and any URL parameters or queries.

Several middleware files were created for the API, middleware is executed by the router before any code from the controller is executed. Middleware can be configured to run on every route or to be executed on a per-route basis. For example, the error handling middleware needs to be executed on every route. However, a piece of middleware such as access control or authentication may only need to be executed on certain protected routes.

[INSERT DIAGRAM TO SHOW ORDER OF EXECUTION OF MIDDLEWARE]

Helpers do not have a designated place in the order of execution of requests. Rather, they are intended to be used by other modules within the API to perform frequent, utility tasks or to enumerate types. For example, there are two enumeration types within the helper module 'role' and 'status'. These are used across many different modules within the API to reduce code duplication.

[INSERT DIAGRAM TO DEPICT THE ABSTRACTION LAYERS]

2.3 Data Model

The data model for the platform is fairly straight forward, the simple, flexible nature of MongoDB certainly assisted in this simplicity. To preface, a 'collection' in this context is comparable to a 'relation' in a traditional relational DBMS. A 'document' is comparable to a 'tuple' or 'record', and a 'field' is comparable to a 'column' or 'attribute'.

The data model contains three collections 'User', 'Booking', and 'Company'. All collections have unique Object IDs and each collection references another. The 'User' collection is likely the largest, it is a general model for Customers, Drivers, and Company Admins. The decision to use a single, larger model for all users, as opposed to a specific model for

each type of user was made early on, due to its simplicity. This is implemented by leaving the type-specific fields as null if they are not applicable to the user. A user's type is stored using the 'role' field. The data model is depicted in the Entity Relationship below:

[INSERT E-R DIAGRAM]

2.4 Android App Architecture Overview

Much of the Android App's design is boilerplate, with the main package split into java code and app resources. There are no significant differences between the app's build variants as it is intended to offer different functionality dynamically, depending on the user's role. It was considered early on to provide two different variants of the app, one for customers and another for drivers. However, as the differences between the two app's functionalities would be very minor it was decided that a single app would be the best approach.

The app's main java package is comprised of 5 packages: model, network, ui, util, and viewmodel.

[INSERT IMAGE OF ALL PACKAGES AND CLASSES]

The model package contains 5 model classes, each representing a type of JSON response that can be received from the API. They are required to enable serialization/de-serialization of JSON objects. The 'User', 'Booking', and 'Company', and 'Response' models are the ones uses most frequently. With 'LoginResponse' only being used in a single special case. The Response model is returned anytime the API is expected to return a generic response with no particularly important data. It is very useful for Error Handling for retrieval of JSON values.

The network package contains the classes required to interface with the API using Retrofit. It contains the 'RetrofitInterface' interface, which is comparable to the Repository interface that would be used in Android's data persistence library. The 'NetworkUtil' class has several overloaded methods that each return a differing instance of the RetrofitInterface. For example, if the method is called with no parameters it returns a Retrofit instance with no HTTP authentication headers. Whereas, if the method is called with an access token as a parameter, it returns a Retrofit instance with the access token in the HTTP headers.

All of the app's activity and fragment classes are located in the ui package. The package itself is split into 5 sub-packages, which are intended to relate to each of the app's screens and dialogs. The app has two main activities Authentication and Main. This is how access control is enforced in the app, the user can only access the main activity if they have successfully logged in via the authentication activity. There are of course other activities within the app, but they are always called on top of one of the two main activities.

The util package contains all of the classes that contain shared code that is intended to be used all across the app. This includes the app's error handler, validator, shared preference manager, customer de-serializers, enum types, and more. The use of a util package with these classes is intended to reduce code duplication.

The Model View ViewModel (MVVM) design pattern is implemented in this app via the viewmodel package. The view models within this package provide a layer of abstraction between the networking and UI classes. In the case that the API's routes were updated, supporting this update in the app is made considerably easier using view models. Therefore increasing maintainability and encouraging good object orientation practices.

[INSERT MVVM DIAGRAM]

2.5 Web App Architecture Overview

The Web Application is likely the simplest system within the platform. Similarly to the Android Application, the main directories are fairly similar to a boilerplate Angular Application. The main directory of the app contains seven sub-directories: '_guards', '_helpers', '_models', '_services', 'components', 'layouts', 'modules'.

The '_guards' directory currently contains a single guard file to enforce authentication within the app.

The '_helpers' directory contains two HTTP interceptors, an error interceptor, and an access token interceptor. The error interceptor is intended to intercept incoming responses from the API and handle any errors appropriately. If the application was of a larger scale, it would likely be more appropriate to also have a designated error handling file. However, due to the scale of the application currently, this solution was selected due to its simplicity. The 'jwt' interceptor is intended to intercept outgoing requests to the API and append the user's access token to the request headers. This means that the token doesn't have to be manually added each time a request is made.

The '_services' directory contains all of the services required by various views. The services are intended to add a layer of abstraction between the view and the HTTP requests. There is also a service file for displaying notifications in the view, it works by injecting a HTML element into the view and running jQuery code to animate the element.

All of the app's angular components reside in the 'components' directory. An angular component represents a 'page' within the app, with components belonging to a module. For example, the 'drivers' page is a component that belongs to the 'main' module. Each component contains a view file, style file, and controller file. These are typically html, scsss, and typescript files respectively.

Any recurring layouts within the app reside in the 'layouts' directory. This includes things such as headers, footers, navbars, etc. Similarly to components, each layout has a view, style, and controller file. However, the layout directory is treated as its own Angular module.

Lastly, the modules directory contains the app's two main modules 'auth' and 'main', these modules create a scope in which components can exist. Each module requires a router and module file, in addition to the files required by standard components. A module's view file will typically contain a placeholder for whichever component the module's router serves.

[INSERT IMAGE DEPICTING ANGULAR APPLICATION LAYOUT]

2.6 User Interface

2.7 Technologies Used

As the developed platform would contain several systems, it was clear that a diverse array of technologies would need to be adopted. Both to aid project management and for use in implementation. This section will discuss the selected technologies for this project.

2.7.1 Implementation

With regards to the technology that would be used to develop the REST API, it was a close decision between Rails 5 and Node.js / Express.js. Although Flask and ASP.net were most certainly fit for purpose and had plenty of documentation to get started. There was a distinct lack of experience in the underlying languages (Python & C#). The project supervisor had a level of expertise in Rails, this would have been an advantage for the adoption of this framework. However, due to personal preference and a slight bias towards Javascript, Node.js together with Express.js was selected. This choice showed great promise, with a considerable amount of online resources and available testing libraries. It would also mean the development of the API would integrate seamlessly with Node.js's default package manager (NPM). This would allow access to hundreds of thousands of javascript packages.

Having researched both Angular and React it was clear they were both equally appropriate for this task. However, Angular's traits were increasingly more appealing after in-depth research. It offered a substantial amount of in-built functionality compared to React. Examples of these functionalities are an In-built XSS protection, a native Router (eliminating the need for a third party library), a form builder, and SCSS compiling. It also supported Google's Material Design, through pre-built components, this was ideal to ensure consistency between the platform's applications. The only considerable drawback was that Angular is based in Typescript, a syntactically strict superset of Javascript; this was an unfamiliar technology. However, given the considerable advantages and previous experience with Javascript, the benefits certainly outweighed the drawbacks. It should also be mentioned that the adoption of Angular would loosely complete the project's adoption of the MEAN stack [37].

2.7.2 Data Persistence

Although the research was certainly a key part of making a fully informed decision; a JSON-like document-oriented DBMS seemed like an obvious fit, having already decided to use Node.js for the API. Therefore, MongoDB was selected. One disadvantage of a NoSQL solution is that complicated queries are typically considered more difficult com-

pared to a traditional SQL DBMS. However, given the simple nature of the queries for this project, this didn't seem overly relevant.

2.7.3 Development Environments

Having researched JetBrains' WebStorm, it was decided it would be a better fit for the development of both the platform's REST API and Web Application. With substantial language-specific support, code completion, syntax highlighting, and a preview of markdown files, it was an easy decision. WebStorm also provides in-built integration with docker, GitHub, and NPM.

2.7.4 Project Management

After researching many project management solutions, a decision was made to use Docker for containerization of the REST API. This conclusion was mainly motivated by the ability to easily replicate the development environment across multiple machines. Due to Extreme Programming being the project's chosen development methodology, a Continuous Integration tool had to be selected. Having researched several options, it was eventually decided that TravisCI would be used. It was by far the most lightweight and easy to understand tool of those researched. It seamlessly integrates with GitHub through the use of a single YAML file and OAuth, making adoption relatively easy for inexperienced users.

2.7.5 Deployment

In order to make an informed decision regarding the deployment method for the API and Web Application, sample applications were deployed using both Heroku and AWS. It was discovered that the process was actually a lot more in-depth than initially anticipated. Together with having to register an account and provide banking details this was somewhat discouraging. Therefore the API and Web Application were both deployed to a home-based headless ubuntu-server box. The API is deployed using Nginx as a reverse proxy and Node.js's process manager, PM2. The Web Application is deployed using an Apache web server.

Although there are certain security implications with a 'homemade' deployment solution. For example, the lack of reliable SSL encryption and physical security concerns due to the location of the server itself. The deployment is only intended for demonstration purposes and will likely only contain fictitious test data. If the platform were ever to be deployed for production, a fully supported commercial solution would be sought out.

Chapter 3

Implementation

The implementation should discuss any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third-party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Provide information in the body of your report and the appendix to explain the testing that has been performed. How does this testing address the requirements and design for the project?

How comprehensive is the testing within the constraints of the project? Are you testing the normal working behaviour? Are you testing the exceptional behaviour, e.g. error conditions? Are you testing security issues if they are relevant for your project?

Have you tested your system on "real users"? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

Whilst testing with "real users" can be useful, don't see it as a way to shortcut detailed testing of your own. Think about issues discussed in the lectures about until testing, integration testing, etc. User testing without sensible testing of your own is not a useful activity.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1 Overall Approach to Testing

4.2 Automated Testing

4.2.1 Unit Tests

4.2.2 User Interface Testing

4.2.3 Stress Testing

4.2.4 Other types of testing

4.3 Integration Testing

4.4 User Testing

Chapter 5

Evaluation

Examiners expect to find a section addressing questions such as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

The questions are an indication of issues you should consider. They are not intended as a specification of a list of sections.

The evaluation is regarded as an important part of the project report; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things in the work and aspects of the work that could be improved. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

Annotated Bibliography

[1] Tutorialspoint, "SDLC - Waterfall Model," https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm, 2019, accessed April 2019.

This web page is a simple breakdown of the SDLC software development methodology.

[2] S. Palmer and J. Felsing, *A practical guide to feature-driven development*. Upper Saddle River, NJ: Prentice Hall PTR, 2002, 0130676152.

This book attempts to provide an in-depth guide to apply Feature Driven Development to a project by explaining the importance of roles, overall models and feature lists.

[3] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed. Boston: Addison-Wesley, 2005, 0321278658.

This book provides an explanation of the practices and principles of Extreme Programming and gives an insight into the philosophy and motivations behind its creation. The book is authored by one of the original creators of the Extreme Programming methodology.

[4] A. Alliance, "What is Extreme Programming?" https://nodejs.org/en/, 2019, accessed April 2019.

This is an entry in Agile Alliance's methodology glossary. It provides an easy to understand overview of Extreme Programming, adapted from literary pieces by Kent Beck and Don Wells

[5] K. e. a. Burke, "Flask-restful," https://flask-restful.readthedocs.io/en/latest/, 2019, accessed April 2019.

Flask-RESTful is an extension to the Flask micro framework. It is written in python and can be used to quickly and easily build REST API's.

[6] D. Hansson, "Ruby on rails," https://rubyonrails.org/, Feb. 2019, accessed February 2019.

Ruby on Rails is a web application framework for server-side use, it is written in Ruby. It will be considered as an alternative to Node.js to create the REST API and backend services needed for the App.

[7] Various, "Node.js Foundation," https://nodejs.org/en/, Feb. 2019, accessed February 2019.

Node.js is an open source JavaScript runtime that can is used serverside. It will be considered as an option to build the REST API needed for the mobile app.

[8] Node.js Foundation, "Express.js," https://expressjs.com/, Feb. 2019, accessed February 2019.

This is a web framework for Node.js, it will be considered as an option to build a REST API together with Node.js

[9] Microsoft, "Asp.net," https://dotnet.microsoft.com/apps/aspnet, 2019, accessed April 2019.

ASP.NET is a cross-platform, open source framework designed to support the creation of web apps, services and APIs using .NET and C#. It is developed and maintained by microsoft and is very prominent technology in industry.

[10] PostgreSQL Global Development Group, "Postgresql," https://www.postgresql.org/, accessed April 2019.

PostgreSQL is a very well established, open source database management system for relational databases.

[11] Oracle Corporation, "Mysql," https://www.mysql.com/, accessed April 2019.

MySQL is a database management system for relational databases widely used in industry.

[12] MongoDB Inc, "mongodb," https://www.mongodb.com/, accessed April 2019.

MongoDB is a free and open source NoSQL database solution. It is document-oriented and uses JSON-like documents.

[13] Square, "Retrofit," https://square.github.io/retrofit/, 2019, accessed April 2019.

Retrofit is a HTTP Client for Java (and by extension, Android). It is commonly used to consume REST APIs and serialize/de-serialize JSON objects into Java Objects.

[14] ReactiveX, "rxjava," https://github.com/ReactiveX/RxJava, 2019, accessed April 2019.

rxJava is an implementation of the ReactiveX asynchronous programming API for Java. It implements the Observer pattern to allow asynchronous programming with observable streams.

[15] Google, "Angular," https://angular.io/, 2019, accessed April 2019.

Angular is a frontend web framework developed by Google. It is TypeScript based and open source, it has an in-built HTTP client and various supports for Material Design.

[16] Facebook, "React," https://reactjs.org/, 2019, accessed April 2019.

React is a Javascript based frontend web framework, it is developed and maintained by Facebook and similarly to Angular it is component based.

[17] E. You, "Vue.js," https://vuejs.org/, 2019, accessed April 2019.

Vue.js is a frontend web framework that is based in Javascript, it is very versatile and is designed to be used in conjuction with other libraries.

[18] Atlassian, "Trello," https://trello.com/, accessed February 2019.

Trello is a web based service that allows the creation of virtual boards, lists and cards to allow organization and prioritization in projects.

[19] JetBrains, "Teamcity," https://www.jetbrains.com/teamcity/, accessed February 2019.

Team City is another CI/CD tool for consideration, it is created by Jet Brains; the same company who develops IntelliJ Idea, the platform on which Android Studio is built.

[20] Travis CI, "Travis ci," https://travis-ci.org/, accessed February 2019.

Travis CI is a continuous integration tool that can be synced with github projects to create a build flow.

[21] Jenkins, "Jenkins," https://jenkins.io/, Feb. 2019, accessed February 2019.

Jenkins is an open source continuous integration and continuous delivery tool in the form of an easily configurable server.

[22] Docker Inc, "Docker," https://www.docker.com/, accessed February 2019.

Docker allows for containerization of software. Containers are isolated and can be used to bundle tools, libraries and configuration files for simple and easy deployment. Docker works very well with CI/CD tools.

[23] Salesforce, "Heroku," https://www.heroku.com/, accessed April 2019.

Heroku is a cloud platform that has support for deploying many web applications, including those written in Javascript, Python, and Ruby.

[24] Amazon, "Aws," https://aws.amazon.com/, accessed April 2019.

AWS is owned by Amazon and provides on-demand cloud computing platforms on a pay-as-you-go basis. Allowing servers to be spun-up and deployed very easily, also allowing fine grain control over resources.

[25] Nginx Inc, "Nginx," https://www.nginx.com/, accessed April 2019.

Nginx is a web server ideal for deploying REST APIs, as it can double up as a reverse proxy.

[26] U. T. Inc., "Uber," https://www.uber.com/gb/en/, 2019, accessed April 2019.

Uber provides a platform that allows registered drivers to receive 'ride' bookings from customers. The platform also allows communication between the customer and the driver regarding the booking (either via in-app messaging or GPS).

[27] M. N. Ltd., "mytaxi," https://mytaxi.com/uk/, 2019, accessed April 2019.

mytaxi is described as a 'taxi e-hailing' app. They operate in many major cities across Europe and allow customers to electronically book taxis with participating companies.

[28] minicabit Ltd., "minicabit," https://www.minicabit.com/, 2019, accessed April 2019.

minicabit is an online service that allows customers to compare quotes for taxi journeys across the UK. It also provides support for making taxi bookings.

[29] R. Evans, "Project blog," http://blog.rhysevans.xyz/, accessed February 2019.

This blog documents the progress of the project on a weekly basis.

[30] K. Auer and R. Miller, *Extreme Programming Applied: Playing to Win*. Boston;London: Addison-Wesley, 2002, 0201616408.

This book is part of the XP Series, it is described as a pragmatic guide to getting started with Extreme Programming. This book will provide guidance on how to apply the important principles and practices of XP in this Project.

[31] K. Duuna, *Secure Your Node.js Web Application*. Dallas: The Pragmatic Bookshelf, 2016, 1680500856.

This book provides guides and best practices to improve the security of Node.js web applications. It provides tips for creating robust database calls, user authentication, and session management.

[32] Mills, C. et al, "Express web framework (node.js/javascript)," https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs, Dec. 2018, accessed February 2019.

This is a user-created guide to using Node.js and Express.js, it is a resource that would be very useful if Express.js/Node.js were used to build the REST API.

[33] C. Sevilleja, "Express web framework (node.js/javascript)," https://scotch.io/tutorials/build-a-restful-api-using-node-and-express-4, Apr. 2014, accessed February 2019.

This is a tutorial for building REST APIs using Express.js, it provides an introduction to routing using Expree.js 4 and the use of middleware.

[34] Adobe, "Adobe xd cc," https://www.adobe.com/uk/products/xd.html, accessed February 2019.

This is an user interface design and prototyping tool made by Adobe, there are material design expansions available. These can be used to quickly create extremely accurate Android Application prototypes.

[35] Fluid UI, "Fluid ui," https://www.fluidui.com/, Feb. 2019, accessed February 2019.

Fluid UI is a web-based user interface design tool. Similarly to Adobe XD, it too has material design UI packages.

[36] "Data protection act 2018," http://www.legislation.gov.uk/ukpga/2018/12/contents/enacted, May 2018, accessed February 2019.

This act controls how personal information is used by organisations, businesses or the government. It is the UK's implementation of the General Data Protection Regulation (GDPR)

[37] Linnovate, "Welcome to the mean stack," http://mean.io/, accessed February 2019.

The 'mean' stack is the name given to a set of open soruce components used for building dynamic web applications.

Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

For example, if you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document as an appendix. In the body of your report you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.

If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification. Perhaps you use stories to keep track of the functionality and the 'future conversations'. It might not be relevant to include all of those in the body of your report. Instead, you might include those in an appendix.

There is a balance to be struck between what is relevant to include in the body of your report and whether additional supporting evidence is appropriate in the appendices. Speak to your supervisor or the module coordinator if you have questions about this.

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what your original work is and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

The following is an example of what you might say.

Apache POI library - The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client's existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

Appendix B

Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

Appendix C

Code Examples

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Project Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

3.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [**?**].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
```

```
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*-------------------------------------------------*/
  /* Minimum Standard Random Number Generator        */
  /* Taken from Numerical recipies in C              */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity   */
  /* Always call with negative number to initialise  */
  /*-------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  double temp;

  if (*idum <=0)
  {
    if (-(*idum) < 1)
    {
      *idum = 1;
    }else
    {
      *idum = -(*idum);
    }
    idum2=(*idum);
    for (j=NTAB+7;j>=0;j--)
    {
      k = (*idum)/IQ1;
      *idum = IA1 *(*idum-k*IQ1) - IR1*k;
      if (*idum < 0)
      {
        *idum += IM1;
      }
      if (j < NTAB)
      {
        iv[j] = *idum;
      }
    }
    iy = iv[0];
  }
  k = (*idum)/IQ1;
  *idum = IA1*(*idum-k*IQ1) - IR1*k;
  if (*idum < 0)
  {
```

```
    *idum += IM1;
  }
  k = (idum2)/IQ2;
  idum2 = IA2*(idum2-k*IQ2) - IR2*k;
  if (idum2 < 0)
  {
    idum2 += IM2;
  }
  j = iy/NDIV;
  iy=iv[j] - idum2;
  iv[j] = *idum;
  if (iy < 1)
  {
    iy += IMM1;
  }
  if ((temp=AM*iy) > RNMX)
  {
    return RNMX;
  }else
  {
    return temp;
  }
}
```