Research Capstone Project

Predicting Dublin Bus Journey Times: A Web Application

Zaur Gouliev, Sean Moiselle, Oscar Fernandez, Bo Tian

A thesis submitted in part fulfilment of the degree of

MSc. in Computer Science

UCD School of Computer Science
University College Dublin

22nd August 2022

Project Specification

Bus companies produce schedules which contain generic travel times. For example, in the Dublin Bus Schedule, the estimated travel time from Dun Laoghaire to the Phoenix Park is 61 minutes. Of course, there are many variables which determine how long the actual journey will take. Traffic conditions which are affected by the time of day, the day of the week, the month of the year and the weather play an important role in determining how long the journey will take. These factors along with the dynamic nature of the events on the road network make it difficult to efficiently plan trips on public transport modes which interact with other traffic.

This project involves analysing historic Dublin Bus data and weather data in order to create dynamic travel time estimates. Based on data analysis of historic Dublin Bus data, a system which when presented with any bus route, departure time, the day of the week, current weather condition, produces an accurate estimate of travel time for the complete route and sections of the route. Users should be able to interact with the system via a web-based interface which is optimised for mobile devices. When presented with any bus route, an origin stop and a destination stop, a time, a day of the week, current weather, the system should produce and display via the interface an accurate estimate of travel time for the selected journey

- Website: `https://oscardbf.eu/`
- Github: `https://github.com/gouliev/dublinbusapp`

Chrome:	Full Support
Edge:	Full Support
Firefox:	Full Functional
Safari:	Limited Support/Limited Functionality. Application not optimized for this browser. See Chapter 5.

Abstract

This paper presents a unique solution to users needs in relation to predicting bus journey times and general journey planning. Discussing the background research, process, technical architecture and testing conducted in order to build the unique web application, optimized for mobile devices, in order to best provide users with the tools to plan their journeys. This project was conducted during a whole semester, with a group of four individuals. Within the technical architecture, machine learning and deep learning methods are implemented to produce incredibly accurate journey prediction models with an r^2 Score 0.958. The eventual application produced was highly user accessible and available online. Within this paper a summary of contribution as well as a critical evaluation of the approach taken and technical stack utilized is provided.

Acknowledgments

The team would like to thank each other for the hard work which has been put into this project. We would like to thank Madhusanka Liyanage, Bartlomiej Siniarski, Fatemeh Golpayegani, Padraig Cunningham and supporting staff for making this module possible and supporting us throughout the semester. We'd finally like to thank our families and friends for supporting us for the duration of this project.

Table of Contents

Chapter 1: **Introduction**

1.1 Description of Problem

The primary aim of the research and project which is detailed in this paper was to create an accurate 'Bus Journey Planning' web application, optimized for mobile, utilizing historical Dublin Bus data to provide predictions. This project aimed to produce an accessible and impactful application which will enable the user to better plan their future journeys with Dublin Bus.

1.2 Discussion of Wider Context of the Problem

Providing individuals with the ability to accurately plan their daily, and unique trips would have a large and academically recognized positive impact on the user: 'Providing real-time and accurate travel time information of transit vehicles can be very helpful as it assists passengers in planning their trips to minimize waiting times.' [1]. Furthermore, providing users with this information would largely improve both 'passenger satisfaction and bus attraction' [2]. As recognized in [2] increasing the accessibility, accuracy and precision of information reduces barriers which prevent individuals from using bus services to commute. This reduction in barriers would potentially lead to an increase in bus usership, resulting in a reduction of cars on the road and reduced traffic congestion [3], which in turn would improve bus travel times [3].

However, the ability to accurately predict bus arrival times is not an easy task with many technologies and techniques being implemented to better improve academic understanding of the issues [1][2][4][5]. These technologies include Regression Models, AI Based Models, Probabilistic and Time Series Models [1][4][5]. The Variables which affect bus travel times includes but is not limited to: dwell-time [4], link-time [4], time of day[1] and weather [5] and of course traffic.

Therefore, there is a defined problem: Individuals require the ability and data to accurately plan their journey with Dublin Bus. The solution being a website, which provides users with the relevant information. Displayed utilizing a user-friendly interface optimized for mobile devices. There is a wealth of research in this specific field [1][2][4][5], which guided and provided this team with the knowledge to create the aforementioned solution .

1.3 Discussion of existing approaches and their limitations

Both Academia and Industry have contributed to this field. Academic solutions primarily focus on the underlying technology which predicts bus travel, dwell and arrival time [1] [2] [4]. While industry solutions implement a prediction technology with the addition of a user-friendly interface and accessible through the web. This project requires the implementation of a web based mobile friendly interface, as such industry solutions will be reviewed.

The most prolific solution would be Google Maps, which currently holds approximately a 70% share of monthly digital navigation users [6]. As per Google Maps own documentation [8] it implements multiple technologies into its web-stack: JavaScript, Json data, Ajax as well as the Geospatial Data Abstraction Library [7]. This provides the user with an incredibly well integrated service,

providing information pertaining to locations, landmarks and all forms of transportation; both public and private. In relation to bus data, Google maps enables the user to both use their current location or a given location as their origin; the user will then assign their destination. Google maps will then provide the user with all available bus options to reach that destination, providing information which includes: bus departure time from origin, bus arrival time at destination station and walking time between the user's destination or location and the two stops (departure station and destination station), as well as the total journey time. Google maps enables the user to gather this information in relation to a given time and day, in essence the user can check all this information days in advance if necessary.In conclusion, Google Maps is a comprehensive service, justifying its large market share.

Similar to Google maps is Apple maps, which has much of the same functionality as Google maps. Dublin-Bus's application allows users to gather information by Stop Number, Bus Route, Address and locality of the user. It will display this information in a 'timetable' format, it can also display the location of the stops on a map. It further provides features such as a Fare calculator. The primary limitations of Google and Apple maps is the requirement to create an account to unlock all of the applications features, furthermore they are stacked with multiple different features creating a UI which some may find difficult to navigate. Finally, the Dublin bus application's user interface is dated, with limited functionality when compared to Google and Apple maps.

1.4 Overview of the Project

This project was conducted as part of the Msc Computer Science Conversion program. Being a Research Practicum this project occurred over the span of a whole semester. The class was split into multiple groups. Each group was tasked with creating a web-based application, optimised for mobile, which aided users in planning their journeys through providing predictions as to the time it would take to make a given journey. These journeys would be taken with the Dublin Bus service. The journey predictions would have to be derived from models created specifically for this project, which utilized a number of variables to give a prediction. The application would then be hosted and accessible to users through the internet.

Furthermore, beyond creating the application this project required each team to work collaboratively with their members and document their work in two forms being presentations and written reports. Some of which were collaboratively made within each team and others were individual assignments. Finally this project submission goes beyond the set requirements, which is detailed in this report.

1.5 Report Structure

Chapter 1: An introduction to the report and project.

Chapter 2: Displays and Describes the features of the application, how the project specifications were met and further innovations beyond these specifications.

Chapter 3: Discusses the personnel and management approach taken in order to create the application and meet the multiple requirements of this project.

Chapter 4: An in-depth review of the 'Four Architectures' used to build the application. The *Data Analytics*, *Web Application* and *Web Hosting* Stack under the *Overall Architecture*.

Chapter 5: Covers the testing processes utilized to verify the functionality of various features. Discussing how mobile porting was tested and the testing of the various prediction models.

Chapter 2: **Description of Final Product**

2.1 Description of Functionality with Screenshots

The application design is purposefully minimalist, having a 'client-first' design. Therefore, decisions were made with the assumption that the user was unaware of how to use the application, emphasising user-friendly and accessible design variations. Due to this, login features were not implemented as they are a "barrier for entry" hiding extra functionalities behind an account requirement. The application provides the user with a tailored and feature rich experience without the use of accounts. Finally the application was built with Mobile interface in mind, but it also supports desktop. *(To see desktop examples see appendix 15.)*

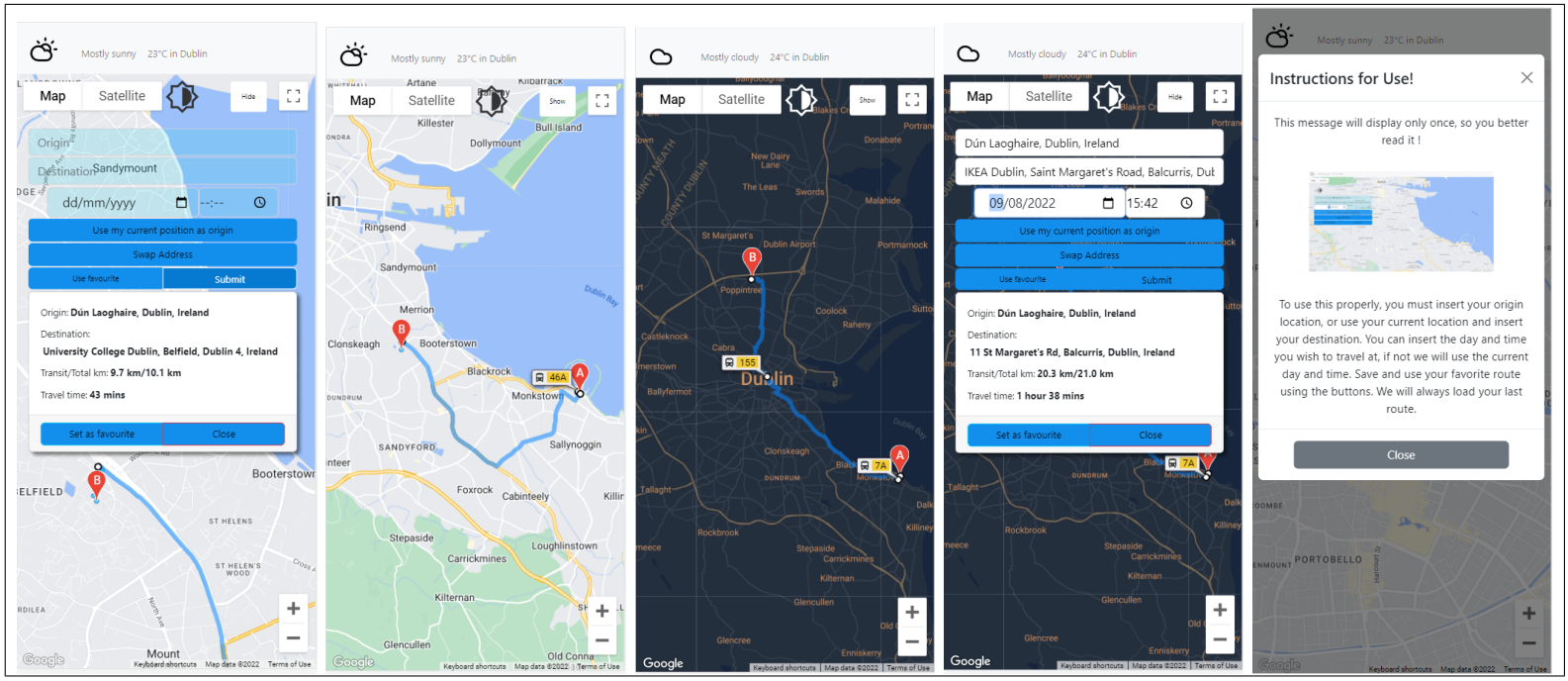


Figure 2.1: Mobile Display Variations and Features

Search and Predict Feature

The primary feature of this application is journey planning. The user must insert an origin and a destination, the application provides an autocomplete function which helps the user define their locations. The user can insert a day and time for which they wish to travel. If they do not provide a day and time, the application will use the current day and time. The user must then press submit. On submit the most efficient route per the input parameters will be displayed on the map. An information 'box' will be displayed, containing details pertaining to *Origin*, *Destination*, *Transit Distance* and *Journey Time*. Finally, this application can handle journeys which require multiple buses to complete, predicting the total journey time across all buses.

First Use Feature

On first use of the application a modal will appear, this displays instructions on how to use the application. The modal also contains a .gif displaying how to use the primary application features. Finally, this modal will *not* appear to the user again, even on reload. *(Figure 2.1)*

Last Route Feature

When the user searches for a given route, this route will be saved. On next load, or the next time the user opens the application, this route will appear on the map with the current prediction for the time it takes to make that given journey.*(See figure 2.1 for example. Displays the same way)*

Favourite Route Feature

Users can, after searching for a journey, set a this route as their 'favourite'. This route will then be saved and can be used by the user at any time, including after closing and reopening the application. When 'Use Favourite' is clicked the application will display that route on the map and provide a prediction for the current travel time. Furthermore, the user can provide a day and time and then click 'Use Favourite', the application will give a prediction for their 'favourite route' at that given day and time.*(See figure 2.1 for button. Displays the same way)*

Swap Address Feature

When the user types in an origin and a destination, they can click 'Swap Address' which will swap the two values. If they click submit and do not type anything following, the 'Swap Address' function will reverse the route and provide a new prediction in the inverse direction. Finally this is integrated with both the Favorite Route and Last Route features. *(See figure 2.1 for search result)*

Weather Feature

The application displays to the user in a 'information bar' which contains details about the current weather conditions in Dublin City. Providing the temperature, a description of the weather and an icon depicting the conditions. This bar will change colour depending on the time of day, having both a day and night mode.*(See figure 2.1 for mobile, appendix 15 for desktop)*

Dark and Light Mode

Finally, the application contains a dark and a light mode. The button, as displayed by a white or black semicircle with a geometric shape around it, when clicked will iterate between the two modes.*(See figure 2.1)*

Use current location

The application makes use of location based services, allowing the user to insert their current location as the origin, through clicking 'Use my current position as origin'. *(See figure 2.1)*

Collapse Feature

Finally in order to enhance user interaction with the application, the mobile variation of the application includes a 'hide/show' button. This button toggles the user interface between two states: display and do not display. Therefore allowing the user to engage with the map feature on mobile. *(See figure 2.1)*

2.2 Description of how project needs have been met

The specifications of this project detail just two primary requirements: The application must be ported primarily for mobile devices, and the application must be able to provide a prediction for a given journey on Dublin Bus.

Firstly the specifications require the application to have a web-based interface for which users can interact with. This interface should be optimized for mobile devices. The application as displayed meets these requirements. Being both a web based interface and one which is primarily ported for mobile devices. It can be seen that the mobile based variation displays all information in a clear and concise manner, providing the user with the tools to engage with all features of the application through the collapse feature. The weather information bar being a constant on the screen, with the user input sections and information box being 'togglable' with the use of the collapse feature. This feature allows mobile users to engage with and utilize the map and route display. This application is confirmed to work on a plethora of mobile and desktop devices which is further detailed in Chapter 5

Secondly the project specification requires the application to give a prediction of a given journey's 'travel time' when provided with a number of variables. This application will provide such a prediction, as made by our models, to the user based off of these variables in an accessible and uncomplicated manner to the user. Furthermore, this application provides predictions for journeys which require multiple buses to make. The application also provides the user with the ability to provide a given day and time to retrieve the prediction for, further expanding on this requirement.

2.3 Explicit description of innovations beyond the specification

There are a number of innovations beyond the specifications which the user can directly interact with being the 'Cookie Features' which are further discussed in Chapter 4. These features are the *First Use*, *Last Route*, *Favorite Route features*, which provide extra functionality to the user. It was the belief of the team to not stack the application with features which at their core have no relation to the core concept of providing a journey planning application. Furthermore, the self imposed minimalist design meant that these features had to be implemented without the use of accounts or storing user data on our servers.

These features were implemented using an intricate system of cookies which held the required information for these features to function. Furthermore, as seen with the Swap Address button and feature the 'Cookie Features' were fully integrated with all other features. Further complicating and increasing the difficulty of implementing these features.

As discussed in Chapter 4, Artificial Neural Networks were trained, and despite these not appearing in the final application it is the belief of the team that these should be considered a further innovation. These Neural Networks were trained utilizing cutting edge technology, being *Tensorflow* and *Keras*, requiring a heavy time and research investment to implement. They were not included in the final application as a result of our very extensive research into best predicting bus journeys, resulting in a very accurate model which produced an r^2 Score of 0.958 and MAE of 129.710. The Neural Networks did not surpass these metrics and therefore were not implemented. This is further discussed in Chapter 4.

Finally there are a number of smaller innovations beyond the specifications which were implemented, such as the *Swap Addresses*, *Use Current Location*, *Light and Dark Mode* features. Which add extra layers of functionality to the user while furthering the application's ease of use.

Chapter 3: **Development Approach**

In this chapter, we will detail the management approach which was taken and the unique alterations which were made to best meet the needs of this project. Furthermore, the management and technological tools which were used to best coordinate working on this project will be detailed.

3.1 Organizational Strategy and Management Approach

In approaching this project, the team implemented an altered variation of the Agile Scrum methodology. Agile was chosen due to being 'best suited for quick and effective development of software' [12] and the assumption that requirements will alter, which the Agile methodology accounts for in its philosophy [13]. Agile most often produces successful outcomes when implemented in small teams [13]. Within the Agile philosophy of management, the Scrum [14] methodology was implemented as it 'offers a customized way of working. . . having advantages such as flexible requirement selection' [12] and enables close collaboration between teammates [15]. The process which was implemented was chosen to maximise coordination and flexibility while minimizing isolating each member and task. Alterations were made to the Agile Scrum process to best meet the specific aims of this project, therefore a number of Agile Scrum principles were implemented in unique ways; customization of these methodologies based on project requirements has been shown to be beneficial [15] by practitioners in both industry and academia.

3.1.1 Implemented Scrum Roles

The scrum workflow requires the assignment of three roles *Scrum Master*, *Product Owner*, and *Scrum team* [12][13][15]. The *Scrum Master* is responsible for ensuring that each Scrum meeting and sprint yields the highest return [15], they are not responsible for 'what' is to be done but 'how' it is done [16, p13]. The *Product Owner* represents the customer, they are responsible that the 'correct' product is built to meet the customer's needs, in essence they are the 'what' is to be built through managing the product backlog [15]. Finally, the *Scrum Team's* responsible for bringing the product to life, working collaboratively and effectively; guided by the *Scrum Master* and the *Product Owner*. During team assignment positions were given to each member by the university; these positions provided a base for assigning these roles:

Scrum Master: Assigned to 'Project Coordinator' – Sean Moiselle

Product Owner: Assigned to 'Customer Adviser' – Tian Bo

Scrum Team: Assigned to 'Code Lead' and 'Maintenance' – Oscar De Burca Fernandez, Zaur Gouliev

Alterations were made in the general implementation of these roles to best fit our needs. For example, there was collaboration between the *Product Owner* and *Scrum Master* in discussing what would be included in each sprint's backlog, a task traditionally assigned exclusively to the *Product Owner* [15]. The *Scrum Master* and the *Product Owner* would discuss what could realistically be completed in each sprint. This enabled more efficient completion of each sprint, as tasks and features were built at the appropriate time to do so and product backlogs were appropriately assigned. This alteration to the Agile Scrum process has shown to be effective in academic research and is considered a positive alteration [15].

Furthermore, the *Scrum Master* engaged in each aspect of the project, enabling accurate tracking and efficient completion of each task. Usually the *Scrum Master* would change each sprint. This was not implemented, the team felt the assigned positions guided the general role each member had to fulfil. Having a single *Scrum Master* would lead to a smoother transition between sprints.

3.1.2 Implemented Scrum Workflow, Meetings and Communication

The Agile Scrum workflow is split into sprints, which last between 1 to 3 weeks [12]. The outcomes of each sprint is dictated by a product backlog [12][15]. Daily meetings or 'scrums' are held in order to keep the project on course and improve or implement various features of the product. Sprint planning and retrospectives are conducted in order to properly structure and end each sprint [12]. *(See appendix 14)*. In implementation the team followed the following sprint structure:

Sprint Planning Meeting: At the beginning of each sprint a meeting was conducted in which the team would plan the sprint which is about to commence. The outcome of these meetings was a commitment by the team to meet the product backlog. These meetings were often guided by the *Scrum Master* and the *Product Owner.* Detailing what had to be done, how it was going to be done and whom it was to be done by. These were implemented as it is considered best practise in the Agile Framework to conduct these meetings and render positive outcomes [15].

Sprint: The team agreed that the length of each sprint would amount to two weeks, keeping with best practises [12]. A week in each sprint was defined as Monday to Sunday, weekend inclusive. During a sprint, each member of the team had the authority to work independently on their tasks. During each sprint, member's tasks progression were tracked using YouTrack.

Bi-Daily Scrum: As per the Agile Scrum methodology, a daily scrum meeting is required [12][13]. However as discussed in [15] effective implementation of Agile Scrum does not inherently require daily meetings. Thus, the team agreed to Bi-Daily meetings, or a meeting every second day. These meetings ran regardless of the day it fell, Scrum meetings were held on weekends too. The logic behind applying this meeting scheduling was in relation to work completed, the team agreed that daily scrum meetings often were inefficient as not enough work could be completed on a day-to-day basis which would be relevant to report. Furthermore, if urgent issues arose and required immediate attention, the team agreed an extra meeting could be called on an 'off-day'.

Meetings ranged between 15 to 45 minutes, topics to be discussed were assigned by the *Scrum Master*. The time variation was affected by whether an individual needed help. If this were the case multiple members would review the problem and arrange a 'pair programming' session. All meetings were held through Google-Meets. Tasks and features were moved from the *Submitted* progression state to the *Verified* progression state upon group review during each meeting. Finally, each meeting would end with a review of the *Burndown Chart*.

Sprint Retrospective: Sprint retrospectives are considered an integral part of the Agile Scrum Process [12], these provide the team with a structured feedback process on the previous sprint evaluating effectiveness and efficiency [15]. For this project the team implemented these upon the conclusion of each sprint, where each member would detail what they have done and often the complexities which other members need to be aware of. As an alteration to the traditional implementation of these retrospectives, the team also provided feedback to each other based on their performance and attendance.

Pair programming: Pair programming is the process by which two people code the same problem congruently [17] and has been shown to produce 'better code, faster' [17]. The team implemented this method under two circumstances: A task has proven to be too difficult for one member and to conduct Data Analysis. The logic behind Pair-Programming the data analysis was to produce the

best possible result from what we felt was the core concept of this project, predicting Bus-Arrival times. We believe the implementation of this uncommon [17] programming technique paid off.

To conclude Scrum Workflow, the team implemented all of the above techniques after researching the Agile Scrum methodology and altering it to fit the project needs. It is worth noting that often a Sprint Review Meeting [15] is listed as a part of the workflow, however as this project had no inherent stakeholders apart from the members of this team who are directly involved in its production this meeting was deemed to be non-beneficial and was never implemented.

3.1.3 Artifacts and Tools Utilized

Artifacts refers to any form documentation which has been utilized in tracking and planning the process which was implemented for this project [15]. Artifacts and the relevant tools to store and create them will be covered congruently.

Sprint Product Backlog: The *product backlog* is a list of all the features which are to be completed and implemented by the completion of a given project [12][15]. While a *sprint backlog* is a list of all features and tasks which are to be completed by the end of a given sprint [15]. In implementation the team utilized 'Youtrack' by JetBrains as it has built in Back-Log functionality. This gave each member access to view the product backlog, and adequately plan the following sprint based on the amount of items and contents of those items in the backlog.

Burndown Charts and Progression Tables: Burndown Charts are tools utilized to measure the progression of each sprint [12][15]. Through utilizing YouTrack, burndown charts were generated based on the 'Time To Complete' values which were applied to each task/feature. The progression table provided by YouTrack allowed for each member to see the current progress state each task or feature was in. Our table had four states: *Open, In Progress, Submitted* and *Verified. Open* indicated a task which was to be completed this sprint but not currently being worked on. *In Progress* indicated a task which was being currently worked on, while *Submitted* was a task which was completed by the individual assigned it and *Verified* was a task which was reviewed by the group during a scrum and confirmed to be completed correctly.*(See Appendix 0 and 2)*

Meeting Planning and Minutes: Meeting minutes were kept for each *Bi-Daily scrum*. These minutes were stored in a document which was shared to each member through Google Drive. Furthermore, minutes were stored in similar documents for the *Sprint-Retrospective* and the *Sprint Planning Meeting*. Allowing for each member to revisit what was agreed upon and discussed after each meeting. *(See Appendix 2 for an example.)*

Github and Code-Sharing: Naturally being a programming research project, code sharing was a central requirement. We utilized a Task-Based branching procedure on Github to share code [18][19], allowing for easy tracking and maintenance of the code base as well as encouraging collaboration. *(See Appendix 6 for branching procedure.)*

Systems Requirements Specification: An SRS was created to help guide the early development stages of the project. *(See appendix 13 for example page.)*

3.1.4 Conflict resolution

An unfortunate reality of working in a group is that conflicts between members occurs. During this project, there were instances of disagreements, however they were handled during group discussions, in particular scrum meetings, to encourage each member to speak their mind on the issue. These disagreements were always solved and the outcome best for the Team as a whole was usually the course of action taken.

Chapter 4: **Technical Approach**

To best discuss the technical approach implemented, each Sub-Section of this chapter is split into three distinct sections: *Data Analytics, Web Application* and *Application Hosting*. While *'Description of the Architecture of the System'* also contains a *'Overall Architecture'* section which details how each of the three sub-architectures/stacks interacts. The technical stack implemented for unit testing is discussed in *Chapter 5*.

4.1 Description of the architecture of the system

First we will discuss at a high level the *'Overall Architecture'* which encompasses, *the Data Analytics, Web Application, Web Hosting* and how they broadly interact with each other. Initially

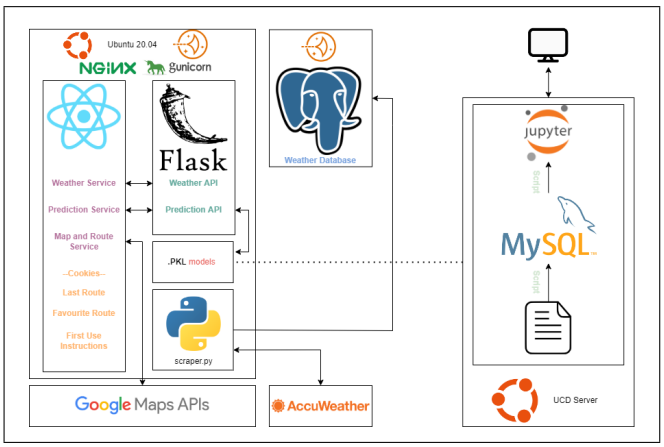


Figure 4.1: Overall Architecture

it can be seen that *Data Analytics* has its own unique stack, this was a result of the project requirements stipulating the provided data must be kept on the UCD server. This technical stack enabled us to conduct Data Cleaning, Analysis and Model Creation within the constraints of the requirements. The results of this tech stack, being the models for predicting bus journey times were then utilized as a part of the *Web Application*. The *Web Application* is hosted using the *Application Hosting stack*, which includes the hosting of the *PostgreSQL* Database.

4.1.1 Data Analytics

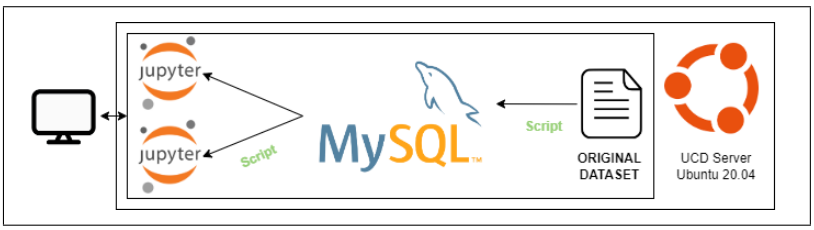


Figure 4.2: Data Analytics Architecture

As part of this project, we were provided with a dataset containing information pertaining to Dublin Bus. As per the regulations surrounding the handling of this data, it was required to be exclusively stored on UCD servers, this requirement was adhered to. This dataset was initially

stored in a csv file where columns are separated by semicolons and rows by newline. This was deemed to be less than beneficial, and the format lacking the functionality a traditional database could provide. Therefore a script was created which inserted the csv files line by line into a *MySQL* Database, this was done as the memory of the provided UCD VM was not large enough to load all the data at once.

Jupyter Notebooks were utilized to do data cleaning, model analysis and model creation. This tool would be utilized through a local browser, which connected to the Jupyter Notebook application hosted on the remote UCD server. This was done through specifying the port to be used during the creation of the SSH connection: port 8888. Therefore specifying that any connections to the local port 8888 were to be forwarded to the port 8888 on the remote server over the secure SSH channel. A bash script was created on the server which activated the correct conda environment and started a Jupyter Notebook on port 8888. Therefore streamlining the start-up and connection process.

With regards to cleaning data, another python script was created. This operated by loading a reduced amount of data, being one bus route in one direction, into a pandas dataframe. This was done by connecting to the previously mentioned *MySQL* database and calling a stored procedure which was created for this purpose. This script would manage the following identified logical issues in the data:

Incorrect First and Last Stop: This script identified the mode (most frequently occurring) first and last stop. These were accepted to be the true beginning and end stations. It would then remove rows which belonged to trips that began at, or finished at stops which were not the identified modes.

Reverse Time Journeys: Some journeys were recorded going backwards in time. These were identified and removed.

New features were also added such as *hour of day, day of week, month of year* and *journey time*. *Journey Time* refers to the time taken to reach a given station or stop from the start of the trip. In essence it is the cumulative amount of time taken to reach a given stop since the beginning of a given bus's journey. This feature was our target feature. *Hourly weather data,* obtained from Met Éireann, was loaded into a dataframe and joined onto the data by the script. Finally, the script removed features that wouldn't make sense in aiding the predictive ability of a given model on the target feature.

The remaining features and data would then be examined through utilizing the previously mentioned Jupyter Notebooks. As discussed in Chapter 3, multiple Jupyter Notebooks were created by different members for the purpose of better analysis and model creation. Upon further examination of the data, it had been shown that in relation to the target feature of *Journey Time* the predictive abilities of *Month of Year, Day of Week, Hour of Day* and *Progress Number* were incredibly strong. *Progress Number* being an integer which represents the cumulative count of that given stop in relation to the journey, for example the 10th stop in the journey would be given the *Progress Number value* of 10. In essence we actually are predicting the amount of time which has passed since the bus left the first station to a given *Progress Number* (or station), eg 600 seconds to get to station 10. *(See Appendix 8 for Progress Number Feature Importance)*

After the creation of multiple models, utilizing packages such as *Sklearn, Tensorflow* and *Keras*, with various different fundamental approaches (further discussed in Section 4.3), after comparing these models a Random Forest Regressor Model proved to have the best overall performance, with the following metrics *(See appendix 11 for screenshot)*:

r^2 Score	0.958
Mean Squared Error	202.811
RMSE	11.259
Mean Absolute Error	129.710

Table 1: Random Forest Metrics

Finally, in exporting these models there were initial difficulties with the size of each .pkl file. Given that we had 246 models, two for each bus route going in either direction, initial individual .pkl file size often exceeded 500MB. This is a result of the chosen model naturally producing a large .pkl file size without proper 'reduction techniques' therefore through iterative attempts to reduce the file size without encountering any loss of accuracy we managed to produce a set of models which had an acceptable size of 5GB in total across all 246 models, the average metrics of these models are shown in Table 1. This was done through setting the following parameters to each Random Forrest Regressor Model:

*n*Estimators: 5	maxFeatures: Auto	oobScore: False	randomState: 1	max depth: 20

Table 2: Random Forest Parameters

The result of this *'Data Analytics Architecture'* being the models, was utilized as part of the *'Web Application Architecture'*.

4.1.2 Web Application

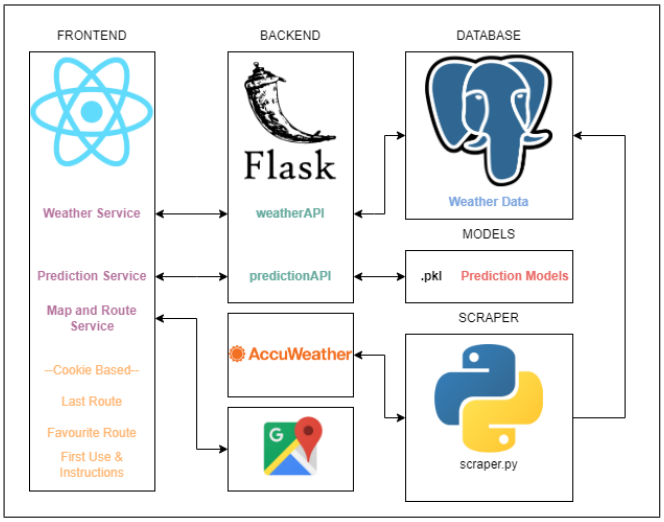


Figure 4.3: Web Application Architecture

In best describing the architecture of the Web-Application, it is best to review each front-end feature and its supporting back-end operations. It is also beneficial to initially recognize that the front-end framework React is divided into several parts. One such part being the public folder, which contains the only html file. Leading to our code being created in a div node with an id of root. A virtual DOM is then made and mapped to the div node. References will be made towards the file structure which can be seen in *Appendix 10*.

A *context* file containing variables which we want to be available globally in the frontend was created in the *src* folder. Due to React passing variables between different components, in the form of passing parent components to descendant components [27] it was required that between the many 'sibling' components a common parent was needed. This allowed us to transfer

variables globally. The *hook file* corresponding to this *context file* enabled our current architecture implementation and also made our code better encapsulated. The *index.js* file is where we map the virtual DOM to the real DOM. *App.js* is the assembly area for all our components. This ability to assemble components within a small file is a reflection of the reusability of React code. Finally, we would like to point out that our code follows very strictly to OOP fundamentals which can be seen in the GitHub, particularly the *Flask* python code.

Map and Route Feature

The search box and map/route component are independent of the weather component, contained within *MapandSearchBox.js*. Google Maps Api is utilized to generate the map which is displayed to the user. Furthermore the search box makes use of this same api in order to provide automatic search completion, for both the origin and the destination. Finally, we make use of geolocation packages to ascertain the current location of the user for utilization as the origin. Form input components are utilized to enable the user to insert the date and time of their journey, if no date or time is selected then the current time is utilized.

Utilizing the integrated technologies contained within the Google Maps api, a request is made for providing the user with the best bus journey at that given time. This journey's route, or bus route is then displayed on the map utilizing the api. It is important to note, no prediction pertaining to the travel time of the bus is ascertained from this API, given that the bus route's data was within the provided dataset.

Prediction Feature

Upon a request being made to the Google Maps Api, information is ascertained from the response. This information is then utilized to make a request to our own Flask api, containing all the necessary information for our models to function: *Iterator, Route, Direction, Stations, Day, Month, Hour*. This causes the application to call the method *jsonPrediction* which belongs to the class Prediction. jsonPrediction activates a multi-step process utilizing multiple class methods: First cleaning the data, Second checking if it can predict this route through ascertaining if a given model exists, Third open relevant .pkl file and utilize the *.predict* function from *sklearn* to get a prediction from that given model. From here the response is formatted into JSON data, containing the route, direction, prediction and is sent back to the frontend. *(See Appendix 3 for JSON)*

Upon receiving the response, the information contained within the JSON data is then utilized to populate the *info.js* variables within the *React* application which are displayed to the user. If there is a distance which is to be walked by the user, the time it takes to do this would be added onto our prediction utilizing the Google Maps API. This process operates for journeys which require multiple bus routes to complete, providing predictions for the total journey time through making multiple API calls. As previously mentioned the React structure allows for parent-child relationships to exist, in this application the information component which displays our prediction and various other data points pertaining to the journey is a child of the *MapandSearchBox.js* file and contained within the *info.js* file.

Weather Feature

The weather service is automatically loaded upon opening the application, as discussed in Chapter 2, it displays the current weather conditions in Dublin. The first step to the functioning of this service is a python script: *scraper.py*. This script is independent of the *Flask* application and is responsible for calling the AccuWeather API at the correct intervals. This is due to the AccuWeather free tier being restricted to a set number of calls a day. Once an hour it calls the static method *insertData* which is a part of the *Weather* class contained in *models.py*. This method deletes the contents of the table *'Weather'* in the *PostgreSQL Database* and inserts the

current weather conditions. To pass data to the front end the method *jsonWeather* which is part of the class *WeatherAPI* in the file *jsonData* was created, contained within the Flask application. Which when called by a web application through the correct URL, made accessible by the function *weather* in the *routes.py* file, would query the postgreSQL database for the most current weather information, format this information to JSON data and respond to the request with this JSON data through the *weather* function in *routes.py*. This response is then utilized by the front-end to display the current weather. *(Appendix 4 for example JSON)*

In addition, there is also an *img folder* contained in the *public folder* of the *React* frontend, which holds images corresponding to each weather type. When the weather API responds with a weather type, the corresponding 'weather condition' image will be identified and displayed on the frontend. This weather component, and other asynchronous programming models in the frontend, are consistent with typical and compliant asynchronous implementations [26]. Finally utilizing auxiliary files in the src folder, we can change these 'weather condition' images to fit with our 'night mode' which activates after a certain time of day.

Cookie Based Features

There are a number of features, which are to be regarded as "extra features" or "further research". These features rely on the use of cookies, which are stored within the user's browser.

First use: This feature detects whether a cookie exists within the user's web browser, if the cookie does not exist it will provide the user with instructions on how to properly utilize the application. The instructions are displayed utilizing an independent component containing a modal.

Last Route: This feature saves the last route which was inserted by the user in a cookie. Upon opening the web-application a check will be made for two cookies, one containing the destination and the other the origin. If they exist this data will be utilized to make a requests to the Google Api and subsequently the Prediction Api. This feature operates as part of the MapandSearchBox.js component.

Favorite Route: This feature allows the user to save their favorite route in a cookie. The user is then able to quickly get predictions pertaining to that specific route, as well as reversing the route i.e swapping origin and destination. This feature is also a part of the MapandSearchBox.js and Info.js components.

4.1.3 Application Hosting

The application hosting is done through utilizing *AWS Lightsail* to host our servers and data, due to the free tier carrying good and secure cost options. Secure, when compared to EC2 instances which can often carry with them hidden costs.

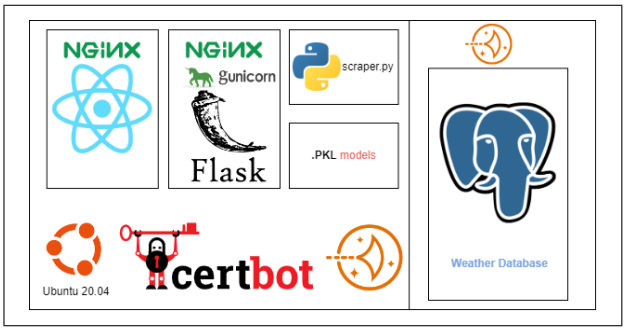


Figure 4.4: Web Hosting Architecture

Our hosting makes use of *ubuntu 20.04* instances for the web application. Within this a number

of software services were made use of primarily *NGINX* and *Gunicorn*, supported by *Supervisor*, *Certbot* and *Uncomplicated Firewall*. The *PostgreSQL* database is hosted on another *AWS Lightsail* instance, this allows for easy integration of the application and the required database.

4.2 Description of the technical stack utilised

4.2.1 Data Analytics

Ubuntu 20.04: The server provided to use was an *Ubuntu 20.04* container hosted on UCD servers. This was utilized to host the various programs and scripts needed to conduct Data Analysis.

MySQL: A *mySQL* database which was hosted on the *Ubuntu* container was used to store the data in a more accessible and manageable format than the initial storage state of the dataset.

Jupyter Notebooks: *Jupyter Notebooks* were used to conduct Data Analysis and model creation. These notebooks were hosted on the *Ubuntu* server and accessed through ssh from a local computer.

Packages: As a subset of the *Jupyter Notebooks* a number of packages were utilized to conduct analysis and model creation: *Pandas* and *Numpy* for analysis and cleaning. *Sklearn* for model creation. *Tensorflow* and *Keras* for deep learning and Artificial Neural Networks. These packages were managed using conda.

Custom Scripts: Custom scripts were created for connecting to *Jupyter Notebooks* from a local machine to the *Ubuntu* container. Furthermore scripts were created for cleaning the data and inserting it into the *MySQL* database.

4.2.2 Web Application

React: *React 18* was utilized to construct the front-end through implementing multiple components which are roughly correlated to each individual feature the application provides the end user. Split into four primary components: *Weather Service, Prediction Service, Map* and *Route Service, Cookie Based Services.*

Front-End Technologies: Other technologies which support the *React* Frontend are *CSS, Bootstrap* and *HTML. HTML* carries the function of DOM and browser interaction.

Flask: *Flask 2.2.1* was implemented for the backend. It harbors three main purposes being the handling of the *Weather API, Prediction API* and declaring the Weather Class for use in the *PostgreSQL* Database.

PostgreSQL: Version 12.10 of *PostgreSQL* was utilized to store data for use in the *Weather API*.

Prediction Models: 246 *Random Forrest* models (.pkl files) were created, two for each route. Essentially one for each direction of every route. These models were utilized to provide predictions for the user.

Python Scripts: A python script named *scraper.py* was used to properly time the calling of the static method *insertData* which belonged to the *Weather* class. Effectively it called the method which made an API query and stored that query into the *PostgreSQL* Database.

External API's: Two external API's were used to provide various functionalities. *Google Maps* api provided search completion, map display and route selection. While *AccuWeather* provided

weather conditions for the Weather API.

4.2.3 Application Hosting

AWS Lightsail: *AWS lightsail* is a service which provides two products of value to this project: *Ubuntu* instances and *PostgreSQL*. It provides extra functionality with regards to the *Ubuntu* instances, allowing users to take 'snapshots' of the instance [36]. Effectively providing a built in back-up feature. Furthermore, it provides easy connectivity between *PostgresSQL* databases and *Ubuntu* instances without the need of exposing the database to the public.

Ubuntu 20.04: Lightsail provides free Ubuntu 20.04 instances with 2GB of memory 1vCPU 60GB of SSD storage and 3 TB of transfer speed.

NGINX: The application makes use of *NGINX* as a reverse proxy server and web serving [35]. Utilized for both the *React* and *Flask* applications.

Gunicorn: Utilized to provide the application with a Python WSGI HTTP Server for Unix [37]. This service was provided to the *Flask* application exclusively.

Supervisor: Implemented to monitor and control a number of processes on the Ubuntu server [38], including the *Flask* application.

Uncomplicated Firewall: The default firewall configuration tool for Ubuntu, utilized to handle connectivity rules such as allowing https connections [39].

Certbot: Utilized to provide the server with SSL certificates in order to allow for a HTTPS connection [40], required for implementing location based services.

4.3 Alternative design and technology solutions

4.3.1 Data Analytics

Database: We considered using a non-relational database such as *MongoDB* [28] or even foregoing the database and working with the csv files directly, possibly using grep commands as a substitute to SQL queries in a database. However the data provided to us was inherently relational, therefore formatting it for a non-relational database such as *MongoDB* would add extra overhead which was deemed unnecessary. While working directly with csv files proved to be non-satisfactory for the functionality we required.

Data Analysis: For Data analysis and preparation we reviewed multiple libraries such as *PyTables*, *h5py, Tabel* and *pandas*. The alternatives to *scikit-learn* as a machine learning library are *mlxtend, mlpy* and *Shogun*. These libraries provided the same functionality with less online support than *pandas* and *scikit-learn*.

Model Creation: As noted previously, two different conceptual methodologies were implemented for model creation. One which aimed to predict total journey time, and the other which predicted journey time up until a given stop. If total journey time was found to be more accurate than up to a given stop; we would then use this prediction and divide the total journey time by the amount of stops travelled. This conceptual approach rendered the following metrics:

Linear Regression	r^2 Score: 0.45	MAE: 386.30	RMSE: 515.54
Random Forest	r^2 Score: 0.33	MAE: 406.50	RMSE: 572.44
K-Nearest	r^2 Score: 0.044	MAE: 522.70	RMSE: 683.96

Table 3: Total Journey Metrics

Within the second conceptual methodology, which was predicting journey time up to a given stop, we eventually chose Random Forest. However, multiple other models were tested as well. These models rendered the following metrics:

Linear Regression	r^2 Score: 0.88	MAE: 543.21	RMSE: 19.64
K-Nearest	r^2 Score: 0.92	MAE: 298.10	RMSE: 351.69
Decision Tree	r^2 Score: 0.88	MAE: 377.46	RMSE: 536.612

Table 4: Between Stop Metrics

It is to be noted that the following is to be considered further research which goes beyond the requirements of this project. After reading multiple papers which implement Neural Networks [2][5][11] we implemented and tested Deep Learning techniques and Neural Networks utilizing libraries *Tensorflow* and *Keras*. This was done with the aim to best predict bus cumulative bus journey time, or journey till a given stop. This was done twice with the same optimizer being SGD or stochastic gradient descent [30] with different *epoch, activation* (eg. sigmoid), *loss* parameters *(See Appendix 12)*. The eventual results were:

ANN 1:	MAE: 437.19
ANN 2:	MAE: 414.31

Table 5: Neural Network Metrics.

These results display a relatively ok performance of the Artificial Neural Network, it is the belief of the team that the neural networks could have been further improved. However with no formal education in the field it was deemed to be beyond our grasp to achieve this goal, as the time and effort investment was rapidly increasing with minimal reduction in Mean Absolute Error.

Finally, alternative features were examined that didn't make it into the final model were dwell time, temperature, rain, humidity and pressure. Despite their initial consideration it was found that they were non-beneficial to the goal of prediction, this was previously recognized and described in academic papers [1].

4.3.2 Web Application

In relation to the web application, the only major competing front-end technologies which was considered was Angular.js. Angular is a 'popular JavaScript framework based on the model - view - controller pattern to construct single page web applications' [31]. Its general support and use has greatly been reduced in recent years [25]. Furthermore, it can also encourage applications to suffer from various technical problems including Unnecessary two-way data binding and wrong watching strategy [31].

MySQL was considered and had been used previously in this project within the *Data Analytics architecture*. It is a purely relational database, in contrast to *PostgreSQL* which features object-relational capabilities.

In relation to the back-end, the team had agreed upon utilizing a *python* based framework. This left us with two major frameworks, and while *Django* was considered it was eventually decided against. *Django* is a monolithic python framework which makes a point of re-usability. Resulting in less code and rapid development, providing a secure and scalable base for any application [32]. In contrast to Angular.js, this framework is incredibly popular, but due to being monolithic it is harder to tailor to the needs of a smaller project. Furthermore it provided tools which the team, upon review, knew we would not make full use of such as the Django Administration feature.

4.3.3 Application Hosting

The team only ever considered utilizing two variations on the eventual hosting structure. Utilizing the provided UCD virtual machine was considered and had the following metrics: 6GB of memory, 4 cores, 80GB of disk space and ran on *Ubuntu 20.04*. *AWS lightsail* was used instead.

Docker was considered for utilization in hosting, *Docker* is utilized for application containerization [34]. The result is an increase in robustness and resilience of the application [34].

4.4 Justification for design and technology decisions

4.4.1 Data Analytics

To begin, as per the requirements of utilizing the provided data we were permitted to only use the Dublin Bus dataset on the UCD network/servers. Therefore the decision to utilize the *UCD Ubuntu Container* was one primarily driven by 'administration' requirements. However, it is to be noted that the provided server reached our expectations and needs in order to best collaborate on the Data Analytics aspect of this project.

For storing the data a *MySQL* database was chosen, a relational database. We decided against a non-relational database such as *MongoDB* as they "don't use relations (tables) as its storage structure"[20], whereas the data provided was structured in a relational manner. Thus after review, the time investment into converting the data into a non-relational format was deemed to outweigh any benefits it may provide. *MySQL* was chosen as it is "extensively used", "popular" and open source[20], which would ensure a high likelihood that any issues we run into have already been encountered and solved. It is also considered a "fast" and "light weight system" [20]. We discarded the option of going without a database as choosing a database utilizing the "relational model is beneficial when it comes to reliability, flexibility [and] robustness" [20], all of these attributes we desired for our data storage solution.

To conduct operations on the data: analysis, cleaning and model creation, we chose Jupyter Notebook as it is the "de facto standard" tool used by data scientists[21]. We also needed to be able to share different approaches and collaborate on ideas when doing the data exploration and analysis, a task where "[Jupyter] notebooks excel"[21]. Finally, we were able to utilize Jupyter Notebooks entirely on the *UCD Server* by connecting to them through an SSH channel. It is for these reasons that Jupyter Notebooks was chosen.

Pandas and *Numpy* were chosen for data cleaning and analysis as they are the standard libraries in python for these tasks. *Pandas* 'has a strong community support, a rich offer of functionalities and no serious competition', and as a result is strongly recommended[22]. For creating the models we used the *scikit-learn* library as it 'is the best library in the field' with 'a large scope of implemented algorithms' [22]. Furthermore, *Tensorflow* and *Keras* were utilized for implementing deep learning and Artificial Neural Networks. *Tensorflow* was chosen due to its focus on deep learning techniques and ability to create Artificial Neural Networks [41], while *Keras* was utilized as it provides the python interface for which Tensorflow is to be implemented with [42].

Finally, the outcome of this technical stack being the *Predictive Models* were in the form of *Random Forest* models following the conceptual approach of predicting the journey time to each stop (as opposed to whole journey prediction), this due to their accuracy and various other metrics outperforming the competing models. As displayed in Table 1 and Tables 4,5 6 it can be concluded that the average r^2 score of 0.958 and MAE of 129.710 which the *Random Forest* produced far exceeded all competing models including the *Artificial Neural Networks*. These results numerically justify the decision to implement the *Random Forest models*.

4.4.2 Web Application

The *React* framework was implemented, specifically *React 18*. *React* is the most widely utilized front-end framework with one of the most active communities[27]; contributing to a high level of support. Furthermore, *React allows* for code reuse it enables ease of implementation of this concept through its component based architecture [43]. *React* is a high-performance technology which uses a number of techniques to reduce the number of DOM operations to update the User Interface [43]. *React* uses the virtual DOM method which updates only the sections of the website which are different from the original DOM, reducing the number of operations. The justification for implementing React is its focus on high-performance and low load times, which increase user satisfaction and its high level of support online enabling and supporting our production.

Flask was utilized because it is a micro web-framework, allowing us to effectively pick and choose what features or tools we require to build the application [44]. This contrasts with *Django* which is a more monolithic framework [32] bringing with it more features, however these features would go unused and as a whole the framework would be 'heavier'. Therefore for the purposes of being an API handler, which is what our backend is, *Flask* proved to be the better choice. Furthermore, there was a personnel aspect to choosing *Flask* as each member of the team had prior experience with the framework. We believed choosing it would enable us to implement it easier and divert time resources to areas wherein we had less experience such as *React, Deep Learning, Artificial Neural Networks* and *PostgreSQL*.

PostgreSQL was implemented due to being an object-relational database [45], allowing for additional custom functionalities to be created by the team for our unique tasks on top of traditional SQL operations. The ability to method overload and create unique aggregate functions [45] were originally viewed as some of the major justifications for implementing *PostgresSQL*. In practise, custom non-aggregate functions were implemented. Finally, *PostgreSQL* is one of the two database management systems which is supported by *AWS Lightsail* therefore it can be said that, to a certain degree, it is also justified by its ease of deployment and hosting. There exists symbiotic relationship between the justifications of *PostgreSQL* and *AWS Lightsail*.

4.4.3 Application Hosting

AWS Lightsail and *Ubuntu 20.04* were chosen; as combined they had additional functionalities such as 'snapshotting' and ease of integration with *PostgreSQL* instances [36] which other options, such as the UCD servers, did not provide. Furthermore, the three free *Lightsail* tiers provided us with more than adequate resources. After extensive use of the UCD server it was also deemed inappropriate to utilize it to host our application, as multiple processes could interfere with the deployment said application. Finally, we also chose *AWS Lightsail* due to its integration with *AWS Route 53* which is a domain providing service. A domain is required for a HTTPS connection, which was needed for location based services.

NGINX was chosen due to being industry standard, widely used and having a lot of online support with more than adequate documentation [35]. It is also a streamlined software, which is easy to deploy and configure. Furthermore, *Gunicorn* was implemented as it is one of the leading Python WSGI HTTP Server for Unix [37] based systems; which *Ubuntu* is.

Finally, *Docker* was not implemented, it is important to detail why. While the team intended to implement *Docker* we encountered issues with it, in particular its interactions with our models as well as the *Flask* backend proved to be too troublesome for the benefit it provided. In effect, the models would create for an incredibly slow dockerizing process due to their size and once complete we would encounter numerous unseen environmental problems as a result of *Docker's* interactions with *Flask*. While we understand that these issues could be overcome, the time investment was deemed to be too great for the positive returns of implementing *Docker*.

Chapter 5: **Testing and Evaluation**

5.1 Unit Testing

First we will discuss the purpose behind testing and evaluating the functionality of the application. The primary reason for conducting this testing was to identify bugs which currently exist in the application, allowing the team to handle any issues which may arise. In essence, due to conducting unit testing we can ascertain if a feature functions correctly under all or most circumstances, and if not the process of unit testing will reveal under which conditions this particular feature or functionality stopped operating. Following this, code performance can also be measured using unit testing techniques, allowing the team to identify 'spaghetti code' and from there optimize and refine the code.

In short, a unit test has the main purpose of isolating a piece of code in our React and Flask app, then testing it to determine if it works as it is intended to do so. It can detect any flaws that will need to be rectified. The need for unit testing is one which is of utmost importance, the time that is spent making a web application can all go to waste if it is not functional outside of the local development environment and even more so it can go to waste if something crashes that was missed by the development team, this essentially could make the web application redundant. During the development of this project we implemented the unit testing libraries *Selenium* and *Jest* as well as *pytest*. Finally hosted application performance testing was conducted.

5.1.1 Selenium Testing

Selenium is an open-source tool that automates web browsers [46]. It can be considered a single interface and it lets us write test scripts in many languages such as Python and Java. *Selenium Web-Driver* executes scripts through browser specific drivers. It contains the entire library, driver, framework and the API [46]*Selenium* was implemented to perform tests from the perspective of the user, scripts were created which tested each feature:

Search and Prediction: Tested through iteratively changing the Origin and Destination.

Set Favorite and Use Favorite: Set and use various different favorite routes.

Dark and Light mode: Iterate through the two modes

Swap Addresses: Iterate between the two modes and get prediction.

5.1.2 Pytest Testing

Upon review, the team considered it best to implement a number of tests utilizing more customizable scripts with *pytest*. *Pytest* allows for us to write small, readable tests which are also highly scalable [48]. The two cases wherein these were utilized was to ensure the proper functionality of the *Flask API's*, being the weather and the bus prediction services. This was done to directly test the operational capabilities of these API's directly without the intermediary step of the *React* framework.

The bus prediction service has a total of $2^{24.3345}$, or roughly twenty-one million, possible predictions which could be provided based on the amount of possible combinations of each variable and given that the average bus route has fifty-five stops. Testing each possible combination was considered, however after review if each model responded within one second it would take 244 days to test each possible combination. This is an unfeasible task. Therefore a very small subset of possible predictions was tested utilizing *pytest*.Upon conducting these tests it was concluded that our API's are very robust, and operate under all circumstances which we could account for. The weather API responded correctly and within an acceptable amount of time to each call, while the prediction service responded correctly and efficiently to each of our subset of possible predictions.

5.1.3 React Jest Testing

Jest is a JavaScript testing framework which works with Node.js and React. Jest is a safe, fast and versatile testing library [47]. Jest was primarily used for 'snapshot' testing which is a process which confirms the robustness of the user interface, ensuring it does not change in unexpected ways [47]. A typical snapshot test case renders a UI component, takes a snapshot, then compares it to a reference snapshot file stored alongside the test. The test will fail if the two snapshots do not match: either the change is unexpected, or the reference snapshot needs to be updated to the new version of the UI component [47].

Jest was implemented to test both the desktop and mobile interfaces of this application. In particular it was used to test the robustness of the form wherein users would input the required information to garner a prediction time. This was done due to that particular aspect of the interface being the essential interactive component of the application.

5.2 Human Interaction Testing

On top of all of these automated tests, the team also conducted their own usability tests.This refers to each team member utilizing the application across multiple platforms to ensure it functions as intended. The team used desktop browser based mobile emulators, such as those provided by edge and chrome, to test across various screen sizes and devices. Furthermore, native user tests were conducted on our own mobile devices, therefore testing on as many platforms as accessible to the team. Due to no members having a Mac, it was only found in the final days of the project that our application does not fully support desktop Safari, this required us to rely on external prediction services for this browser. Firefox was found to have a minor UI difference to chromium based browsers.

5.3 Model Testing

Finally conducting testing with reference to the Prediction models directly was done. The results of these tests were used to influence which model was chosen for the final application. As previously discussed in sections 4.2.1 and 4.3.1, and displayed in tables 1, 3 and 4 there was extensive testing conducted to find which models performed best. These tests utilized r^2 Scores, Mean Absolute Error, Root Mean Square Error and Mean Squared Error [49].

In choosing a model, a subset of Bus Routes were manually viewed and analyzed with regards to the above metrics across the various models which were implemented: *Random Forest, ANN, Linear Regression* and *K-Nearest Neighbors*. The *Random Forest* model was chosen based on its performance across this subset of bus routes. The performance of the *Random Forest model* was then confirmed with a custom script which generated an average value for each metric across all bus routes. These average metrics are displayed in Table 1. It was found that the average performance was actually better than the previously reviewed subset of bus routes.

Bibliography

[1] W.Fan and Z. Gurmu, *"Dynamic Travel Time Prediction Models for Buses Using Only GPS Data"*, International Journal of Transportation Science and Technology, vol. 4, no. 4, pp. 353-366, 2015. Available: 10.1016/s2046-0430(16)30168-x.

[2] M. Yang, C. Chen, L. Wang, X. Yan and L. Zhou, *"BUS ARRIVAL TIME PREDICTION USING SUPPORT VECTOR MACHINE WITH GENETIC ALGORITHM"*, Neural Network World, vol. 26, no. 3, pp. 205-217, 2016. Available: 10.14311/nnw.2016.26.011.

[3] A. Russo, M. Adler and J. van Ommeren, *"Dedicated bus lanes, bus speed and traffic congestion in Rome"*, Transportation Research Part A: Policy and Practice, vol. 160, pp. 298-310, 2022. Available: 10.1016/j.tra.2022.04.001.

[4] S. Rashidi, S. Ataeian and P. Ranjitkar, *"Estimating bus dwell time: A review of the literature"*, Transport Reviews, pp. 1-30, 2022. Available: 10.1080/01441647.2021.2023692.

[5] T. Nimpanomprasert, L. Xie and N. Kliewer,*"Comparing two hybrid neural network models to predict real-world bus travel time"*, Transportation Research Procedia, vol. 62, pp. 393-400, 2022. Available: 10.1016/j.trpro.2022.02.049.

[6] K. Riley,*"The Businesses Where Google Is Biggest"*, Wall Street Journal, 2020.

[7] Open Source Geospacial Foundation, *GDAL Documentation.* [Accessed 2 July 2022].

[8] Google,*"Maps JavaScript API."* [Accessed 2 July 2022].

[9] J. Patnaik, S. Chien, and A. Bladihas.*"Estimation of bus arrival times using APC data"*Journal of Public Transportation, 7(1), 1–20, 2004.

[10] M. Chen, X. Liu, J. Xia, and S. Chien.*"A dynamic bus arrival time prediction model based on APC data"*Journal of Computer-Aided Civil and Infrastructure Engineering 19 (5), 364-376, 2004.

[11] S. Chien, Y. Ding, and C. Wei.*"Dynamic bus arrival time prediction with artificial neural networks."*Journal of Transportation Engineering, Volume 128, Number 5, 429-438, 2002.

[12] A. Srivastava, S. Bhardwaj and S. Saraswat,*"SCRUM model for agile methodology"*, 2017 International Conference on Computing, Communication and Automation (ICCCA).

[13] A. Srivastava, S. Bhardwaj and S. Saraswat,*"SCRUM model for agile methodology"*, 2017 International Conference on Computing, Communication and Automation (ICCCA).

[14] J. Lopez-Martinez, R. Juarez-Ramirez, C. Huertas, S. Jimenez and C. Guerra-Garcia,*"Problems in the Adoption of Agile-Scrum Methodologies: A Systematic Literature Review"*, 2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT), 2016.

[15] Cobb, Charles G.,*"The project manager's guide to mastering Agile: Principles and practices for an adaptive approach."* USA: John Wiley Sons, 2015.

[16] D. Pauly and D. Basten, *"Do Daily Scrums Have to Take Place Each Day? A Case Study of Customized Scrum Principles at an ECommerce Company"* Hawaii International Conference on System Sciences, pp. 5074–5083, 2015

[17] R. Pichler, *"Agile Product Management with Scrum."* Creating Products that Customers Love, 4th ed. Upper Saddle River: AddisonWesley, 2011.

[18] L. Williams, R. Kessler, W. Cunningham and R. Jeffries, *"Strengthening the case for pair programming"*, IEEE Software, vol. 17, no. 4, pp. 19-25, 2000.

[19] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German and D. Damian,*"The promises and perils of mining GitHub"*, Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014, 2014.

[20] I. Darussalam and Y. Widyani, *"GitMonitor: a Software for Monitoring Project Schedules with Git Data"*, 2021 International Conference on Data and Software Engineering (ICoDSE), 2021.

[21] N. Jatana, S.Puri, M. Ahuja, I.Kathuria and D.Gosain. *"A Survey and Comparison of Relational and Non-Relational Database"*, International Journal of Engineering Research Technology, vol. 1, no. 6, pp. 1-5, 2012.

[22] J.M. Perkel. *"Why Jupyter is data scientists' computational notebook of choice"*, Nature, vol. 563, no. 7732, pp. 145-146, 2018.

[23] I. Stančin and A. Jović. *"An overview and comparison of free Python libraries for data mining and big data analysis"*, 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 977-982, 2019.

[24] BrowserStack. *"Selenium: Definition, How it works and Why you need it."*Available at: ¡https://www.browserstack.com/selenium¿ [Accessed 22 July 2022].

[25] Medium. *"Responsive React."*Available at: ¡https://medium.com/@mustwin/responsive-react-9b56d63c4edc¿ [Accessed 22 July 2022].

[26] Insights *"Stack Overflow Insights"*Available at:Available at: ¡https://insights.stackoverflow.com/trends?¿ [Accessed 24 July 2022].

[27] Xenonstack.com. *"ReactJs Project Structure and Final folder — Boilerplate Setup. "*Available at: ¡https://www.xenonstack.com/insights/reactjs-project-structure¿[Accessed 24 July 2022].

[28] Reactjs.org. *"Lifting State Up – React."*Available at: ¡https://reactjs.org/docs/lifting-state-up.html¿[Accessed 24 July 2022].

[29] B. Jose and S. Abraham,*"Exploring the merits of nosql: A study based on mongodb"*2017 International Conference on Networks amp; Advances in Computational Technologies (NetACT), 2017.

[30] T. Garg, *"Pandas for Data Analysis"*Knoldus Blogs, 2021. [Online]. Available: https://blog.knoldus.com/pandas-for-data-analysis/.

[31] L. Bottou, *"Stochastic Gradient Descent Tricks"*, Lecture Notes in Computer Science, pp. 421-436, 2012.

[32] M. Ramos, M. Valente and R. Terra, *"AngularJS Performance: A Survey Study"*,, IEEE Software, vol. 35, no. 2, pp. 72-79, 2018.

[33] I. Bairagi, A. Sharma, B. Rana and A. Singh, *"UNO: A Web Application using Django"*2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), 2021.

[34] S. Agarwal, S. Jain and A. Kumar, *"GUI Docker Implementation: Run Common Graphics User Applications Inside Docker Container"* 2021 10th International Conference on System Modeling; Advancement in Research Trends (SMART), 2021.

[35] "nginx documentation", *"Nginx.org"* 2022. [Online]. Available: https://nginx.org/en/docs/.

[36] "Lightsail Documentation", *"Lightsail AWS"* 2022. [Online]. [Online]. Available: https://lightsail.aws.amazon.com/ls/docs/en_us/overview. [Accessed: 05- Aug- 2022].

[37] "Gunicorn 20.1.0 documentation", *" Gunicorn Documentation"* 2022. [Online]. Available: https://docs.gunicorn.org/en/stable/. [Accessed: 05- Aug- 2022].

[38] "Supervisor 4.2.4 documentation", *"Supervisord"* [Online]. Available: http://supervisord.org/running.html. [Accessed: 06- Aug- 2022].

[39] "UFW Documentation", *"ubuntu"*[Online]. Available: https://help.ubuntu.com/community/UFW. [Accessed: 06- Aug- 2022].

[40] "Certbot 1.29.0 documentation", *"certbot"*[Online]. Available: https://eff-certbot.readthedocs.io/en/stable/install.htmlabout-certbot. [Accessed: 06- Aug- 2022].

[41] "TensorFlow Documentation", *"TensorFlow"*[Online]. Available: https://www.tensorflow.org/about. [Accessed: 06- Aug- 2022].

[42] "Keras documentation", *"Keras"* 2022. [Online]. Available: https://keras.io/api/. [Accessed: 06- Aug- 2022].

[43] "React", *"Reactjs"* 2022. [Online]. Available: https://reactjs.org/. [Accessed: 06- Aug- 2022].

[44] A. Singh, R. Akash and G. V, *"Flower Classifier Web App Using MI amp; Flask Web Framework"*, 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2022.

[45] "PostgreSQL Documentation" *"PostgreSQL Documentation"*2022. [Online]. Available: https://www.postgresql.org/docs/current/intro-whatis.html.

[46] "WebDriver" *"Selenium Documentation."*[Online]. Available: https://www.selenium.dev/documentation/webdriver/. [Accessed: 07- Aug- 2022].

[47] "Snapshot Testing · Jest", *"Jestjs.io"* [Online]. Available: https://jestjs.io/docs/snapshot-testing. [Accessed: 07- Aug- 2022].

[48] "pytest documentation", *"pytest"* [Online]. Available: https://docs.pytest.org/en/7.1.x/. [Accessed: 07- Aug- 2022].

[49] "Metrics and scoring: quantifying the quality of predictions", *"scikit-learn."* [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html. [Accessed: 07- Aug- 2022].

Appendix 0

Figure 5.1: Appendix 1

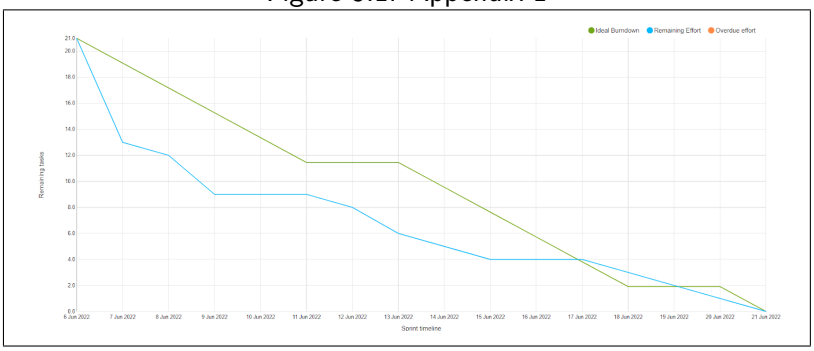


Figure 5.2: Appendix 2

09/07/22

Everyone Present

Front-End

Check on Zaur and Tian Bo's progress. With regards to front-end features and mobile development.

Connecting front to backend:

Discuss how this will be done, most likely it will be done through using rest API's carrying JSON data.

Unit Testing on The Backend:

Discuss the approach to testing the backend.

Documentation:

Discussed standards and the breakdown of the files.

Appendix 3


```
{
  "bus_route": "46A",
  "direction": "2",
  "i": 1,
  "travel_time": 1755.7718985507247
}
```


Appendix 4


```
{
  "IsDayTime": true,
  "WeatherIcon": 6,
  "WeatherText": "Mostly cloudy",
  "weatherMetric": 16
}
```

Figure 5.3: Appendix 5

09/07/22

Everyone Present

Front-End

Check on Zaur and Tian Bo's progress. With regards to front-end features and mobile development.

Connecting front to backend:

Discuss how this will be done, most likely it will be done through using rest API's carrying JSON data.

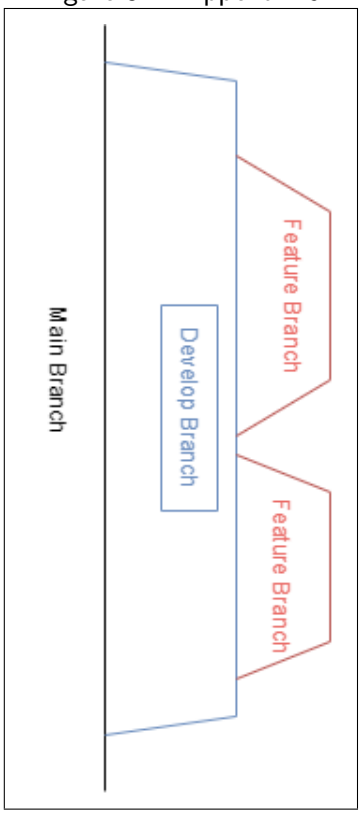
Unit Testing on The Backend:

Discuss the approach to testing the backend.

Documentation:

Discussed standards and the breakdown of the files.

Figure 5.4: Appendix 6



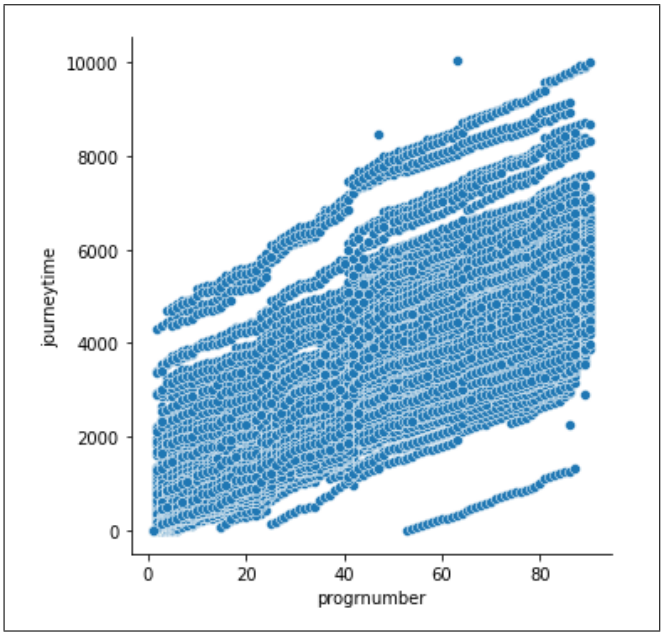


Figure 5.5: *Appendix 7: Progrnumbr (Progression Number) vs Journey Time (s)*

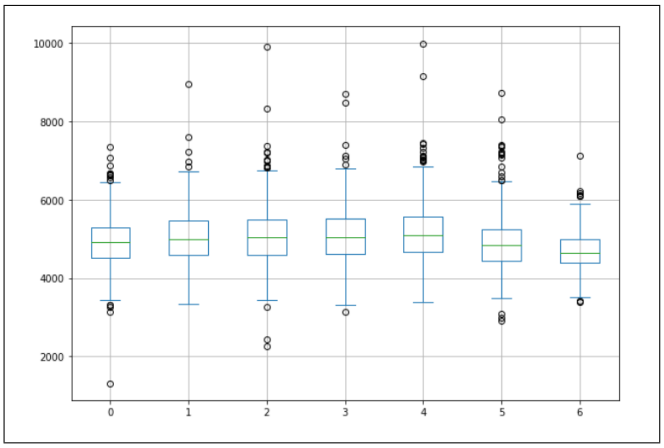


Figure 5.6: *Appendix 7: Weekday (0=Monday, 1=Tuesday, etc.) vs. Journey Time (s)*

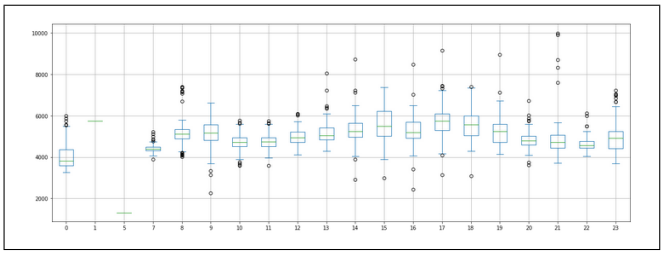


Figure 5.7: *Appendix 7: Hour vs. Journey Time(s)*

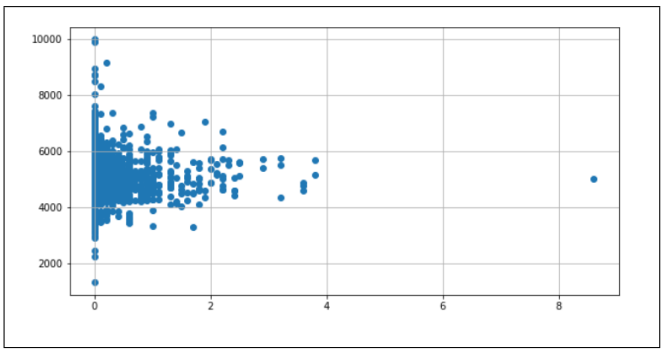


Figure 5.8: *Appendix 7: Hourly Rain (mm) vs Journey Time (s)*

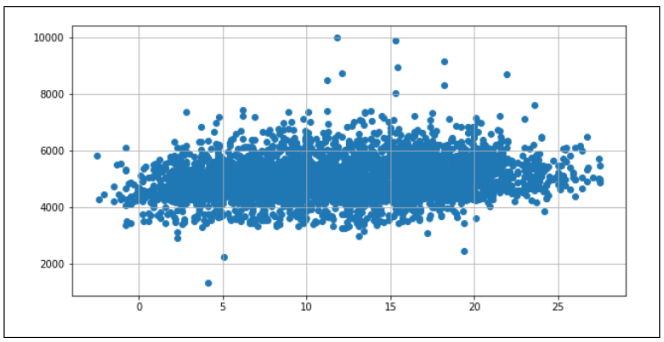


Figure 5.9: *Appendix 7: Temperature (°C) vs Journey Time (s)*

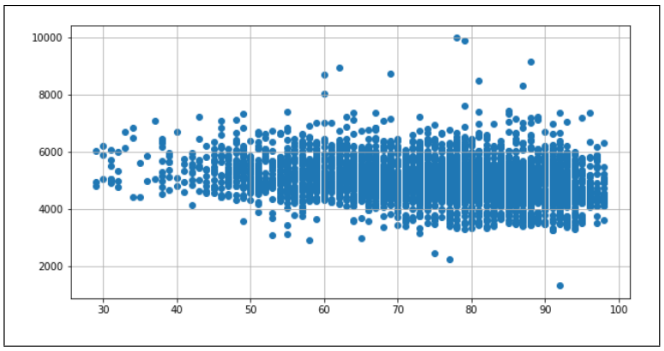


Figure 5.10: *Appendix 7: Humidity (%) vs Journey Time (s)*

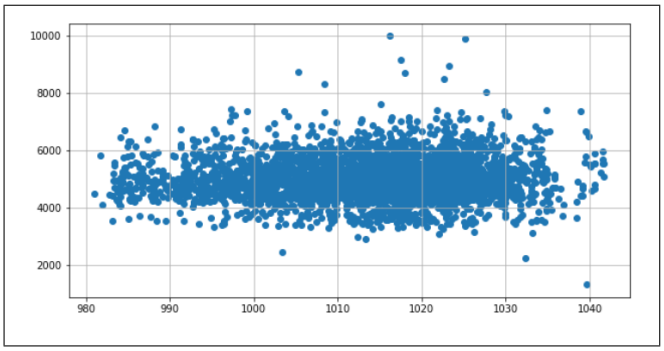


Figure 5.11: *Appendix 7: Pressure (hPa) vs Journey Time (s)*

Figure 5.12: Appendix 8

	feature	importance
0	progrnumber	0.892781

Figure 5.13: Appendix 10

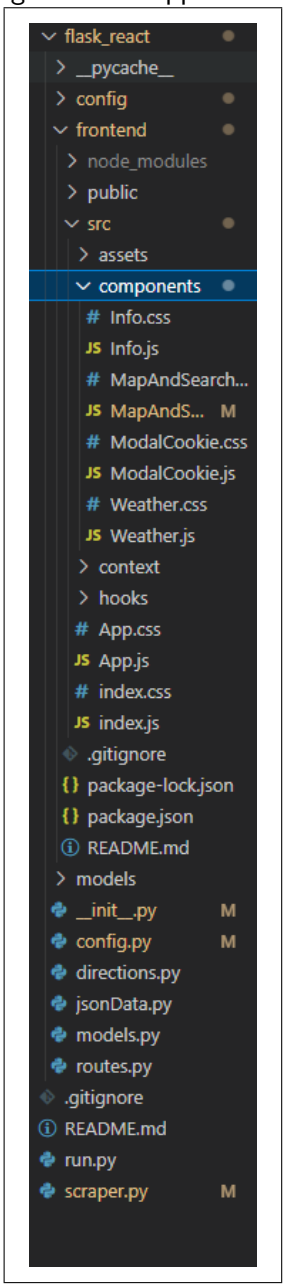


Figure 5.14: Appendix 11

```
print("========================Average Metrics For Random Forest========================")
print(r2Value/len(R2scores))
print(mseValue/len(MSEscores))
print(maeValue/len(MAEscores))
print(rmseValue/len(RMSEscores))
```

```
========================Average Metrics For Random Forest========================
0.9586126326135013
202.8111243866171
129.71002889926578
11.259191212747451
```

Figure 5.15: Appendix 12

```
model.fit(normed_train_data, train_labels, epochs=100,
          validation_split = 0.2, batch_size=50, verbose=0, callbacks=[early_stop])
```

Figure 5.16: Appendix 13

1 Introduction

This document provides a detailed account of the requirements to build a Dublin Bus route and arrival prediction web application using the Flask framework, which will be referred to as DBA for the purpose of this document. This document has not defined a commercial name for the application, but may do so. A full list of abbreviations, definitions of the various technologies and terminologies can be found in the next section 1.1. This document lays out the different sections which correspond to the different requirements needed to build a minimum viable product (MVP). The basis of any prediction or tracking apps are simple, they are used to give an accurate time of travel and arrival to a given location for the user to reduce any anxieties and stress associated with missing buses or being late. This application has great utility for users, and users could benefit.

This document in it's aim is an explanation with high level resolution and it attempts to explain what will be needed technically for the creation of such an application. The DBA aims to create a seamless user experience where any user can geographically pinpoint their current location, find a list of buses near them and also enter where they would like to go, and a prediction (given and calculated in minutes) will appear on the screen. The user should also have an option to actually enter a physical location. Other features such as weather will be displayed. A more detailed account of this is given in later sections of this software requirement specification.

1.1 Document Conventions, Terminologies and Definitions

The following terminologies are used in this document:

- `DBA`: This refers to the actual Dublin Bus application, which will be used many times through this text for ease of writing.
- `MVP`: Minimum Viable Product. This refers to the version of the product that would have the sufficient features for a launch so that users can use and provide results/feedback which can in turn be used to further develop versions.
- `Ecosystem`: The ecosystem is a term to describes the entire user system. This includes things like the buses, the users of the app, the development team and essentially any entity that is involved in the application.
- `Flask`: will be used, flask is a Python web framework, the flask when connected with the our VM and acts like our server and is the main connection point from front end to back end.

Figure 5.17: Appendix 14

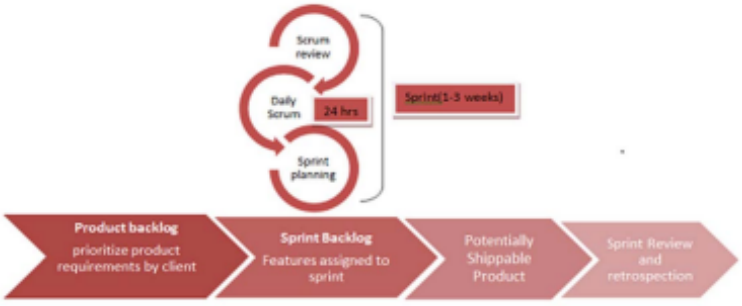


Figure 5.18: Appendix 15