Research Practicum Project Report

Bus Journey Time Prediction

Henry Lewis, Cormac Reilly, Brian Ryan, Jamie Reynolds

A thesis submitted in part fulfilment of the degree of

MSc. in Computer Science (Conversion)

Group Number: 9

COMP 47360

UCD School of Computer Science

University College Dublin

April 20, 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 (http://dublinbus.ie/Your-Journey1/Timetables/All-Timetables/46a-1/). 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

Abstract

Congestion is ubiquitous in cities, particularly Dublin City due to ever increasing traffic volumes. Such congestion is brought about by stochastic variation in traffic flows resulting in inaccurate and unpredictable bus arrival times at stops. To overcome this challenge, we have developed a user friendly and functional mobile-friendly web application that allows users in Dublin to plan their routes with accurately predicted journey times. Alongside its predictive capability, there are a number of innovative features for travel users, such as real-time weather, tourism features based at stops, events and attractions pages alongside a Spotify plugin which allows the user to plan a diverse range of events through our application. The approach taken is to use machine learning to develop a predictive model that predicts the travel time between stop pairs along the Dublin Bus Network. The predictive modelling uses Linear Regression Models for stop pairs and Random Forest Models where stop-pairs from different bus routes overlap. Evaluation of the results indicate that the application provides accurate and robust predictions using the aforementioned modelling approaches.

Website Link: https://explore-dublin.com/

GitHub repository: https://github.com/jamie-reynolds-UCD/Dublin-Bus-App-Team9

Acknowledgments

We would like to thank all those who provided support and encouragement throughout the project, particularly Laura Dunne, for her guidance and support with key elements of the project. In addition, we would also acknowledge the guidance given throughout the mentoring process by Dr Gavin McArdle and Dr Madhusanka Liyanage which was invaluable in completing the project.

Table of Contents

Chapter 1: **Introduction**

Cities such as Dublin are experiencing ever-increasing levels of vehicles resulting in high levels of traffic congestion and decreasing quality of life for residents. One way of combating increased traffic is to get more people to use Bus Transport more regularly by providing an accurate real-time information regarding bus arrival and departure times at stops. However, a key problem in predicting bus travel times is the complexity due to the stochastic events, such as traffic congestion or accidents are difficult to predict. Attempts by Dublin bus and Transport for Ireland (TFI) using live information to overcome the problem of unreliable information have received poor reviews in terms of accuracy and functionality (See reviews in Appendices (A.1) and (A.2)).

More recently, researchers have produced a dearth of studies examining the problem of predicting bus arrival times using a diverse range of techniques with historical data models (Reich et al., 2019), Linear regression Models (Jeong and Rilett, 2004), (Shalaby et al., 2002). The popularity of Artificial Neural Networks (ANN) has been increasing in the literature for their high accuracy and their general superior performance when compared to other methods (Reich et al., 2019). Overall, the consensus is that historical models and linear regression models perform poorly when compared with other ANN based models. However, they require extensive training along with longer prediction times and have been shown to be no better than historical models in heavily congested cities (Julio et al., 2016). A central criticism of these studies is that they are restricted to a specific route or small collection of routes and have not been extended to capture the dynamics of an entire network (Mori et al., 2015)

We have established that public transport services in Dublin do not provide reliable or accurate live information. Therefore, to overcome this deficit, the main aim of our project is to create a functional mobile-friendly web application that allows users in Dublin to plan their routes with accurately predicted journey times. This resulted in Explore Dublin, an application that allows users to not only explore nearby services, events, and tourist attractions, but to easily plan a route to the desired location and receive an accurate journey time prediction to get there on time. The prediction elements and model choice of the application was informed through in-depth data analysis process as we explored and created various features in the data to create a precise predictive model for our application. In addition, our application will provide an intuitive friendly user experience, which emphasises a mobile-first user experience to provide application users with an on-the-go app via their smartphone to effortlessly plan their routes.

Our project evolved considerably over its course. After adequately meeting the project specification requirements, different avenues in which we could take our application were explored. Innovation was at the forefront of our minds as we wanted to develop an application that provided something truly beneficial to users and that serves a need which is not currently met by existing applications.

This report will outline the evolution of Explore Dublin and begins with an application overview in Chapter 2 where existing transportation applications and our original design mock-ups are discussed. Following this, a detailed breakdown of the final product is provided which includes a description of its functionality, how we addressed the project specification requirements, and innovations beyond the requirements. Chapter 3 outlines our development approach from a high level and how we implemented SCRUM methodology. The details of the four primary components of our system architecture are provided in Chapter 4. Finally, the process of testing and evaluating our application is described in Chapter 5.

Chapter 2: **Description of Final Product**

The finished application takes on board the central tenets derived from the research of other transportation planning applications alongside the best practise conducted by the major mobile phone manufacturers guidelines for development of both iOS (Apple, 2020) and Android based applications (Google, 2020) to produce an application that is capable of delivering a reliable and robust predictive transportation planner alongside an event planner that is easy to use for all.

2.1 Application Overview

The web application was designed using a mobile-first strategy. This was carried out by focusing on the layout of components to make sure they made sense from a mobile point of view. Responsiveness was a priority so we used media queries on every component to ensure a responsive design. The final design is broadly based on our initial mock-up designs as well as best practises guidance in UX Design. Each component of the application and its functionality will now be detailed in the sections below.

2.1.1 Homepage

The Explore Dublin homepage, Figures 2.1 and 2.2, provides users with an easy-to-use interface that is packed full of features allowing users to discover locations and plan routes efficiently. We will now explore and breakdown each feature of the homepage individually.

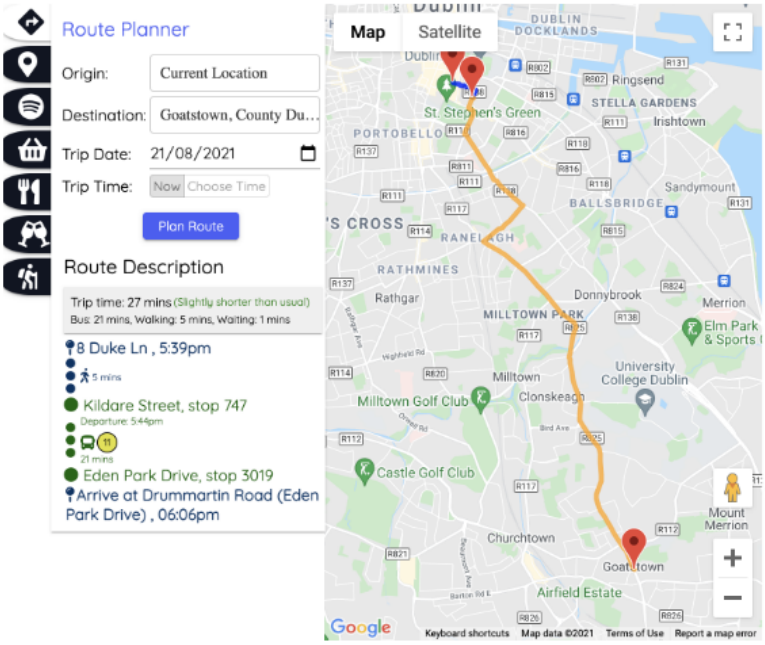


Figure 2.1: Desktop

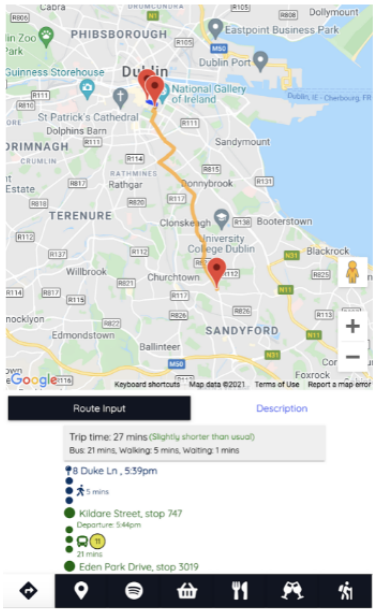


Figure 2.2: Mobile

2.1.2 Route Planner

The primary functionality of our final product is to plan and estimate the time of journeys between an origin and destination in Dublin using the bus as a primary source of transport.

We provide the user with origin and destination input boxes to plan their route. The user can also choose a date and time for their trip.

When the user clicks on "Plan Route", a request is sent to our api endpoint "/getroute" with parameters for the origin and destination coordinates as well as the date and time of the trip. This endpoint queries the google maps directions api, specifying the transit method as bus and retrieves the route instructions. Before the route instructions are sent back to the frontend, the bus journey estimated forecast times are updated to predictions from our machine learning model.

2.1.3 Model Implementation

Forecasting Journey Times

The directions retrieved by Google are a JSON object with descriptions for all legs of the journey. We then search the legs to find bus journeys on the route. For each bus journey we find the start and end stop and generate the full path of the journey which includes all intermediate stop to stop pairs. Finally, we look-up the models for each stop to stop pair and generate the predictions. These are primarily linear regression models, however we have random forest models for some of the most frequently used trips. The sum of the predictions from each stop to stop pair is our estimated total time for the bus journey.

Regenerating Routes for missed connections

If there are multiple bus journeys on the trip returned by Google, we must check if our predicted time for the first bus journey results in a missed connection. If it does, we regenerate the route from the endpoint of the first bus journey and repeat the process. This is required so that we don't send an invalid trip back to the user.

.

2.1.4 Displaying the Route

The final route directions including the estimated bus journey times from our model is returned to the user. This route is rendered on the map and the route description is displayed in the sidebar. The route description contains color-coded legs of the journey. The estimated time for walking is directly retrieved from the google directions and the estimated bus journey times are from our machine learning models. We also display the bus route and the name of the start and end stop of each bus journey to the user.

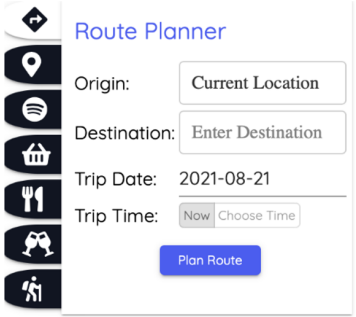


Figure 2.3: Route Planner Input

2.2 Addressing the Project Specification

The project specification indicated that the project should create an application that uses the static historical datasets provided by Dublin Bus to provide an accurate journey time prediction. When presented with an origin and destination (while also taking into account the day of the week, and current weather condition), the application provides directions to the origin bus stop, and an accurate prediction of how long the journey will take from origin to destination. when the user enters and origin and destination into the application. The approach taken to produce journey time estimates was to use machine learning models to predict the travel time by breaking the total journey into a series of stop pairs. This is a more realistic interpretation of actual passenger usage of Dublin Buses, journey segments consisting of groups of stops as opposed to using a specific route in its entirety. This section of the report details how the project specification was met regarding the predictive modelling.

2.2.1 Predictive Modelling with Data Analytics

A diverse range of approaches to modelling are detailed in the literature. Overall ANNs dominate in terms of accuracy over other methods. However, they require extensive training along with longer prediction times and have been shown to be no better than historical models in heavily congested cities (Julio et al., 2016). To overcome these deficits we decided to use a more simplified modelling approach using linear regression models which have been shown to be more accurate than simple historical models (Jeong and Rilett, 2004) and random forests for the most visited stop pairs since they have a higher accuracy than linear regression for complex non linear datasets but would not be computationally efficient to use on each stop pair.

Our goal for the data analytics was to create accurate prediction models for each stop pair throughout Dublin. We decided to go with this approach instead of predicting the total travel time for an entire bus route, based on the logic that most users of bus don't use the entire bus route. Rather, they usually use the bus for segments of a bus route. Therefore, modelling segments of a bus route made more sense from a user's perspective than modelling complete bus routes.

Modelling Strategy

The architecture of the Dublin bus route network is essentially a hub and spoke layout with most of the routes going from the suburban and periurban boundaries to the city centre. Our approach to the data preparation at a high level was for two members of our team to try different approaches to feature creation. We would then test the results of these approaches on the same three routes each. The routes (70,123 and 104) were chosen as they represent an appropriate sample of routes that are currently run by Dublin Bus. Route 104 is a purely suburban route, whilst the 123 is a purely urban route that traverses the city from Griffith Avenue to Walkinstown. Route 70 is a limited stop route from Dunboyne to the city centre. Whichever approach returned better metrics would be the one that we would use for model creation for the project.

One approach was to create as many features in the data as possible, while the other approach was to use as few created features as possible. The initial features created were the weekday (numbered from 0 to 6, starting at Monday), school summer holidays (0 if no, 1 if yes); morning rush hour (if the time of the journey was between 7.00AM 9.00PM) and evening rush hour (if the time of the journey was between 4.00PM and 6.00PM), as well as precipitation (the amount in millimetres of rain and/or snow).

We then compared the metrics produced by models made with both approaches. As we can see in figures 2.4, 2.5 and 2.6, the metrics for the fewer-features approach performed better than the more-features approach. We then created models for all stop-pairs using the fewer-features approach.

As can be seen in figures 2.7, 2.8 and 2.9, the metrics for random forests using the fewer-features approach models performed slightly better over all, but the computational cost of using random forests models for the entire application would be too high. Instead, we took the most-used stop-pairs for each direction and created random forests models for just these stop-pairs. This meant that the accuracy for the most-used stops was as high as possible, while also saving on computational costs for the app. We went with this approach because this project is made with users of the app in mind. It is important that the web app's prediction models do not take too much space, or use too many computational resources. Otherwise, the app may not function correctly, which would make for a poor user experience. This is why we had to prioritise the most-used stops for the random forests models.

All models were trained using a train/test split of 70 percent training data to 30 percent test data. The results of the models' performance on the test data was used for evaluation of the models. Evaluation of a model's performance on the test data is a clearer indicator of how well the model performs, as it is a measure of the model's accuracy on data it has not been exposed to before.

Data Exploration

Initially, we took a short route and long route, the 120 and 46A respectively. We removed rows with null values and duplicate rows. We removed duplicate and constant columns, as well as features that were uninformative such as TENDERLOT and BASIN. Finally, we removed features where almost all of their values missing such as JUSTIFICATIONID and SUPPRESSED.

Prior to data exploration the trips, leavetimes and weather dataset was combined into a single dataframe. As part of the process of data exploration we created a number of features from the original data sets to better capture the relationships between the data. For any stop-pairs that had negative journey times, we removed the entire trip that the negatively-timed journey was in. It was likely that one negative journey time would affect the rest of the trip, so we thought it better to keep only trips where we were sure that the data was correctly recorded. For this reason, we also removed any trips that were missing stop-pairs in the middle of the journey. In addition, outliers were removed where they were 3 standard deviations above the mean. These were stop-to-stop journeys that took so much longer than the average, we inferred that there was a mistake with the bus's AVL data for this journey. Most of these journeys occurred between the second-last and last stops of a trip, so we felt it necessary to just remove the outlying stop-pair and not the entire trip.

Data Collection and Analysis

The data is a historical dataset collected from Dublin Bus vehicles from January 2018 to January 2019 using an Automated Vehicle Location (AVL) system. This system records specific parameters based on the vehicle's position on the network enabling data on individual trips to be collected for each stop. In addition to the bus data, we also gathered matching historical weather dataset from OpenWeatherMap. This contained the weather information at hourly intervals. The weather dataset contained a large number of parameters such as the levels of rain and snow, wind speed, humidity and a general weather description. We decided to use the OpenWeatherMap historic data, because it has more data features than the Met Eireann historic data, and is in the same format as the weather data we would need as input for our prediction models. The bus dataset

because of its size was located on an external server and was disaggregated over a number of tables. Of principal importance, was the Trips and the Leavetimes datasets. A single trip is disaggregated across both leavetimes and trips datasets, with elements such as the line id and route id being part of the trips dataset, whilst both stop arrival and departure times are contained within the leavetimes dataset.

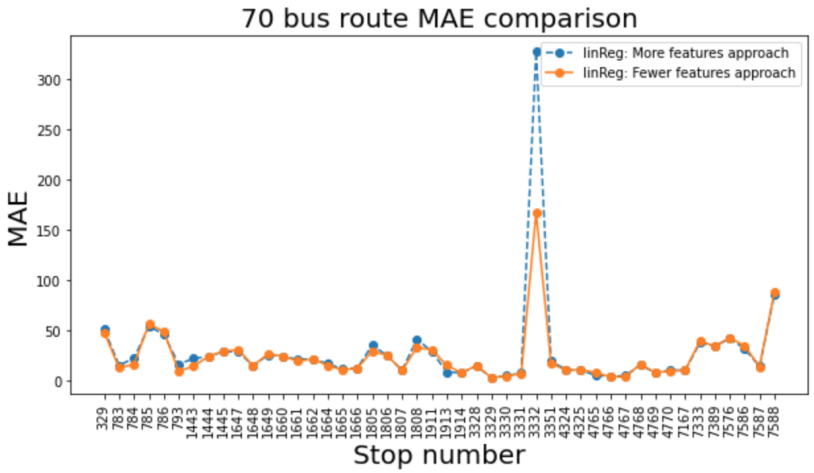


Figure 2.4: 70 Linear Regression Comparison

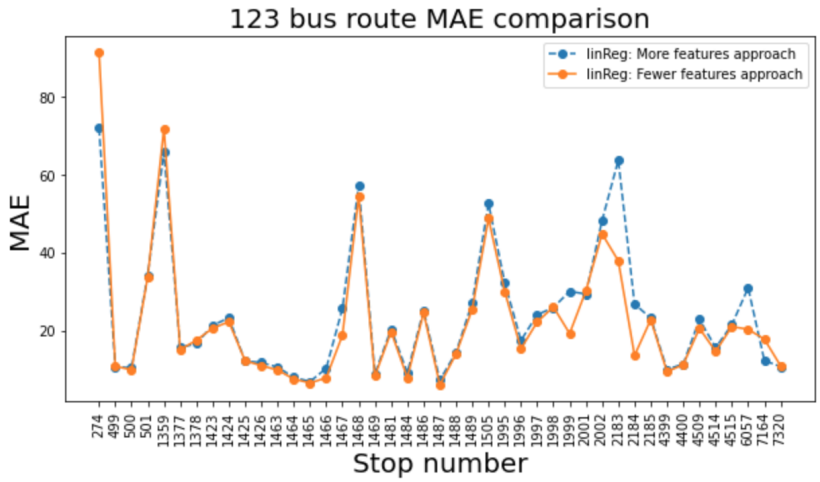


Figure 2.5: 123 Linear Regression Comparison

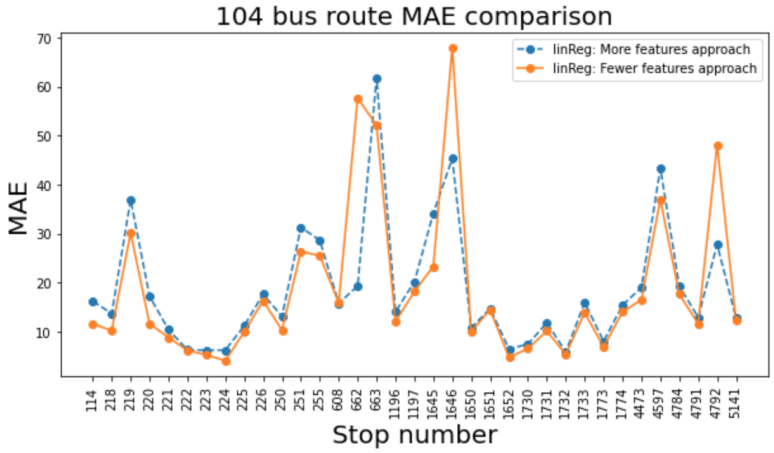


Figure 2.6: 104 Linear Regression Comparison

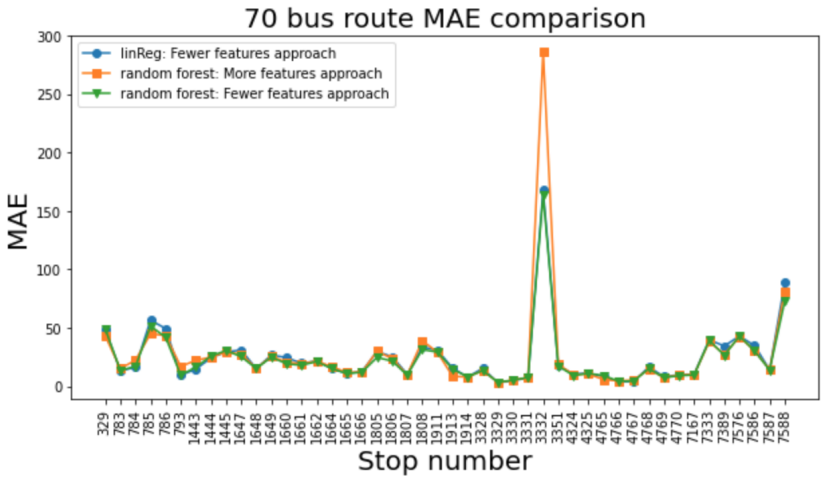


Figure 2.7: 70 Random Forests Comparison

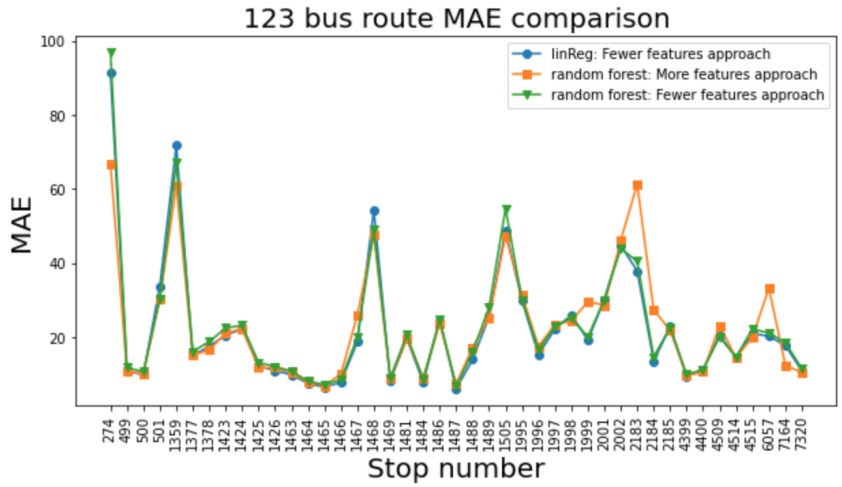


Figure 2.8: 123 Random Forests Comparison

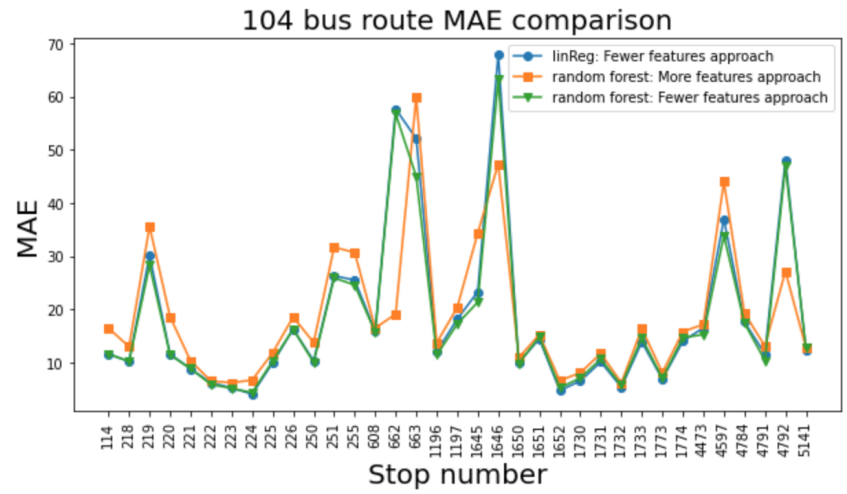


Figure 2.9: 104 Random Forests Comparison

2.3 Application Innovations

2.3.1 Suggested Activities

When the user visits the site, their location is detected and a number of activity suggestions are loaded under various categories. This is achieved by making use of the places service offered by google's API. A keyword search is conducted for each of the categories. The results include places to eat, places to drink, supermarkets, and outdoor walks.

When a user clicks on one of these tabs, the name, rating and price level of the establishments in this category are displayed to the user. They can click a button called "Get there" beside the location to quickly plan the route from their current location.

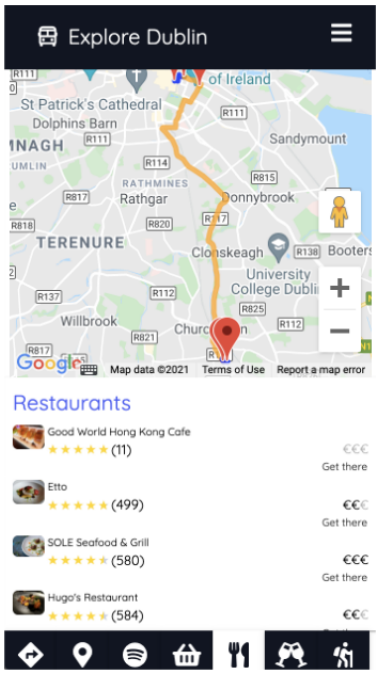


Figure 2.10: Restaurant suggestions

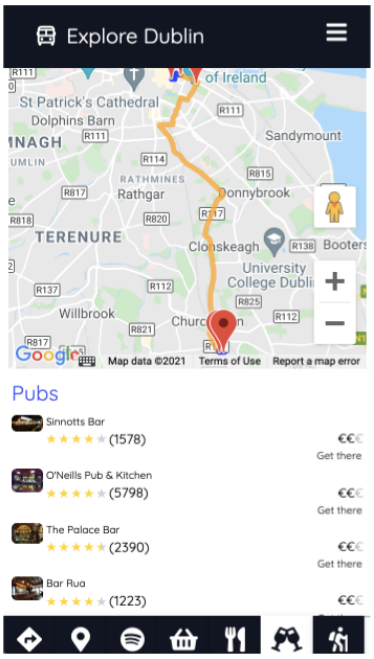


Figure 2.11: Pub suggestions

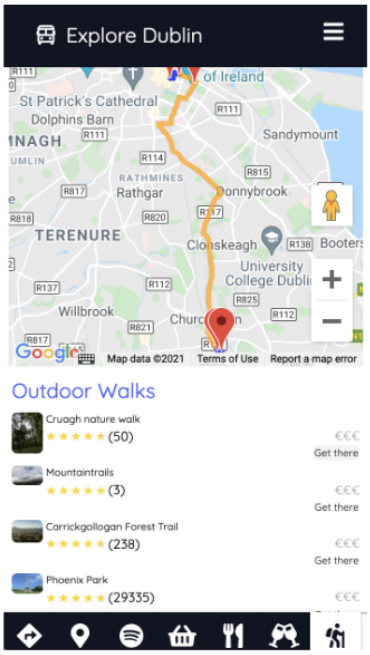


Figure 2.12: Walk suggestions

2.3.2 Frequently Used Locations

If a user regularly uses the website they have the option to save their locations such as for Home, Work or College. This means in the future they can plan their route to the destination in one click. We implemented this using cookies so that we did not have to require a user login. To use this feature, a user must simply click on "Add Location" and enter the full name of their frequently used location along with a pseudonym. This will show up as a destination in the locations tab from that point onwards. If a user clicks on the location it will be set as the destination in the route planner and a route will be displayed (see figure 2.13)

.

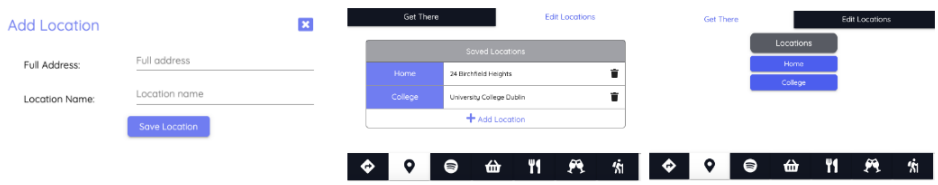


Figure 2.13: Frequently Used Locations Interface

2.3.3 Spotify - Dublin Podcasts

Our website is intended to be a place where people can find out what is happening in Dublin and quickly plan their routes to get there. One of our target audiences is therefore people who are coming to see the city. We decided that we could integrate the spotify API into our application to allow the user to listen to dublin-based podcasts discussing current affairs and stories from around Dublin.

This functionality is only available to users who have a spotify account. If a user clicks on this feature, they are prompted to connect their spotify account. This process is detailed in figure 2.14.

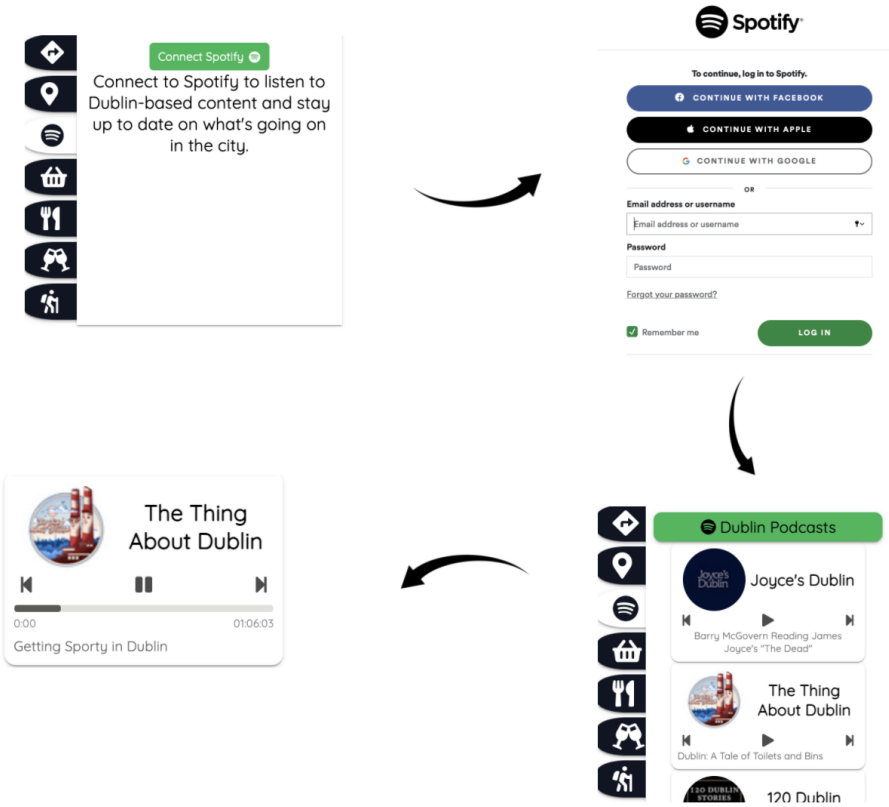


Figure 2.14: Spotify Podcasts

.

After authentication through Spotify, an access token and refresh token is saved in our database and associated with the user through a session key. Whenever the user comes back to the site, a new token can be requested automatically using the refresh token supplied during the initial authentication. The Spotify playback will take place in the browser on desktop or within the Spotify application if the user is on mobile and their Spotify app is active.

2.3.4 Events

Explore Dublin's events page allows users to discover a wide variety of upcoming events in the city. Using the Ticketmaster API, all the major events in Dublin, from concerts to football matches, are displayed on an easy-to-use interface so that users can find an event, buy tickets, and plan a route all from one application.

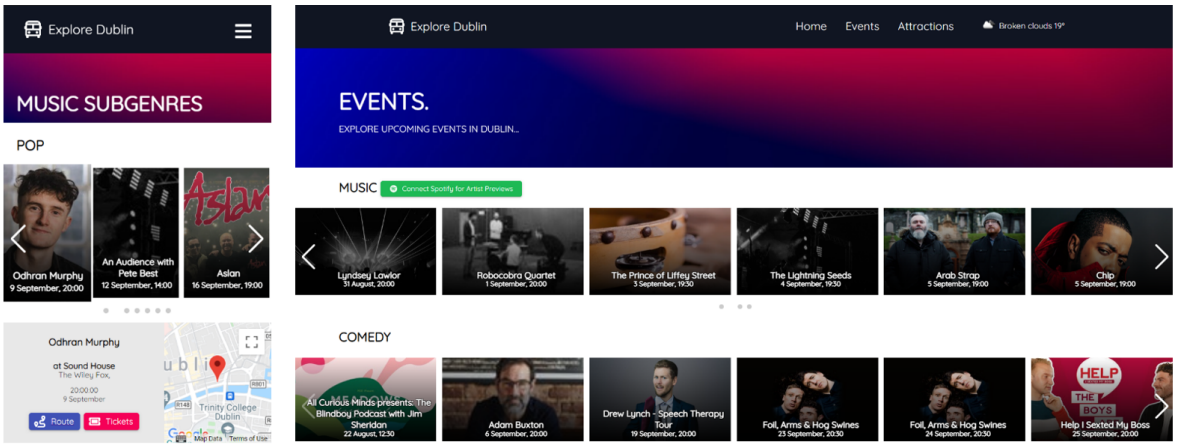


Figure 2.15: Events Page: Mobile and Desktop

.

Users are presented with various rows of event categories in which they can swipe through to find events that may interest them. The user interface is fully responsive, and the event cards adapt to all screen sizes. Music genres are categorised further down the page for further exploration as shown in Figure 2.15.

When an event is clicked, a window is shown below the row which provides the user with further information including the venue's name, address, and a marker on an interactive map. Two buttons are also shown for added functionality. The tickets button brings users directly to the event on the Ticketmaster website or application where they can purchase tickets and get further information. The route button brings the user to the homepage and a route description is instantly provided. The route description automatically uses the user's location and predicts a route directly to the venue fifteen minutes prior to the event's start time on the day of the event. The user can of course alter these details to their desire.

2.3.5 Spotify

The Spotify API is also utilized on the events page. After a user connects their Spotify account, a play button is shown on the top right of music artists' cards as shown in Figure 2.16. The user can press play to instantly listen to one of the artist's top songs via the Spotify application on mobile and desktop or via the Spotify web player. This allows users to explore new artists, genres, and discover upcoming concerts they would like to go to.

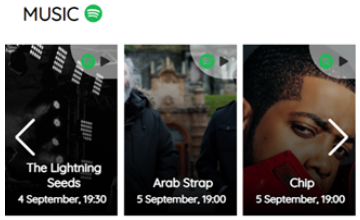


Figure 2.16: Basic layout

2.3.6 Attractions

The Explore Dublin attractions page is designed to allow users to easily find suitable tourist attractions throughout Dublin. The page utilises open data available from Fáilte Ireland to provide users with a list of high quality attractions. During our research, we found that current tourist attraction websites for Dublin were outdated and did not provide a pleasant user experience. Our page boasts a responsive modern interface that allows user to easily find the right attraction for themselves, and to easily get a route and accurate journey time to the chosen attraction.

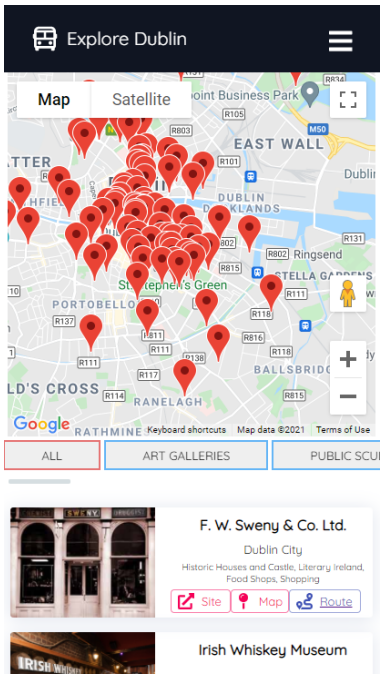


Figure 2.17: Attractions page

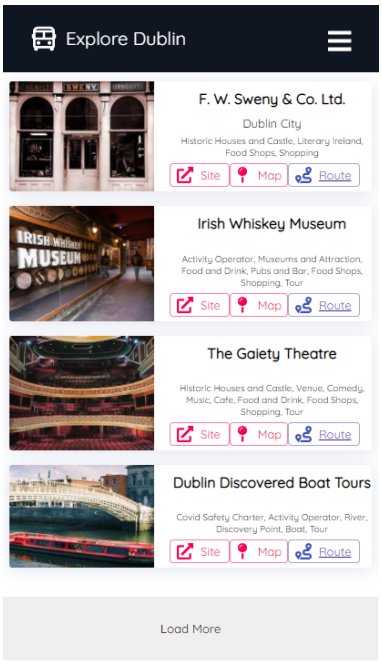


Figure 2.18: List of attractions

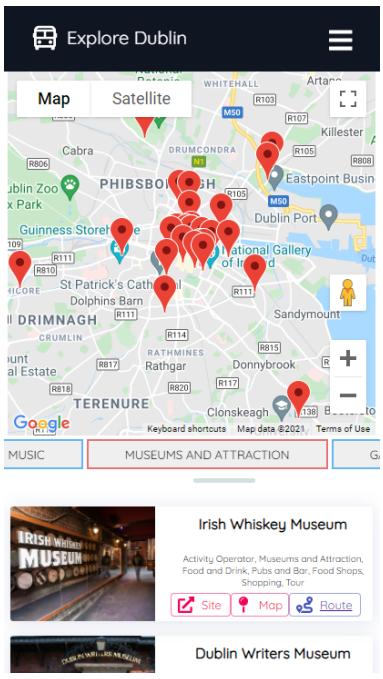


Figure 2.19: Filtered list

Users are presented with an interactive map at the top of the page which shows markers for all of the attractions. Users can use this map to see if there are attractions nearby or discover others throughout Dublin. Cards for each attraction are displayed below the map which provide the user with information about the attraction including its name, locality, and tags, as shown in Figure 2.15. There are three buttons on each card which allow the user to go directly to the attraction's website, view only the attraction's marker on the map, or to plan a route to the attraction instantly via the homepage.

A selection of tag buttons allow the user to filter the list of attractions to their desire as shown in Figure 2.16. When a tag is selected, the new list of attractions is loaded and the map is refreshed to show only those markers related to the tag. This allows users to quickly find quality attractions that interest them and to easily plan a route.

The page is fully responsive and the design changes depending on the screen size. On the desktop version, the shape of each attraction card changes and they are displayed in rows as shown in Figure 2.17.

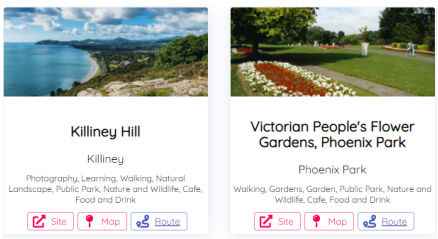


Figure 2.20: Desktop

Chapter 3: **Development Approach**

3.1 Development Approach

The project used the SCRUM methodology to develop the application. As part of this process, the project was divided into 5 sprints with each Sprint being 2 weeks in length. As part of the Scrum process we allocated a SCRUM master for each sprint who was responsible for providing detailed notes for each stand-up meeting as well as a sprint review meeting at the end of each sprint to detail the project progress. As part of each sprint we undertook daily stand-up meetings to discuss progress and any blockages that would prevent these from occurring. Outside the formal stand-up meetings we remained in constant contact over the discord text channel throughout the day should any urgent issues arise. To assist in the SCRUM management process we initially discussed the most appropriate methods to manage the tasks within each sprint. The two options that were proposed were the Trello board and a Jira were discussed as options for the management process and task distribution between members. In the end we felt that using a Trello board was the best option as it is easier to learn than Jira. It also allows users to log tasks using a three part hierarchy of boards, lists and cards. To ensure users are kept up to date Trello sends notifications based on who is mentioned on a card or task. This made the Trello board ideal for tracking task progress of each feature within the project and to monitor backlogs.

To streamline the workflow and establish everybody's responsibilities and the procedures that we would follow throughout the project we drew up a team contract. The contract also specified how we would deal with disagreements and conflicts within the team. It also outlined the communication methods that we would use throughout the project. This took the form of using the discord text channel for stand-up meetings and for any other contact throughout the project period. The justification for utilising Discord is that the entire team was familiar with its operation and it provides easy access text and group chat options, alongside audio and visual options including screen sharing among group members when other members have problems. Initially we all came into the project with predefined roles within the team but we agreed that we would not be so rigid and the roles became more fluidic as the project progressed.

After the preliminary matters were concluded, we began to make the high level decisions about the functionality the application should have to fulfil the requirements of the brief. The first steps were to understand the high level components needed to deliver the basic application functionality, particularly the architecture of the application as well as the interaction of the back and front end components alongside the database management system. Once the fundamental backbone of the application was established we then set about developing additional features of the events and attractions pages. A fuller description of the application architecture is set out in Chapter 4.

Chapter 4: **Technical Approach**

4.1 System Architecture

This section will describe the technical approach that the team took in developing a solution to the problem, and will include an in-depth discussion of the architecture of the final application as well as the technology stack used. There are four primary components of the system architecture:

1. Web server and web application framework for backend logic and serving static files.
2. Frontend framework for displaying information to the user on the client-side, responding to user actions and interacting with the web server.
3. The database for storing information required for the application to function such as bus stop details and access tokens for external APIs.
4. The modelling process and final models used by the web server to make predictions on travel times.

The first section here will discuss a broad overview of the full architecture of the system and how each component interacts with each other, and this will be followed by a more detailed analysis of each aspect of the system including a discussion about the technology stack used.

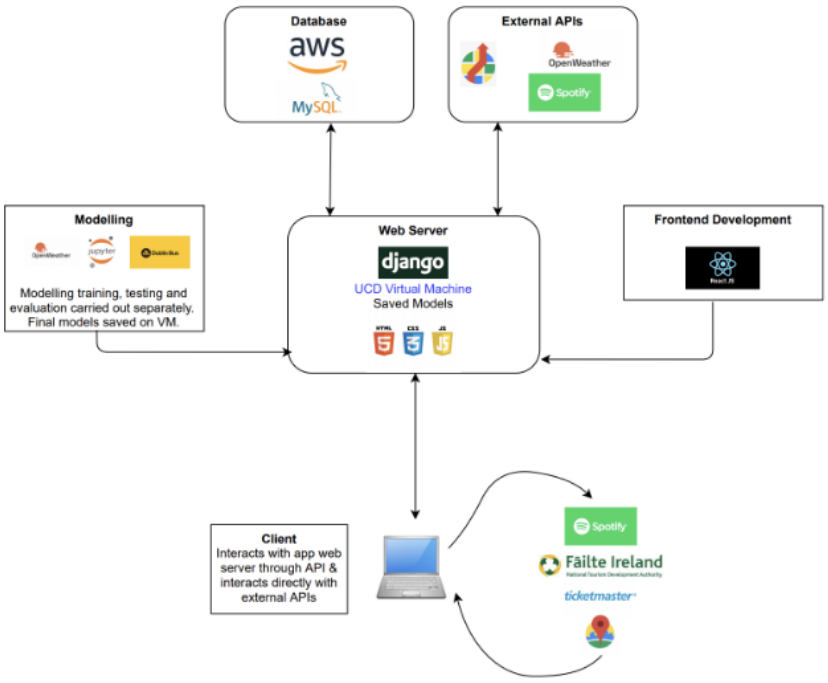


Figure 4.1: Full System Architecture

.

4.2 Architecture Summary

When a client visits the website, the static HTML, CSS and Javascript files are served to the client's laptop. These static files are the output of our frontend development which was carried out in React, where the React code is compiled down into static CSS, Javascript and HTML files which can be interpreted by the browser. There are then a number of calls to API routes on our server, as well as directly to external APIs to retrieve data to be displayed to the client on the web page. As the client interacts with the page this results in further API calls, such as to plan a journey from one destination to another. On the web server, trained models will be saved from our modelling process, and these will be used to predict journey times given the time of day, day of the week, start and stop stations. Our web server will also interact with a database hosted on AWS where there is data stored related to bus stations (bus station names and stop IDs) and authentication (such as access tokens for the Spotify API). The server will also directly interact with various external APIs for the purposes of retrieving route directions from google maps, retrieving access tokens from Spotify, and retrieving weather forecasts for the predictive model.

4.2.1 Web Server

The web server is hosted on the UCD virtual machine. The web application framework which we used is Django. We chose Django for a number of reasons. The first is simply that all members of the team are familiar with Python, and this allowed us to focus on learning and understanding the web framework instead of the syntax of a new language.

Django has many useful packages which enable a quick development process without compromising security, and comes with an intuitive object-relational mapper which makes it very easy to interact with our mySql database. Another benefit of Django is that it comes with an inbuilt session management framework, allowing us to introduce personalised aspects of the database without dealing with login credentials. We focused on utilising session keys and cookies so that all functionalities of the application were available to all users. The main tradeoff with Django is simply that Python is not a fast language, and poorly designed api routes and logic can result in a slow website. We kept this in mind throughout the development process and tried to spot potential bottlenecks before they became an issue. In production, we configured gunicorn as the WSGI server and nginx as the reverse proxy with https for encrypting traffic to and from the server.

4.2.2 Frontend

For our frontend framework we used React. React uses compilers to compile the code down into html, javascript and css which is what is served by the website to the client. React has a number of advantages over pure javascript and html. Firstly, it helps with overall code structure and reusability of components. React allows you to build custom components which can be used within larger components using tags. This makes it very easy to segregate different aspects of the web page into separate components which helps with debugging, reusability, and code readability. React also makes it more efficient and easier to manage state and update the user interface using the virtual DOM. The virtual DOM allows the UI to update only those elements and children which have changed in the UI. In comparison, if the DOM is manipulated directly the full real DOM is updated and this can be expensive. React deals with optimising these UI updates efficiently and this proves to be extremely useful in managing a responsive web page.

Alternatives to using React included using pure Javascript, CSS HTML, or using another popular framework such as JQuery or Angular.js. We were focused on providing an extremely responsive and user friendly UI for this project. We felt that leveraging the inbuilt packages and benefits of a frontend framework would help us to achieve this goal. We focused on the learning curve, performance and reliability when choosing our frontend framework. Angular is a widely supported framework, however it is Typescript based and has a steeper learning curve than React. Our research found that component management, complex syntax and complicated core features all contributed to a more difficult learning process. JQuery was another viable candidate which we considered using. The primary benefit we observed for using React over JQuery was the ease of state management through the virtual DOM compared to JQuery which interacts with the DOM directly. In our design it was clear that we would have many linked components - including route input, google maps component, plan route buttons for events and attractions along with others. React appeared to be the best option for ensuring consistency and responsiveness across these components throughout the application.

4.2.3 Database

We used a MySql database hosted on AWS to store any required permanent data for our application. We did not require a significant amount of stored data in our database as we decided to allow personalisation of the application through cookies instead of login functionality. The primary tables in the database included static data related to Dublin bus routes. This data was relational in nature with stops being associated with trips, and trips being associated with routes etc. This meant that a relational database was a perfect choice for our application, and MySql is supported by Django's ORM so it was a good option in the context of our overall architecture.

4.2.4 Data preparation and analysis

For data preparation and analysis, we used Jupyter Notebooks to run our Python code. Jupyter notebooks are useful for data analysis, as they allow for separation of code into blocks that can be run individually. This makes editing code easier, as mistakes or modifications are easier to spot within the smaller blocks of code. For the data preparation, we used the Pandas Python library. For creating machine learning algorithms for predicting bus journey times, we used the scikit-learn Python library. We chose both of these libraries as we had experience of them before, and were able to keep all data-related code in the same notebooks.

4.2.5 External API

We made use of a number of external APIs in the application, including Spotify, Google Places, Google Directions and Ticketmaster. We primarily loaded the interactions with these external APIs on the client side so as to reduce the load on our server. This means that the client's device generally queried these APIs and this populated the static files. For example, we wanted to include artist previews for upcoming events in Dublin and to do this we used the Spotify API. To display an accurate record of what song is playing and the progress of the playback, fresh information related to the user's account needs to be queried regularly. If we performed these requests server-side, the server would quickly become overwhelmed. To overcome this issue we made use of a functionality in React which allowed us to query Spotify from the client-side on behalf of the user at set intervals. This data was shared throughout the application, maintaining an accurate record of the currently playing track on the user's Spotify account. The primary reason for any calls to external APIs from the server-side involved refreshing and retrieving access

tokens for the user's Spotify account. The following diagram illustrates the lifecycle of the Spotify integration within our application

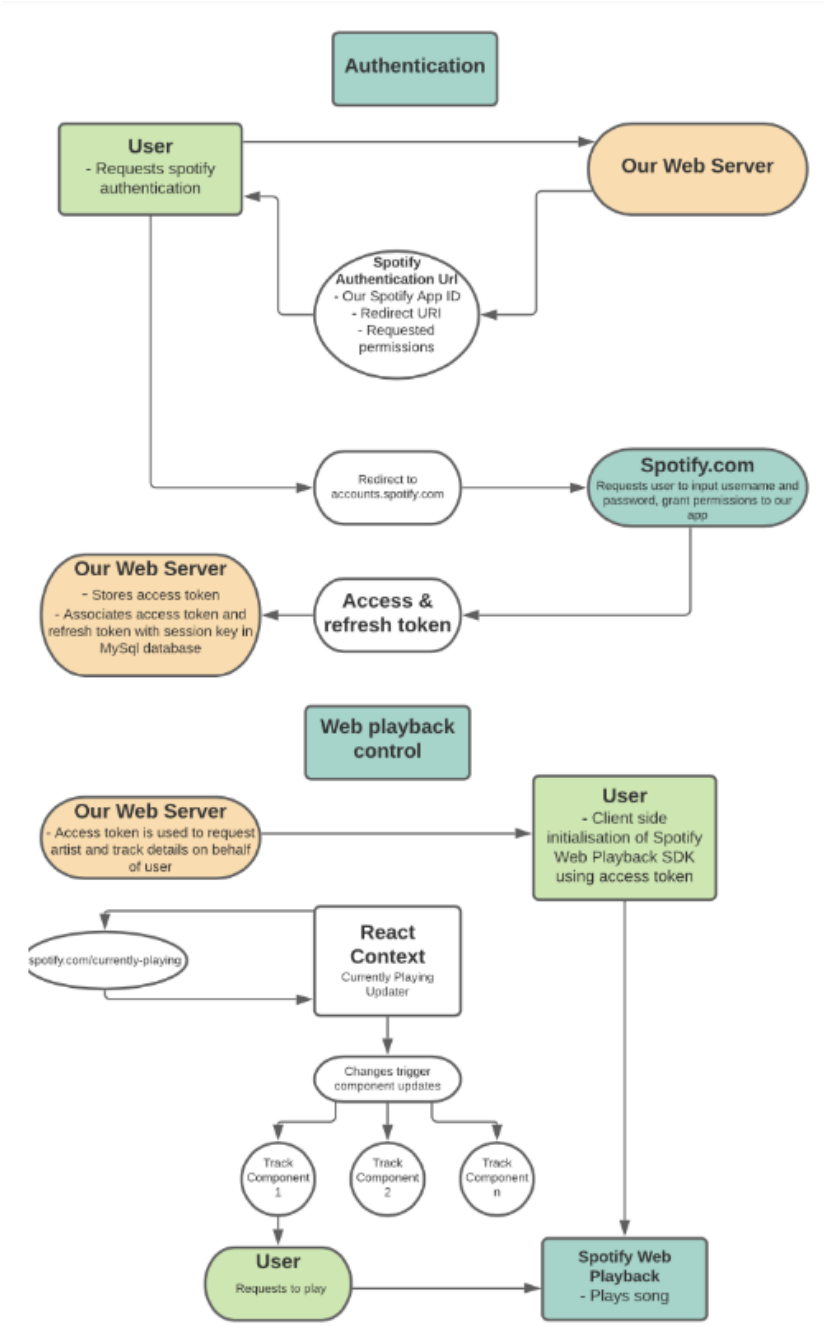


Figure 4.2: Spotify Architecture

.

Chapter 5: **Testing and Evaluation**

API Unit Testing

With regards to the web application, we wanted to design tests which evaluated the correctness of the Django views and we also wanted to test the frontend functionality of the app. The primary views in our application dealt with generating route directions, making predictions, retrieving weather forecasts and sending back route instructions to the user. We broke this down into granular tasks and designed unit tests to test specific functionalities including the following:

- Given an origin and destination pair of coordinates, do we retrieve a valid route directions object from google which includes the legs and steps of a journey using only walking and the bus as forms of transportation
- Given a stop name, is this successfully converted into a unique stop number which can be used as a lookup for associated models.
- Given a valid departure stop number and a valid arrival stop number on a route, do we generate the correct full path For example if we pass in stop number 2021 and stop number 4566 on route 46a, do we generate an array of [2021, 2022, 4565, 4566]
- Given a valid stop to stop pair, is the corresponding linear regression model successfully retrieved.
- Given a datetime, is the correct current weather or weather forecast retrieved from OpenWeatherMap.
- Using the trip time and the stop to stop pair, is this data combined with the weather data to create an input variable of the correct dimension and data type.
- Is a valid prediction time retrieved (in seconds) for this input data.
- Does our overall prediction methodology result in a predicted trip time which is within some reasonable bounds relative to Google's default estimated time

We tested these functionalities using both valid and invalid inputs. We were seeking confirmation that for any valid example the functionality would produce the expected result, and for invalid examples (such as providing an origin stop number and destination stop number which are not connected based on our data) the method would fail so that we could default to Google's estimated trip time. Some of these functionalities resulted in different blocks of code being tested depending on the input. For example, if a stop's name ended in an integer such as "Foxrock Grove, Stop 2020" it was easy to retrieve the stop number. However, a more difficult example would have the input "Foxrock Church" on route "46a" in which case retrieving the correct stop number is less trivial. We made sure to test both hard and easy examples in these cases.

We used Django's testing framework which is based on the unittest library in python. We created test classes which inherit from the TestCase class within the django.test package. This allowed us to automatically setup and tear down test databases when the suite of tests were run, creating an environment for testing which replicated our production environment.

Within our api application in the django project, we created a `test_api` file which is automatically discovered when we run ./manage.py test api in Django. Within this file we created the following classes:

- TestJourneyPlanning, inheriting from TestCase, with the following methods `test_correct_route_id_retrieved_easy`, `test_correct_route_id_retrieved_hard`, `test_correct_path_generated`.
- TestWeather, inheriting from unittest.TestCase (since it did not require the database replication) with the following methods - `test_weather_current`, `test_weather_forecast_3day`, `test_weather_forecast_2week`.
- TestJourneyPrediction, inheriting from unittest.TestCase, with the following methods - `test_model_lookup`, `test_model_inputs`, `test_model_prediction`.

This setup provided us with a suite of tests which would provide a high degree of confidence that our functions and views were working correctly and as expected if all tests passed. We ran these tests throughout development to confirm that our logic was correct, and we also ran these tests whenever code was refactored or modified to ensure that there were no breaking changes

5.0.1 Website Functionality and Usability

After website deployment, we put together a short survey for users to fill out. This included a number of simple questions such as "How would you rate the overall look and feel of the website?", "What did you like/dislike about the application?" and "Where could we improve?".

This provided us with feedback from users on the usability of our application and resulted in a number of changes to the frontend. For example, one piece of feedback stated that there was a lack of a connection between the sidebar and the route description for planned journeys. We updated the frontend to include the route description in the sidebar underneath the inputs for planning a route. A second critique noted that we had not included the route number of the bus journey in our route description. We then added in a clear icon indicating the route number for any bus journeys on our route. Another user noted that our inputs for origin and destination had dropdown arrows which made them believe it was not a textual input, and this caused some confusion in how to actually enter destinations. We responded to this by removing the dropdown errors and changing the default text to more meaningful statements.

The anonymous survey was extremely useful in receiving unbiased feedback on our application and highlighted a number of important flaws which we had overlooked in development.

Chapter 6: **Major Contributions**

My technical role on the team was in data analysis and modelling. In terms of the web development aspect of the project, I remained in daily contact with the team and gave high-level opinions and ideas. However, I was not involved in any web development at a technical level.

In our first team meeting, we introduced ourselves and informally discussed what we would like our roles on the team to be. It worked out neatly: Brian was interested in back-end development; Cormac, in front-end development, Henry wanted to do a bit of everything; and I was mainly interested in the data analytics and machine learning.

We also discussed what technologies we wished to use. This was an important choice to make. We are still beginners, so we considered learning how to use new technologies to be an important part of the project. At this point, I was not confirmed at just doing data analytics, so I did research on what development tools we could use. We all had previous experience with Python Flask, and plain JavaScript from the software engineering module in the previous trimester. After research, we decided that for front-end, we would use React: a JavaScript framework; and for back-end we would use Django: a Python framework. We chose these forto learn new frameworks for web development, and because of the popularity of these frameworks in the industry.

After our first meeting, I set up a Github repository for the team, and also prepared a Github cheat sheet for members of the team to look at if there was anything about Github that they had forgotten.

My role from here became more focused on the data we had. I knew from the project spec that we had to use historic Dublin Bus and weather data from 2018. I also looked into other sources of traffic-related data, such as the Smart Dublin General Transit Feed Specification Realtime (GTFS-R) API, and SDCC Traffic Congestion Saturation Flow Data.

It was my decision – which the team agreed with – to just use the Dublin Bus data and OpenWeatherMap data for prediction of journey times. There were multiple reasons for this. Firstly, the more data used in creating prediction models, the more data needed as inputs for the models. As we had a limited amount of storage to host our web app, working with less data for model creation was seen as an advantage. Secondly, the approach to most other studies on this topic was to train models using historic data and to use the current ime and weather conditions as inputs for prediction in realtime. This could be seen as a good reason to use historic traffic data in modelling, and realtime traffic updates in realtime prediction – it would be a different approach to a lot of these types of studies that have already been done. However, due to the concerns about storage space for the app, and my desire to keep the modelling process simple, I made an early decision to focus just on the historic bus and weather data in model creation.

Having made the decision on which data we would use, I turned my attention to data exploration. For the next few weeks, I investigated the data. My initial goal was to create prediction models for each stop-to-stop journey in a single bus line.

Why did I model stop-to-stop journeys rather than model entire bus lines? Intuitively, this way made more sense to me, as I will now explain. The other option would be to model the entire trip from start to finish, and although these kinds of prediction models tend to be lower in error over all, I did not see it as being the best option for a user application. Most people take the bus for a portion of its entire route, rather than from the first stop to the last stop. As well as this, models of the entire line would be less accurate in specific parts of the journey. For example, if someone

gets the bus from Nassau Street in the centre of Dublin, to the suburb of Castleknock during rush hour, it is possible that the part of this journey that goes through the city centre is going to proportionally take more time than the rest of the trip. Traffic conditions in the city centre are generally worse than on main roads and suburban areas. This is due to more traffic being in the city; there are more traffic lights; there are fewer lanes for traffic to travel in. Therefore, if the journey takes one hour in total, half of that may be in the city centre portion of the journey despite less than half of that journey taking place in the city. My reasoning here, was that models of stop-to-stop journeys will more accurately reflect the traffic conditions in a specific area than models of entire trips would do.

Why did I choose an entire bus line first? With data for an entire bus line, the data is all related. Journey times can be compared to one another in a way that makes sense. As well as this, mistakes to look out for in the data could be seen when all journeys were part of the same bus line.

Some trips were missing stop-to-stop journeys. For example, there may have been no data recorded between the fifth and seventh stop of the trip. These entire trips were removed in case the missing data for journeys had knock-on effects for the subsequent journeys.

However, it was also important to note the different ROUTEIDs in the data – variations in the bus trip, but still part of the same bus line. These variations in bus lines had different combinations of stops. Some ROUTEIDs were missing a value for the PROGRNUMBER. The PROGRNUMBER is described as the 'sequential position of the stop point in the trip' in the documentation for the historic datasets'. So, PROGRNUMBER 1 is the first point of the trip between the first and second stops, PROGRNUMBER 2 is the second point of the trip bewteen the second and third stops, and so on. The 46A_62 ROUTEID was missing PROGRNUMBER 30. That is to say there are no recorded journeys in the entire dataset for that ROUTEID with PROGRNUMBER 30. Therefore it would appear that these journeys are incomplete. However, if there is no data for a specific PROGRNUMBER in the whole dataset, we can take it that that particular ROUTEID does not have the PROGRNUMBER as part of its journey. Thus, some trips that appear to be incomplete, are not. It was important to check for this before removing seemingly incomplete trips from the data.

After being sure that I was only working with journeys from complete, consecutive trips, I added the JOURNEYTIME feature. This was made by getting the arrival time at each stop and subtracting it from the arrival time at the stop after it in the dataset – the dataset was sorted by PROGRNUMBER – provided the TRIPID and date were the same. This had the added benefit of taking the bus dwell time information into account.

After creating the JOURNEYTIME feature, then I found more data that needed to be removed. For example, there were stop-to-stop journeys that were recorded as arriving at their destination stop before they had left their origin stop – giving them a negative journey time – which is obviously impossible. Any trips that had a stop-to-stop journey with a negative journeytime were removed. There was a lot of data to work with, and if one stop-to-stop journey was incorrectly recorded, it was likely to have a knock-on effect on the rest of the stop-to-stop journeys for that trip. This is why it was decided to remove the entire trip and not just the negatively-timed stop-to-stop journey.

I also found a lot of journeys that most often occurred between the second-last and last stops of a trip that had abnormally long journey times. Some were as long as 2 hours. I suspected this to be a mistake, especially when comparing these journeys to the average. So it was also my idea to remove outliers in terms of journey time. For this, I removed any journeys that were more than three standard deviations from the mean of all equivalent stop-to-stop journeys. This removed journeys that I suspected were badly recorded – if the average journey time between two stops was 100 seconds, and there was a journey between the same two stops that took over

5000 seconds, it would be safe to assume that this was an incorrectly recorded journey. I chose to remove everything more than three standard deviations from the mean, based on (Jajodia, 2017) from kdNuggets.

I would say that these data insights were my main contribution to the modelling process. I was responsible for the creation of other data features, such as daySegment: I divided the day into 30 minute segments, as a way of categorising what time the journey took place; publicHoliday: a binary categorical feature that had a value of 1 if the date was any of the dates here, and 0 otherwise; schoolterm: a binary categorical feature that had a value of 1 if the date was during school term, and 0 otherwise; rush hour for morning and evening: both binary categorical features that had a value of 1 if the time of the journey was between 7.00 AM and 9.30AM, or 4.00 PM and 6.30 PM respectively, and 0 otherwise. However, when testing models made using these features, versus not using them, the metrics were better for the models that did not use the created features.

My main technical contribution – the data insights are more a high-level contribution – was creating linear regression models for all stop-pairs for the following bus lines: 47, 79, 83A, 63, 4, 120, 41C, 70, 84A, 220, 39X, 32X, 68A, 84X, 38, 102, 270, 51X, 33X, 75, 26, 66A, 31A, 111, 14C, 114, 76A, 44B, 161, 7A, 43, 25, 104, 33A, 16C, 42D, 31B, 66X, 31D, 33D, 41B, 40B, 7D, 46E, 38D, 118, 51D, 15D, 41A, 25D, 66B, 38B, 236, 7B, 41X, 69X, 68X, 25X, 40E, 70D, 116, 77X, 16D, 33E, 41D, 46A.

I also created random forest models for the 90 most-used stops, using the same data features as the linear regression models. I simply found which stop-pairs from the leavetimes dataset had the most entries, and made models for these. I decided that random forest models would be a good addition provided they did not take up too much space. They performed slightly more accurately than the linear regression models. With the random forest models made for the most-used stop-pairs, 85 had MAE values of 30 or less, and all were had MAE values of less than 35.

Chapter 7: **Background Research**

7.1 Literature Review

In this section, I will review works that address a similar problem to the one this project addresses – particularly within the realm of prediction. I will point how this project addresses prediction differently, and why we chose to do things the way we did. The literature will be categorised according to the methods used.

Historical averages: Used by (Fan and Gurmu, 2015), (He et al., 2019). This method assumes that traffic conditions will remain the same. It will generate the current and future travel time based on historic bus travel time data, assuming the traffic conditions will not change. These predictions were used a baseline model for this study, to compare kalman filtering and artificial neural networks against. However, the performance of the models are weak unless the traffic pattern in the area is consistent and not subject to sudden changes in traffic conditions. As a result of this model being too inaccurate, except for when traffic conditions remained the same, and because we were trying to create prediction models for buses around an urban area – where traffic conditions change often – we decided not to use historical averages for this project.

Artificial Neural Networks (ANN): Used by papers (Fan and Gurmu, 2015), (Ma et al., 2019). This method finds relationships in the data that are not immediately obvious. It is becoming more popular in prediction modelling because of its high accuracy. (Fan and Gurmu, 2015) found ANN models to be the best at predicting bus travel time using GPS data, scoring the lowest mean absolute percentage error across three segments of a bus route in Macae, Brazil, when compared with models made using historical averages and klaman filtering methods. However, ANNs are computationally expensive to run and to generate. They are also not very adaptable: that is to say they will perform well for specific bus routes, or for stop-pairs, but they will need to be trained for each of these bus routes or stop-pairs, which will use up more computational resources. This project was a user application, and we had limited space and processing power to work with. ANNs would have taken more space than we had, so we decided not to use them.

Kalman Filtering: Used by paper (Fan and Gurmu, 2015), (Kumar, Vanajakshi, et al., 2017), (Kumar, Jairam, et al., 2019). This method is used to estimate the future state of the target feature (journey time) based on past examples. They are widely used and have the advantage of being continuously updated. However, it requires real-time updates continuously to work well (Ma et al., 2019, p. 538). As well as this, in (Fan and Gurmu, 2015) the results for kalman filtering models did not perform as well as ANNs in bus journey time prediction across three bus route segments. As per the project specification, we were tasked with creating prediction models, using historic bus data. Using real-time data on top of the historical models was something we decided against, as it would be complicated to use both live and historic information.

Support Vector Machine (SVM): Used by paper (Yang et al., 2016), (Ma et al., 2019), (He et al., 2019), (Bin et al., 2006), this is a supervised machine learning algorithm that is mostly used for classification problems, but can be used for regression problems as well. Ultimately, we decided against using SVMs because they can take long to train and to predict. This app was made with usability in mind, so models that can be trained and make predictions faster were prioritised for testing. (Ma et al., 2019) split up bus journey time prediction into transit time and dwelling time. The paper found this method to be the best at predicting bus transit time versus ANN and k-NN

algorithms, scoring the lowest MAE (seconds) across three bus routes in Xi'an, China.

Linear regression: Used by (Yang et al., 2016), (He et al., 2019). This is a machine learning method that predicts a target feature, based on other features, such as weather and traffic conditions. In (He et al., 2019), These models work with the absence of traffic data as they take into account multiple features that are independent of one another in predicting the target feature. These models are quick to train and run and do not take up much space. Because of this, we decided to generate linear regression models as a baseline to compare random forest models against.

Random Forests: As used in paper (B. Yu et al., 2018). Random forests is made up of multiple decision tree algorithms. Each decision tree takes data from the dataset at random, and a subset of the dataset's features at random. For a regression problem, the average prediction of each created decision tree is used as the prediction value. This random choice of data and subsets of features make random forests less affected by changes in the dataset than decision trees. It was because of this, that we decided to use random forests to build our models for the most-used stops. An example of random forest being used for bus travel time prediction is in (B. Yu et al., 2018), where two bus routes are modeled using random forest based on near neighbour method. The results of these models are then compared with linear regression, k-nearest neighbours, support vector machine and classic random forest models on the same data. While the random forest-near neighbour approach scored the best metrics for both routes, the time it took to run the models was very long: 44,301 seconds for one of the routes. Classic random forest performed the second-best of the models for both routes – it was less than two percent higher than random forest-near neighbour in MAPE – and took far less time to run: 1,241 seconds versus 44,301 seconds for the same route.

7.2 Technologies and Programming Tools Used

As part of the project, we were given access to a high-speed server for data operations. All data analysis and modelling was done in Python Jupyter Notebooks. Running Jupyter Notebooks on the remote server, I isolated specific bus routes from the entire leavetimes dataset so that I could work on them locally.

For data analysis and preparation on these samples from the dataset, I used the Pandas library with Python. I have experience with Python and Pandas from the Data Analytics module in the Spring Trimester.

For plotting results and statistics, I used the Matplotlib Python library in Jupyter Notebooks. Again, I have previous experience with Matplotlib and find it to be a useful tool to visualise results and statistics.

For modelling, I used the scikit-learn Python library in Jupyter Notebooks. This library has a selection of ready-to-use models and is in Python. I also had previous experience with scikit-learn, which is why I decided to use it.

Chapter 8: **Critical Evaluation**

8.1 Overall conclusions

From my perspective, the overall project specification was met: we created a web-based application that was optimised for mobile devices. The app produces dynamic travel-time estimates for the complete route or selections of the route, based on models created using historic Dublin Bus and weather data.

The prediction models are reasonably accurate, but not as accurate as could be, if space and processing power were not an issue. For example, model creation using ANNs or SVMs may have proven to be more accurate than even random forest models, but due to space and computational constraints, I decided not to create these types of models.

There was innovation on the web development side of things, with the ability for people to check what events are on, and to preview music artists on Spotify to see if they would like to see them.

8.2 Weakness of approach

There were a number of things I would do differently now that I have completed the project.

Firstly, I would look into all model approaches that I mentioned in the background research chapter and create models from all of these approaches. I would be curious to see how accurate each approach is, while also seeing how long they take to make a prediction, and how much space each would take up. I think that after background research, I was too set on what my approach was going to be: linear regression and random forest models. If I had been more broad in my approach, I would have had more models to compare metrics with. This would have made me surer that the approach I went with was the correct approach.

I also think it would have been worth trying different amounts of data for the creation of models that normally take up more space, like random forests. So, instead of using as much data as possible for each stop-pair model, it might have been worth trying with half the data, and seeing how the metrics compared.

In hindsight, it also would have been interesting to see if traffic data would have been useful for prediction in combination with the historic bus and weather data. As I established before, I was trying to keep things as simple as possible when it came to model creation. I was also concerned about the amount of space taken up by the models, so keeping the data to as few sources as possible was a priority. However, it is possible that using traffic data would have worked well for prediction. If it did, it may have been possible to use a portion of the data for each stop-pair while also using traffic data for prediction. This could have saved on space, while also improving prediction metrics.

After cleaning a single route and gaining the insights I did about the bus data in particular, I would generate stop-pair models from the entire leavetimes dataset, rather than generating models for

each stop-pair based on the route it is in. Having a better knowledge of the dataset – as well as a greater fluency with the Pandas library – would allow me to do this easily. It would allow for lots of models to be made with access to more data across different bus lines. I think that the approach we went with – creating models for each stop-pair across each route – was a wasteful and less accurate approach.

I spent a lot of time with data cleaning and exploration. However, I think I could have spent longer on it. Throughout the project I felt a time pressure to get models that returned accurate results. But the time pressure did hinder my ability to see things from a broader perspective at times. An example of this would be for the removal of outliers in the data. I could have found a better approach than removing entries with journey times three standard deviations away from the mean, as the dataset we were working with did not have a standard distribution. The way I chose to remove outliers did work quite well, but due to time pressure, I felt that I wasn't able to find an approach more suited to our data.

As well as this, I think we as a team could have been more collaborative over all. We communicated well and often. However, Henry and I who both did most of the data analytics did not explain our approach or rationale for our decisions to the other team members very often. I think trying to explain one's approach to people who aren't involved with the same part of the project is a good way of catching potential blind-spots and a good way of maintaining a big-picture focus, rather than getting too focused on minute details.

8.3 Future Work

For the future, I would look at other modelling approaches, and I would make models based on stop-pairs from the entire dataset rather than stop-pairs from each route. I would also look into making models with less overall data and compare the metrics with the models we already made.

I would also try to incorporate traffic data into the modelling to see if the models could be made more accurate that way.

Bibliography

Apple (2020). URL: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.

Baptista, Arthur Trigueiro et al. (2011). "Towards building an uncertainty-aware personal journey planner". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems*. DOI: http://dx.doi.org/10.1145/2858036.2858558.

Bin, Yu, Yang Zhongzhen, and Yao Baozhen (2006). "Bus Arrival Time Prediction Using Support Vector Machines". In: *Journal of Intelligent Transportation Systems Technology, Planning, and Operations* 10.4, pp. 151–158. DOI: 10.1080/15472450600981009.

Cristóbal, Teresa et al. (2019). "Bus Travel Time Prediction Model Based on Profile Similarity". In: *MDPI Sensors*.

Dai, Zhuang, Xiaolei Ma, and Xi Chen (2018). "Bus travel time modelling using GPS probe and smart card data: A probabilistic approach considering link travel time and station dwell time". In: *Journal of Intelligent Transportation Systems Technology, Planning, and Operations* 23.2, pp. 175–190. DOI: 10.1080/15472450.2018.1470932.

Dandekar, K., B. Raju, and M. Srinivasan (2003). "3-D finite-element models of human and monkey fingertips to investigate the mechanics of tactile sense." In: *Journal of biomechanical engineering* 125 5, pp. 682–91.

Dobres, Jonathan et al. (2018). "The effects of visual crowding, text size, and positional uncertainty on text legibility at a glance". In: *Applied Ergonomics* 70, pp. 240–246. ISSN: 0003-6870. DOI: https://doi.org/10.1016/j.apergo.2018.03.007. URL: https://www.sciencedirect.com/science/article/pii/S0003687018300681.

Fan, Wei and Zegeye Gurmu (2015). "Dynamic Travel Time Prediction Models for Buses Using Only GPS Data". In: *International Journal of Transportation Science and Technology* 4.4, pp. 353–366.

Gal, Avigdor et al. (2017). "Traveling time prediction in scheduled transportation with journey segments". In: *Information Systems* 64, pp. 266–280. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2015.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0306437915002112.

Google (2020). URL: https://developer.android.com/guide/topics/ui.

He, Peilan et al. (2019). "Travel-Time Prediction of Bus Journey With Multiple Bus Trips". In: *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS* 20.11. DOI: 10.1109/TITS.2018.2883342.

Jajodia, Punit (2017). *Removing Outliers Using Standard Deviation in Python*. URL: https://www.kdnuggets.com/2017/02/removing-outliers-standard-deviation-python.html.

Jeong, R. and R. Rilett (2004). "Bus arrival time prediction using artificial neural network model". In: *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pp. 988–993. DOI: 10.1109/ITSC.2004.1399041.

Julio, Nikolas, Ricardo Giesen, and Pedro Lizana (2016). "Real-time prediction of bus travel speeds using traffic shockwaves and machine learning algorithms". In: *Research in Transportation Economics* 59, pp. 250–257. DOI: 10.1016/j.retrec.2016.07.019.

Kay, Matthew et al. (2016). "When (ish) is My Bus? User-centered Visualizations of Uncertainty in Everyday, Mobile Predictive Systems". In: *CHI Conference 2016*. DOI: http://dx.doi.org/10.1145/2858036.2858558.

Kumar, B. Anil, R. Jairam, et al. (2019). "Real time bus travel time prediction using k-NN classifier". In: *Journal of Intelligent Transportation Systems Technology, Planning, and Operations* 11.7, pp. 362–372. DOI: 10.1080/19427867.2017.1366120.

Kumar, B. Anil, Lelitha Vanajakshi, and Shankar C. Subramanian (2017). "A hybrid model based method for bus travel time estimation". In: *Journal of Intelligent Transportation Systems Technology, Planning, and Operations* 22.5, pp. 390–406. DOI: 10.1080/15472450.2017.1378102.

Ma, Jiaman et al. (2019). "Bus travel time prediction with real-time traffic information". In: *International Journal of Transportation Science and Technology* 105, pp. 536–549.

Mori, Usue et al. (2015). "A review of travel time estimation and forecasting for Advanced Traveller Information Systems". In: *Transportmetrica A: Transport Science* 11.2, pp. 119–157. DOI: 10.1080/23249935.2014.932469. eprint: https://doi.org/10.1080/23249935.2014.932469. URL: https://doi.org/10.1080/23249935.2014.932469.

Pandurangi, Ankhit et al. (n.d.). "Design and Development of an Application for Predicting Bus Travel Times using a Segmentation Approach". In: *GISTAM 2020 - 6th International Conference on Geographical Information Systems Theory, Applications and Management*.

Reich, Thilo et al. (2019). *Survey of ETA prediction methods in public transport networks*. arXiv: 1904.05037 [cs.CY].

Shalaby, Amer et al. (2002). *Bus Travel Time Prediction Model for Dynamic Operations Control and Passenger Information Systems*.

Sinn, Mathieu et al. (2012). "Predicting arrival times of buses using real-time GPS measurements". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 1227–1232. DOI: 10.1109/ITSC.2012.6338767.

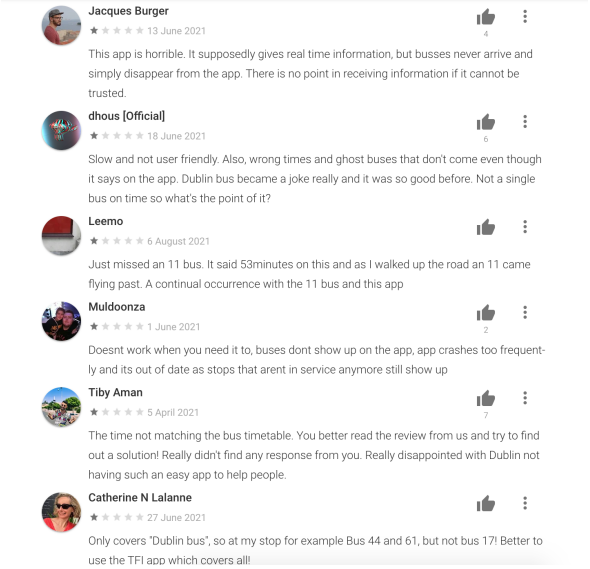
Yang, M. et al. (2016). "BUS ARRIVAL TIME PREDICTION USING SUPPORT VECTOR MACHINE WITH GENETIC ALGORITHM". In: *Neural Network World* 3, pp. 205–217. DOI: 10.14311/NNW.2016.26.011.

Yu, Bin et al. (2018). "Prediction of Bus Travel Time Using Random Forests Based on Near Neighbors". In: *Computer-Aided Civil and Infrastructure Engineering* 33, pp. 333–350.

Yu, Haitao et al. (2013). *A Bus-Arrival Time Prediction Model Based on Historical Traffic Patterns*.

Appendix A: **Appendix A**

A.1 Dublin Bus Application Reviews



Jacques Burger
★☆☆☆☆ 13 June 2021
4
This app is horrible. It supposedly gives real time information, but busses never arrive and simply disappear from the app. There is no point in receiving information if it cannot be trusted.

dhous [Official]
★☆☆☆☆ 18 June 2021
6
Slow and not user friendly. Also, wrong times and ghost buses that don't come even though it says on the app. Dublin bus became a joke really and it was so good before. Not a single bus on time so what's the point of it?

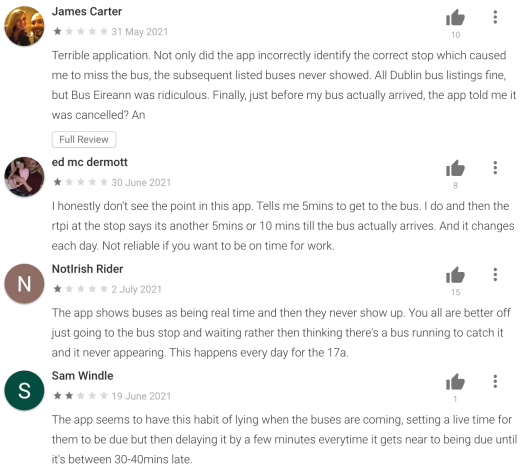
Leemo
★☆☆☆☆ 6 August 2021
Just missed an 11 bus. It said 53minutes on this and as I walked up the road an 11 came flying past. A continual occurrence with the 11 bus and this app

Muldoonza
★☆☆☆☆ 1 June 2021
2
Doesnt work when you need it to, buses dont show up on the app, app crashes too frequently and its out of date as stops that arent in service anymore still show up

Tiby Aman
★☆☆☆☆ 5 April 2021
7
The time not matching the bus timetable. You better read the review from us and try to find out a solution! Really didn't find any response from you. Really disappointed with Dublin not having such an easy app to help people.

Catherine N Lalanne
★☆☆☆☆ 27 June 2021
Only covers "Dublin bus", so at my stop for example Bus 44 and 61, but not bus 17! Better to use the TFI app which covers all!

Figure A.1: Dublin Bus app reviews



James Carter
★☆☆☆☆ 31 May 2021
10
Terrible application. Not only did the app incorrectly identify the correct stop which caused me to miss the bus, the subsequent listed buses never showed. All Dublin bus listings fine, but Bus Eireann was ridiculous. Finally, just before my bus actually arrived, the app told me it was cancelled? An
Full Review

ed mc dermott
★☆☆☆☆ 30 June 2021
8
I honestly don't see the point in this app. Tells me 5mins to get to the bus. I do and then the rtpi at the stop says its another 5mins or 10 mins till the bus actually arrives. And it changes each day. Not reliable if you want to be on time for work.

NotIrish Rider
★☆☆☆☆ 2 July 2021
15
The app shows buses as being real time and then they never show up. You all are better off just going to the bus stop and waiting rather then thinking there's a bus running to catch it and it never appearing. This happens every day for the 17a.

Sam Windle
★★☆☆☆ 19 June 2021
1
The app seems to have this habit of lying when the buses are coming, setting a live time for them to be due but then delaying it by a few minutes everytime it gets near to being due until it's between 30-40mins late.

Figure A.2: TFI app reviews