Research Practicum Project Report

Predicting Dublin Bus Journey Times

Shen Chen, Wu Di, Daniel O'Byrne, Sybilla Robertson

A thesis submitted in part fulfilment of the degree of

MSc. in Computer Science (Conversion)

Group Number: 4

COMP 47360

UCD School of Computer Science

University College Dublin

August 20, 2018

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, day of the week, 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 GPS data and weather data in order to create dynamic travel time estimates. Based on data analysis of historic Dublin bus GPS data, a system which when presented with any bus route, departure time, day of the week, current weather condition, produces an accurate estimate of travel time for the complete 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 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

On an average week day, Dublin Bus transports approximately 400,000 passengers with 1,000 buses and 5,000 stops. Dublin Bus is currently responsible for 70 percent of all bus commuters in Dublin during peak times [6]. Given the citys reliance on these services, providing accurate and relevant journey information to users is of the utmost importance.

According to Transport for Ireland (TFI), the system that provides Dublin Bus users with journey time information has a margin of error that is less than one minute for predictions of five minutes or less [15]. However, as regular Dublin Bus users will be acutely aware, this system can often provide misleading prediction times. Such misinformation may leave users waiting aimlessly for a bus that should have arrived several minutes previously. Apart from the inconsistencies in travel time predictions, the application Dublin Bus provides is limiting. Its value can only be realized by regular users who know exactly where they are going. Tourists and users who want to plan a journey with Dublin Bus will have to look elsewhere for a useful application.

This report outlines the project undertaken to solve this problem. The described application, also known as Dublin Bashi, is available at www.dublinbashi.com.

Acknowledgments

The team would like to thank the staff in UCD School of Computer Science for supporting us throughout this project. We are grateful for the presentations, materials, assistance and advice provided by the project mentors and demonstrators throughout this research practicum.

Finally, a special thanks to Daniel our Coordination Lead, Wu Di our Code lead, Shen Chen our Customer Lead, and Sybilla our Maintenance Lead for putting up with each other for twelve weeks to deliver an application that meets and exceeds the given requirements.

Table of Contents

Chapter 1: **Introduction**

This project has presented the opportunity to address the local issue of predicting travel times for Dublin Bus services. Regular users will be painfully aware of this issue as actual travel/arrival times deviate from the provided scheduled times, all too often. A familiar scenario for these users would involve their bus being due to arrive for several minutes. Or worse still, the bus disappearing from the schedule and never arriving. For more casual users, such deviations can be costly in trying to plan a once off journey. These issues are particularly poignant in suburban areas where services are less regular.

The goal for this project is to use historic Dublin Bus data, our expertise and our research capabilities, to create a web-based application that will accurately predict travel time for a given journey. This prediction will be calculated using a wide range of data sources, including weather, real-time and historical data. The application will be optimised for mobile use, given that users for such an application would likely need this information on the move.

The application will be a reliable, quick, easy-to-use and stylish application that goes beyond the minimum requirements. Besides creating a great application, other aims include: thoroughly but effectively make decisions as a team; apply Agile methodologies to aid our progress; overcome any conflicts in a professional manner; embrace the experience of working full-time on a full stack project; gain experience using new technologies; and gain a greater appreciation for research and its role in projects such as this.

1.1 Project Motivations

With almost thirty percent of the countries population now located in the capital, pressure continues to mount on public transport services across the city. Dublin Bus has been an integral part of this transport infrastructure since its inception in the 1980s. However, the service's integrity can often be scrutinised by users, due to the consistent misinformation the company provides regarding journey times. Although bus usage figures suggest a positive trend in recent years [4], improvements in their journey prediction times would likely heighten the extent of this trend.

Increased use of Dublin Bus services is vital for the country to reach carbon emission targets, set out in the Governments National Mitigation Plan. Ireland is currently on track to miss their 2020 target [12]. Increased investment in this area is likely the most appropriate solution to increase the use of public transport. However, if users can access a reliable prediction of journey times for the current services, this will make their experience much smoother, thus leading to increased use.

1.1.1 Existing approaches and Limitations

There are many applications which already provide journey times to Dublin Bus users. The Dublin Bus application being the first port of call for most. The application can search buses by stop IDs, stop addresses or route IDs. Thus, in the event that a tourist wanted to plan a journey from

landmark A to B, it would be quite difficult unless there are bus stops nearby. The application can only handle stop-to-stop journeys that are on the same route. Without providing walking instructions to stops or cross-route functionality, the application is not a suitable route planner.

Google Maps is the leader in the world of journey planning. Their time predictions are very accurate given the amount of data they have amassed. Due to the size of capabilities Google itself provides, the application comes with a plethora of extra features, such as hotel or bar recommendations. Such features are not necessarily important to Dublin Bus users and can often be a distraction to the route planning itself.

Hit The Road was the application we took most inspiration from, out of the existing approaches. Although it does also give other forms of public transport, it gives a selection of different route options for each journey. The application is simple with few frills to distract the user from quickly retrieving their journey times. It also gives valuable information about bus fares and carbon emission savings versus taking a car journey.

1.2 Overview of Project

With these downfalls of existing applications in mind, we set out to build an application that uses historic Dublin Bus data to accurately predict journey times for users. Several modelling methods were explored, and the most accurate and reliable method was chosen. These modelling techniques were then deployed through a web application. Routes are selected by the user where the time and date can also be chosen. The application acts as a convenient alternative to the aforementioned existing approaches. By focusing solely on Dublin Bus, we feel regular Dublin Bus users and tourists will find great value in the application.

1.3 Report Structure

There are five chapters in our group report. The first chapter is the introduction to our project, which includes the problem being addressed, the aim of the project, the limitations of the existing approaches and the overview of project.

Chapter 2 describes the application's functionality in detail. This section is divided into how the project specifications were met and how innovations were applied beyond the specifications, with relevant screenshots presented.

Chapter 3 outlines our development approach. This chapter includes how the team applied Agile methodologies on a daily basis and throughout. Conflict resolution, team communication and documentation methods are also discussed as part of the development approach.

Chapter 4 presents the technical approach taken in creating the application. This section discusses how the application's prediction models were iteratively developed and finalised. The technical stack utilised is also discussed, with justifications and alternative solutions at each stage.

Chapter 5 discusses the testing and evaluation methods deployed in the application's development. Testing of various forms occurred in the modelling process as well as in the creation of the front and back-end structures. The usefulness of the application, its weaknesses and areas of potential future work are also discussed in chapter 5.

Chapter 2: **Description of Final Product**

2.1 Addressing the Project Specification

The Dublin Bashi application uses Artificial Neural Network (ANN) models (discussed further in Chapter 4) to predict bus journey times for a requested route. These models have been trained using historic Dublin Bus data. The application takes an origin and destination input by the user (Figure 2.1). The user also inputs the time and date of the journey they would like information for.

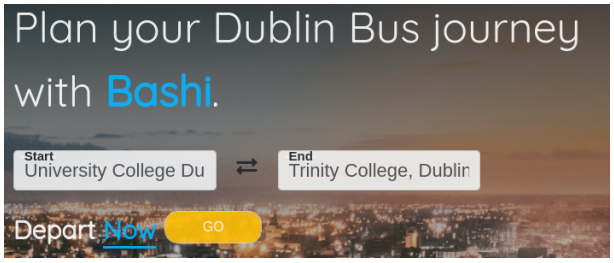

Figure 2.1: User input area

This information is then used to calculate a time prediction for the journey, returned from the back-end structure of the application. Predictions can be calculated for a whole route or parts of a particular route.

Figure 2.2: Route option information

The application is a web based interface that the users can interact with, as seen in the Figures below. The application is optimised for mobile devices. When a screen of mobile size is used, bootstrap methods [3] are utilised, adjusting elements to be sleekly laid out. The navigation bar collapses into a dropdown menu, and the user input area re-formats to the screen. The routes are shown on the map as does the desktop version, with each route option appearing vertically(Figure 2.3), as opposed to horizontally in Figure 2.2.

2.2 Innovation Beyond Specifications

The Dublin Bashi application was created using the web-design philosophy of the Three-Click Rule [13]. The user should be able to retrieve information regarding their required route in as few clicks as possible. To achieve this goal, the application defaults the route's origin to the users current location, and the time of departure to Now, being the current time. The user then has

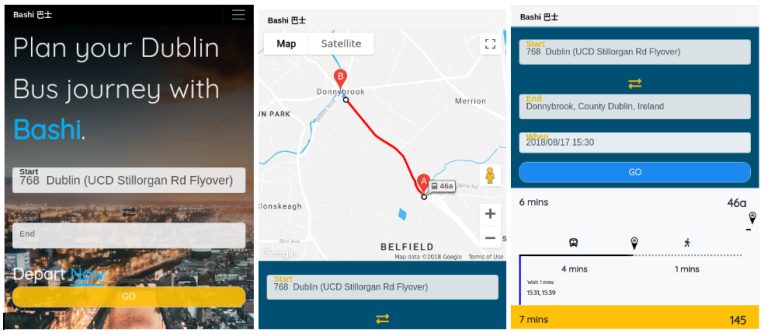

Figure 2.3: Mobile phone view

to simply enter a destination, and click GO.

This design philosophy does however make some assumptions about our potential average user. Such a user would use this application at times where they are looking to get a bus now, from where they currently are. These observations come from personal experiences within the team, as well as interviewing other bus users who have used similar applications in the past.

Furthermore, this application can just as easily be used as a trip planner where the user can input any origin/destination and departure time. The Start and End input boxes above show the ability of the application to take exact addresses. Users can also enter stop ID numbers or stop addresses. Whatever is entered, the application will show autocomplete suggestions for both bus stops and exact addresses (Figure 2.4).

The departure time and date can then be selected from the dropdown provided, rather than the defaulted Now value. The user can select a date up to five days from the current day. This time period is chosen as only five days of accurate weather forecasting is available.

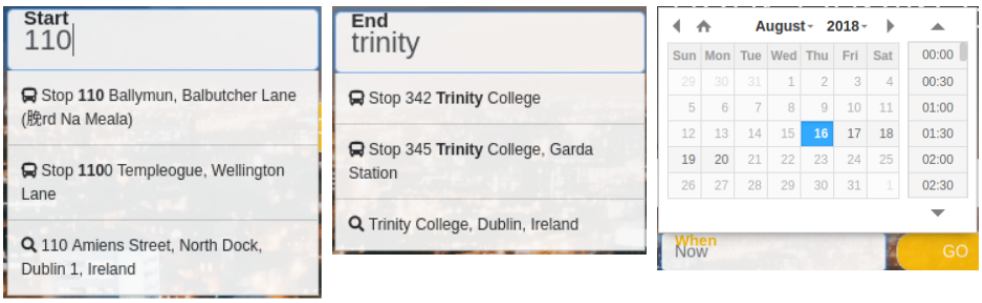

Figure 2.4: Origin/destination/time inputs

Upon clicking GO, the origin/destination, time and weather are sent to the applications back-end to be used in returning journey prediction times. The user will be presented with a map, powered by Google, with the requested route drawn clearly.

The application does not allow the user to choose the route line they would like to take. Journey planner applications are used to accurately inform the user of the quickest way to get to a requested destination. Thus, we felt giving the user this option would be redundant. Rather, the application chooses the best route for them.

The application uses Googles ability to calculate and map the routes between the given origin and destination. On clicking GO, the route shown on the map is the route option that Google has evaluated to have the lowest journey time. Below the map, we show all possible route options returned by Google. However, we present our own journey time predictions, split into each of its parts.

As can be seen in Figure 2.1 and 2.2 above, there is a switch button beside the origin/destination input boxes. This allows the user to evaluate the return journey, where the time selected can be adjusted to some time in the future.

The order in which the routes are listed is according to Googles evaluation of journey time. As the application takes 1-2 seconds to return journey times for all route options, the team did not want to show the user a blank screen for that time. Thus, the routes are shown in Googles order. The user can see the route for those number of seconds and if they are in a rush, they can begin walking to the origin stop. The user can retrieve walking instruction by hovering over the walking icon on the route option plan (Figure 2.5).

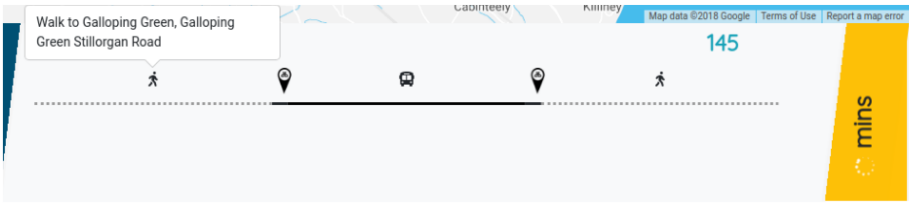

Figure 2.5: Route option loading

As the order shown is decided by Googles response, it is possible that the application's predictions wont be in line with Googles. This results in the route options not being shown in order of fastest-to-slowest. However, the application clearly shows the total time for each option. Thus, if there is some discrepancies with Googles results, the user can simply click into the fastest route option.

When the user selects one of the route options, that route is shown on the map. Each route is mapped in its own distinct colour. This allows the users to view exactly how each route differs in direction, if they decided to get off the bus early for example (Figure 2.6).

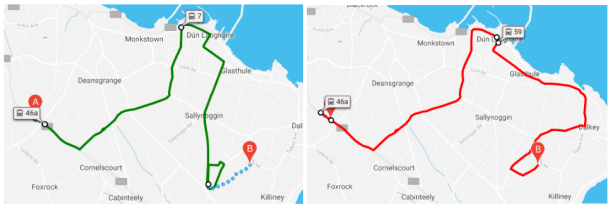

Figure 2.6: Alternate route options

As described in our modelling section in chapter 4, our application predicts journey time by predicting the time from the route's origin to the user's origin and destination. Using this method, we have been able to show waiting times to the users. This is possible as the application has been linked with Dublin Bus timetables, using General Transit Feed Specification (GTFS) data. Real-time information, or planned times of arrival could have been used to assess when a bus would arrive at a particular stop. However, the team decided that our application would predict these times. Although the project specifications outlined how the application should provide accurate estimate of travel time, the team felt waiting times were another key issue which frustrated Dublin Bus users. Therefore, the application uses the travel time predictions in place to more accurately calculate waiting times vis-a-vis the timetables provided. This will hopefully provide more accurate waiting times for the users versus that of the current system.

The application predicts whole or parts of a single route. However, it can also evaluate journeys with several bus routes used. Given the number of people travelling from outside of Dublin to work/study, commuters increasingly have to use more than one route on their regular journey. Thus, this feature is imperative for a Dublin based bus application (Figure 2.7).

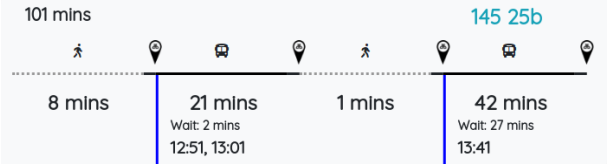

Figure 2.7: Route option information

Bus Tracker

The Bus Tracker option on the navigation bar provides users with a live and interactive experience, allowing them to visualise their bus position. The speed of the bus is based on the scheduled routes and our model's prediction times interval between each stop, making it possible to locate the bus position on the road.

The challenge of the tracker is handling the large amount of data, consisting of both the static geographic routes and real-time predictions. Timetables have been created and stored in the application's database. These timetables are based on the planned timetable from GTFS data, combined with our model's predictions. Although, the database and queries have been optimised more than once, the response still requires at least ten seconds, due to the capacity of our server. For the optimal user experience, a moving animation has been implemented to indicate to the user that their request is being processed (Figure A.1).

When the request is finished processing, a Google map of Dublin is loaded with markers indicating each bus movement. The routes are dynamically drawn from the departure stop to the destination stop. Each bus is represented by a SVG marker with its particular route number. These markers are animated to represent the vehicles trajectories and current position as estimated from the timetable (Figure 2.8).

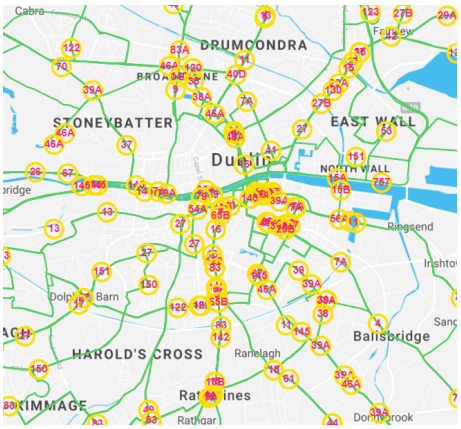

Figure 2.8: Bus Tracker map

Each marker can be clicked to provide the user with more detailed information about that particular bus route. This information is displayed at the bottom of the page, which shows all stops along the route. This stops guide also transforms from a horizontally to vertically oriented feature on mobile devices (Figure 2.9) .

Figure 2.9: Bus tracker phone view

Chapter 3: **Development Approach**

3.1 Agile Methodology

The design of the application is outlined in the previous chapter. In order to apply this functionality, the team had to organise the development of the application. This development included communicating our progress and managing our time. We employed the Agile paradigm to aid the team in this process.

Agile Software Development spawned from a selection of methodologies that were deployed in the early 1990s. These methods were a response to the heavyweight management tools that allowed for micro-managed projects. Such tactics resulted in prolonged time to market, far slower than the business cycle required [16]. The Manifesto for Agile Software Development [2] compiled these techniques into a core set of principles which defined how Agile processes should be carried out. These principles are based on delivering a working prototype in the shortest period possible, iteratively adjusting the prototype thereafter.

Having experienced the benefits of such Agile methods in the past, the team committed to rigidly applying the paradigm during this project. The manner in which these methods would be applied, was set out during the initial planning phase. The project was divided into two-week sprints. Each sprint was managed by a scrum master, the role rotating among members across the course of the project.

Stand-up Meetings

Daily stand-up meetings were imperative to keep track of the teams progress. These meetings provided us with continuous feedback in regards our progress versus our plan set out for each sprint. The stand-ups also signalled areas where team members needed assistance or advice. In these events, issues that arose were discussed amongst the team. The team would then agree upon an approach that should be taken to tackle the issue, with the most appropriate personnel assigned to carry out the approach. This tactic resulted in issues being resolved quickly.

Sprint Organisation

A sprint review and retrospective would follow each sprint, which the scrum master had the responsibility of writing and presenting to the team. The retrospective meetings highlighted the teams progress in applying Agile methods, as set out in the planning phase. This ensured that these methods were applied more effectively in the next sprint. The sprint review meetings brought attention to technical mistakes that were made recurrently. For example, the third sprint saw several avoidable GitHub mishaps. During these meetings the coordination lead also presented the burndown chart for the sprint. This gave us a visual representation of our progress through the product backlog and exposed areas where the most time was spent.

The division of work was seamless during the planning phase of sprint one. Team members suggested areas of the project they would be most interested working on, which then decided team roles for the first sprint. For the following two sprints, two members would rotate between back-end and front-end production. This allowed for team members to specialise in the area they

were working on for two consecutive sprints. Thus, smooth integration was possible when a team member switched from front to back-end, for example. The back-end specialist had all required knowledge to bring other members up to speed. This process ensured that each member became familiar with all aspects of the project, while also preparing them for the individual presentations.

For the last sprint, three members primarily worked on the deliverable product. This was mostly comprised of the efficiency of the application, running tests, fixing bugs and adding some final styling effects. The other member focused on the report and project management materials, helping the application team where required.

The team embraced an iterative approach in developing the application. Initial research was undertaken to make informed decision on the what the most appropriate approach was at each stage. Using this research, we developed a working application as quickly as possible. This allowed us to iteratively add functionality to the application. We continually created more efficient functions and investigated methods that would improve our applications design.

3.2 Project Management

3.2.1 Communication and Meetings

During the initial planning phase, the group established expected working hours to meet on campus from 10 am to 5 pm, Monday through Friday. It was agreed that occasional days working remotely would occur, once team approval was sought and granted. The team placed a high level of importance on working in direct contact each day. From previous experience, we had agreed that meeting in person held much greater benefits than communicating over virtual platforms. This allowed problems to be quickly and effectively solved and ideas easily being explained and understood. Thus, we limited working remotely to a minimum. Daily stand-up meetings were therefore conducted in person. We also documented each stand-up using the Slack application. Our team communicated outside of these hours through WhatsApp and Slack platforms. This type of communication was informal in nature, regarding issues of organisation and general enquiries if a team member was working on the application after hours.

3.2.2 Conflict Resolution

Although the team gelled well together from the outset, we were prudent when it came to conflict resolution. Concrete guidelines were set out as part of the team contract, in the event of a disagreement within the team. These guidelines were centered on the team's agreement that verbal communication would be the most effective way to solve any conflict. In the event of conflict surrounding decisions about the application, a vote would be taken. The scrum master would have the final say in the event of a split vote.

Thankfully, the team faced no real conflict throughout the course of the project. Any disagreements that did occur were based on how a particular feature should be implemented. For example, in the latter sprints, two members of the team focused on the applications appearance. On occasion, a decision could not be made between just the two members. In such cases, the rest of the team was consulted and a decision would be made according to the voting process outlined above. Although structured, this was mostly an informal process.

3.2.3 Documentation

The team placed great importance on accurate and regular documentation, throughout the project. As previously mentioned each scrum master was in charge of ensuring daily stand-up meetings were properly documented. This emphasis was justified when it came to writing the sprint reviews/retrospective as each scrum master had a clear representation of the issues that arose and how the team could learn from them. Our group documentation was stored on a shared Google Team Drive. The drive housed our team contract, presentations, screen shots, background research and any additional materials. Within this shared drive, we created a Google spreadsheet where our sprint and overall product backlogs were documented. After estimating the number of hours required for each feature on the sprint backlog, the actual hours spent would be recorded at the end of each day. With these hours, a burndown chart was created for each sprint. The summation of the predicted and actual hours for the project as a whole can be seen in Figure 3.1.

Figure 3.1: Project burndown chart

The burndown charts were invaluable throughout the sprints as the team saw how time was spent each day and how we could better utilise and predict the use of our time. These learnings allowed us to have a much greater sense of what features could be achieved, especially coming into the last two sprints. As the chart shows, the first two sprints involved much higher actual hours than predicted, showing the team's eagerness from the outset. Time was managed much more efficiently in the middle two sprints as the team became more familiar with the tasks at hand. The final two weeks then saw an increase of hours spent due to extra styling and debugging work required.

3.2.4 Version Control

The group used one GitHub repository with a master branch, and as the application progressed during the third sprint, additional branches were created for separate team members in order to prevent merge conflicts. As we met in person daily, we verbally communicated our GitHub commits in order to mitigate such conflicts. Although precautions were taken, there were times where conflicts did occur, leading to avoidable time wasted. Thankfully, no code was ever lost as we made sure to always have a backup on at least one of the local machines. The difficulties we did face served to give us further exposure to Gits functionality and usefulness in controlling such a team project.

Chapter 4: **Technical Approach**

4.1 Overview

As described in Chapter 1, our goal for this project was to create a web-based application that would accurately predict travel time for a given journey. The technical development to accomplish this was constructed across three main phases, using Agile techniques. This Chapter addresses these phases and our technical approach to each: data analytics, back-end and front-end.

4.2 Data Analytics

To understand and manage the data, the team implemented the CRISP DM methodology. This strategy compartmentalised business understanding, data acquisition and preparation, modelling, and evaluation into individual sections. These silos provided a paradigm and flow for our data cleaning and modelling process. In regards to functionality, Python, Jupyter Notebooks and data analytics based packages were utilised by the team for this task.

4.2.1 Business Understanding

The business problem was to provide dynamic bus schedule estimations. Dynamic travel time is the actual travel time that the bus would experience during a particular trip. These estimates should then display for the selected journey, via the interface. In order to provide such a bus schedule, a prediction model with great accuracy is required given the dynamic nature of this problem.

4.2.2 Data Acquisition

Dublin Bus provided historic data regarding bus journeys. This was the core data utilised. Additional data was also retrieved that was useful in building our prediction model. The data used in developing the application was as follows:

- Historic. 2016 and 2017 historic Dublin Bus data for 'leavetimes' and 'trips'.
- Weather. From three different weather stations in Dublin, compiled by Met Eireann [11].
- GTFS. Format to describe static schedule data and associated geographic information for transit networks [7]. TransitFeeds provides Dublin Bus GTFS data that updates twice a month [14]. Here we could acquire all necessary data about each of Dublin Bus stops.
- Real-time Passenger Information (RTPI). System provided by the National Transport Authority (NTA). All Dublin Buses are GPS equipped which allows an on-board software to

predict the arrival time at the next bus stop. This information is sent through a central control system, and is then displayed to users. These updates are generally transmitted at 30 second intervals [15].

4.2.3 Understanding Data

With the given project specification for our prediction model, our domain concepts initially decided were weather, dates and bus routes. Bus routes being defined as an established series of stops connected by an origin and a destination. Our target concept was arrival time. Jupyter Notebooks and Python modules were constructed to clean, combine and visualise datasets.

The historical data provided was combined into one trips file and one leavetimes file for each year. However, the files were too large to handle - trips files were 211 MB, leavetimes were 8.71 GB. Therefore, the team initially decided to focus on one route, being 145, as a case study to understand the data before employing any Data Quality Plans. We created Python scripts to split each route's data from the text files, rather than querying the database directly. This was much more efficient and flexible.

Stop data from the GTFS was combined with the weather data retrieved from Met Eireann, to pair each stop to the closest weather station through their respective coordinates. Any variance in weather would then be reflected in the data. The weather, GTFS data and historical data were all combined into one dataset in a Jupyter Notebook to further analyse the specific route. Data quality issues were identified, allowing the development of a Data Quality Plan, before training a model for each route.

4.2.4 Preparing Data

After creating the initial dataset, data reduction was required given the volume of data. Data management issues and strategies included:

- Incomplete data. The historical data only consisted of months between January and June for both 2016 and 2017. We decided to only keep June as its the only month to operate the Summer timetable.
- Unclear contents for 'note', 'tenderlot' columns. They were dropped.
- Outlier columns - 'suppressed' and 'justification'. They are officially outliers in the dataset. The rows with non-null value in these columns were dropped.
- Low Stop ID cardinality. Six stop IDs accounted for over 95% of all destination stops, suggesting that the other 5% did not complete their journey. As this is such a large chunk, all rows that did not have destinations as one of these stops, were dropped.
- Bank Holidays. Dublin Bus operates unique sets of timetables on bank holidays. Compared to a normal day, these days can be regarded as the outliers in the data. Thus, they were dropped.
- GTFS vs Historic. Both data sources were required. However, the GTFS data did not have a 'routeID' column as the historic data did. This was required for our modelling. This problem was solved by using a combination of the 'headsign', start 'stopID', end 'stopID' and 'direction' features to uniquely identify each row in the GTFS.
- Null values. Roughly 3% of data entries experienced null values, which were dropped also.

- Missing stops for a given trip. The feature progrnumber in the data is the sequence number for a particular trip. In the historical data, multiple stops were skipped for a given trip. Any trip that skipped a sequence number in their journey were dropped.

Target Feature

The selection of our target feature and training features were obtained from both the historic and real-time data. With our target concept being arrival time, the target feature was chosen as trip duration. The duration is defined as the difference between the models time prediction to the users requested origin and destination stops, from the routes origin stop. The time the bus leaves the routes origin is retrieved from the planned departure times in the GTFS data (Figure B.1).

This approach was taken as the trip duration and the number of stops had an almost linear relationship. We also assumed that it would be beneficial to users to incorporate timetables into the application. Thus, waiting times could be provided for the user, vis-a-vis the timetable. Moreover, travel time predictions from the routes origin can also be utilised to create the Bus Tracking part of the application, as seen in Chapter 2.

4.2.5 Modelling

Background research suggested the following two approaches for this dynamic travel time problem:

Linear Regression Model

Regression models predict and explain a dependent variable with a (linear) mathematical function to explain a dependent variable, formed by a set of independent variables. Unlike historical data based models, these are able to work satisfactorily under unstable traffic conditions [1]. These models have been used by many authors in bus travel time predictions. However, as a result of transit system variables generally being inter-correlated, regression model applications can be limited [8].

Neural Network Model

The Artificial Neural Network (ANN) is a machine learning model that makes a prediction of the parameters of the model through a learning process on existing data. The inspiration behind the ANN model is to emulate the intelligent data processing ability of human brains [1]. The artificial neurons, interconnected by weights, map the input-output relationship for the analysed system automatically [8]. The benefit of this model is its ability to process complex non-linear relationships and complicated data.

4.2.6 Model Development and Deployment

A Linear Regression Model was used as a baseline, as the team had experience deploying such a model in previous work. An ANN model was then created to compare findings. The ANN showed only slightly more satisfying results compared to the the linear regression, which is discussed in Chapter 5. However, given the aforementioned literature recommendations of this approach, the ANN model was preferable.

After preparation of the data and implementation of the ANN model, the model itself was deployed for every route. This was done by training the model and saving it as a joblib file. Joblib is a tool that enables the saving of the model in order to reuse it again when the application needs to. A script was deployed to create joblib files for each route. The Jupyter Notebook routesAndModels.ipyb was used initially to visualise, clean, combine and then train the ANN model for the dataset. Once this process was complete, the code was used in a Python script, train_models.py, for time and efficiency purposes. This data cleaning and modelling process is illustrated in Figure B.2.

4.3 Front and Back-end Architecture

The front-end structure is created using a combination of HTML, CSS, JS, JQuery and Bootstrap. All files are housed by the Django project structure. The left half of Figure B.3 shows what information is required when the user clicks the 'GO' button on the index.html: origin/destination; time/date.

Googles response may contain the same bus line several times, each relating to a different bus on the timetable. The JS function checks to see what bus lines are required for the route, so the HTML content is not created for the same route more than once. Instead, the application will show one bus route option, but give several bus times where appropriate.

These routes are then sent to Djangos back-end structure using AJAX, which is received by the script views.py. This script is the workhorse of our application. This is where journey time predictions for each route option are returned to the index.html. Before calling our prediction models, there are a number of steps involved in preparing Googles response. We use a preprocessing.py script to do a lot of this preparation. This scripts tasks include: mapping Googles latitude/longitude results to bus stop IDs; retrieving the weather from the given timestamp; querying the timetable database for possible journeys. All this information is then used to call the relevant joblib files for each route, which returns a prediction time for each bus journey in a route. These predictions, along with any other relevant information (mode of transport, error code, etc.), are sent back to the front-end functions. The response is analysed and formatted according to the details of each journey. A simplified version of this process is shown in Figure B.4.

4.4 Technological Justification and Alternative Solutions

4.4.1 Python, Django

Firstly, the team decided that a framework running on Python would be preferable. This was due to the teams prior experience with Python compared to other languages. Python also has a selection of easy-to-use data analytics packages, which worked very well in the creation of the application.

Django was an obvious choice for the development of this web application. Django and Flask were

two frameworks we initially considered, as they both relied on Python and had many benefits. We had all previously deployed a lightweight Flask application, which made this approach seem favourable. Further research suggested Django had more advanced functionality, which we felt was important for a project such as this. Thus, we opted to deploy a Django application. Where necessary, we took advantage of this wide range of functionalities. The SQLite database that is built in to Djangos framework was of particular use for storing timetable information. The application queries the database multiple times for each requested route. Djangos integration with this database therefore allowed for almost instant query times.

However, there were some functions Django provides which we regretfully did not take full advantage of. Django includes an integrated function called models, which provides a template to store all data related to your application. In the latter half of the project we used these tools for the Bus Tracker, however, they would have been valuable in the early stages for creating the front-end structure. Although the final product would not have been different, greater consideration from the outset would have saved time.

4.4.2 HTML, CSS, JS, Bootstrap

Combining the use of HTML, CSS, JS and Bootstrap has become common practice for our team. Thus, this was the most appropriate selection of tools to create the front-end of the application. They are also the most commonly used tools for this type of development, while being supported by all major browsers. We used the JS library, JQuery, across our application also. JQuery allows for much faster document manipulation and event handling than regular JS. JQuery was only familiar to some of the team. This meant the other members took on a steep learning curve to implement its functions. Greater implementation of such functions may have lead to a greater user experience of the application. Another library we used for the front-end component was Bootstrap. It provides elegant HTML and CSS based design templates for mobile web application development.

An alternative to these tools that we originally considered, but decided to avoid, was the use of ReactJS. React is a JS library used for building interfaces. Its functionality surpasses that of the combination mentioned above. The library also provides React Native which allows you to build IOS and Android apps. Although using React tools would have given the application greater functionality, the team felt the learning curve was too steep for the scope of a twelve week project.

4.4.3 NGINX, uWSGI

The application is running on a remote server in UCD. It achieves this through the use of the web server, NGINX. The alternative web server we initially investigated using was Apache. Like Flask, Apache had been implemented by the team previously. NGINX, however, showed greater performance qualities versus Apaches process driven approach, creating a new process for every request. Research suggested that the combination of NGINX and Django was a widely popular choice. A plethora of documentation exists online regarding their set up and instructions to solve regularly occurring issues.

The process of choosing a WSGI, to allow Django and NGINX to communicate, took a similar shape. The uWSGI implementation provided greater performance and ease of use [9] compared to its competitors, such as Guincorn. There was again a detailed selection of online documentation to implement uWSGI with Django and NGINX.

Chapter 5: **Testing and Evaluation**

5.1 Testing Strategy

The application has been created using an iterative approach to testing. Throughout the modelling process, various measures of accuracy were documented and compared. In creating the front and back-end structure of the application, all functions were iteratively written until the team felt they were most effective. For us, this meant that any errors would be handled gracefully, without the users experience being tampered. After creating each of the functions, they were tested using Python unit testing.

5.1.1 Data Analytics

The models that return prediction times were created using an iterative approach of testing and adjusting. Once the initial Linear and ANN models were made for the 145, various tests were run to assess their accuracy levels. These tests often highlighted data related errors which had to be dealt with. Once the sample 145 route provided reliable, satisfactory results, we created models for all routes. Further tests of accuracy were then run on these routes to verify if the 145 accuracies were representative of the entire population. These tests were invaluable as they drew attention to errors relating to time predictions for journeys of only a few stops.

5.1.2 Front-end

The JS functions were written incrementally, with console.log() statements at each juncture. This allowed us to inspect the format of particular data in order to create necessary functions. When the user clicks GO, a request to the Google API is made. A dictionary is returned with all information on the possible route options. As this response is in dictionary format, retrieving information was simple. Therefore, we implemented many if/else statements to check the contents of the dictionary. Different content would then be handled appropriately. This meant we did not deploy a huge amount of try/catch blocks for our JS. Every data scenario needed to be handled differently, so if/else statements were largely the most effective error handling here. The code to implement Google Maps' functions came with inbuilt error handling, which proved useful. This saved time in the applications development and ensured the user would be duly notified when Googles services were down.

5.1.3 Back-end

When Googles data is processed and the relevant HTML is created, a dictionary data structure is created and sent to the back-end script, views.py, for each of the route options. Again the dictionary's ability to quickly retrieve information is fully utilised here. The dictionary's content is assessed, with different content calling required functions. Thus, if/else statements were again the most appropriate form of error handling. Print statements were used in development, each

time data had to be manipulated. This ensured we knew the format of the data at all times. Try/except blocks were then implemented on the necessary functions in the preprocessing.py.

When all routes are evaluated, the content of the dictionary is sent to views.py, which is changed to include the model's results and timetable information. This dictionary is sent back to the HTML/JS where once more, if/else statements do most of the necessary error handling.

Djangos pytest plugin was employed to test if each of the functions in views.py and preprocessing.py worked as required. Using this plugin allowed for a clean testing structure. A test folder needed to be created, where each file with 'test_' appended to the front of its name, would be found by the plugin. Each of these files included tests with simple assertion statements, for every small part of the functions being tested. This approach ensured data was correctly formatted after each function call was made.

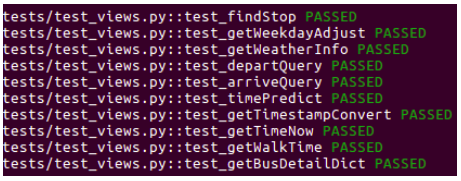

Figure 5.1: Unit-test results

5.2 Evaluation Results

5.2.1 Data Analytics

The goal of this project was to design a web-based application that provided Dublin Bus users with dynamic travel time information. Our application meets and exceeds the requirements, providing accurate journey time predictions. The accuracy of the models was measured by the R-squared and Mean Absolute Error (MAE) values. R-squared measures the closeness of the data to the fitted regression line [10]. While the MAE is calculated based on absolute difference between predicted and observed travel time given the actual travel time [5]. Table 5.1 shows the results from the aforementioned linear regression and ANN models, with the ANN approach being selected given recommended research in the area.

Final Model Iteration	R-squared	MAE
Linear Regression	0.9148	304.3713
ANN	0.9335	208.31308

Table 5.1: Model Results

These figures were taken from the results of the route 145, which was used to test and create the initial models. As previously discussed, although results were similar for both models, our research suggested the an ANN model was most suitable for this type of problem. This decision was justified by the results of the models that were created for each route. Taking a sample selection of these models for fifteen of the routes, R-squared values ranged from 0.82-0.95. Thus, confirming the legitimacy of the high R-squared value seen for route 145.

5.2.2 Additional Data

Although these results are satisfactory, additional factors may affect the accuracy of journey times which were not considered. Such factors include: irregular traffic patterns; passenger demand; road accidents; events in Dublin such as concerts or parades; roadworks. All of these factors may impact Dublin Bus travel times and some have readily available data. Such a selection of sources were not fully utilised for this application. This has resulted in a less than optimal model being produced. An area of future work for the application could be to assess the effect of such events through the historical data, providing a model for days of scheduled or unscheduled abnormal events.

5.2.3 Google Reliance

Google Maps API served as the foundation for our route mapping. This was extremely useful given our technical and time restraints. Google provides superior functionality than could ever be matched in this time period, which can implemented at relative ease. However, this also makes the application dependant on Google. If Googles server experienced problems, our application would be ineffective as a result. This also limits us in providing a unique service in route mapping, as Dublin Bus, TFI, and many existing public transportation applications utilise Google Maps.

5.2.4 Data Value

The data that has been used to create these models is also static and already outdated. This means that if bus passengers have changed their usage routines since, these models are obsolete. Furthermore, the timetable implemented as part of this application is from May 2018. Thus, once there is a timetable change, user experience may suffer. On a wider scale, the NTA recently announced plans to radically change the Dublin Bus network. This involves changing route names, directions, and timetables. This would again result in little usefulness of this application for a Dublin Bus user.

The Irish Government has also initialised a 2040 strategy to overhaul the current transport system. Besides the timetable and route name issues this would present, the increase in the use of other forms of transport would also prove difficult. The application strictly provides information for journeys that involve Dublin Bus as the form of transit. With the potential spread of services like the Luas and Dart, increasing numbers of commuters will take journeys that involve cross-service transport. Therefore, the application would need to provide prediction models for all the primary transport providers in Dublin to fully satisfy the needs of Dublin Bus customers.

Bibliography

[1] Altinkaya, M., Zontul, M. (2013). *"Urban bus arrival time prediction: A review of computational models"*. International Journal of Recent Technology and Engineering (IJRTE). 2. 164-169. Web. Retrieved 28 June 2018, available here.

[2] Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, ...Thomas, D. (2001). *Manifesto for Agile Software Development*. Web. Retrieved 28 June 2018, available here.

[3] Bootstrap. (2018). Web. Retrieved 16 June 2018, available here.

[4] Bus tha Cliath. (2017). *Bus tha Cliath Annual Report and Financial Statements 2017*. Web. Retrieved 16 June 2018, available here.

[5] Celan, M., Lep, M. (2017). *Bus arrival time prediction based on network model*. EUSPN. Web. Retrieved 18 June 2018, available here.

[6] DublinBus. (2018). *RTPI - Real Time Information - FAQs*. Web. Retrieved 14 June 2018, available here.

[7] Google Developers. (2018). *GTFS Static Overview*. Web. Retrieved 13 June 2018, available here.

[8] Fan,W., Gurmu, Z. (2015). *"Dynamic Travel Time Prediction Models for Buses Using Only GPS Data"*. International Journal of Transportation Science and Technology, 4,(4), 353-366. Web. Retrieved 17 June 2018, available here.

[9] Griffiths,K. (2012). *uWSGI vs. Guinicorn, or How to Make Python Go Faster than Node*. Web. Retrieved 17 June 2018, available here.

[10] Keller,D., Pammer, C., Scibilia, B., Steele, C., Stone, B. (2013). *Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?* Web. Retrieved 10 July 2018, available here.

[11] Met Eireann. (2018). *Available Data*. Web. Retrieved 16 June 2018, available here.

[12] O'Sullivan, K. (2017). *"Global warming: Ireland set to miss 2020 emissions targets"*. Web. Retrieved 15 June 2018, available here.

[13] Porter, J. (2013). *"Testing the Three-Click Rule"*. Web. Retrieved 16 June 2018, available here.

[14] TransitFeeds. (2018). *Dublin Bus GTFS*. Web. Retrieved 16 June 2018, available here.

[15] Transport for Ireland (TFI). (2018). *Real Time*. Web. Retrieved 15 June 2018, available here.

[16] Varhol, P. (2018). *"To agility and beyond: The history - and legacy - of Agile development"*. Web. Retrieved 17 June 2018, available here. http://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit

Appendices

Appendix A: **Chapter 2 Appendix**

Figure A.1: Loading GIF

Appendix B: **Chapter 4 Appendix**

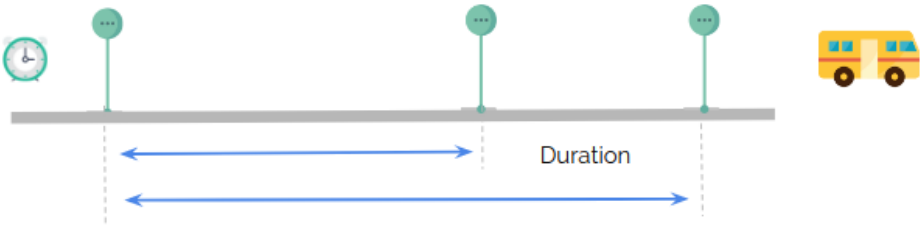

Figure B.1: Target feature

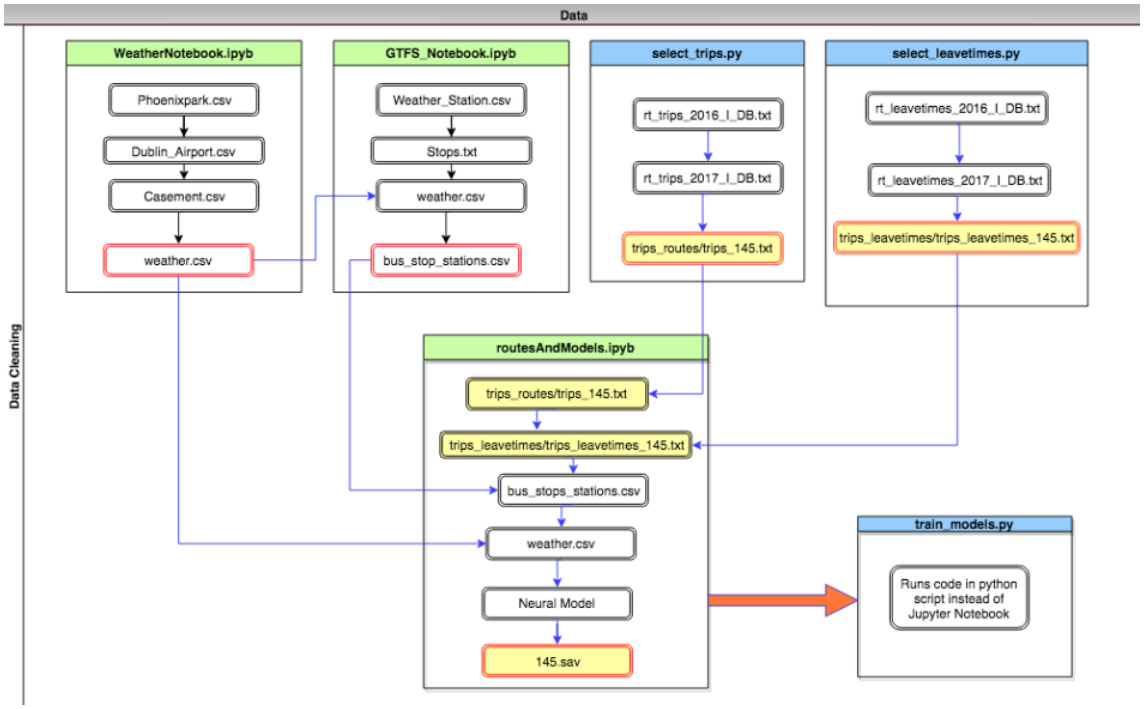

Figure B.2: Data Cleaning and Modelling with Python and Jupyter Notebooks

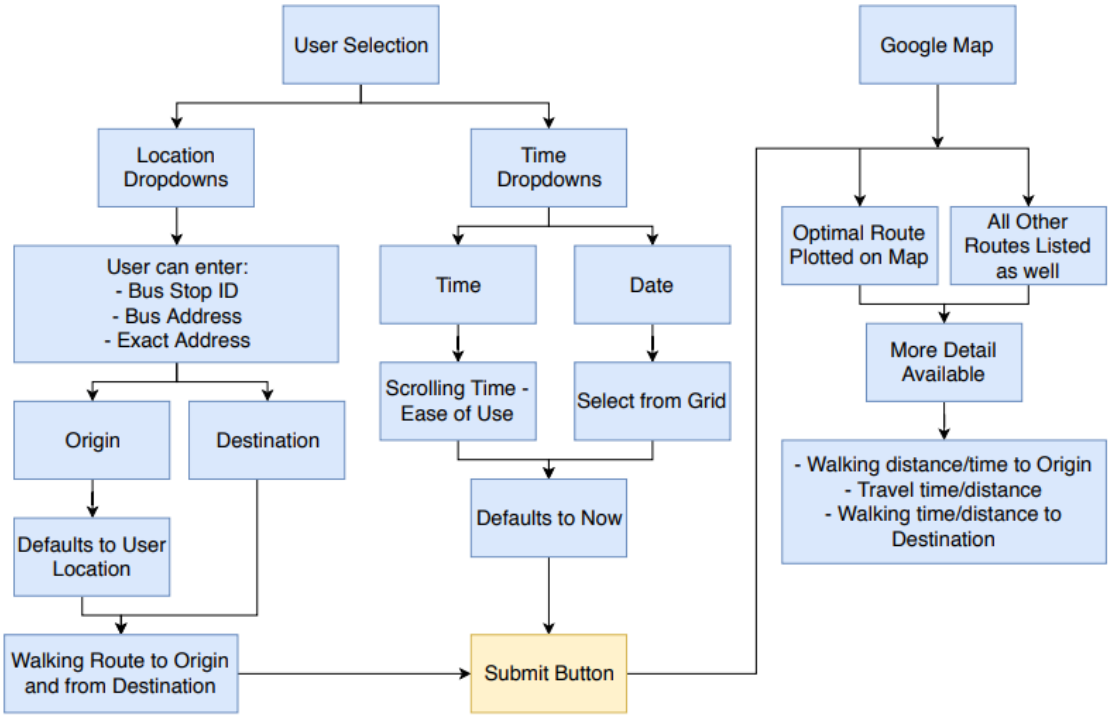

Figure B.3: Front-end functional requirements

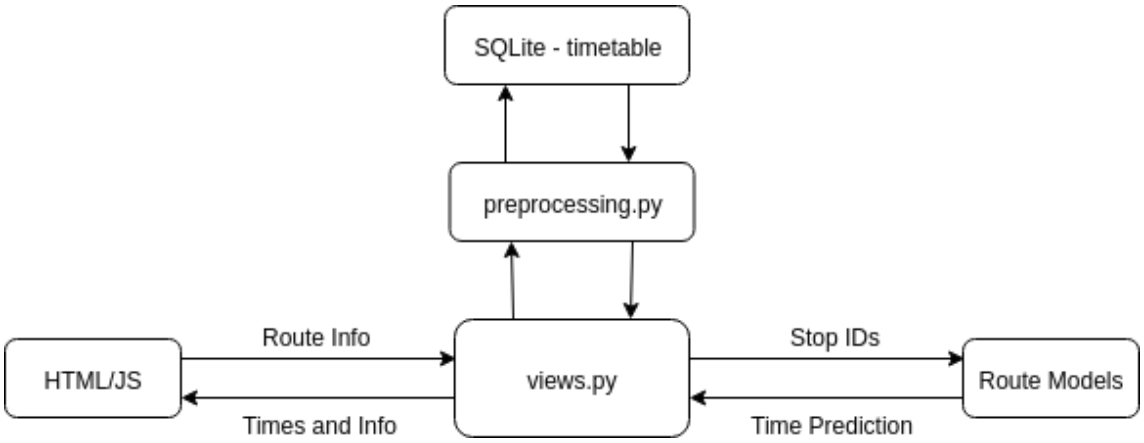

Figure B.4: Back-end work flow