Research Practicum Project Report

# Predicting Dublin Bus Journey Times

Eoghan Cullen, Shane Dunne, Finnian Rogers, Ross Kingston

A thesis submitted in part fulfilment of the degree of

**MSc. in Computer Science (Conversion)**

**Group Number:** 5

COMP 47360

UCD School of Computer Science

University College Dublin

March 26, 2023

# 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

The work presented in this report encompasses the design, development and testing of a web application for Dublin Bus oriented travel information. The application aims to present the user with more accurate predicted arrival times than what can be found in static timetables. Dublin bus historic data from 2018 facilitated the training of K-Nearest Neighbour models for full routes, in each direction of travel. These models are shown to dynamically predict journey times with a mean absolute error of 7.5 minutes.

The application aims to facilitate a customised user experience by providing user accounts which personalise the application according to user preference. These user accounts are not required to use the application, but enhance the functionality of many of the application's innovative features. The project is presented to the user in the form of a mobile-optimised web application and was developed over four three-week sprints using Scrum Methodology. The application is accessible at thedeparted.games and the GitHub repository for the project is available at github.com/Eoghan-dev/comp47360_wotb.

# Acknowledgments

We would like to thank the lectures first and foremost for their guidance and insights into our project. The mentor meetings provided us with stability and focus, which ensured we stayed on track with the project. We are grateful for the services UCD provided us during the project, which facilitated development.

We would also like to thank Laura Dunne for all help provided in the role TA. She provided insight into the more intricate parts of the project, built upon her own past experiences. This allowed us to better build our final application to a standard we were happy with.

Finally, we would like to thank all friends and family who completed our various surveys and tested the application. They found errors and provided feedback to improve on the usability of the application.

# Contents

# Chapter 1: **Introduction**

## 1.1 Objective

Transportation is one of the leading causes of greenhouse gas emissions globally, with road transportation contributing 21 percent of the EU's total emissions of carbon dioxide, the main greenhouse gas [1]. Added to the environmental concerns surrounding road transportation, traffic congestion is a major issue facing major cities around the world. One way of tackling these issues is to encourage the use of public transport to reduce the number of vehicles on city roads.

Our bus networks can be a hugely valuable resource in this context. Bus services are the most commonly used form of public transport in the European Union and are a highly cost-efficient and flexible form of public transport. They form a vital part of a city's transport infrastructure for the following reasons:

- They improve social inclusion by providing access to education, employment, and healthcare
- They are a vital link in the overall transport infrastructure of a city
- They are important contributors to tourism
- One bus has the capability to replace 30 cars on the road which can help ease congestion[2]
- Buses are the most sustainable form of motorized transport with the lowest carbon footprint per passenger[2]
- They are the safest mode of transport accounting for only 2 percent of fatalities on EU roads[2]

It is therefore vitally important that citizens are encouraged to use bus services such as Dublin Bus. One way of achieving this is by providing an app that will allow bus users to better plan their journeys to improve their user experience. The objective of this project is to build upon existing approaches to develop a richer application that will enhance the Dublin Bus experience for its users.

## 1.2 Existing approaches and their limitations

In order to achieve the stated aim of the project, an analysis of existing solutions was carried out. The key findings of the analysis are outlined below and focus on the Dublin Bus web application itself and the approach taken by the Transport for Ireland application.

The following key limitations were identified:

The use of static generic travel times provided by many bus companies. The key example in this case being the static timetable displayed on the Dublin Bus website that provides estimated travel time to its customers. This approach is clearly limited by the many dynamic factors present in a real-world scenario that affect bus travel times. These factors include traffic conditions affected by the time of day, day of the week and the month of the year. Weather conditions also have a significant impact on journey times.

The second major issue found with existing approaches centred on the ease of use or personalisation of the solution. This issue was evident on the Transport for Ireland application, which provides a much more dynamic solution than the static example cited previously. However, after analysing this, it was concluded that the application was limited by its lack of personalisation and therefore lacked the ease of use that is believed a busy modern user would benefit from.

## 1.3 Overview of the project

To address these limitations, real-time bus and weather data was analysed to create dynamic travel time estimates, which are made available to users through a web-based interface optimized for mobile devices. To further augment the usability of the application, the functionality for users to create their own personalized accounts will be provided. These accounts allow users to save their favourite routes and stops, making this information easily accessible for commuters on the go. The option for users to indicate their user payment status will be provided so that they can receive personalized fare calculation estimates, and the option to receive bus cancellation information for favourite routes will also be provided A user features survey and a final usability survey will be conducted to gather feedback as the application is developed.

The project was carried out over a 12-week period in a team consisting of 4 people, with the time-period broken into four sprints of equal length. Four distinct roles were created within the team, a code lead, a coordination lead, a maintenance lead, and a customer lead. The success of the project relied on the quality of the research, the technical application, and the strong team dynamic within the group.

## 1.4 Report Structure

The purpose of this document is to outline in detail the solution to the problem outlined above. The report will include an in-depth description of the final product including the problem addressed, the broad functionality of the product and the innovations that elevate the product above the problem specification. The document will outline the development approach taken by the project team, how the process worked and what strategies were employed. The technical approach taken to outline the architecture of the system, the technical stack utilised and the justifications behind the design and technology decision made throughout the process will be described in detail. Finally, for the group portion of the report, the description of the testing and evaluation strategy for accuracy, efficiency, and usability of the final product.

# Chapter 2: **Description of Final Product**

## 2.1 Application Overview

### 2.1.1 Directions Feature

The application has a range of features, which both address the minimum specification of the project and provide several innovative functionalities which go beyond the minimum requirements. The core feature of the application is a directions feature, which is what the user is greeted with upon loading the web page with the option to move between the Directions, Routes and Stops tabs to use each feature. This feature allows the user to enter an origin and destination location manually, or use their current geolocation as their origin location. Upon entering origin and destination locations along with a date and time, directions for each step of the journey are displayed to the user along with the estimated time and cost of each step. Finally, the user is also shown a predicted total journey time and cost prediction.

The path of the route is also shown on the map, which is handled using the Google Maps API (see figure 2.1a). Each step of the journey can either be a walking or bus step, the predicted time taken for the bus steps are generated using the predictive models trained on historic data, whereas the walking step time predictions are taken using the predictions returned by Google Maps. Journey directions and predictions can be generated for up to five days in advance.

By default, the fare prediction displayed to the user is for an adult fare with no leap card. If the user is logged in, they can change these settings to get more accurate fare predictions. Fare predictions are generated by the number of buses taken, and the distance travelled on each bus throughout the trip. This information paired with whether the user is an adult or child and has a leap card or not allows us to make a fare prediction.

The stops feature allows the user to select a route from an autocomplete text search or by choosing a favourite route saved to your account if logged in. Upon selecting a stop, it should be displayed on the map as a marker and an info window should be displayed to the user. This info window should contain the stop name and number, all routes that serve that stop and the next several buses that will arrive to the stop and how far away they are (see figure 2.1b). The time remaining for the next several buses to arrive is calculated using a combination of predictive models and real-time data from the General Transport Feed Specification (GTFS) API provided by Transport for Ireland (TFI).

The routes feature has a similar interface to the stops feature, where the user can select a route from an autocomplete text search or by choosing a favourite route saved to your account if logged in. Upon selecting a route, the user can see the path the route takes on the map along with all the stops on the route (see figure 2.1b). The user can select any stop on the map to activate the stops feature and see the information displayed in figure 2.1c.

### 2.1.2 User Accounts Feature

As mentioned previously, the application has user accounts which are not mandatory but allows the user to avail of additional features upon registration. If a user registers for an account, they then gain access to a dedicated My Account page, which consists of two tabs, favourites, and user settings. From the favourites tab, the user can add or remove favourite stops and routes for easier access in each feature and click a link to see the timetable for any favourite route (see figure 2.1d). From the user settings tab, the user can edit their fare status as adult or child and whether they have a leap card or not, which will be taken into account during fare calculation in the Directions feature. The user can also change their password from this page.

## 2.2 Project Specification

The minimum specification for this project was to create a mobile-optimised web application which, when presented with a bus route, origin and destination bus stops, a departure time, a day of the week and the current weather, should produce and display an accurate time prediction for the journey. The application achieves this specification, but in a more user-friendly way than manually entering all of these parameters when making a request to the system. This minimum specification can be interacted with through the Directions feature, which takes origin and destination locations, a date, and a time as inputs. These locations can be selected using a text search input, which automatically suggests places using Google Places API as you type.

Alternatively, the origin location be taken as the user's current geolocation. The date and time inputs are automatically set to the current date/time by default, but the user can request predictions for up to five days in advance, as that is as far in advance as weather predictions can be generated. Weather forecasts are generated using the Open Weather API, on an hourly basis for up to five days in advance, and stored in a database.

When the user enters two locations, a date and a time, a request is given to the Google Maps Directions API, which returns directions and information for each step of the journey (either walking or bus). For all bus steps, upon receiving this information from the API the application fetches weather data from the database for the appropriate time, matches the origin and destination locations to their nearest stops and gets the route number for the suggested route and feeds this information to the model, giving back an estimated time for that step.

Depending on the locations entered by the user, more than one bus may be necessary for the journey, in which case this step is repeated as many times as necessary. For walking steps, the journey time is taken automatically from the prediction generated by the API response by Google. Google's predictions for bus routes were used in lieu of the custom classifier predictions (described in 4), when it was unable to make predictions. An example of such a failure is when the bus routes were missing from the historic data used for training the models.

After this information for each step is gathered, directions, estimated time taken and a predicted fare for each step of the journey are displayed to the user, along with a total time and fare prediction for the entire journey (see figure 2.1a).

## 2.3 Innovations

### 2.3.1 Directions Feature

As mentioned in the product overview, there are a number of innovations that go beyond the minimum project specification in the application. However, before looking at these innovative features, it is worth noting that the Directions feature which encompasses the minimum specification also has a number of additional features built into it.

Firstly, it allows for a user to enter any two locations (or geolocation for origin location) and get predictions and directions based on this, rather than having to explicitly enter a start and end stop and route number. The minimum specification does not require directions either, which is an innovation built into this feature with the Google Maps API.

The fare predictor is also an innovative feature which is always displayed along with the time prediction. By default, this generates a prediction for an adult with no leap card, but the fare status is changeable from adult to child along with whether or not you own a leap card. The predictor is an estimator and is not 100% accurate due to the way Dublin Bus have set up their pricing tiers. They have separate prices for adults and children, with different prices within each depending on if they have a leap card or not. Children also have a different set price if they are travelling during hours they have defined as "school hours".

All of these factors were relatively straightforward to account for in this feature, but what complicated things was that the price per journey changes depending on the amount of "stages" the bus has passed through. These stages appear to have no clear definition, not seeming to relate to a number of stops, but instead to be a predetermined distance in metres. As a definition for what a "stage" is according to Dublin Bus could not be found, it was decided to take a stage to be a bus stop, as this appeared to be relatively accurate from the research conducted. In short, for any given bus journey, the number of stages was directly mapped to the number of bus stops passed. Plans are in place to continue to do research on what consists of a "stage" but for the meantime it was felt that this is a reasonable compromise which could be easily updated in future work.

### 2.3.2 Routes Feature

The Routes feature is another one of the main innovations, which can be found on the main page of the application. The Directions, Routes and Stops features can all be accessed using tabs at the bottom of the screen from the home page. As mentioned previously, routes can be selected either through an autocomplete search field or by selecting a favourited route if logged in. Upon selection, the routes feature displays the path any bus route takes on the map, along with all the stops on the route as markers (see figure 2.1c). Any of the markers can be selected to see the same information for that stop that can be found using the stops feature. The line on the map for the route path is generated using the Google Maps Directions API.

This is accomplished by obtaining the start and end stops for the selected route and checking the timetable for that route to see when it should next leave the start stop of the route from the static JSON files. Once this information is obtained, a request can be sent to the Google Maps Directions API from between the co-ordinates of the start and end stop of the route and with a desired departure time of whenever the bus is next scheduled to leave. This worked in all tests except for a scenario where a route is requested at a time of day when there are no more buses running on that route, as the application is then unable to set the time of the next departing bus from the timetable as the desired departure time as there are no more departing buses for that day. In this case, all the markers for the route are still shown, but the line generated from the API is not displayed on the map. It is planned to work on this in future work, but for the time being it seems to work well in most scenarios, except for a few select routes that only run early in the morning or stop running particularly early.

### 2.3.3 Stops Feature

The Stops feature can be interacted with in the same way as the Routes feature, but instead of displaying the route and all its stops on the map, it displays that stop and that stop only on the map along with an info window for that stop. The info window displays the stop name and number along with the routes that serve that stop, but the most innovative component of the stops feature is that it also displays the next several buses that will arrive to that stop and how far away they currently are. A view collects a list of all upcoming buses in the next hour from the static GTFS information and feeds this list to the predictive algorithm described in 4. The generated results are then ordered, and the view returns the next four buses past the current time. This feature can be considered an in-app equivalent to the electronic signs at many bus stops, which display the next arriving buses and how far away they are.

### 2.3.4 User Accounts Feature

Users have the option to create an account to avail of additional features in the application. The main features they can avail of are the ability to save favourite routes and stops for easier access from their respective features, and the ability to update fare and leap card status for more accurate fare predictions. All accounts are handled using an extended version of the built-in User model in Django and stored securely in the database. This allows us to use many of Django's built-in tools for authentication and account management, which facilitates secure and easy administration for this component of the application with full password hashing.

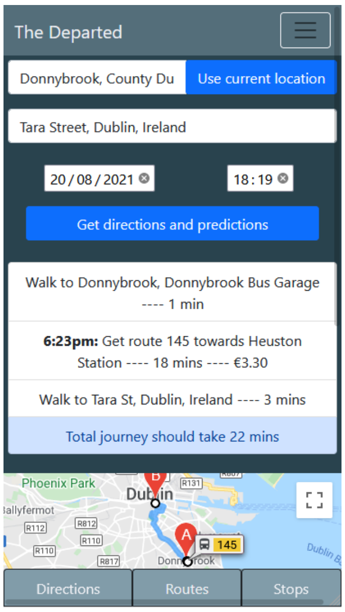
When registering for an account, users are asked for their first and last names, their email address and their password. Fare status (adult or child), leap card (yes or no) and favourite routes and favourite stops are additional fields which can be modified from the My Account page after registering. Fare status is set to adult and leap card set to no by default, while the favourite routes and stops are empty by default. Favourite routes and stops are stored in the database as comma separated strings, which can then be parsed as an array when handling this data in either Python or JavaScript. The application also contains a fully functioning forgot your password feature, where you can have a password-reset link sent to the email you signed up with. Alternatively, if you know your password and just want to change it, you can do so from within the My Account page in the application.

### 2.3.5 Cancelled Routes Feature

The application has a cancelled routes feature where currently cancelled bus routes can be found. This feature can be accessed from the routes tab on the home page of the application, whereupon clicking a button, an overlay screen with a table containing all cancelled routes is displayed to the user. This is accomplished by the server querying the GTFS Real-time API every five minutes and storing the results to the database. When the homepage of the application is opened, the database is queried to get all routes that are currently cancelled, and this is then parsed on the front-end and formatted into a table, so the user can view it once they click the button. The table displays the route name and head sign, along with the previously planned departure time from the first stop on the route.

### 2.3.6 Additional Innovations

In addition to the aforementioned innovations, there are several smaller innovations in the application. The first of which is a timetable feature where you can see a full timetable for any particular route listing the departure times for each stop using static Dublin Bus timetables which were generated using the static GTFS data. There is also a dedicated Twitter page where live updates can be viewed from the Dublin Bus Twitter account.



Donnybrook, County Du

Tara Street, Dublin, Ireland

20 / 08 / 2021 — 18 : 19

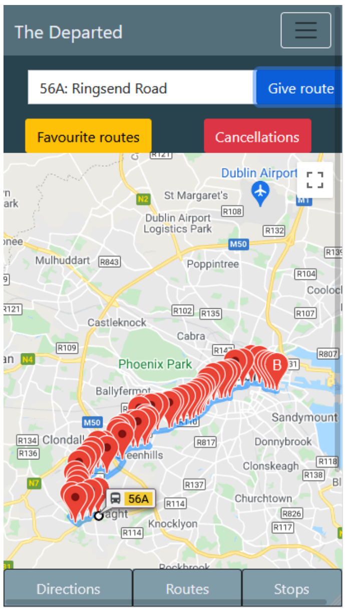
Walk to Donnybrook, Donnybrook Bus Garage ---- 1 min

**6:23pm:** Get route 145 towards Heuston Station ---- 18 mins ---- €3.30

Walk to Tara St, Dublin, Ireland ---- 3 mins

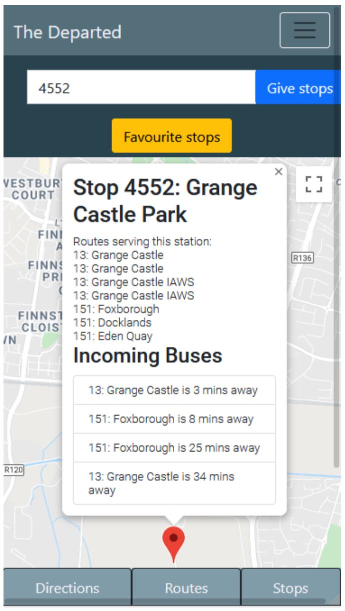
Total journey should take 22 mins

(a) Directions Feature



56A: Ringsend Road

(b) Routes Feature



4552

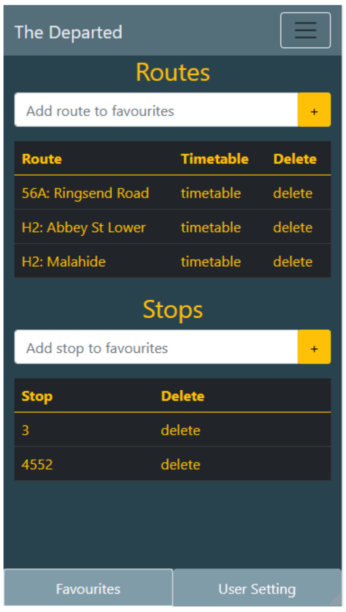
**Stop 4552: Grange Castle Park**

Routes serving this station:
13: Grange Castle
13: Grange Castle
13: Grange Castle IAWS
13: Grange Castle IAWS
151: Foxborough
151: Docklands
151: Eden Quay

**Incoming Buses**

13: Grange Castle is 3 mins away
151: Foxborough is 8 mins away
151: Foxborough is 25 mins away
13: Grange Castle is 34 mins away

(c) Stops Feature



Routes

Add route to favourites +

| Route | Timetable | Delete |
|---|---|---|
| 56A: Ringsend Road | timetable | delete |
| H2: Abbey St Lower | timetable | delete |
| H2: Malahide | timetable | delete |

Stops

Add stop to favourites +

| Stop | Delete |
|---|---|
| 3 | delete |
| 4552 | delete |

(d) Account favourites

Figure 2.1: Application pages

# Chapter 3: **Development Approach**

This section of the document will cover in detail the work processes the group employed to deliver a final product. An outline of the development strategy used to manage the process and the operational tools employed to aid in the implementation of this strategy.

## 3.1 Development Strategy

The development strategy the group agreed on was the agile scrum methodology. All four members of the team had used this methodology before and were satisfied that it worked well to manage a software project to a successful completion. As mentioned in the introduction, the process was broken down into four sprints of equal length (3 weeks). Each sprint was top and tailed by pre- and post-sprint meetings. Each team member assumed the role of scrum master for one sprint, ensuring that the scrum framework was adhered to throughout.

## 3.2 Sprints

### 3.2.1 Sprint 1

During the first sprint, the group concerned itself with establishing a strong foundation for the rest of the project. This meant establishing good project management documentation which was done using Confluence, Jira, and Google Drive. An architecture for the project was agreed, and the technologies outlined in chapter 4 were chosen. Each member of the group got set up to use the relevant technologies. Communication and conflict resolution protocols were agreed. All relevant data was collected for the predictive model to be created.

### 3.2.2 Sprint 2

In the second sprint the project concentrated on establishing a solid front and backend communication, this was done successfully. A basic model was created with a pickle file to be used when building out the remainder of the application. Views were set up on Django that served predictions to the front-end. At this stage of the project, the group generated ideas for features and conducted a user features survey to ascertain feedback on the ideas generated.

### 3.2.3 Sprint 3

Sprint 3 proved to be a very productive period for the team. The frontend UI was brought close to completion with only some minor styling issue to complete. Fully functioning predictions were served to users at the frontend. During this sprint, the team carried out a number of long form meetings to intensify progress. All four members of the team participated in these meetings, and they proved very successful in progressing the project.

### 3.2.4 Sprint 4

Work in sprint 4 focused on testing the application's functionality and fixing bugs resulting from testing. A usability survey was carried out to test the usability of the application, and final polishing of the user interface was completed.

## 3.3 Communication Method

Robust communication protocols were established very early in the project. Daily stand-up meetings took place at 10am every weekday. A discord server was set up to conduct the group's regular meetings, the server was also used for communication outside formal meetings. Stand-up meetings were attended by all team members each day. These meetings ensured the team successfully tracked the progress of the project daily, while also fostering a very strong team ethic within the group. Code reviews were carried out every Friday to ensure all team members were kept a breast of the workings of the various components of the project. During sprint 1 a larger meeting was scheduled for Wednesday evenings however as the weeks progressed, these meetings were found to be redundant, and they were replaced with meetings called for specific issues as they arose. Conflict resolution was handled by discussing the pros and cons of an issue and coming to a consensus based decision.

## 3.4 Organizational Tools

Organizational tools were chosen to aid with communication as well as the management of the project. The tools used were Google Drive, GitHub, Confluence and Jira. Google Drive was used to store documents, so all team members could access and them as necessary. Confluence was used to record meetings including daily stand-ups, code reviews and any other meetings. Jira was used in tandem with confluence to plan and track sprints and to distribute task across the team. GitHub was employed to handle version control using a development branch, branches for each feature in progress and a final main branch.

# Chapter 4: **Technical Approach**

## 4.1 Overview of architecture

For the project, the team used a range of technologies to aid in both the design and the functionality of our application. A brief overview can be seen in Figure 4.1, which shows the interactivity of these various technologies. At the centre of the application is Django, this joins and operates all the various technologies in unison. The web application the team have designed is aimed for mobile users, however, with the grid box methodology in bootstrap the app should also be clean and accessible for a user through a tablet or desktop.

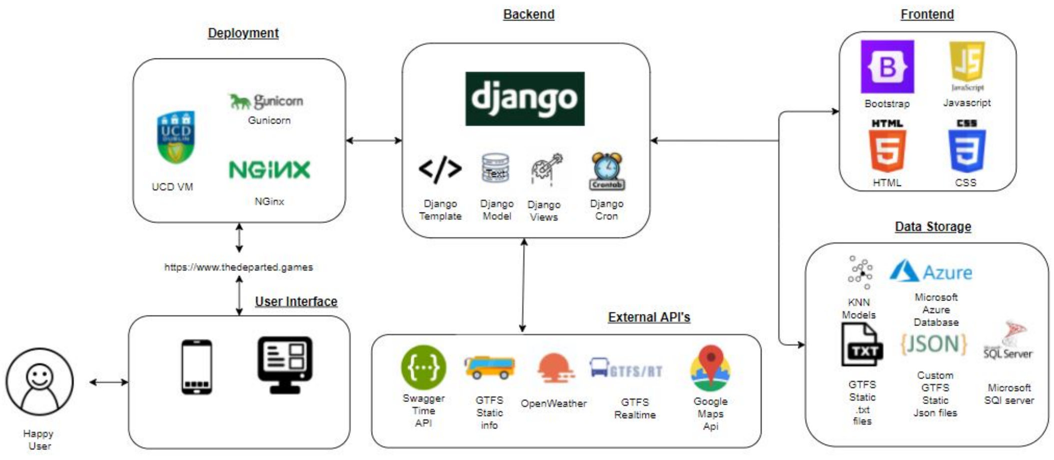


Figure 4.1: Application Stack

## 4.2 Backend

Django was used as the backend framework for this project. Django follows a models/templates/views architectural format for delivering a working application to the user. Django comes with a deep source of documentation pages available with it thanks to the application first being created in 2005 and being widespread in industry (Instagram, YouTube, NASA). It allowed for those working on the frontend to quickly come to speed with the base features available on the framework, as well as quickly troubleshoot many common problems that occurred over the course of the project.

The "model-view-template" is a software design template that Django uses. Models handle the database and provides customisation with how someone accesses the data and subsequently uses it. Template section deals with how information is presented to the user in the form of the UI. Views are used to complete tasks on the backend and collects data from the models; along with rendering templates with passed information. This provides a basic structure needed to create a web-based application.

Within the models, data was scraped from various online API's. The data was then either; migrated to the application's relational database on Microsoft Azure or stored as JSON files locally. Data stored on the models could then be readily accessed through the Django ORM that can interact with a multitude of different relational databases. The ability to interact with a multitude of different relational databases allows the application to be highly adjustable and to quickly adapt to a new database. This was important within our project due to the use of Microsoft SQL Server, which is not a natively supported database with Django. This ensured that if any issues arose later, the database that Django was interacting with could be easily changed to one with a natively supported dialect of SQL.

Views provided a way to interact with this data saved within the models or locally and to access the required information. This could then be used to feed the data to the user. It also allowed us to account for users actions and to return the desired information to the customer from our templates. Views provided us with the ability to access our stored data whether this was from pickle files, models or JSON files and then customise the data to allow it to be correct rendered with the template.

## 4.3 API's

The main source of the Data we used for current information surrounding Dublin bus came from the General transit feed Specification or GTFS. GTFS allowed the specification of which transit mode was desired to call information for, for this project we contained collected data to on buses from Dublin bus. The information was received as text files on download. These files were converted to JSON files to facilitate simple integration with the frontend, along with an uncomplicated upload to the database on necessity. The information contained within provided an accurate source to find all bus stops and the routes served by each stop. It provided a way to find the official departure and arrival times for not only the start and end points but also for each stop along the route and save these in custom JSON files for quick access. This was essential for interaction with the predictive model, which required all this information to form a prediction of a route at any time. These JSON files can be scraped whenever major changes occur to the transportation route, but does not have to occur often as the current information is valid until 16/10/2021 per the calendar.txt file which gives the dates for bus trips and how long they are valid until.

For real time information relating to Dublin Bus, data was scraped from the GTFS-R API. This API did not contain an exhaustive list of buses. Rather, it contained information on cancelled buses or known delays on a route, instead of providing information on every bus currently running at a given time. This information contained less information that could be used reliably, but provided the application with a way to supplement the given information by informing the user if a bus was cancelled along a route.

OpenWeather API was used for scraping current, Forecast, and historical weather. It only gained information for one location across Dublin. This was due to the historical data being over a year-old, meaning that if more than one co-ordinates was wanted for accurate weather at each stop, then a fee of 10 dollars would need to be paid per location accessed. This meant that the information paired to work with the historical data would relate only to the general weather data from a single location in Dublin. As this is what the predictive model was based on, it was decided to use the same single location for the current and forecasted weather. Data from this location would then be fed into the predictive model to return an estimate.

The Google Maps API was used for the display of markers and routes to the user on the home page. This provided us with important information such as transit routes, along with information relating to the next bus along a route from a particular stop. This was used heavily when a user called the directions search feature, as this could direct a user which bus stop to walk to, and from there it would then feed the information to a view that would access our data to get the estimated journey time of this bus if possible. It also provided us with a fail-safe that we can simply default to google maps if for some unforeseen reason the view fails to return a value.

An API called swagger time API was also implemented late into the project. This was to deal with the one hour thirty-minute time difference in the UCD server compared to real-time. It was chosen over a simple time delta as it ensured that no matter the server the application is running on, the app will always be correct for parsing information to models for Dublin.

## 4.4 DATA

The Database used for this project was provided by Microsoft Azure, using Microsoft SQL Server as the relational database language. This was chosen as it was provided cheaply with the student version of Microsoft Azure and provided the most functionality at an affordable price to ensure that the credits given would not go over the limit throughout the project. This came with the issue of not being natively supported on Django as a language supported with ORM. This was overcome by using an earlier version of Django (3.0) that came with the ability to install mssql-django that allowed interaction with this database. This at an early stage still needed to be tested to ensure no major issues arose throughout the project but thanks to ORM provided with Django along with the model system it allowed us to use this database with the knowledge that if issues arose, we could adjust our chosen database to a natively supported one later in the project.

JSON files were also used for a lot of our static information, this facilitated simple management of the data needed for creation of bus stops across Dublin and timetables for given bus stops. It also provided a simple way of adding data to the database via Django fixtures(automated Django feature to allow certain information to be instantly transferred to the database) or custom models. From here we could easily see issues with the files scraped from the API from the GTFS API such as times over a 24-hour period and could easily provide fixes and catches, so these could be amended within the JSON files. These provided a convenient way to access this information, which could be read on load of a page, and which was needed for fluid interaction on certain pages, as can be seen with stop markers on the homepage or for various stop times in the timetable page or the loading of the data lists for routes and stops.

In total, the application contained 6 JSON files pertaining to general information about bus information, along with a JSON file representing bus times for each individual route. Meanwhile, the application had 2 consistently updating tables on the database: real time bus info, and weather forecast for the next 5 days in 3 hour intervals of accessible information. It also had a table that stayed static, which was the historical weather data that was paired with the historical bus information in 2018 to generate predictive models.

## 4.5 Predictive Modelling

36% of American smartphone users abandon a mobile transaction if it is too slow [3]. This concept of long wait times not being favourable is not shocking and was echoed during the user feature survey. It was very important to the group that the models make predictions fast but also maintain an acceptable degree of accuracy. Finally, due to the storage plan, deployed space on the hosting machine was limited and so the models were also required to be compact. In order to satisfy these requirements, the K-Nearest Neighbour (KNN) was used to classify the historical data. KNN was the model used in the final implementation, although many others were evaluated during the data preparation phase. The KNN method is a simple approach and so the total model memory usage was minimal, once compressed, amassing to 32 MB. The principle of KNN finds the closest k features to the target feature and classifies new cases based on the closest target features from the training data [4].

The structure of the preparation phase was centred around the model approach used. Again with the memory and efficiency constraints in mind, a full route model approach was chosen. With 130 bus routes, this would amount to 260 models due to the need to model in each direction as the traffic flow varies on either side of the road and the routes may differ in opposite directions.This number of models is far less than the  6000 needed for stop-2-stop. Due to the historic data being from 2018, without spending money there was no option to get a variety of locations around Dublin, therefore the weather is only taken from one location in central Dublin. This may not be a factor for most routes, however some busses come from the greater Dublin area where the weather may be different. The historic data was stored in the high performance server provided by UCD and was converted to a SQLite format to both reduce the load on the server and allow the data to be queried using simple SQL commands.

To build a full route model, the appropriate data had to be queried from the historical bus data. For example, for the route '56A', this route is queried using its ID and stored in a pandas' data frame. The weather for each trip is then added to the data frame based on the departure time and day of service. The day of service is then split to two features, day (represented numerically from 0-6) and month (represented numerically from 1-12). The final data frame used in modelling contains these features as well as the temperature, weatherId (which relates to the weather experienced at that time e.g. rain), and the actual/planned departure and arrival times.

After some cross validation of the k number in the classifier, it was set to 5. This provided an acceptable mean squared error when compared to k=10 (92 minutes$^2$ vs 140 minutes$^2$). One thing to note that while this mean squared error was improved at a low k number (71 minutes$^2$), this is due to oversampling, which could severely negatively affect future predictions [5]. Through a combination of python libraries, pickle and gzip, all classifiers were serialized and compressed to reduce their size as much as possible. Finally, a method made in views python file could be made to seamlessly integrate the predictions to the site.

## 4.6 Frontend

Ultimately, the user interacts with the application through the HTML pages outlined in Django templates. The user interface is determined by these various pages, with a large amount of functionality occurring through various JavaScript functions. These took user inputs and interacted with various views in Django to return the user requested information to be displayed on various pages through parsing through the returned JSON objects sent via views. This meant that the information a user needs to download locally was kept to a minimum to ensure that the user can access the application with minimum internet access as might occur with the app as they will likely access it on the go where their connection may not be as strong as at home.

Most of the CSS was handled through bootstrap, the use of bootstrap allowed us to optimise the page to ensure that it was consistent design for not only mobile users as per the project specification but also for desktop users. This was achieved with bootstraps grid system that allows us to specify the number of columns we wish a piece of HTML code to occupy on a page via rows and columns for specific device size using predefined grid classes. This allowed us as a group to use the space most optimally for each device. Initially we used bootstrap studio to build the pages which provided us with a great base to build functions on the page, but we quickly found this cumbersome and restricting as we tried to settle on the final design of the page and started taking a much more hands-on approach with applying the bootstrap.

The web application contained 6 HTML pages with 5 of these relating to unique pages:

- Index html – Routes, directions, stops, map, Favourites, prediction displays and journey time
- MyAccount – favourite routes, fare status, login for additional features, register
- Timetable list - Full list of timetables available, with search feature to find desired route
- Timetables – Times for each departure time at every bus stop along route (based on static data)
- Twitter – Feed of Dublin bus twitter page with relevant info contained from there
- AboutUs – Page about us and the roles we took on the team + link to GitHub

## 4.7 Deployment

When the application began to take shape, it had to be deployed. Gunicorn and Nginx are used to serve and distribute the Django application. Gunicorn is a Web Server Gateway Interface (WSGI), which provides the instructions for how a web server can communicate with the application. Gunicorn strongly suggests using a proxy server to handle load balancing and serve static files and allow SSL handling, these last two are crucial to the functionality of the application. Django applications have a WSGI file which Gunicorn can bind directly to. Binding is the process in which a local address is associated with a socket. This project uses Unix domain socket, which is an interprocess communication mechanism that allows the bidirectional data exchange required to serve the static files [6]. Nginx is the medium through which the users of the application will communicate and is at the edge of the Internal network of the Linux server provided by UCD. In order to use the geolocation feature, the application must be using the https protocol and so a Secure Socket Layer (SSL) certificate is required. The domain name and corresponding SSL certificate was received from names.com. The Nginx configuration file then had to be designed as to redirect all traffic to the domain name, including converting http requests to the https protocol. After this configuration was completed, the application was then accessible from the domain name, thedeparted.games. Deploying the app allowed for user tests and load testing.

# Chapter 5: **Testing and Evaluation**

From the beginning of development, testing and evaluation were recognised as integral parts of deploying a successful application. Firstly, a distinction should be made between testing and debugging. Debugging homes in on a specific aspect of code that is not functioning as intended and attempting to find the root cause and possibly employ preventative measures for future cases. In the case of this project, the root cause of many bugs resided in the python and JavaScript scripts, as they controlled the greatest number of moving parts of the application. Debugging relies on defects in the code which have been found, whereas testing is the process in which a system is subject to operations in order to find these bugs [7].

There are a multitude of different testing options available in order to derive the most meaningful aspects of the software component tested. Different stages of the application require more intense testing, for example if the application is to be realised to a large user base, appropriate load testing must take place. In this project, it was important that the tests were not intrusive, due to a short life cycle of application development. A mixture of manual and automatic testing was used during evaluation, each providing useful insights into the script function. This testing mainly fell under the categories of functional testing and non-functional testing. The former tests that the code is producing expected outputs, and the later testing the behaviour of the application under strain [7].

As mentioned in chapter 4, code accessibility was essential to code together and crucially opened up the team to collaborative testing. The peer review meetings held by the team highlighted issues that needed to be further tested and provided the team with a recommended approach to the topic. The testing and evaluation can be broadly broken down into two sections, Web application and predictive model testing. As described previously the web framework used in development was Django, this framework facilitated a variety of testing. The performance of the web application could then be testing using developer tools and load testers. Finally, the predictive model testing and evaluation was conducted using the sklearn metric library.

## 5.1 Application testing

Django received http POST requests from the front end of the application via views. These views would receive information from the URL based on the action in the application. Django has libraries which simply test this function. By importing the aptly named 'SimpleTestCase' to a test python file, a series of test cases can be made for the application. A Class was made which inherits the SimpleTestCase and based on the http response received from a known query the Django requests which govern the flow of the framework. The SimpleTestCase generates a coverage report based on the amount of responses that were expected. The current coverage of the application is 44%. This number is very low and below our standards however, the percentage is being drastically effected by the scraper files (2) which get current info and reformat into customised JSON files to suit the application's needs. Without these files the coverage increases to 75%, this is a much more acceptable figure. This test case should have been implemented sooner in development, to make retrospective testing more manageable. As developers wrote new functions appropriate test cases should have been made alongside them, this would have greatly increased efficiency in the later stages of development.

Like a newly released car, most people just want to know how fast it is. Like this newly released car, not only was it important for the application to be fast, but to be reliable too. A Google developer tool called Lighthouse was used to conduct a performance analysis of the site in production. These performance tests ensure that the application meets the teams' standards for efficiency. The categories of this test can be described as service-orientated, and efficiency-orientated. Where service would judge the availability of features and their response time and efficiency would judge the capability of an application to make use of its canvas [8].

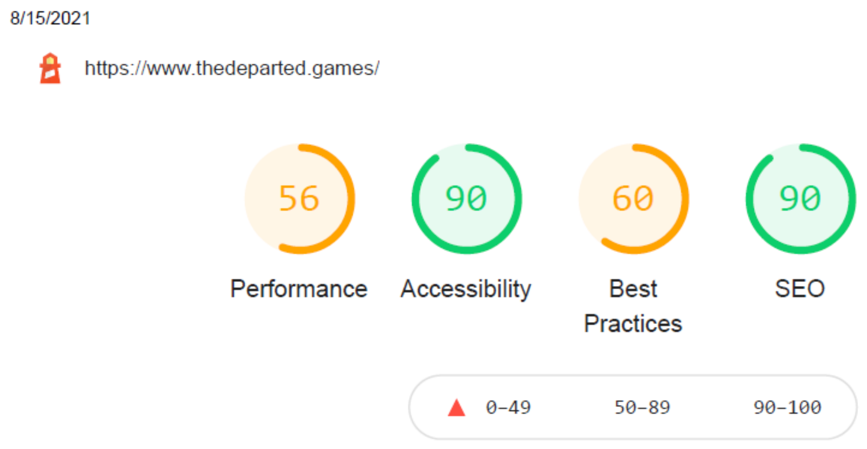


Figure 5.1: Lighthouse performance test results

As seen above in figure 5.1, the application performs very well in accessibility and search engine optimisation (SEO). The accessibility category can broadly be improved by going back through the code and adding labels to everything for screen reading, and improving the colour contrast to make elements pop. These improvements were noted and would have to be tackled before a public release. Again, the SEO can be improved by like additions as in a meta description to the HTML. Performance and best practices suffer in score. There is a large amount of style sheets being transferred onto the site, which blocks render times for more important features, as most of the CSS may not be used. The feature that is the most intensive for the application is the map. The large element size increases memory usage and hence load times, since this is a core feature this load time was deemed necessary by the team until further work can be done to improve load times. Lighthouse quotes possible savings of 1 second when those two issues can be addressed. However, the third-party blocking time from loading the map needs more work into fixing without sacrificing size. Finally, the best practices low score is less troublesome, with the application losing marks on geolocation request upon load and JavaScript libraries that are deemed untrustworthy. In summary, the score across the board will improve once the unnecessary files are removed from the site, which may take some time to weed out. Lighthouse was mainly used towards the end of the project, but if this tool was used earlier, there would be less work in cleaning the code at the end of production.

Throughout the peer reviews and live development of the JavaScript elements to the project, manual testing was employed to ensure the correct information was being passed through the site. This testing was most commonly a product of debugging in development. It is difficult to put a figure on performance of this type of testing due to the predominant factor being the person running the tests [9]. Using console logs through JavaScript was effective for small bugs, but were also used for larger problems and came at a high cost, specifically when attempting to join the front and back-end work for the first time.

Once deployed, the application was opened up to another type of testing. Load testing. This testing was mainly to see how the application and software used to deploy it could handle the strain of consistent usage. Using a stress testing application called "Webload" an investigation into the strain caused by multiple calls to the application was determined. Webload requires a script to be recorded before running this script end on end for a specified time. Webload also specifies an amount of hosts to use during this time to simulate users. The script is simply recorded by operating the application, and the software records the calls made and adds them to the script.

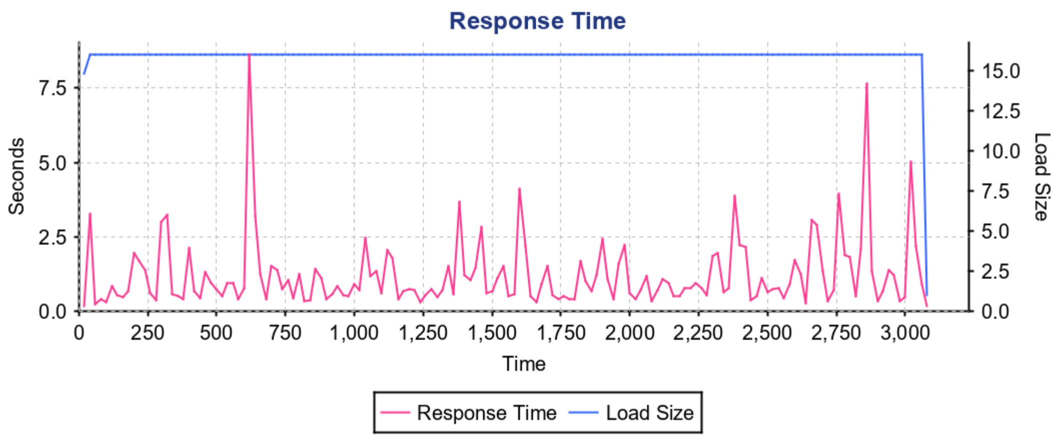


Figure 5.2: Webload: The response time of the application to repetitive calls over the course of an hour

Figure 5.2 shows the response times recorded over an hour-long testing with a 16 user load. The script in this case went through all the main features of the application including directions/predictions, stop/route search and user login and favourite saving. The large calls normally take around 2 seconds, which is shown in the trend. There are also occasional spikes where the response time exceeds 7 seconds. After some investigation this seems to be a database drawback as the data transfer units were set low due to cost. 16 calls to the database at the same time was causing the usage percentage to increase above 80%. In the future of the application, this data transfer rate should be increased to facilitate a higher number of calls.

## 5.2 Predictive Model - KNN

As mentioned in chapter 4, KNN was chosen for the predictive model due to its small file size and therefore greater speed of prediction. The models make predictions in less than half a second. This is an enormous advantage, and hence why they were chosen. The only question is their accuracy.

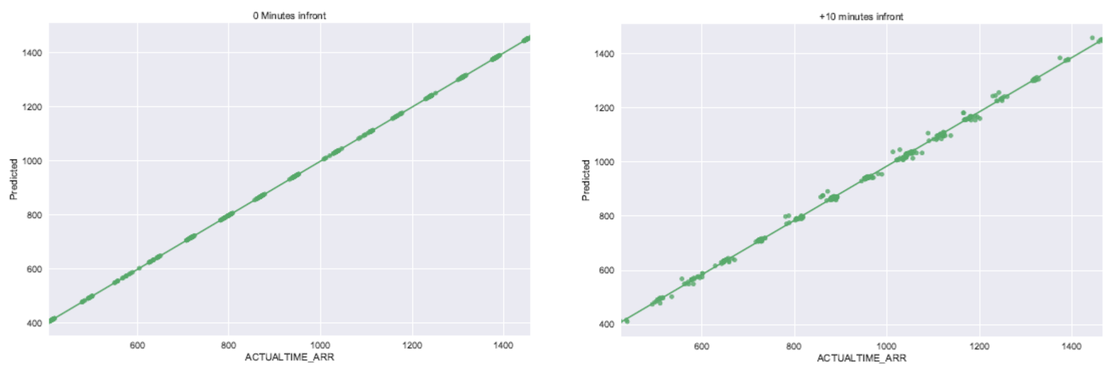
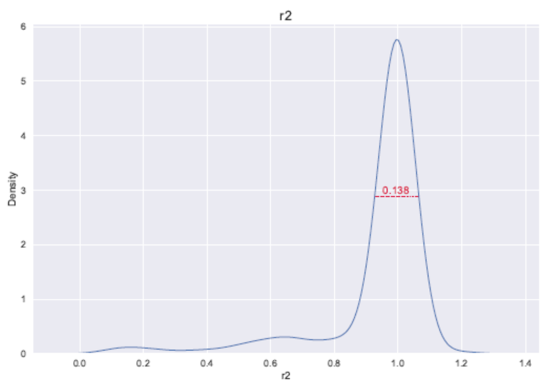
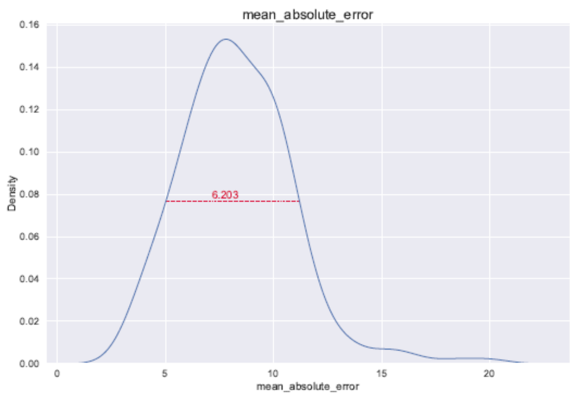


Figure 5.3: Route 56A: Actual vs Predicted arrival times

When testing the output of the KNN model for route 56A, the delay times can be summed from the actual and predicted values to give delay times. Grouping these delay times 5 minutes apart and plotting them as seen in figure 5.3 shows an expected trend. As the delays increase, the spread between the points grows. This illustrates the drop in accuracy of the model when subject to unforeseen delays.



(a) Coefficient of Determination



(b) Mean Absolute Error

Figure 5.4: Cross Validation Metrics

The kernel density estimate, shown in figure 5.4, displays the cross-validation scores for the coefficient of determination (R2) and the mean absolute error (MAE) for every model. While not the most telling of metrics, R2 can quickly display how the predictions deviate from the line of regression [10]. Figure 5.4a represents a large portion of the models show some degree of linearity, which confirms the scatter plot findings of such a tight grouping in figure 5.3. The MAE, a common metric for evaluation and so is much more important to the results [11]. Figure 5.4b shows a large density of models with a MAE around 7.5 minutes. The full width at half max (shown in red) for this plot is relatively large at 6.203 minutes.

The size and speed of these models is exceptional and very favourable for the application in providing users with fast prediction times. The speed is also important for the next buses feature, which calls on the models. The models' mean absolute error could be reduced, but more importantly the full width at half max being reduced would greatly improve the reliability of the models. This improvement could be achieved by a more extensive feature analysis, and perhaps the most telling would be congestion data on the routes in question.

# Bibliography

[1] EU Commission Expert Group. *Road transport: Reducing CO2 emissions from vehicles.* URL: https://ec.europa.eu/clima/policies/transport/vehicles_en. (accessed: 21.08.2021).

[2] The European Automobile Manufacturers' Association. *Buses: what they are and why they are so important.* URL: https://www.acea.auto/fact/buses-what-they-are-and-why-they-are-so-important/. (accessed: 21.08.2021).

[3] Jess Hohenstein et al. "Shorter Wait Times: The Effects of Various Loading Screens on Perceived Performance". In: May 2016, pp. 3084–3090. DOI: 10.1145/2851581.2892308.

[4] Zhongheng Zhang. "Introduction to machine learning: k-nearest neighbors". In: *Annals of Translational Medicine* 4.11 (2016). ISSN: 2305-5847. URL: https://atm.amegroups.com/article/view/10170.

[5] Alexandre Miguel de Carvalho and Ronaldo Cristiano Prati. "Improving kNN classification under Unbalanced Data. A New Geometric Oversampling Approach". In: *2018 International Joint Conference on Neural Networks (IJCNN).* 2018, pp. 1–6. DOI: 10.1109/IJCNN.2018.8489411.

[6] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks.* 5th. USA: Prentice Hall Press, 2010. ISBN: 0132126958.

[7] Thompson Morgan Samaroo. *Software Testing : An ISTQB-BCS Certified Tester Foundation Guide.* Ed. by Brian Hambling. BCS Learning and Development Limited, 2015.

[8] Tanuska Pavol, O. Vlkovic, and Lukas Spendla. "The Usage of Performance Testing for Information Systems". In: *International Journal of Computer Theory and Engineering* (Jan. 2012), pp. 144–147. DOI: 10.7763/IJCTE.2012.V4.439.

[9] Juha Itkonen, Mika V. Mantyla, and Casper Lassenius. "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing". In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007).* 2007, pp. 61–70. DOI: 10.1109/ESEM.2007.56.

[10] Dalson Figueiredo, Silva Júnior, and Enivaldo Rocha. "What is R2 all about?" In: *Leviathan-Cadernos de Pesquisa Polútica* 3 (Nov. 2011), pp. 60–68. DOI: 10.11606/issn.2237-4485.lev.2011.132282.

[11] Weijie Wang and Yanmin Lu. "Analysis of the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) in Assessing Rounding Model". In: *IOP Conference Series: Materials Science and Engineering* (Mar. 2018). DOI: 10.1088/1757-899x/324/1/012049.