



Team 9: Wheelie Good - Dublin Bikes

COMP 30830 – Software Engineering
16/04/2021

Jason Ballantyne, Leah Earley, Jonathan Farrell

Table of Contents

1. Introduction	1
1.1 Objectives.....	1
1.2 Application Functionality Outline	1
1.3 Target Audience	2
1.4 Audience Expectations.....	2
1.5 Unique Selling Points	3
2. Architecture of Application.....	9
2.1 Technology Used.....	11
2.1.1 Frontend: HTML, CSS, JavaScript, jQuery, Ajax.....	11
2.1.2 Backend: Python, Flask, MySQL	11
2.2 Issues Faced and Solutions	11
3. Design.....	13
3.1 Feature Requirements	13
3.2 Unique Features.....	14
3.3 Quality-of-Life Improvements.....	15
3.4 Frontend Development.....	15
4. Data Analytics	17
4.1 Machine learning model	17
4.2 Data Cleaning	18
4.3 Model Training.....	19
4.4 Model Evaluation	20
4.5 Data flow – Inputs and Outputs.....	22
4.5.1 User Input	22
4.5.2 Model integration with flask.....	23
4.5.3 Application Output.....	23
4.6 Issues and Solutions.....	24
4.6.1 Seasonal Data.....	24
4.6.2 Difficult Data	24
4.6.3 Predictive Weather Limits.....	24
4.6.4 Excessive Zero Bikes.....	25
4.6.5 Pickle File Size	26
5. Process:	26
5.1 Overview:	26
5.2 Sprint Planning Meetings:	26

5.3 Product Backlog	28
5.4 Burndown Charts	30
5.5 Issues / Resolutions.....	32
6. Bugs.....	32
7. Future Work.....	33
8. Conclusion.....	34

1. Introduction

The purpose of this project is to create a clean and functional web application for a user to navigate the Dublin Bikes self-service rental system. The user can find the closest available bike, plan their bike route, obtain weather information, visualise graphs on availability as well as a host of additional features which will be discussed later in this report. The report covers detailed information regarding Architecture, Design, Data Analytics, Process and Retrospective/Future Work.

The live deployment of “Wheelie Good – Dublin Bikes” can be accessed via the link below:

<http://ec2-3-85-18-119.compute-1.amazonaws.com:5000/>

The GitHub repository of “wheelieGood” can be accessed via the link below:

<https://github.com/JFarrell/wheelieGood>

1.1 Objectives

Primary objectives for the application were as follows:

- Display map markers for each station
- Display a corresponding onclick info window per station marker.
- Provide drop-down select menus for choosing bike station.
- Provide tabular data to display further information per station.
- Provide predictive data from a machine learning solution, for the user to plan their journey.

1.2 Application Functionality Outline

The application provides two web pages for the user to interact with. These pages display information about the Dublin Bike Stands, both current and predictive, which the user can use to assess and plan their journey.

The first page provides current bike station details. It provides occupancy details for a chosen station, some statistics, and a suggested “next nearest station”. The “Plan Your Journey” page provides predicted future occupancy details and a route planner functionality.

The application uses a variety of technologies to implement. The back end of the web application is handled by python/flask, while the front end is handled by JavaScript/HTML/CSS. The application is a web-based application run on an AWS EC2 Linux Ubuntu instance.

1.3 Target Audience

Dublin City has a population of roughly 1,228,000 people. In recent years, it has become a tech hub for big technology companies such as Google, Facebook and Accenture. These companies have attracted many people to come work in Dublin City. As a result of this, many of these people may require numerous transportation modes, including cycling. Cycling has the added benefit of being able to avoid traffic congestion in comparison to driving in Dublin and as such, is an attractive mode of transport. Dublin City is also home to one of Ireland's largest University's, Trinity College Dublin, home to thousands of students and with Dublin City being the capital of Ireland, thousands of tourists flock to Dublin to see its sights. Therefore, our target audience is the commuter population, student population and tourist population that will be using the application on a daily basis.

With the advent of COVID-19, we estimate our numbers will not be as high as they would be pre-COVID times. With many people forced to self-isolate and work from home, we estimate user numbers may be in the tens or hundreds on a daily basis and will probably not reach the numbers it had prior to COVID-19.

1.4 Audience Expectations

- The users must be able to locate a bike station with an available bike and bike stand without spending too much time finding out details like station name and address. The map should be clear and show all bike stations.
- The user must be able to find the next nearest station with availability if a station they have chosen has no availability.
- The user must be able to see average availability trends in the app to better inform their decision making.
- A user must be able to check the weather on the app to decide whether to use a bike instead a more dry mode of transport.
- People who are planning to go to the city in the future will be able to see predicted availability for use based on his/her expected travel plan, good visualization of the availability trend must be provided so users can modify his/her plans.

1.5 Unique Selling Points

This section will discuss the unique selling points of the Wheelie Good Dublin Bikes Web Application by comparison with the official Dublin Bikes Web Application. Prior to beginning the build of the web application, the Wheelie Good Team brainstormed on ideas of how to create a unique web application that offers the client more readily available features that would provide a better overall user experience.

Home Page

It was evident from the home screen of the official Dublin Bikes web application that the map itself was a key feature of the web application (Figure 1). This is a feature we also felt needed to take priority in the web application as it provides the user with a quick visual, which displays the bike station information quite readily and quickly. However, we noticed in the Official Dublin Bikes application, that not all bike stations were shown on the homepage but rather a zoomed in display of several markers representing bike stations was shown. We felt it was important to allow the user to see all bike stations when the home page was accessed (Figure 2). We feel this is a unique selling point as it allows the user more freedom in choosing an appropriate bike station.

Weather Widget

An additional feature we created which we believe to be a unique selling point is the current Weather Widget that is displayed on the Home page of Wheelie Good Web Application (Figure 2). We decided to implement a current Weather Widget as we felt as a user, knowing the weather is a key decision factor in deciding whether to take a bike from one of the Dublin Bikes stations. We feel this is a unique selling point as it provides the user with more information that may impact their decision to use Dublin Bikes.

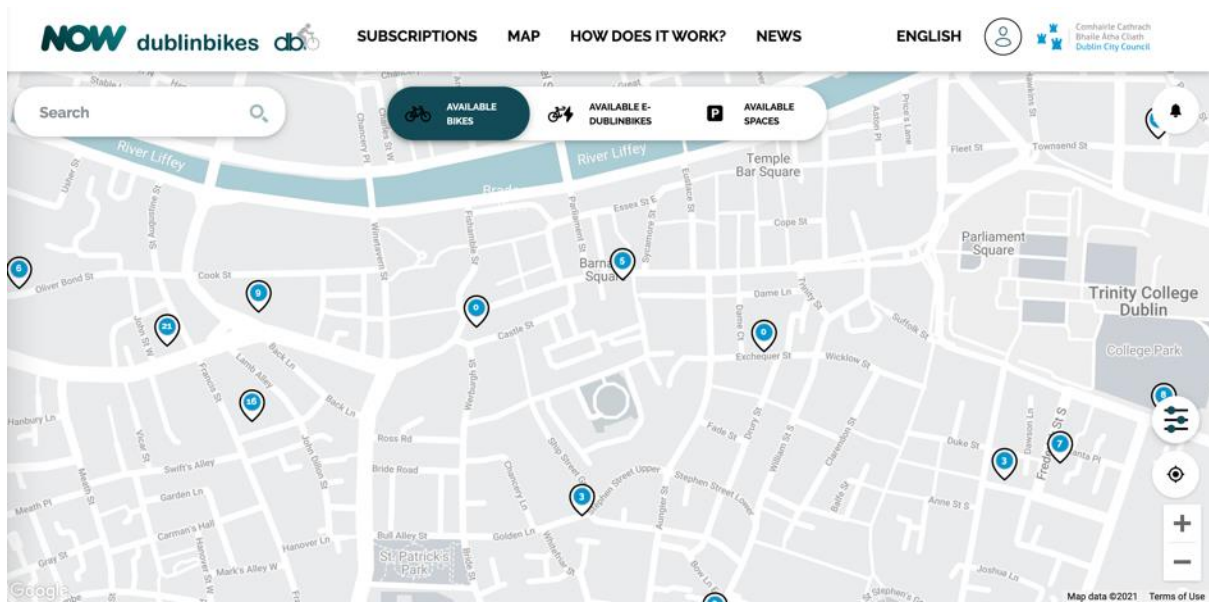


Figure 1: Home page of Official Dublin Bikes Application.

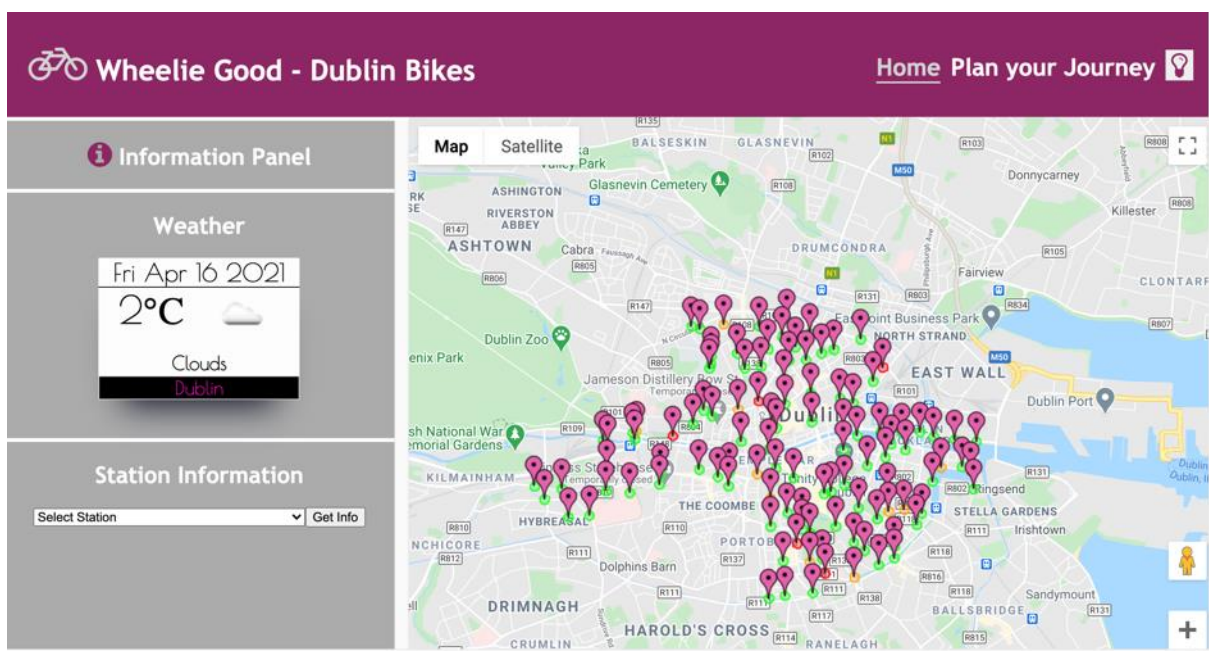


Figure 2: Home page of Wheelie Good Dublin Bikes Web Application.

Analytics Graphs

An additional selling point of the Wheelie Good Web Application is the implementation of analytics graphs on the Home page of the application (Figure 3 and 4). When a user selects or clicks a station for information, two graphs are displayed with information such as Daily Average number of bikes per station and Hourly Average availability information per bike station. This is a feature that is not included in the Official Dublin Bikes Web Application

(Figure 6) and as such we feel it is a unique selling point of the Wheelie Good Web Application. The user has more information available to them that will make allow the user to make an informed decision on the best times and days to use a Dublin bike or return a Dublin bike to a station.

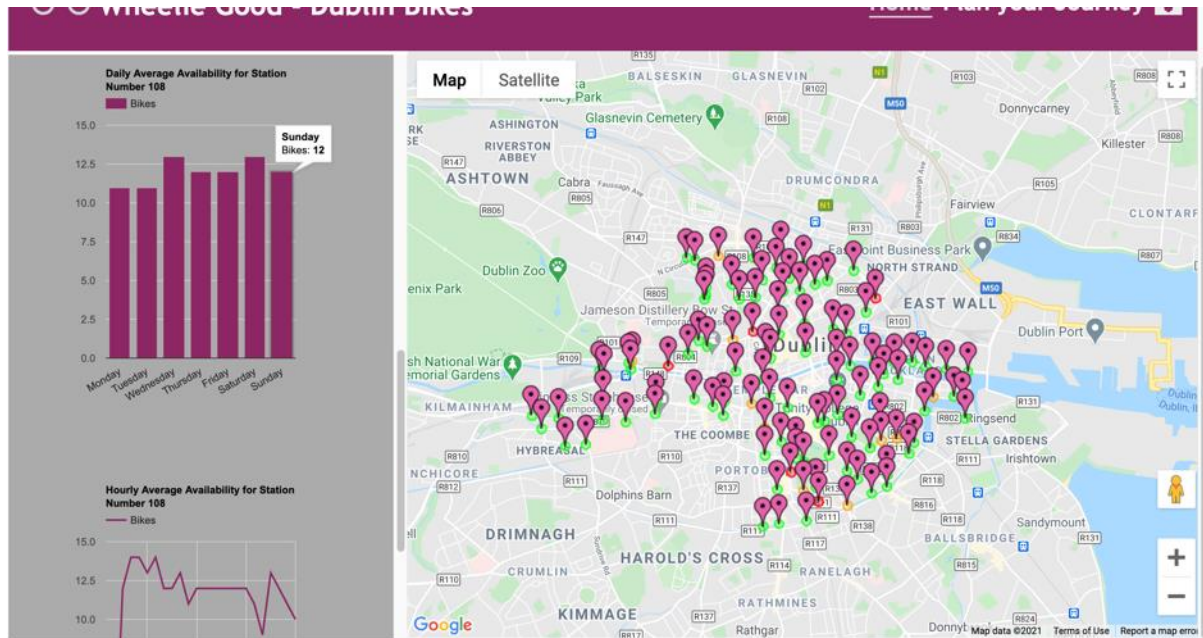


Figure 3: Daily Average Availability Information for Stations on Home page of Wheelie Good Web Application.

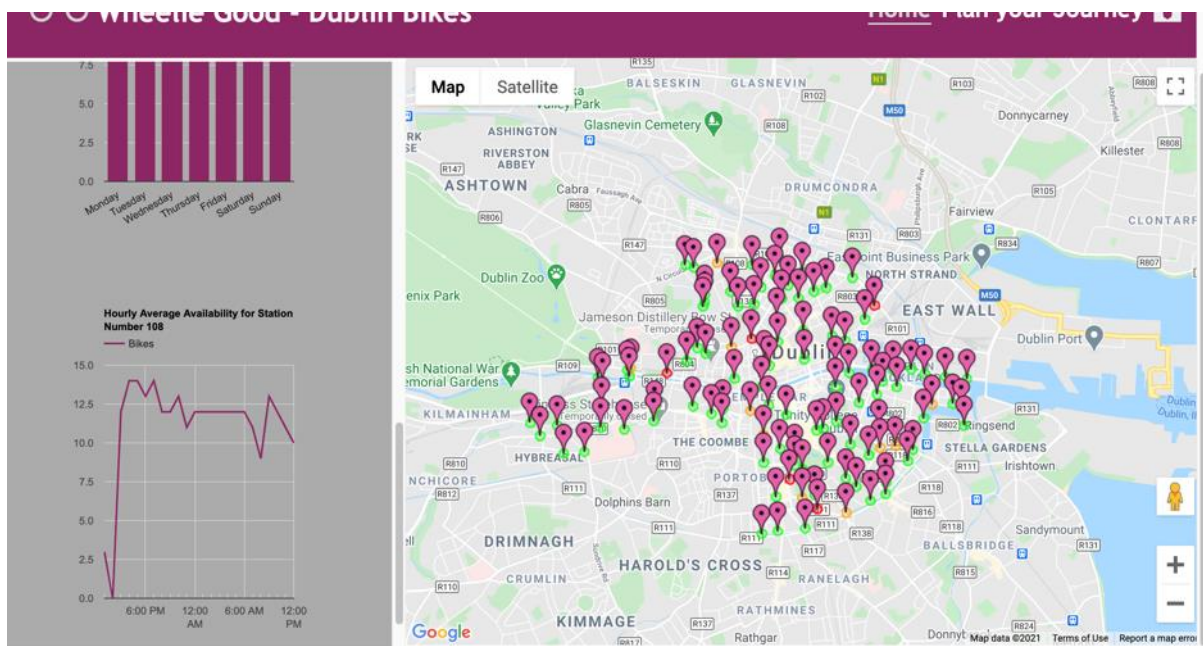


Figure 4: Hourly Average Availability Information for Stations on Home page of Wheelie Good Web Application.

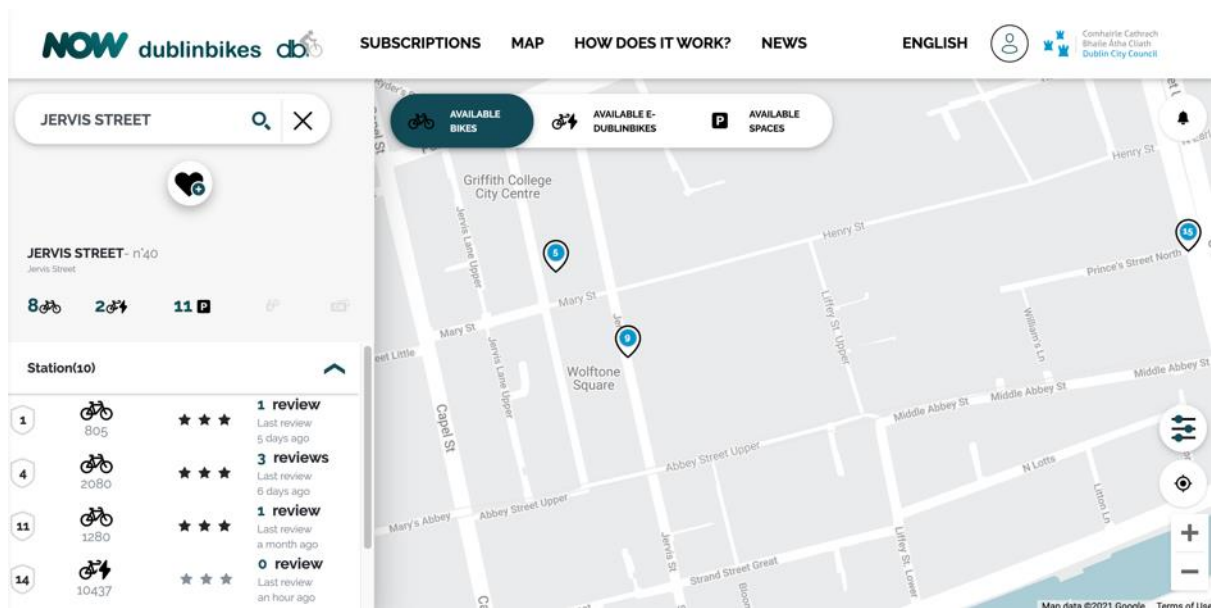


Figure 5: Information Panel in Official Dublin Bikes Application, no analytics graphs are displayed.

Next Nearest Station with Availability

A unique feature that we felt may impact user experience was the addition of a Next Nearest Station with Availability (Figure 6). This is a feature that is absent on the Official Dublin Bikes Application. This information is very useful to the user as it can provide information to the user of which station to go to next if the station they select has no available bikes or bike stands.

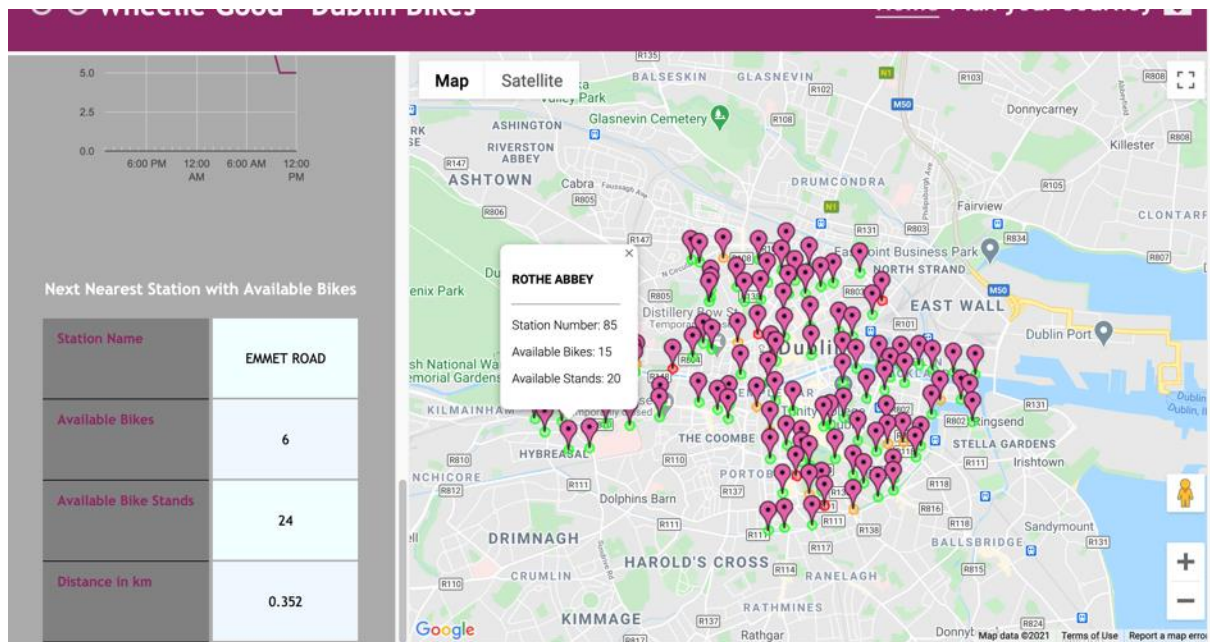


Figure 6: Next Nearest Station Feature on the Wheelie Good Dublin Bikes Application.

Plan your Journey Page

The Wheelie Good Web Application also offers another unique selling point in the form of a Plan your Journey page. We felt that the user should be able to have information available to them that would allow them to make an informed decision on how to best plan their bike journey. Information such as predicted number of bikes and available stands at a given station and directions to and from their chosen bike stations (Figure 7). We felt the directions feature was only a useful feature if the directions were provided in the information panel, as otherwise, the user would just see a route from their starting point to destination without any tangible, useful directions that they could use to actually get to their destination (Figure 8). This is a feature that is not included in the Official Dublin Bikes Application and as such we feel it is a unique selling point of our application.

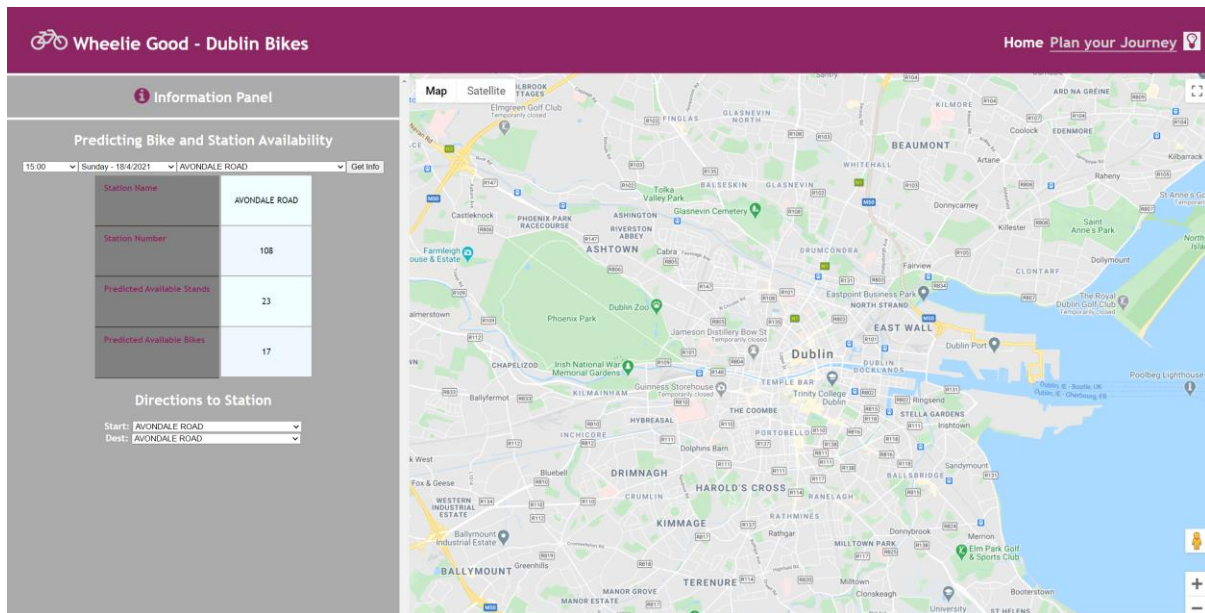


Figure 7: Predicted Availability Information on the Plan your Journey page in the Wheelie Good Web Application.

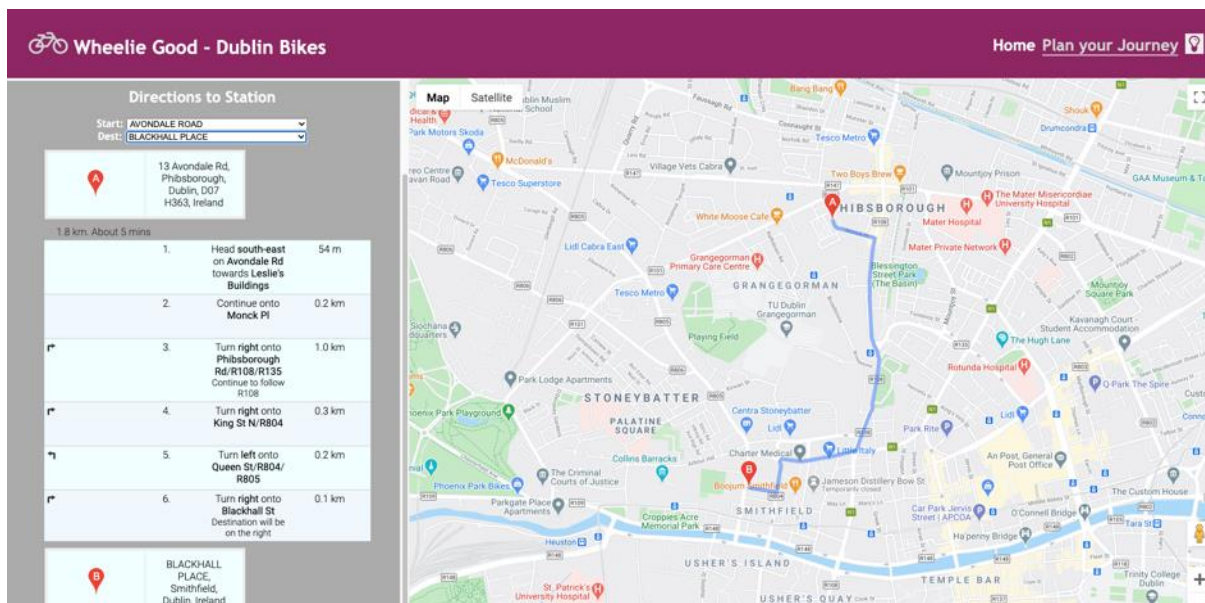


Figure 8: Directions panel on the Plan your Journey page in the Wheelie Good Web Application.

Dark Mode Feature

The final feature that we believe sets our web application apart from the Official Dublin Bikes Application is the addition of a dark mode toggle (Figure 9). In recent times, more people

spend increasing amounts of time looking at screens every day. This can cause eye strain and eye fatigue. We felt a modern feature would be to implement a dark mode feature as many users may access the web application at night prior to the day they use a Dublin Bike. This allows the user to access the Wheelie Good Dublin Bikes Web Application in a dark mode that will not cause as much strain on the eyes.

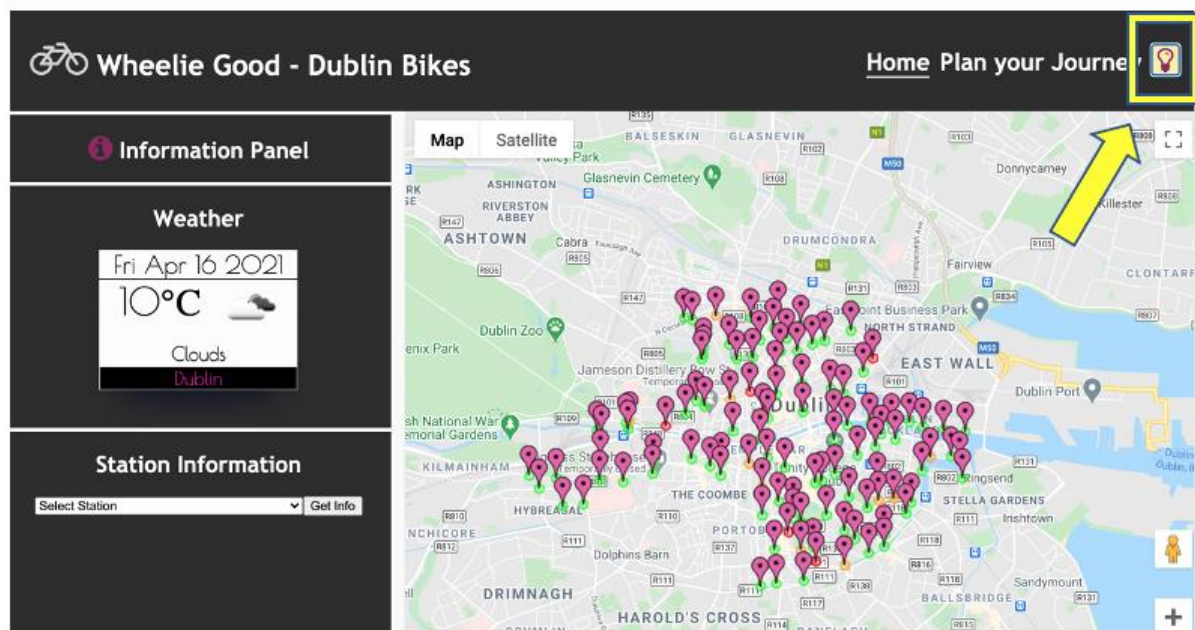


Figure 9: Dark Mode Feature in the Wheelie Good Web Application.

2. Architecture of Application

The architecture of a web application determines how application components communicate with each other to ensure successful and efficient web application deployment. The web application architecture describes the interactions between applications, databases and other component software. It ensures that multiple applications work simultaneously and it is an extremely important mechanism as it ensures communication between the client and the server. The architecture of the application was an important component to understand prior to building the web application. As a team we made sure to understand how the different applications and databases would communicate with the web application which ensured that we did not waste time implementing a component that was not architecturally sound, possibly causing delays to application launch.

The architecture of the Wheelie Good Dublin Bikes Web Application consists of several components (Figure 10). The most important component of the Wheelie Good Dublin Bikes Web Application is the server. For the purpose of this application, we created a flask based server. The flask server communicates with two Application Programming Interfaces, JCDecaux, which stores current Dublin Bikes Station information and Openweathermap.org, which stores current and forecast weather data. The flask server requests this data and upon request success, sends this data to an RDS database periodically via a nohup scheduler that runs two python scraping scripts to extract the information needed from the data. This information is then sent to the front-end of the application where the client can access the information they need. The information is displayed using dynamic JavaScript and HTML. The flask server is hosted on an Amazon AWS EC2 instance. An EC2 instance is a virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the Amazon Web Services (AWS) infrastructure. Throughout the building of this web application, Git Version Control was used to records changes to a file or set of files over time to ensure recall of specific versions.

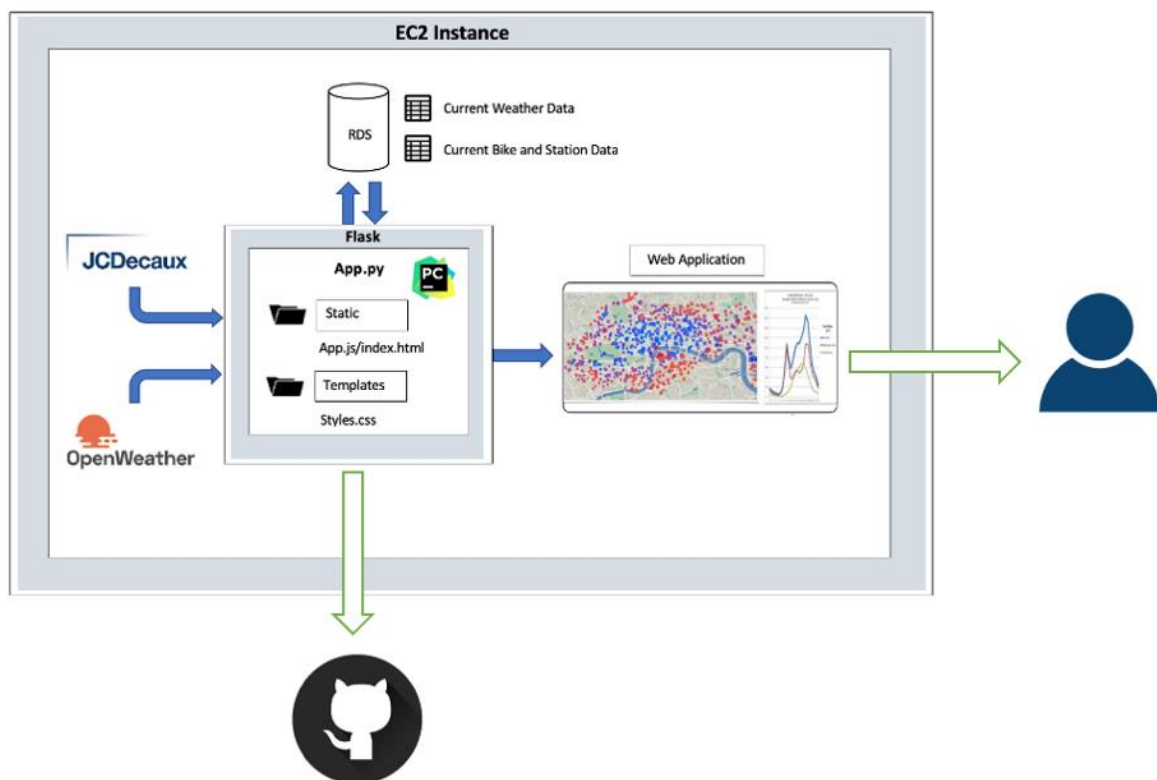


Figure 10: Architecture of the Wheelie Good Dublin Bikes Web Application.

2.1 Technology Used

2.1.1 Frontend: HTML, CSS, JavaScript, jQuery, Ajax

- HTML and CSS was used to style the web application front-end information. This was built from scratch, no template was used. The team decided we wanted a web application that was relatively unique.
- JavaScript and jQuery are used to process data passed to and from Flask and insert data into HTML components.
- An Ajax font awesome icon library was used to display icons on the web application.

2.1.2 Backend: Python, Flask, MySQL

- The Flask application on the EC2 was designed modularly so that debugging and scaling would be much easier with different functional modules separated. This was done by implementing re-usable functions in the app.py python file.
- MySQL was used to store the scraped bike station and weather data that could then be sent to the Flask server via SQL queries in the app.py python file.
- Python was used for implementing a predictive data model. See Data Analytics Section.

2.2 Issues Faced and Solutions

- Scrappers could only run for 30 minutes due to an infinite for loop. Solution: Removed loop and added a nohup command that would run the scraper script every 5 minutes.
- Displaying API keys in GitHub repository, did not want this to be public. Solution: added a config file with all the API keys that ensured they were private.
- Various module versions on different computers meant scripts were not running properly. Solution: add a requirements.txt file with all modules and implement a yml environment.
- Google maps was not loading. Solution: added loading map asynchronously which meant the map was rendered successfully.

- GPS location not working on EC2 instance. EC2 requires a security certificate which we didn't have using HTTP protocol. Solution: We decided to remove the user location feature.
- Web browser was not clearing cache after file changes, therefore, changes were not being shown. Solution: rectified by clearing the cache.
- When pulling data from database, it took a long time due to requesting all data in the database. Solution: Decided to separate information into two databases based on static and dynamic data so the data was received quicker.
- Issue with bootstrap implementation, JavaScript features were not working when a bootstrap template was used. Solution: Built styling using CSS and HTML.
- Issue with screen responsiveness when displaying application, the information was not changing according to size of the screen. Solution: Added a class container to the HTML and set max width to ensure screen responsiveness.

3. Design

3.1 Feature Requirements

As a team, we met all the technical and user requirements proposed for this app and proceeded to add features that we felt would enhance to the user experience while contributing to the aesthetic.

FEATURES	IMPLEMENTED
Scraping Data	
Data Collection through JCDecaux API	Yes
Data Stored on RDS	Yes
Weather Data Scraped	Yes
Weather Data Stored on RDS	Yes
Javascript Features	
Weather widget. Showing current weather	Yes
Predictions based on ML Model	Yes
Alphabetical Dropdown Searches	Yes
Scrollable Side Bar	Yes
Google Map Integration	Yes
Coloured Map Markers	Yes
Populated info windows	Yes
Coloured Circles on Markers for Bike Occupancy	Yes
Close Info Window when new Info Window is clicked	Yes
Hourly & Daily Bike Charts	Yes
Next Closest Station Recommendation	Yes
Route Planning Implementation	Yes
Geolocation Feature	Yes
Darkmode Toggle Integration	Yes

Figure 11: Full list of Features

3.2 Unique Features

One such unique feature we included was a dark mode toggle that displays light text on a dark background. Dark mode is especially helpful for those cyclists viewing the app at night as the reduced brightness can reduce eye strain in low light conditions.

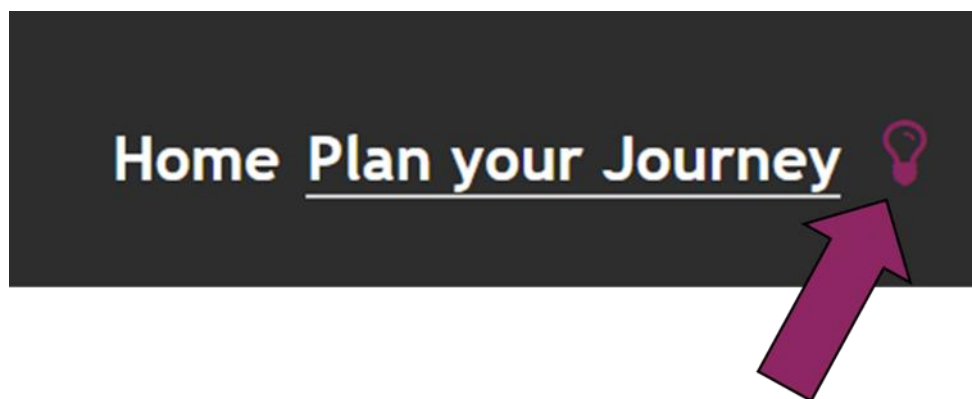


Figure 12: Dark Mode Toggle

Another unique feature that was implemented was the ability to recommend the next closest station to the user. In a situation where the user is looking to get a bike and the station is empty, the cyclist might need to identify where the next closest station is. Similarly, if the user is returning a bike and the station is full, they will also need to identify the next closest station to return the bike to before an additional charge takes hold.

A unique feature which adds to the functionality of the app was the addition of coloured circles around the map markers. These coloured circles correlate to the bike availability at that particular station. For example, if the number of available bikes is greater than 5, the circle around the station will be displayed as green. If the number of available bikes is greater than 2 but less than 6, the circle will display as orange and if the number of bikes is 2 or less, the circle will appear as red. This helps the user identify from first sight which stations have bikes available to the user.

3.3 Quality-of-Life Improvements

Some design implementations, while not enormous, can contribute majorly to the everyday use of the application. An example of such an improvement was making the app more efficient.

In order to increase the overall performance, an emphasis was put on the specificity of our SQL queries to the database. We reduced the number of queries and only the data that was absolutely essential to the running of the app was called from the RDS. As well as this, we selected only the columns required and used “Limit” statements for each query. The benefit gained from doing so, meant that the SQL queries were quick which led to a more responsive experience for the user.

Another Quality-of-Life improvement added was ensuring each of the drop-down menus were ordered alphabetically. This was done using two methods depending on where we needed to reuse the data. The first method was through SQL using “ORDER BY *column* ASC”. The second method was through pandas by using “df.sort_values” combined with “drop_duplicates”. Both methods were effective and added to the readability of the drop-down menus for the user.

3.4 Frontend Development

Our initial early Wireframe mock-ups (See Fig 13) involved creating multiple pages to navigate between the Home page, the Bike Route Planning page, the About page and the Contact page. As we progressed with the application, we agreed that this felt tedious and unnecessary and ultimately settled on just a two-page application, Home and Route Planning. By combining all the features in a more confined space, we believe this adds to the user’s ability to navigate our application. Essential details that a cyclist may immediately require.

For our frontend, we decided to opt for a design where the map was the focus. A scrollable sidebar situated to the left comprised of each of the features. This led to the application having a clean and minimalist look while also being extremely functional for the user. No bootstrap was used in the implementation and only consisted of HTML, CSS, JavaScript, Ajax and jQuery.

Early Wireframe Mock-ups

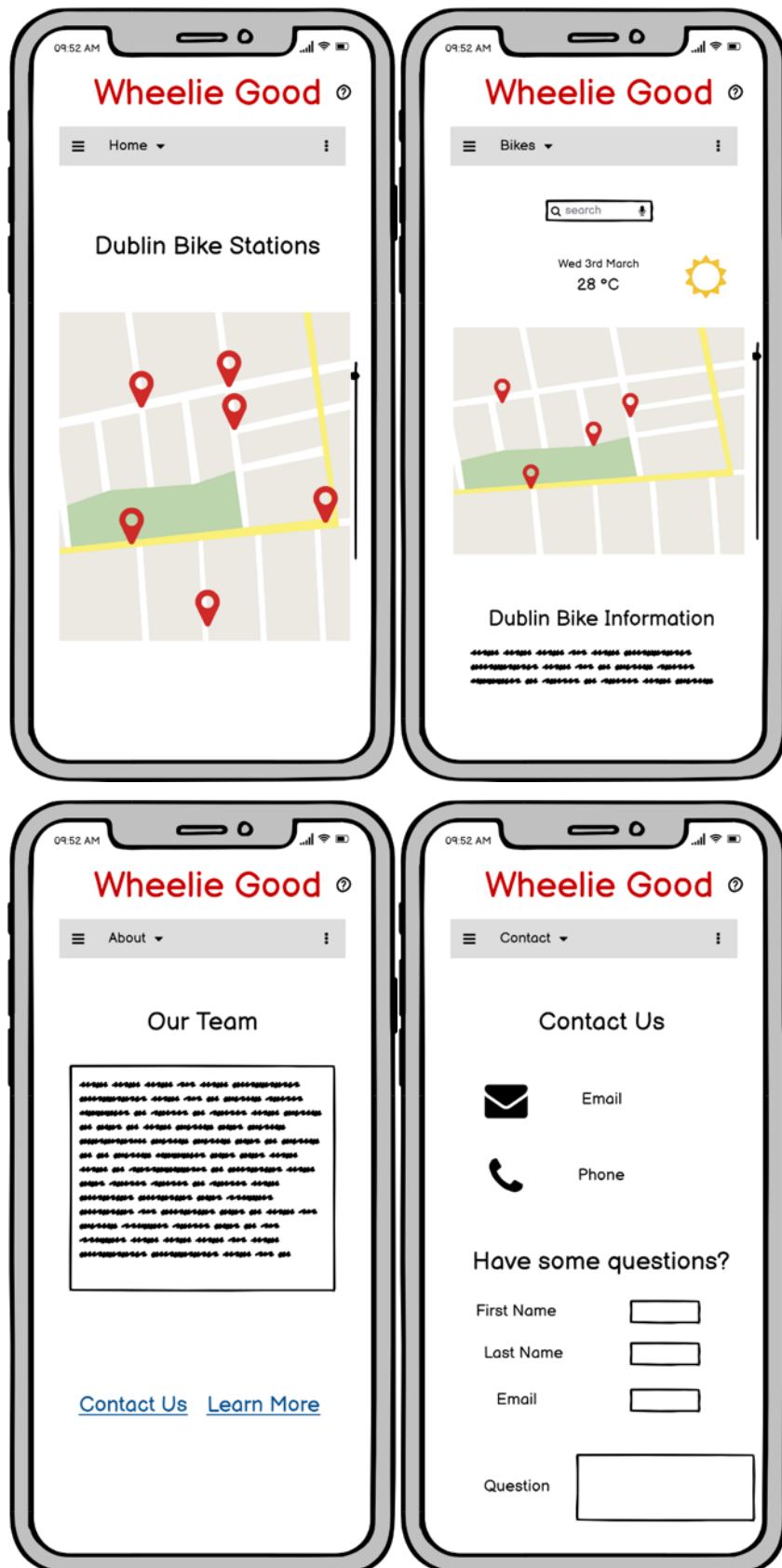


Figure 13: Initial Wireframe Mock-Ups

4. Data Analytics

The machine learning model used in our application was able to provide the user with a predicted number of bikes and bike stands likely to be available on a given day at a given time. It considered historical data of bike station data on given days and times, and the corresponding weather.

To collect this data, we ran a scraper script on an ec2 instance, which collected weather data from the open weather maps API, and bikes data from the JCDecaux Dublin Bikes API. This data was collectively stored in Amazon RDS Database and used to train the predictive models used in our application.

The models were stored as pickle files, in a designated directory, and integrated with the flask script.

The following sections will detail the processes and segments involved in building and integrating our model with our application, in the hopes of providing an accurate and useable prediction for the user.

4.1 Machine learning model

We opted for a random forest regression algorithm, from the scikit-learn library. The reasons for opting for this model are as follows:

- We sought to predict a specified number of available bikes, which is a regression task.
- We aimed to use a simple and effective model implementation, as the data preparation phase would be bulky. The implementation of these models was straightforward, and the results were convincing.
- The linear regression and KNN regressor models both provided poor results when tested with r^2 .
- Decision tree and random forest provided an average of about 98%.
- Random forest, as an ensemble model, provides a slight advantage over a single decision tree.

4.2 Data Cleaning

The stored data set, at the time of building the models, exceeded one million rows in the historical bikes data. The weather data was much smaller as it was only one row per call, while the bikes model was over 100 rows per call.

The historical bike data set had been built in a compact format, and only contained essential rows.

	number	available_bike_stands	available_bikes	last_update
0	42	22	8	2021-03-08 13:44:23
1	30	16	4	2021-03-08 13:45:14
2	54	31	2	2021-03-08 13:36:34
3	108	37	3	2021-03-08 13:44:45
4	56	31	9	2021-03-08 13:44:20

Figure 14: Historical Bike Data Sample

By comparison, the weather dataset needed significant cleaning before it could be paired to bikes on their nearest related time cells.

The weather dataset was purged of data not considered essential to the model. Additionally, a new set of binary value columns were added that indicated the weather possible “weather main” scenarios.

These values were Clouds, Clear, Snow, Rain, Drizzle, and Thunderstorm which were given a value of 0 or 1. These values were taken from the Open Weather Map API documentation, to avoid our data set only containing seasonal weather types. Being Ireland, these column values were generally “Clouds.”

coord_lon	float64
coord_lat	float64
weather_main	object
weather_description	object
weather_icon	object
main_temp	float64
main_pressure	int64
main_humidity	int64
main_temp_min	float64
main_temp_max	float64
visibility	int64
wind_speed	float64
wind_deg	int64
clouds_all	int64
dt	datetime64[ns]
sys_type	int64
sys_id	int64
sys_country	object
sys_sunrise	datetime64[ns]
sys_sunset	datetime64[ns]
city_id	int64
city_name	object
cod	int64
Current Time	datetime64[ns]

Figure 15: Historical Weather Data

The data frames were sorted, datetime columns were used to merge the two datasets and columns were appended for the weekday (0-6) and the hour of the day (0-23). Unneeded columns were dropped. The remaining columns and number of unique values for each is depicted below.

number	110
available_bike_stands	41
available_bikes	41
last_update	281954
wind_speed	27
main_temp	1001
main_humidity	24
dt	4609
Clouds	2
Clear	2
Snow	1
Rain	2
Drizzle	2
Thunderstorm	1
weekday	7
hour	24

Figure 16: Merged Dataset

Lastly, the models appeared to return a suspiciously high level of zeros for certain stations (as high as 31%). These zero values were trimmed. More detail on this process can be found in section 4.8, “Issues and Solutions.”

4.3 Model Training

Several factors contributed to the decisions around the end-model structure. Firstly, the input data needed to be intuitively relevant to the output. The weather, for example, is of significant relation to people deciding to cycle. Another evidently contributory factor is the time of day and the day of the week. The bikes usage trended higher during working hours, despite the coronavirus pandemic lockdowns. Below is a chart plotting the bike availability over 24 hours.

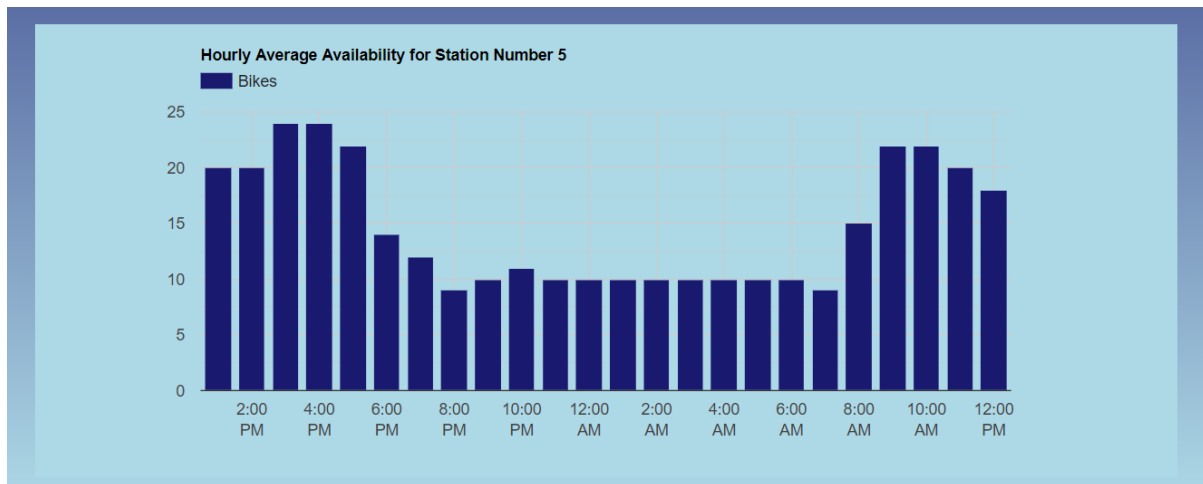


Figure 17: Sample Daily Trend

As the trend shows, there is higher rate of usage during the afternoon. These trends were important to provide to our model.

The model's equation can be summarised as follows:

Available_bikes = random_forest.predict(Temp, Wind Speed, Humidity, Clouds, Clear, Snow, Rain, Drizzle, Thunderstorm, weekday, hour)

Clouds, Clear, Snow, Rain, Drizzle, and Thunderstorm were all calculated as binary values. Weekday was imputed as 0-6, and the hour from 0-23. The number of available bikes stands was calculated by subtracting from the predicted available bikes from the total bike stands.

Although the data provided by the hourly forecast was for 8 days, not 7, the overlap of integers would cause a conflict. For simplicity, this 8th day was erased, and a limit was set at 7.

4.4 Model Evaluation

The models needed to be assessed, at least primitively, for accuracy. Fortunately, the scikit library offers some very convenient metrics. We measured accuracy using an r2 score and feature_importance. Although this was a limited spectrum of testing and no action was taken on the feature importance test, the r2 gave very convincing and decisive results.

The models tested and their initial accuracy results are displayed below.

Model Type	R2 Accuracy Test
K-Nearest Neighbour Regressor	70.32%
Linear Regression	3.67%
Decision Tree Regressor	98.04%
Random Forest Regressor	98.35%

Figure 18: Models vs Accuracy

Evidently, linear regression appeared entirely useless, and KNN offered nowhere near as good results as the two tree-based models. At this point, the decision was made to test the models on a per-station basis and compare the results of the two tree algorithms. The results are as per the graphs below.

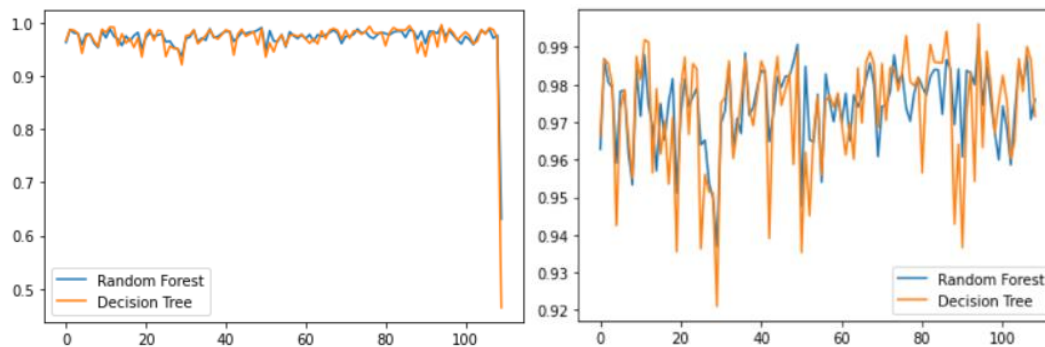


Figure 19 and 20: Accuracy trends per station

As the first image shows, there is a single significant outlier. This is a recently added single hybrid bike stand. The second image presents the data without that stand, for a closer look. As can be seen, there is little in terms of major deviances between the models. Both are highly successful. However, there is slightly greater aberration in the decision tree model, which is to be expected given the random forest's more reliable ensemble nature.

We also used a feature importance test. The rest returned a list of features and their importance to the predictive algorithm.

As can be seen in the figure, “Snow” and “Thunderstorm” provide no contribution to the model. It is likely they should be removed. However, this lack of contribution is likely due to there simply being no data of this type available.

As they did not appear to detract from the model’s performance, they were left in.

main_temp	0.285955
main_humidity	0.243244
weekday	0.171406
hour	0.165779
wind_speed	0.116738
Rain	0.007883
Clouds	0.006912
Clear	0.001779
Drizzle	0.000304
Snow	0.000000
Thunderstorm	0.000000

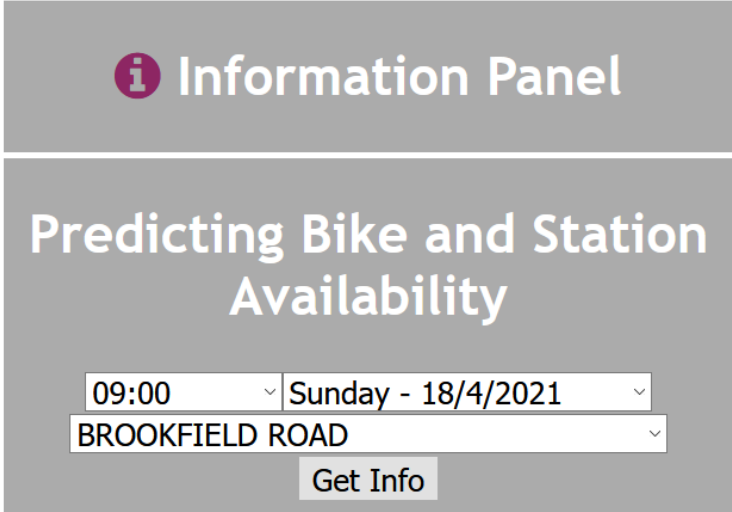
Figure 21: Feature Importance Test

4.5 Data flow – Inputs and Outputs

This section will detail an overview of the flow of data involved in the integration of the machine learning model with the application. It will detail how the application receives an input from the user, provides these inputs to the model and returns predictions to be provided to the end user again.

4.5.1 User Input

The user supplies their request variables via a series of drop-down select menus provided on the “Plan your Journey” page. The drop-down menus are for hour of day, day of week, and given station.



The screenshot shows a web interface titled "Information Panel" with the subtitle "Predicting Bike and Station Availability". Below the title, there are three dropdown menus. The first dropdown menu shows "09:00", the second shows "Sunday - 18/4/2021", and the third shows "BROOKFIELD ROAD". Below these dropdowns is a button labeled "Get Info".

Figure 22: Station Prediction Dropdown Menus

The times range from 00:00 – 23:00, the dates span a week from the current day (0-6) and the stations are alphabetised. The “get info” button submits these dates as variables to a flask route path. This data was then passed to flask to be used as parsing parameters for forecast data.

4.5.2 Model integration with flask

The flask function provides a vital pathway between the front-end and back-end. The day and time variables passed to it from the front end are used to filter the appropriate forecast weather data, while the station number is used to determine the appropriate data model to use.

The forecast json material returned is very dense. It was necessary to parse this data and find specific material relating to the day and time the user had requested. The data returned from the API was translated to a pandas dataframe and passed to a formatting script, `df_formatting.py`. Parsing the data was handled here by a series of functions, with the objective of returning a list that could be passed to the relevant predictive model.

A limitation of the forecasting data was that hourly data was limited to 48 hours. Thereafter, only daily data was available. We considered it most optimal for the model to be able to make full use of the forecasting material. Therefore, these parsing functions were designed to parse the data according to hourly information provided by the user and, where this data exceeded the hourly limitations, it would default to the daily forecast.

4.5.3 Application Output

That predictive models are divided into subdirectories for hourly capable models, and not (daily models). They are then further divided on a per station basis. Once the appropriate predictive algorithm is selected, a list of inputs is provided, and an output returned.

The output is packed into a json array with several other relevant data points, such as station number and particular weather details, and returned as the output of the flask route. This data is then accessible from the front-end JavaScript, where it is output as tabular data for the user.

Station Name	BROOKFIELD ROAD
Station Number	84
Predicted Available Stands	14
Predicted Available Bikes	16

Figure 23: Station Predictions

4.6 Issues and Solutions

4.6.1 Seasonal Data

A significant constraint to the model was the relatively small timeframe it had to collect data. The dataset was, although large, collected over a period of roughly 8 weeks, from the 18th of February 2021, to approximately the 12th of April 2021. This data is very seasonal. Further, it was collected during a global coronavirus pandemic lockdown.

The data, therefore, is not totally thorough. There is little that can be done about this, but it would be remedied over time with further collection. We are confident that the application is still sufficiently accurate, given it can be continuously updated as further data collects over time.

4.6.2 Difficult Data

Two pertinent pieces of data, the forecasted weather type, and the temporal data were challenging to parse appropriately. The weather data information required careful iterations to pull and append to the correct rows.

Additionally, the timestamps provided by the two APIs, although helpful for merging the data frames, could not be used in the model. They needed to be converted to integers. Fortunately, the Pandas library provides a convenient function to do this conversion, and we assigned a numeric digit to each day and daily hour.

4.6.3 Predictive Weather Limits

The Open Weather Map API call provides hourly data for 48 hours and daily data for 8 days. We wanted to provide optimal predictions, as best our resources would allow. Initially, this limitation appears to provide two solutions:

- First: Limit predictions to 48 hours in the future.
- Second: Limit prediction accuracy to daily only.

However, these both felt dissatisfactory. The resulting solution was to implement models that could do either and choose the correct model depending on the parsing outcome. The algorithm for this was as follows:

- Parse the API data from json to a Pandas data frame.
- Parse the hourly subsection of the data frame according to the user inputted metrics (hour, day, station).
- If the parsing returns a data frame, return the items required for the hourly model.
- Else (the data frame returned empty) Reparse the data frame according to the daily data, excluding the hour from before.
- Pass the resulting list to the daily model.

4.6.4 Excessive Zero Bikes

There was a large quantity of zero values being pulled to the database for specific bike stands. The issue was noticed after initial deployment, when testing the application.

The stations were targeted for model prediction testing, and the results failed to accurately portray the historical narrative. For example, station number 25 returned 0 available bikes all day on Saturdays. However, an investigation of the database told us that the station typically had between 8 and 12 bikes available on Saturdays.

This seemed erroneous and incentivised a cautious purging of 50% of the rows where a particular station on a particular day had data returning zero for available bikes in more than 15% of its overall rows. This appeared to greatly enhance the efficacy of the models. Although large amounts of zero values still existed for certain scenarios, we considered this an acceptable, though not ideal, compromise.

4.6.5 Pickle File Size

The initial model build was a single random forest model. My initial impression was that this model could take station information as a predictive factor, thus cutting the need for more models and dividing the dataset. However, the immediate hurdle was a GitHub max file size limitation of 1gb, and the model exceeded this.

This fortunately prompted further thought about how best to implement the model. Had this hurdle not been encountered, the model would not have received as thorough interrogation and the result would not have been as good. The resulting changes was that the model was divided on a per station basis, and to daily our hourly models based on the day chosen (less, or greater than 48hrs in the future) by the user.

Although this required a little more creative python scripting, the end models were more accurate and significantly much more manageable.

5. Process:

[Trello – Sprint 1](#)

[Trello – Sprint 2](#)

[Trello – Sprint 3](#)

[Trello – Sprint 4](#)

5.1 Overview:

A full log of our process could not be outlined here, and we would encourage you to visit our Trello boards for each of the Sprints which are linked above. These boards provide an overview of the daily stand-up meetings, backlog items, sprint planning, our tasks for the sprint, tasks in progress, completed tasks, minutes from formal meetings with our product owner, team meeting minutes and general queries/issues.

5.2 Sprint Planning Meetings:

For each Sprint, we assigned a Scrum master as detailed on the respective Trello board. The scrum master's responsibilities included organising the daily stand ups, communicating what was to be completed for that sprint, participating in meetings, and capturing feedback on the

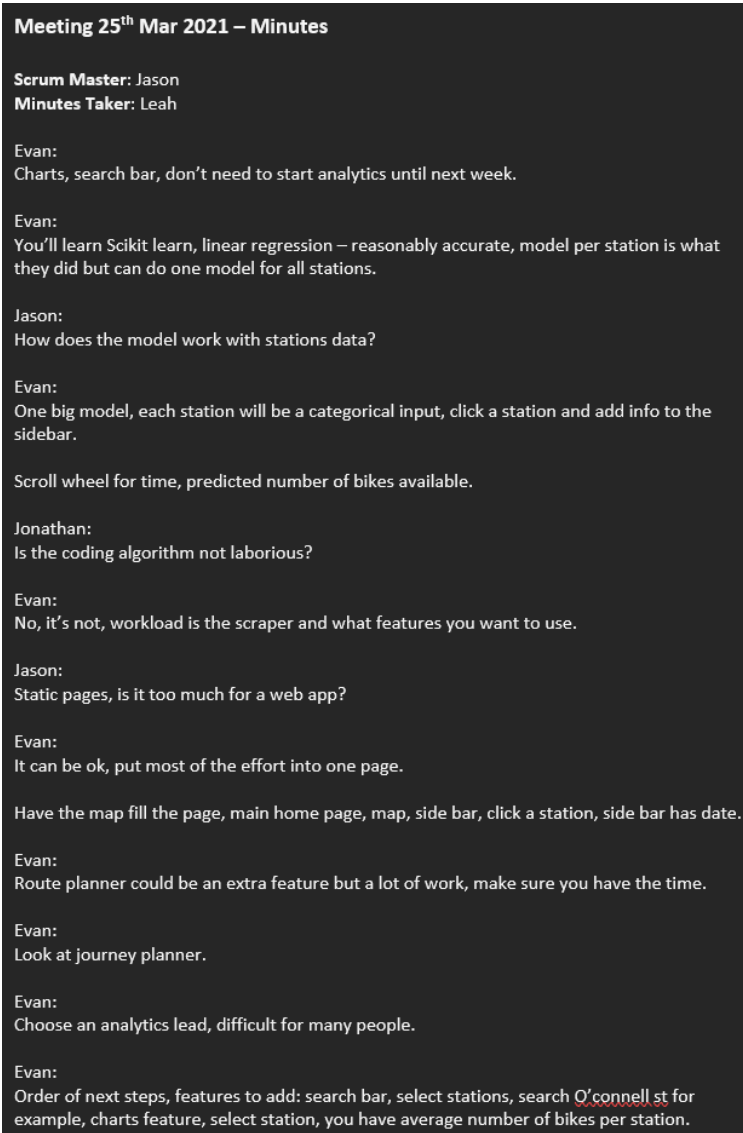
Trello boards. It was also the scrum master's responsibility to admin the Trello board for that sprint and action items for future sprints.

Our team attempted to participate in daily stand-ups where we would briefly state what we were working/focused on. Meetings would occur when we needed to discuss next steps or highlight issues. On average, we would have about meetings 4 times each sprint. As well as these meetings, we would have our weekly formal meetings with the product owner to discuss our progress and next steps. All daily stand-up, team meeting and formal meeting minutes can be found on Trello.



Figure 24. Screenshot of Stand-Ups for Sprint 3

Before each formal meeting with the product owner, a team meeting would take place. We discussed who would take the minute meetings and prepared a set a set of questions for the product owner. This was in order to utilise all of the weekly 20 minutes meetings and to highlight what was the pressing concerns within the team.



Meeting 25th Mar 2021 – Minutes

Scrum Master: Jason
Minutes Taker: Leah

Evan:
Charts, search bar, don't need to start analytics until next week.

Evan:
You'll learn Scikit learn, linear regression – reasonably accurate, model per station is what they did but can do one model for all stations.

Jason:
How does the model work with stations data?

Evan:
One big model, each station will be a categorical input, click a station and add info to the sidebar.

Scroll wheel for time, predicted number of bikes available.

Jonathan:
Is the coding algorithm not laborious?

Evan:
No, it's not, workload is the scraper and what features you want to use.

Jason:
Static pages, is it too much for a web app?

Evan:
It can be ok, put most of the effort into one page.

Have the map fill the page, main home page, map, side bar, click a station, side bar has date.

Evan:
Route planner could be an extra feature but a lot of work, make sure you have the time.

Evan:
Look at journey planner.

Evan:
Choose an analytics lead, difficult for many people.

Evan:
Order of next steps, features to add: search bar, select stations, search O'Connell st for example, charts feature, select station, you have average number of bikes per station.

Figure 25: Formal Meeting Minutes

5.3 Product Backlog

At the start of each sprint plan, a backlog list was made which consisted of tasks required to be completed during that sprint. As we became more familiar with Trello, we began attaching

coloured labels to these items from sprint 2 which detailed the type of work that was involved to complete that task, e.g. JavaScript Features, Data Analytics.

When a team member opted to work on a task, they would add their name to the card and move it to the “Tasks in Progress” and eventually “Completed Tasks” once the task had been carried out. If a task was still unfinished by the end of the sprint, it was added to “Tasks Carried Over to Sprint X”. This task would then be added to the Trello board for the following sprint. We did this to ensure that Trello board was closed off and each team member are working together on the same board.

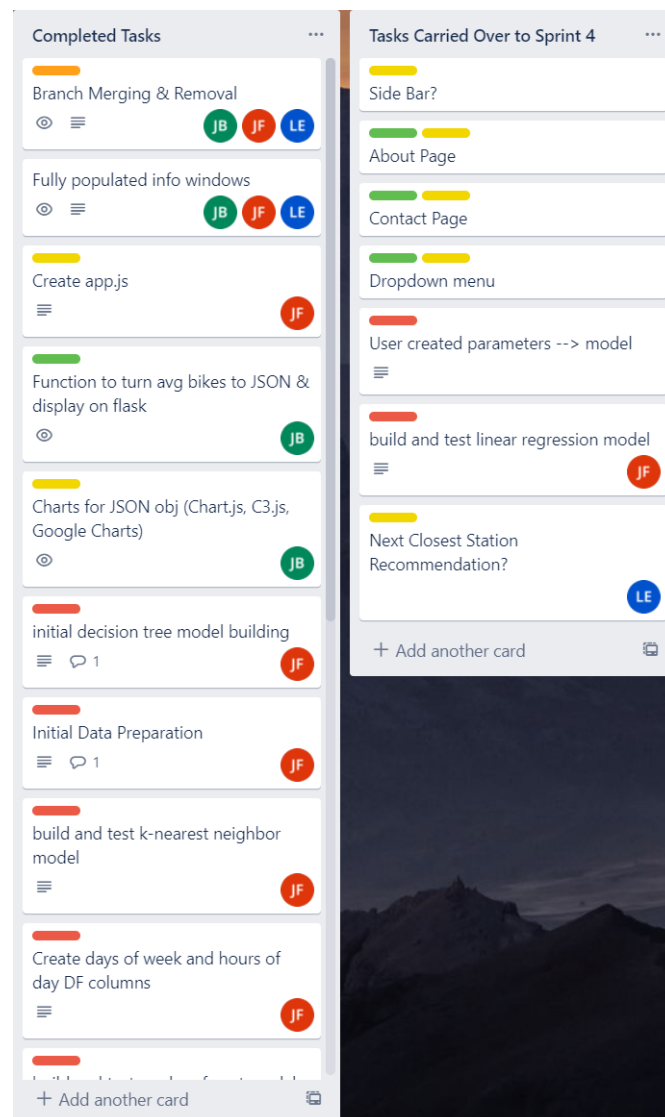
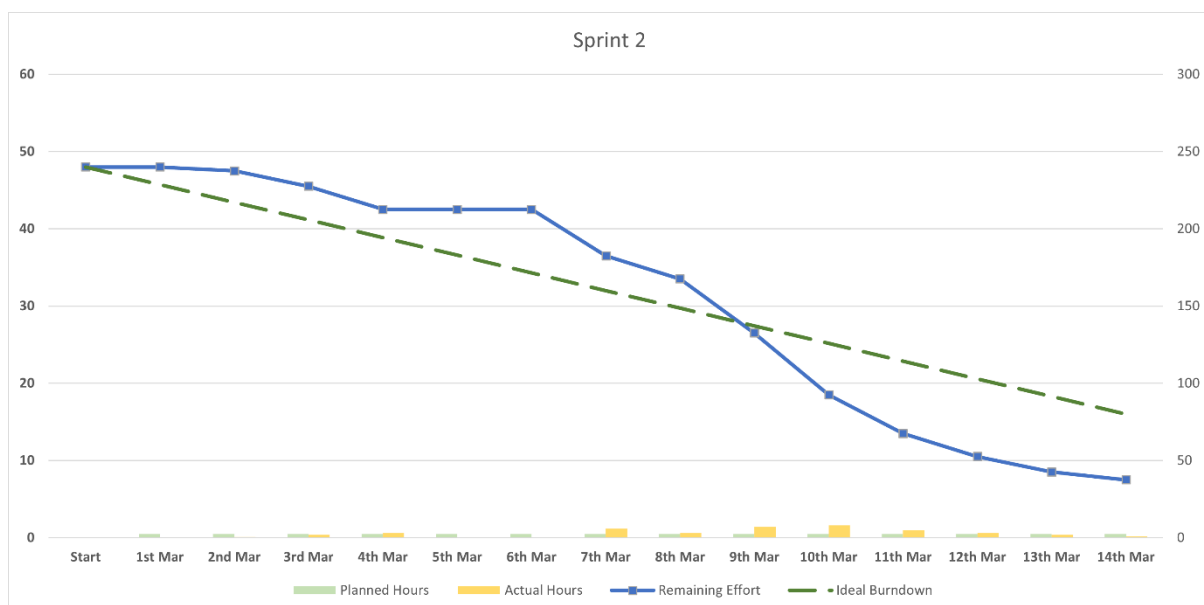
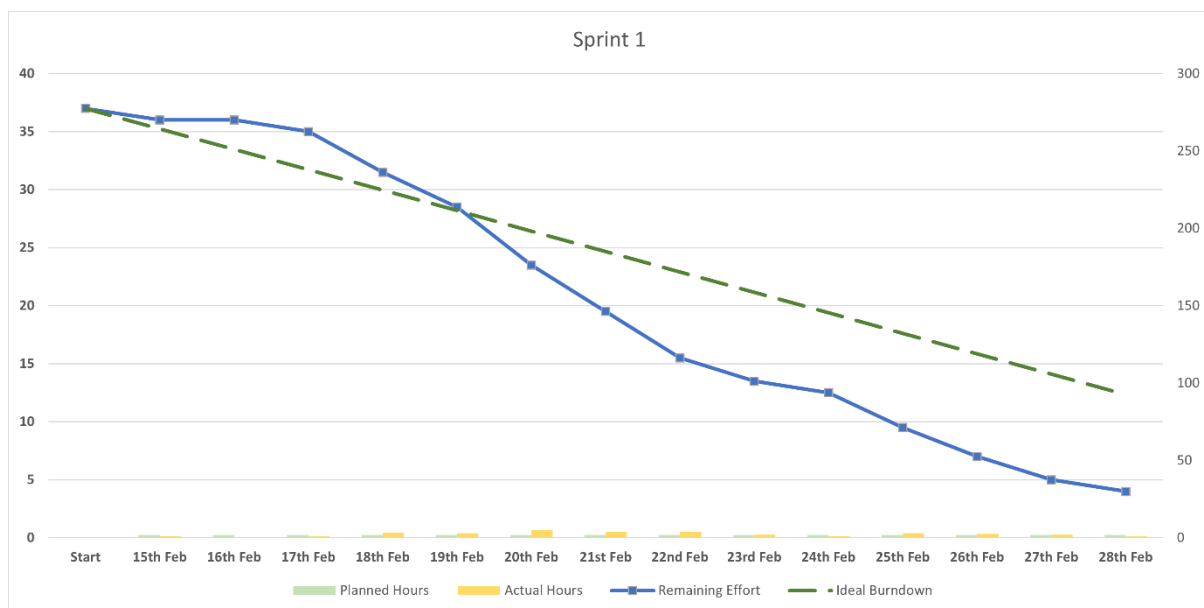
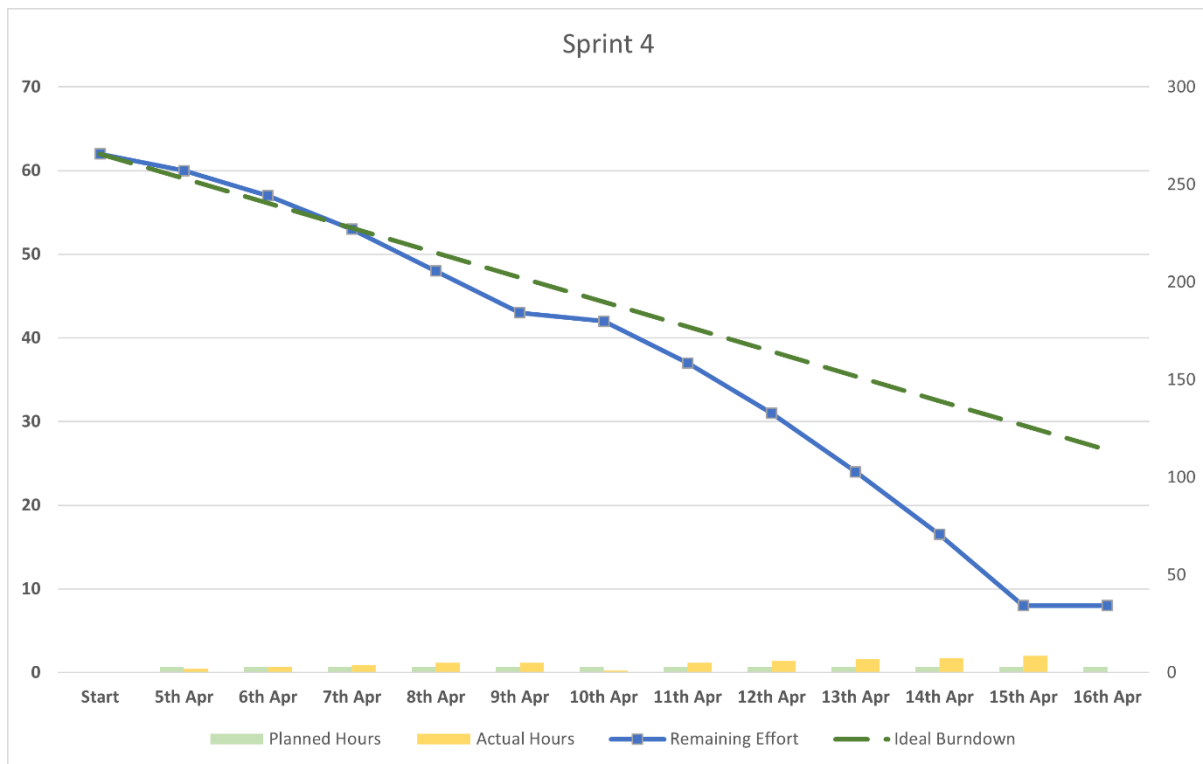
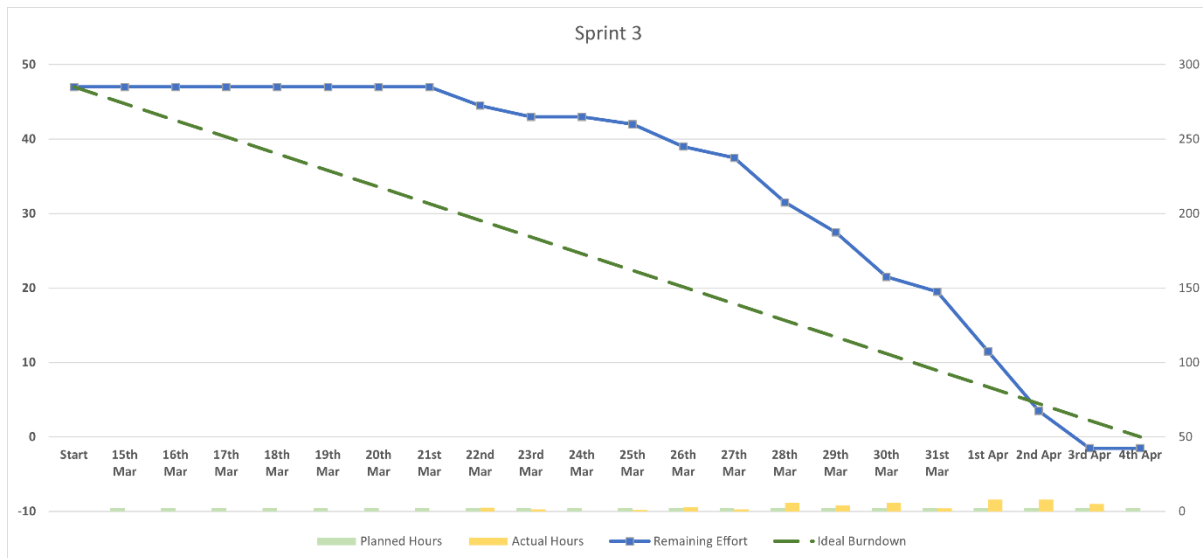


Figure 26: Trello Completed Tasks

5.4 Burndown Charts

Burndown Charts were used to keep the team on track for the sprint. Having graphical representations of outstanding work helped give perspective to our performance. During our sprint planning phase, an estimation would be added to our burndown chart as to how long each task would take. When a task was added to completed tasks, the team member would update the burndown chart with the actual time it took them to complete that task and what day the hours were allocated on.





Figures 27-30: Burndowns for Sprints 1-4

5.5 Issues / Resolutions

Our Trello board was used to deal with issues and resolutions. Our “Queries” list detailed current problems we were experiencing. Other team members were then made aware of the issue and could aid in a resolution before marking it as complete.

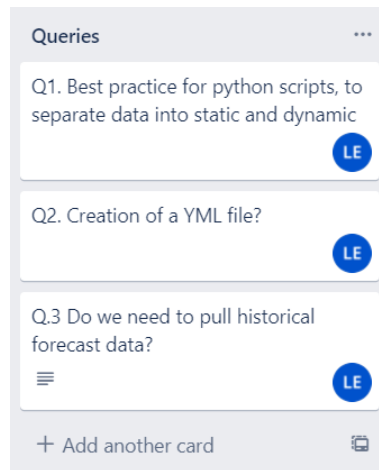


Figure 31: Trello Board Queries

6. Bugs

- Element rendering in certain features such as map and graph rendering is somewhat slow. We tried to implement more specific SQL queries to reduce latency, but it is still a bug in the application. This may be due to performance times of certain JavaScript functions that may need to be further developed in the future to reduce latency.
- The next nearest station with availability feature only works on the click of marker. Initial plans had been to implement this feature when the user selects a station from the dropdown menu. However, after multiple attempts to implement this with the dropdown menu, a bug still existed where the next nearest station with availability did not change after a new station was selected from the dropdown menu. Therefore, we made the decision to exclude this feature with the bug and only have this feature working on the click of a marker. This will be further assessed in future work.
- Another bug that existed in the next nearest station feature was that it only gave the next station information with availability. The JavaScript function written actually includes an array of next nearest stations with availability within 500 metres, so the

user has even more information to better inform their decision. We had this feature almost working. We had issues displaying the array. The bug was that after every marker click the array would be added to the last station's array, instead of resetting and just giving the next nearest stations within 500 metres to the selected station. This will be further assessed in future work.

7. Future Work

- Splash screen page integration.
- Incorporating React to enhance the frontend of our application.
- Adding Forecast weather data widget to predictions feature. As JSON data is already available in the console for forecast data, an integration with predictive bikes and weather would benefit the user greatly.
- CO2 Emission Info?
- Incorporating Next Nearest Stations within 500 metres with availability. We had this feature almost working. However, a bug prevented us from using this feature to its full capability and as such we implemented a Next Nearest Station with availability.

8. Conclusion

In conclusion, the report detailed an overview of the functionality of our Dublin Bikes “Wheelie Good” application. Section 1 provided a summary overview of the functionality of the application. It also detailed the unique aspects and key unique features of our application and outlined the target audience.

Section 2 provided an outline of the architecture, the component interactions, and the layers of the application functionality (front-end and back-end). There were many layers and moving parts involved in the process and it was essential that they integrate well together.

Section 3 outlined the design elements of the application, detailing the functional features in our design. Also included was the quality-of-Life improvements, such as efficiency maximisation and alphabetically sorted dropdowns. Lastly, it provided initial mock-up designs and a brief outline of changes thereafter.

Section 4 provided a detailed breakdown of the data cleaning and machine learning model built into our application. It details the type of model use, the motivations behind this choice, and the inputs and outputs of the model framework. It also details the processes of cleaning the dataset and formatting the weather forecast and user inputs to provide as a model input.

Section 5 is a detailed description of the development process. It contains an overview of the work completed on the Trello board and how each of the meetings were organised for sprints (stand-ups, team meetings and formal meetings). Further, it details the product backlog items, how we planned for upcoming sprints, the burndown charts and how issues and resolutions were dealt with, within the team.