Dublin Bikes Report

Assignment 2

GROUP 15

EC2 address: http://100.26.169.199/

GitHub: https://github.com/pedro-morachacon/Comp30830GroupProject.git

TEAM MEMBERS AND CONTRIBUTIONS:

XIUPING XUE	22200549	33%
XUHUI AN	20211294	33%
PEDRO ANTONIO MORA CHACÓN	22203184	33%

COMP30830: Software Engineering (Conv) 2022/2023

UCD School of Computer Science

5 MAY 2023

Table of Contents

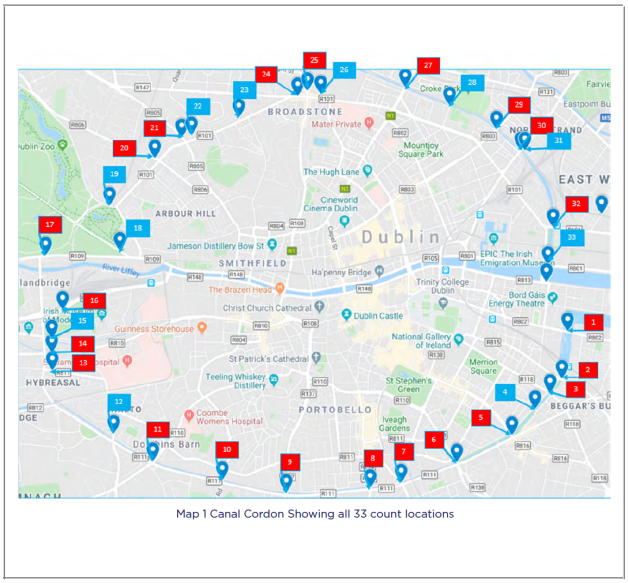
Project Overview

Introduction

Our project aims at providing additional features / services to the users of bicycle rental provider Dublinbikes. The use of bicycles is a convenient way to commute to the city where parking is often scarce and petrol prices are high. Providing additional features like future bicycle journey stations' bicycle / stands predictions and weather forecasts might be of interest to their clients considering the increasing city cyclist trends.

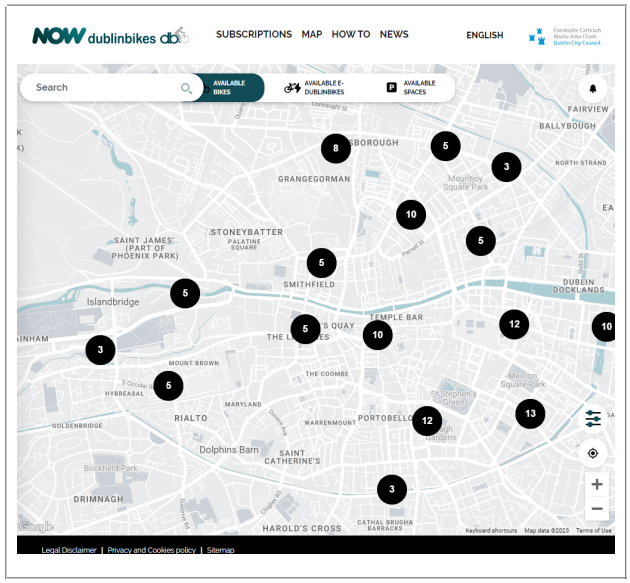
The annual National Transport Canal Gordon report is a detailed account of "traffic counts at 33 locations around the cordon formed by the Royal and Grand Canals. The counts are conducted during the month of November each year. Since 1997 the counts have been conducted over the AM peak period between 07:00 and 10:00." (The National Transport Authority, 2022) This annual report covers roughly the same area as the one covered by the bicycle rental provider Dublinbikes. See below.

Map 1 illustrates the Canal Cordon and the 33 locations on the Cordon (The National Transport Authority, 2022)



Map 1 Canal Cordon Showing all 33 count locations

Map 2 illustrates Dublinbikes coverage map. (JCDecaux Ireland Ltd., 2023)



Their 2021 Gordon report highlights that in Dublin, the use of bikes has been steadily increasing while the use of cars has been declining steadily except for the years impacted by COVID. (The National Transport Authority, 2022) During 2020 and portions of 2021 there were lockdown restrictions which had an immediate impact as only essential activities were permitted for the benefit of the general public's welfare. Even now in 2023, many employers continue to offer their personnel hybrid remote work models in an attempt to accommodate for lingering COVID circumstances.

According to the report, " There had been a steady year on year growth in the number of cyclists crossing the cordon since 2010 with the exception of a slight dip in 2018 until 2019. In 2021, a downward shift was observed with 7,597 cyclists crossing the cordon in the AM peak period. Even with these lower numbers in 2021, this still represents a significant growth of 57% when compared with 2006." (The National Transport Authority, 2022)

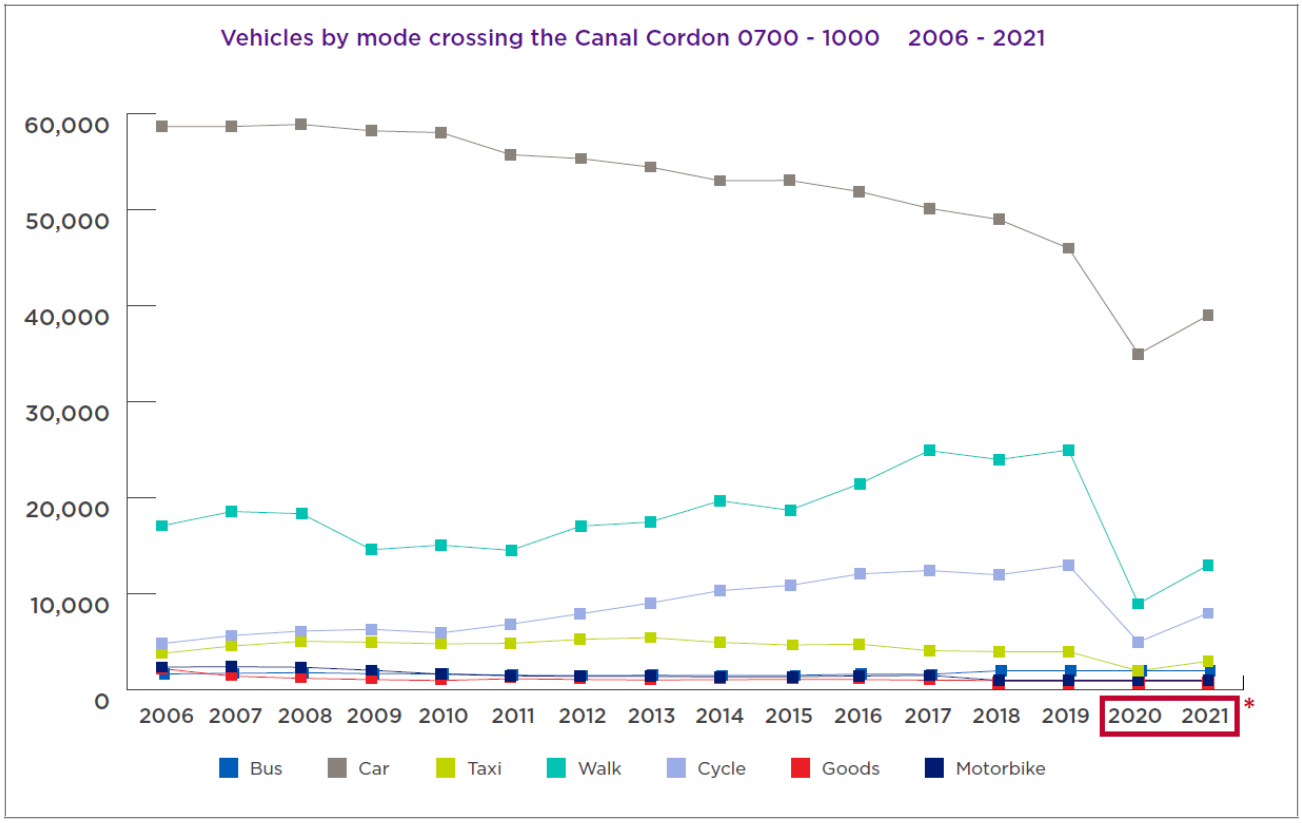


Fig 1 – Vehicles, cyclists and pedestrians crossing the Canal Cordon by mode of travel 2006-2021 (The National Transport Authority, 2022)

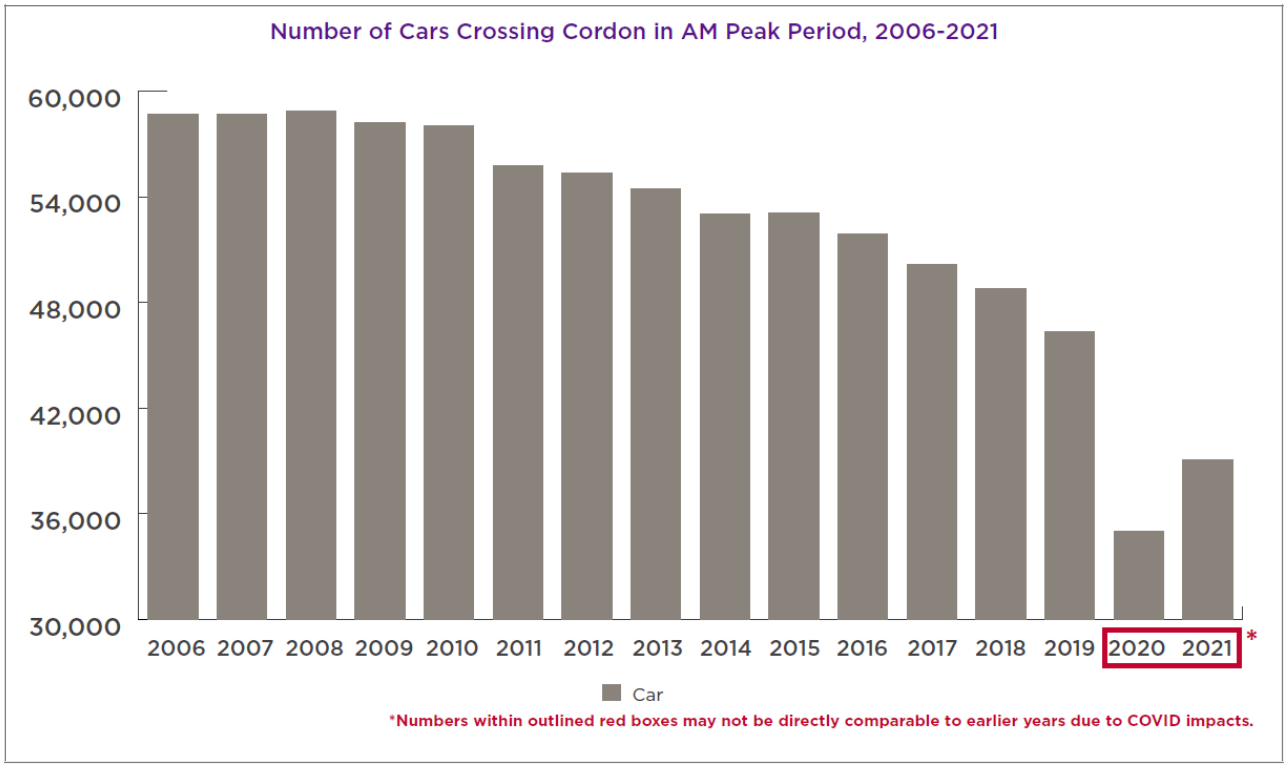


Fig 2 – Cars crossing the Canal Cordon by mode of travel 2006-2021 (The National Transport Authority, 2022)

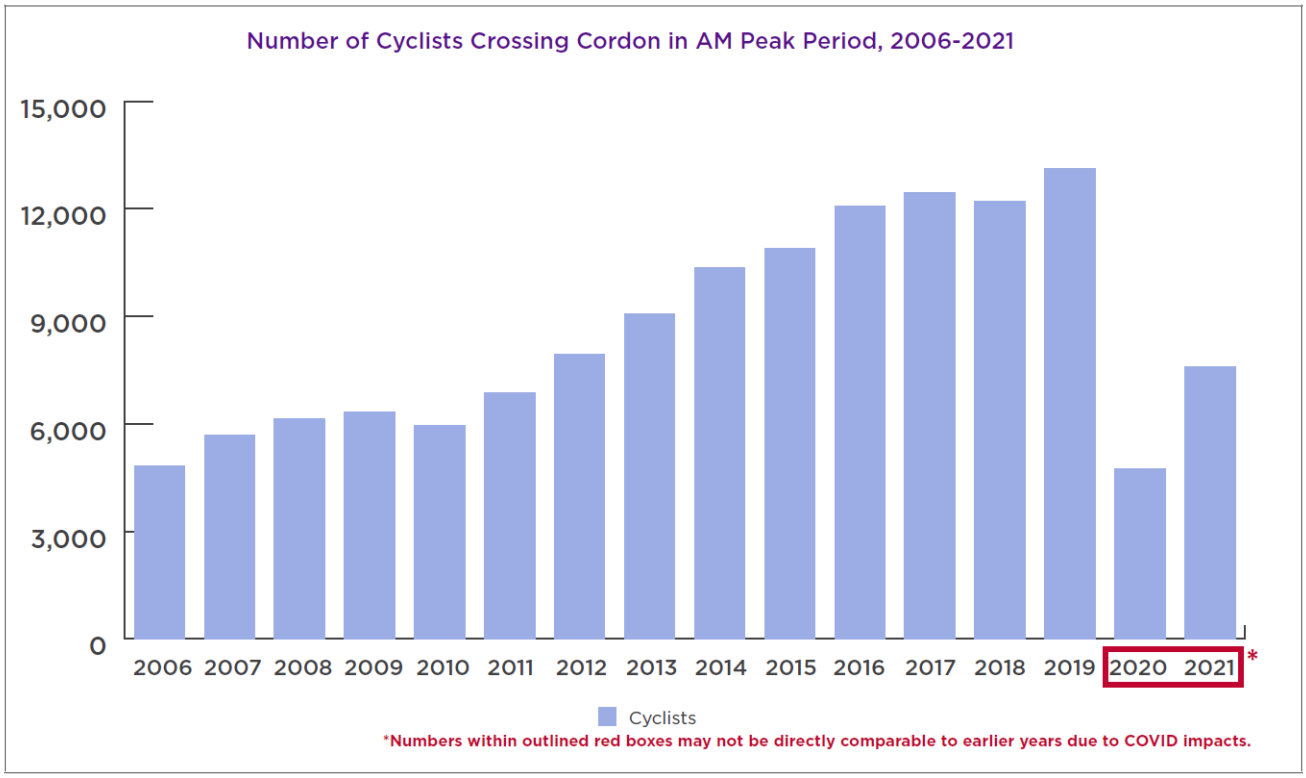


Fig 3 – Cyclists crossing the Canal Cordon by mode of travel 2006-2021 (The National Transport Authority, 2022)

These trends speak favorably to increase demand of bicycle rental services in general and the increase probability that at some point it may be valuable for their clients to be able plan their journeys ahead of time.

Target User:

The target audience for this app includes individuals who are looking for real-time and forecasted weather information for a specific location, as well as those who are interested in monitoring the availability of bicycles at bicycle stations based on weather conditions. This may include commuters, travelers, motorists, and other individuals who rely on weather-related information and each bicycle station's availabilities to plan their journeys effectively. The app may also cater to users who are concerned about weather-related impacts on bicycle availability at bicycle stations, such as during extreme weather events or seasonal variations. Additionally, the app may be useful for individuals who prefer to use shared or public transportation and need to know the availability of bicycles at specific bicycle stations for their travel plans. Overall, the app aims to provide valuable information to a wide range of users who are seeking accurate and timely weather and bicycle station availability updates to enhance their travel experience.

Features

1. Provide real-time and forecast information: The app aims to provide users with up-to-date and accurate information on the availability of bicycles and the number of bicycle stands at specific stations presently, or what would be based on travel time and weather forecast. This information allows users to plan their journeys more effectively and make informed decisions about their travel options.

2. Enhanced user experience: The app is designed to be user-friendly and intuitive, with a user-friendly interface that allows users to easily access the information they need. The app's layout, navigation, and features are designed to provide a seamless and convenient experience for users, making it easy for them to obtain the information they need quickly and efficiently compared to common map websites.

3. Improve travel planning: The app aims to assist users in planning their journeys by providing them with accurate travel time estimates based on real-time data. This allows users to better anticipate travel times and plan their trips accordingly, helping them to avoid delays and disruptions.

4. Increase awareness of weather conditions: The app aims to raise awareness among users about the current weather conditions in Dublin, providing them with real-time weather updates. This information allows users to plan their journeys more effectively, taking into consideration weather conditions such as rain, snow, or extreme temperatures.

5. The app offers users access to real-time and 5-day weather conditions, along with comprehensive status updates for each bicycle station as they are influenced by the local weather conditions. This includes valuable information such as the number of available bicycles and the number of bicycle stands (or bicycle parking spaces) that are currently available. By providing this detailed data, the app empowers users with up-to-date information to make informed decisions about their travel plans, taking into consideration the weather conditions and the availability of bicycles at their desired bicycle station. Whether it's checking the weather forecast for the next week or planning for available parking options, this feature-rich app ensures that users have the most relevant and timely information at their fingertips to make their bicycle rental experience more convenient and efficient.

Architecture

Overview

The Dublin Bike App project is a web platform application that uses the three-tier infrastructure architecture model. Within these presentation, logical and data tiers, the application’s software functional layers uses multiple development technologies that ultimately provide the user the App features mentioned earlier. In this section we will discuss in detail the different development technologies and infrastructure that make up the App’s architecture and how they interact with each other to achieve the app’s requirements.

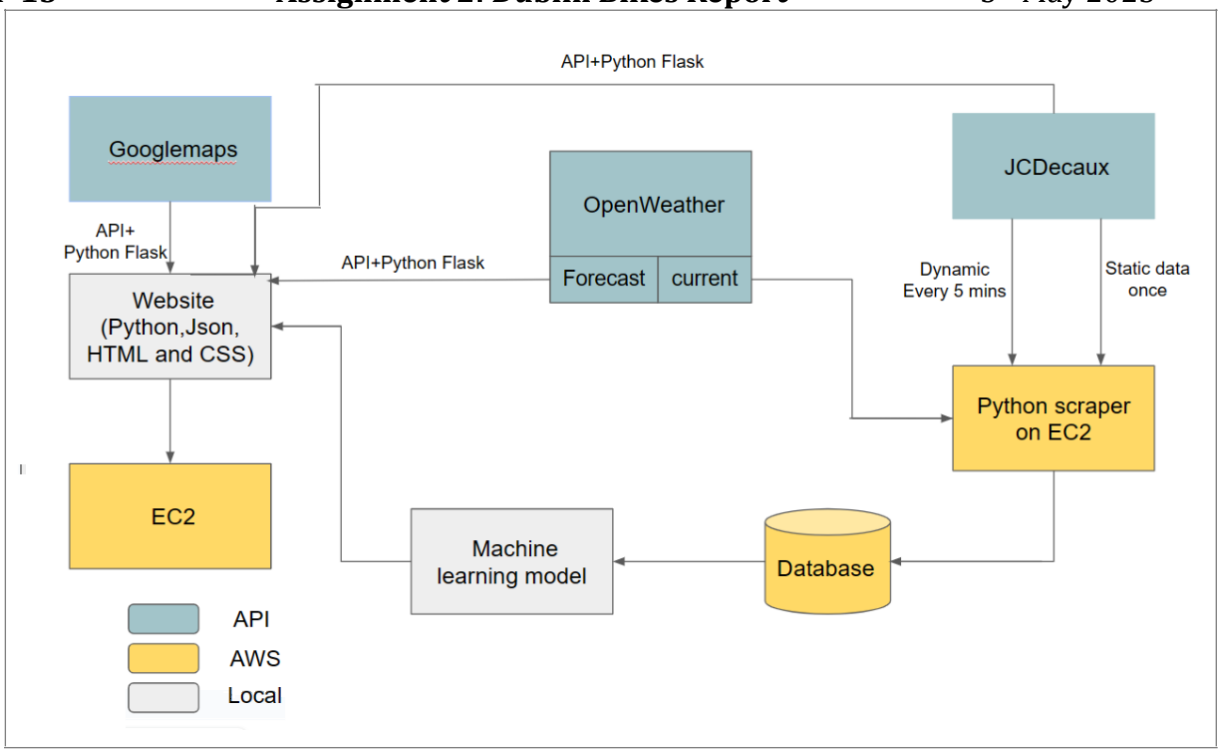


Fig 4 – High level view of Group 15's Dublin Bike App software layers architecture.

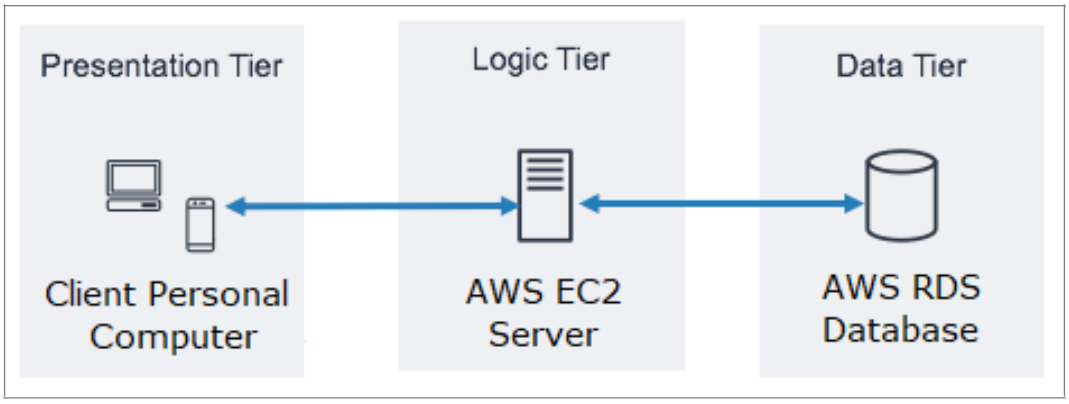


Fig 5 – High level view of Group 15's Dublin Bike App three tiers architecture infrastructure. (IBM, 2023)

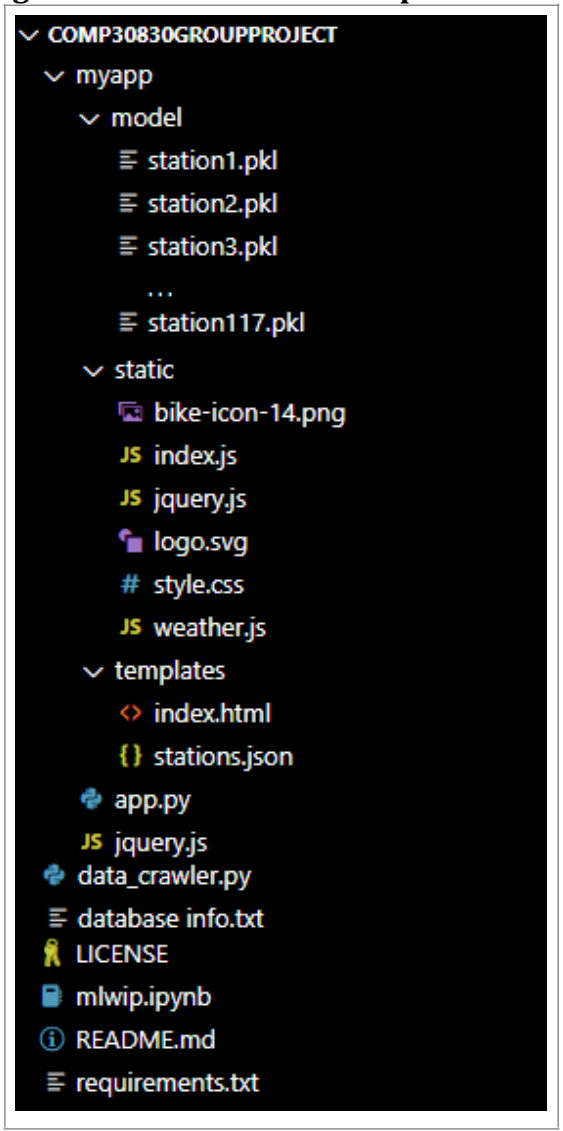


Fig 6 – Dublin Bike App project Server files structure tree.

Three tier architecture model

Briefly, the Dublin Bike App architecture at its lowest level utilizes the classic three tier infrastructure design as the model to run the application. There is a presentation tier that is in charge of processing the user interface function of the app and it is hosted by the client's personal computer. There is a Database tier that is in charge of Data storage. In particular, the collected API's data and this tier is hosted by an AWS RDS cloud instance. Finally, there is the Logic tier that is in charge of handling API Scraping, the machine learning function, the Client/ user interface requests and the responses.

Functional implementation and corresponding technologies

This part will give a more specific and detailed description of the technologies involved in creating the application's functional layers.

User interface layer technologies

The Dublin Bike application provides a dynamic user experience by combining Google Maps' comprehensive set of features, and JavaScript logic to load on the HTML5/CSS web canvas the latest weather and bike stations' data, load the data in the map, listen for user input, forward these to the Server, and update the canvas with the responses. The current weather, the current

bicycle station markers, and the Heatmap showcasing bicycle station current bike availability status are dynamic features that are independent of user actions whereas the dark/light modes, the daily/hourly weather five day forecasts and the date/bicycle station input related charts are triggered only by the user. All of these client-server requests and responses utilize JSON files carrying the appropriate parameters needed for each particular case.

Data collection and storage technologies

The Dublin Bike application, in order to provide the predictions feature, required collecting data to be used for training a predictive machine learning model. This training data information was required to be set up as early as possible since the JCDecaux API for the bicycle rentals does not have any historical data available. This prompted the early setup of both an AWS EC2 and an AWS RDS set of instances. The Dublin Bike application's RDS instance is a distributed MySQL DBMS to store and manage the training data. The EC2 cloud computing instance serves as a server for which we installed Ubuntu as the operating system, the Miniconda/Python IDE to create the program to start collecting the data from the JCDecaux API and the weather from Openweather's API. The program was written to collect the data from both APIs every five minutes, all day, every day.

Web app framework and risk management

The Dublin Bike's EC2 server manages the logic aspect of the application and uses Python's Flask to serve the logic in a web framework. The server manages various client tier requests from JavaScript on the User interface, gets or calculates the answer and forwards the response back to the User interface. Requests relating to current or past information are sent to RDS using SQLAlchemy's database engine connections, manipulates the data and forwards the response back to the User interface. Similarly, requests relating to predictions require the server to make an API request for the weather forecast first, manipulates the weather API data to combine it with the other prediction arguments (bicycle station number, time of day and day of week), gets the corresponding bicycle station prediction model, runs the model, manipulates the model results and forwards it back to the user interface.

To assist managing website reliability and security Apache2 was installed as a proxy server. We did Apache over Nginx because it is among the most popular and we installed it on our app before the lecture and slides on Nginx happened. We thought it would be good experience to have, it adds security against attacks and helps reliability by managing the server workloads.

Tests

Below are test result pictures from a brief test in google lighthouse

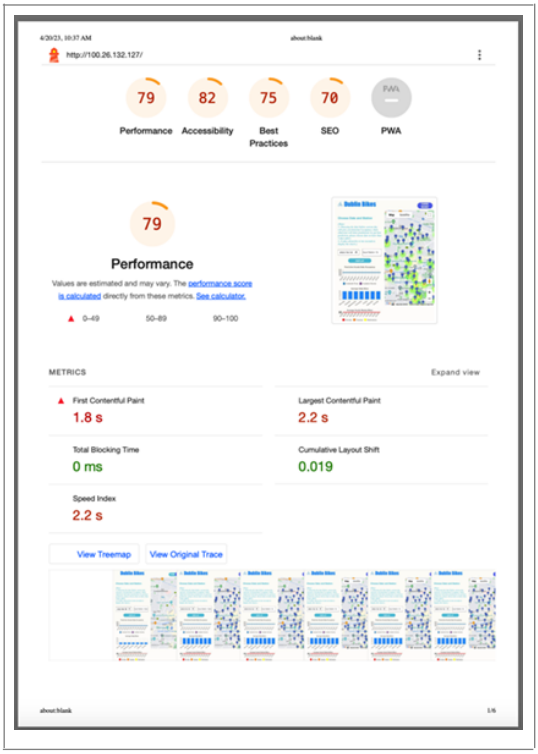


Fig 7 Test results in google lighthouse

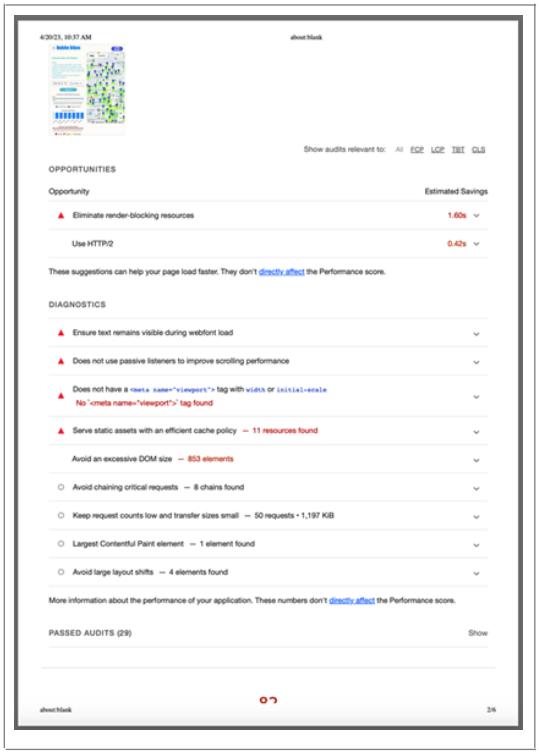


Fig 8 Test results in google lighthouse

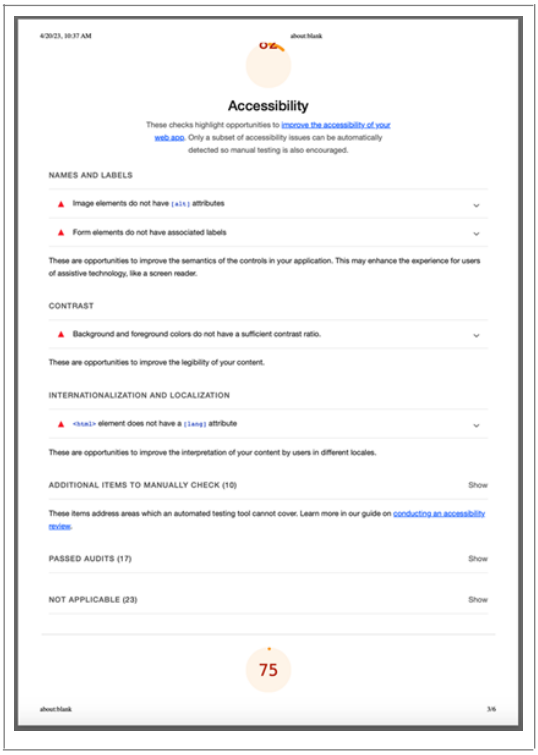


Fig 9 Test results in google lighthouse

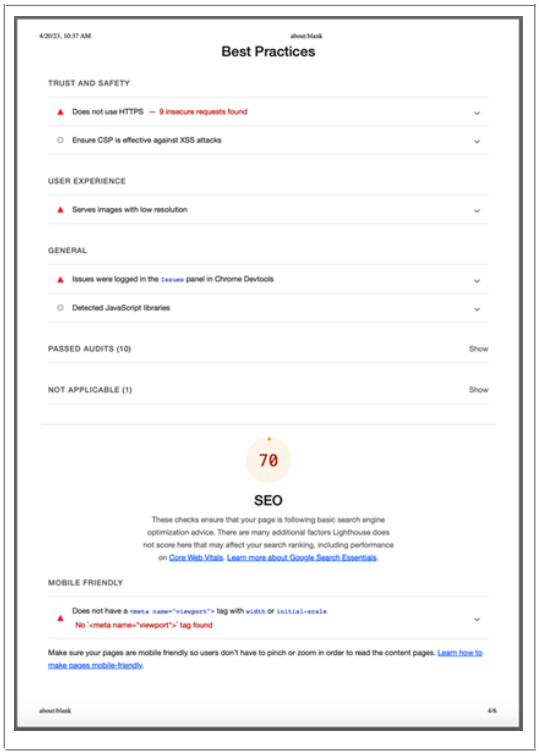


Fig 10 Test results in google lighthouse

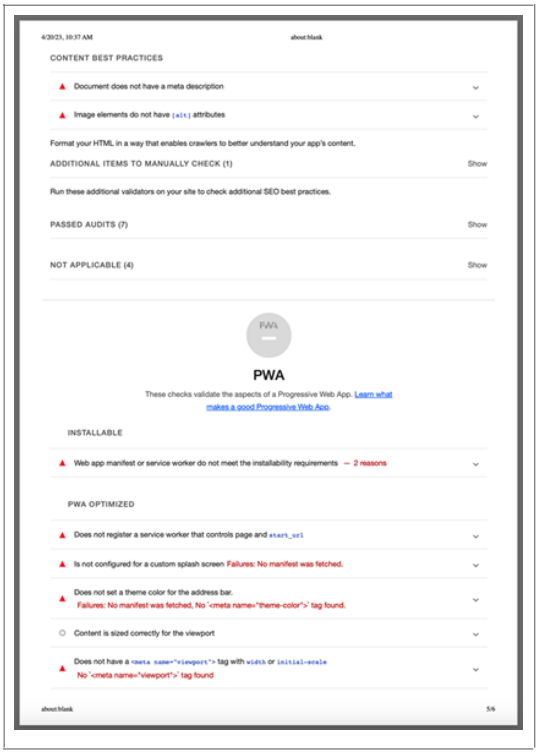


Fig 11 Test results in google lighthouse

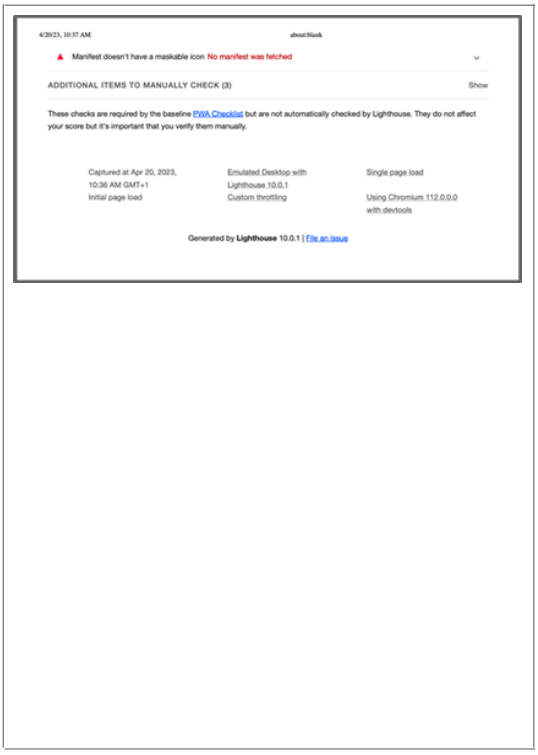


Fig 12 Test results in google lighthouse

User Interface Design

Early Mockups

Here are two sketches of the design of the web page.

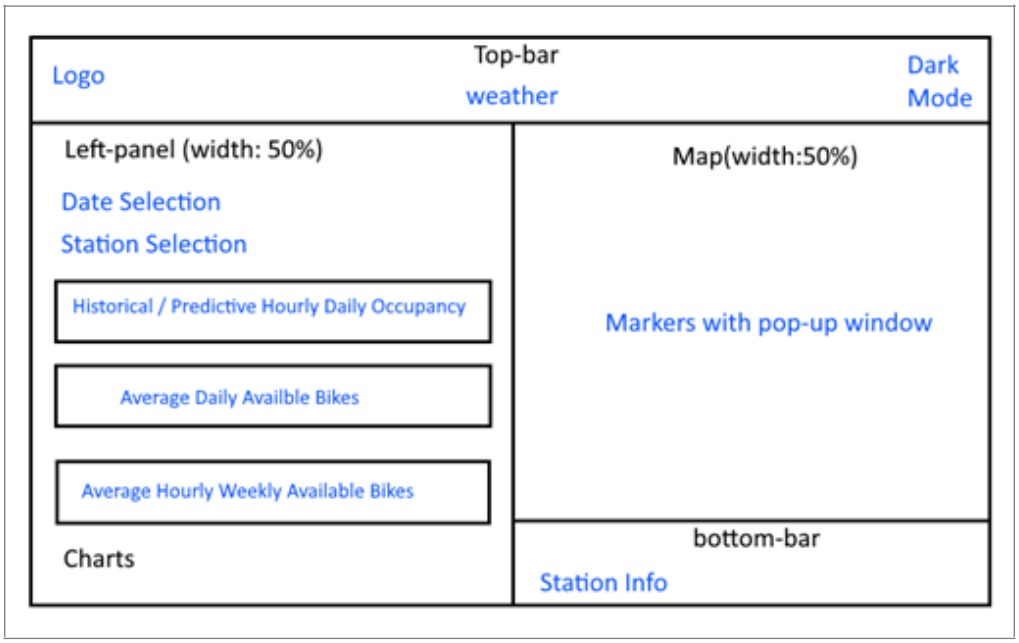


Fig 13 Frontend design-1.

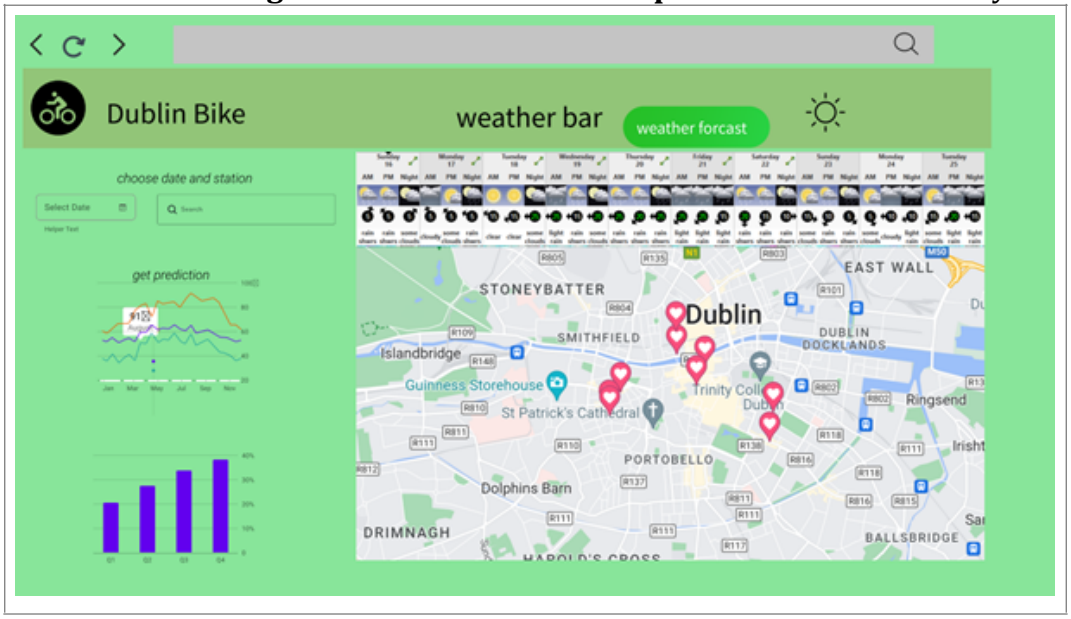


Fig 14 Frontend design-2.

User Flowchart

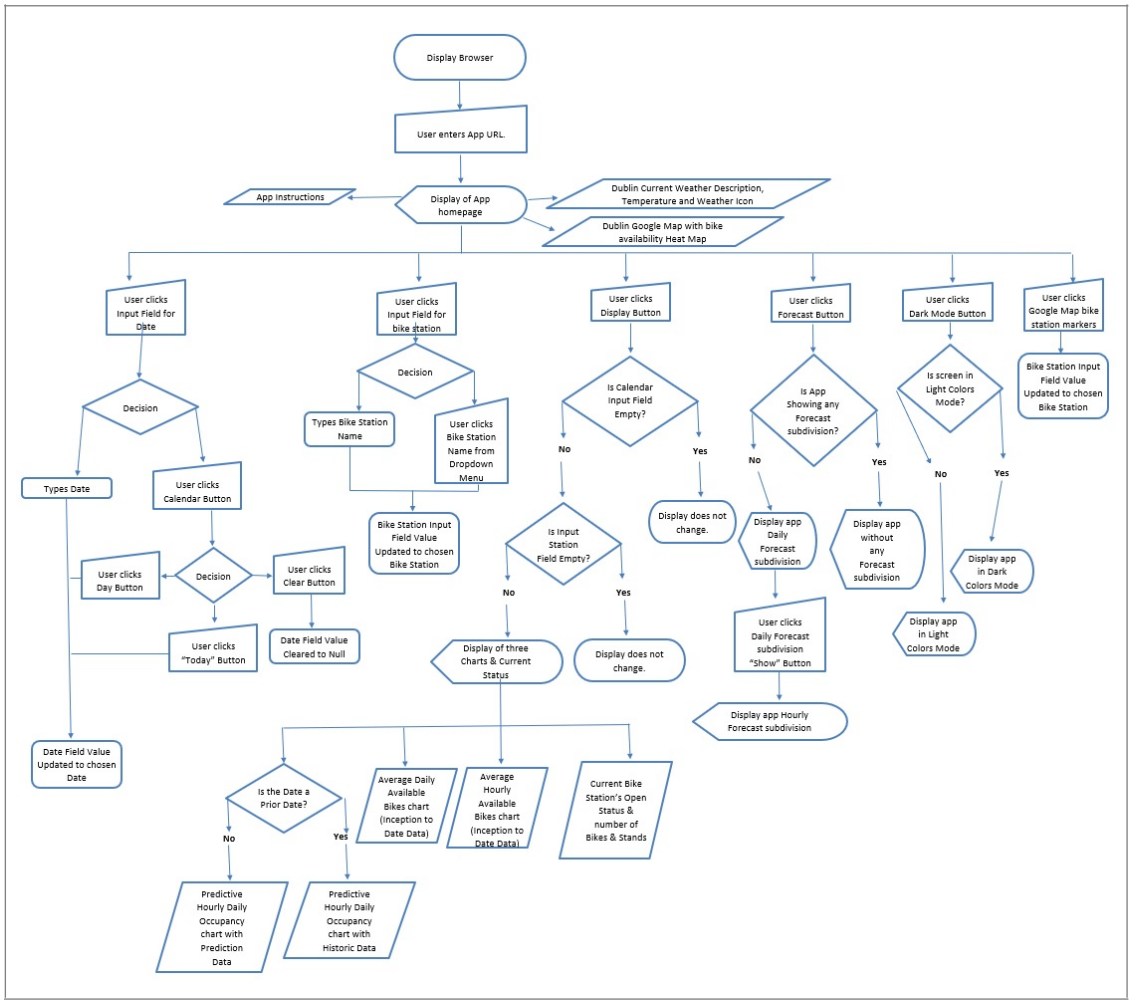


Fig 15 Bikes Dublin App User Flowchart diagram.

In the preceding Bikes Dublin App project user flowchart, the layout of the user interface is shown for reference. The application is intended to be simple to use and efficient for the user to be able to make their decisions quickly.

Front End

The website is divided into two main sections: the left and right portions.

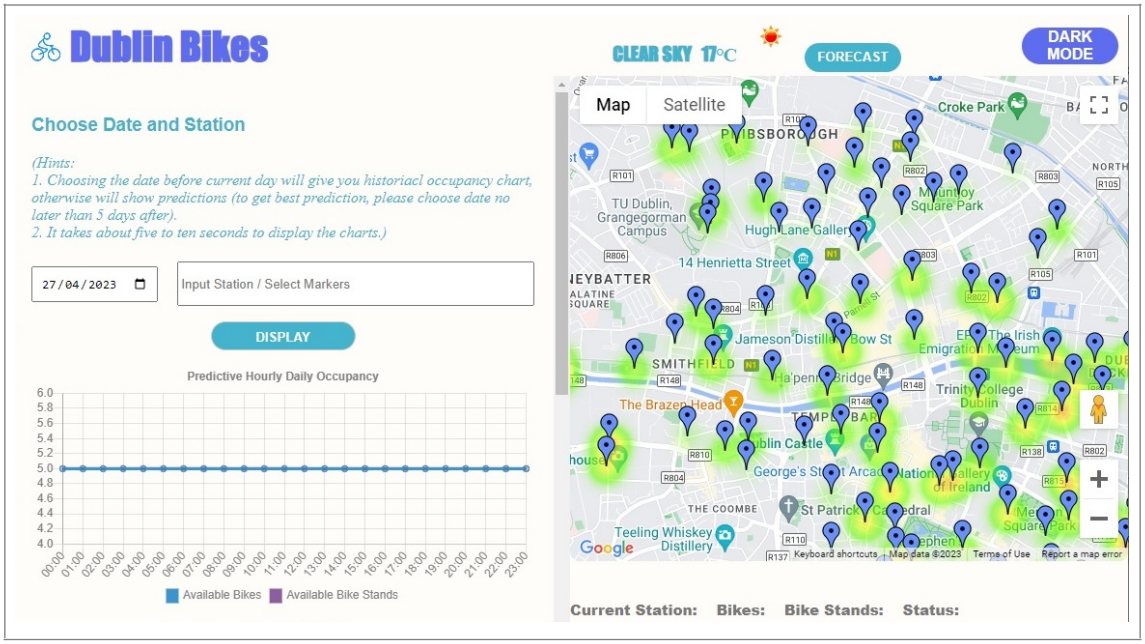


Fig 16 The left portion has the user input and the bicycle and stands charts. The right portion has the Map information.

The left panel of the app provides information to the user based on their inputs. Here, users can choose their desired station by picking from a dropdown menu or by simply clicking on a bicycle station on the map. For the Date of their journey, they may use today's date which is already filled in, type a different date, or choose a date from the calendar pop up button.

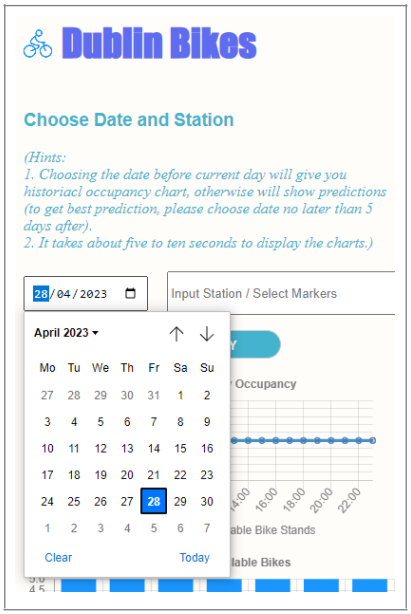


Fig 17 The calendar pop up button.

After receiving the user input, the website generates three charts to provide the users with relevant bicycle station occupancy information. The charts include the average hourly availability of bikes throughout the week, the daily availability of bikes, and the predictive occupancy of the chosen station. See Below.

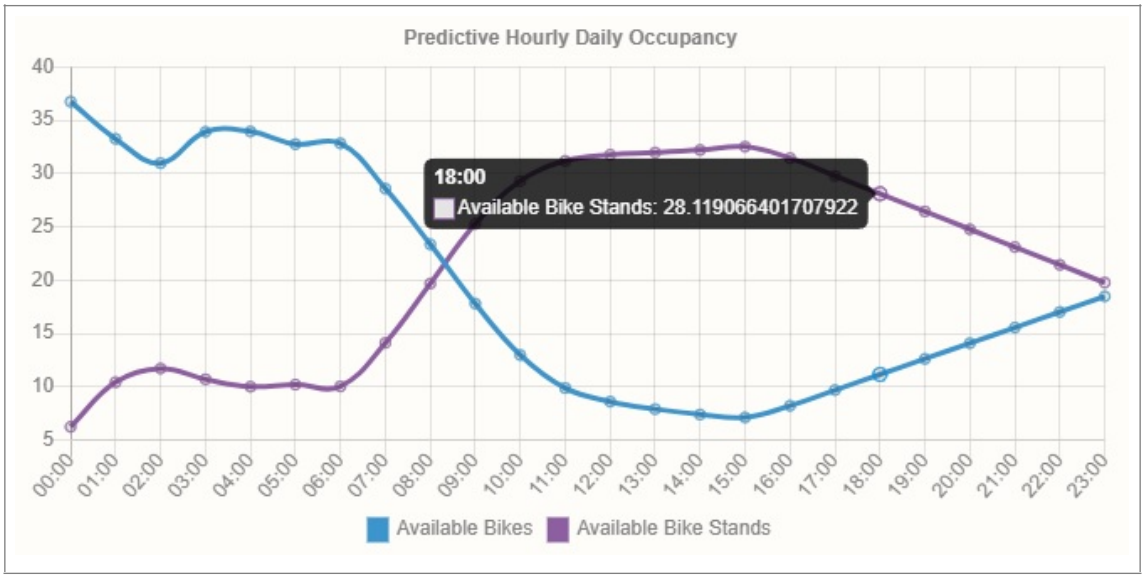


Fig 18 The Predictive Hourly Daily (Bikes and Stands) Occupancy chart.

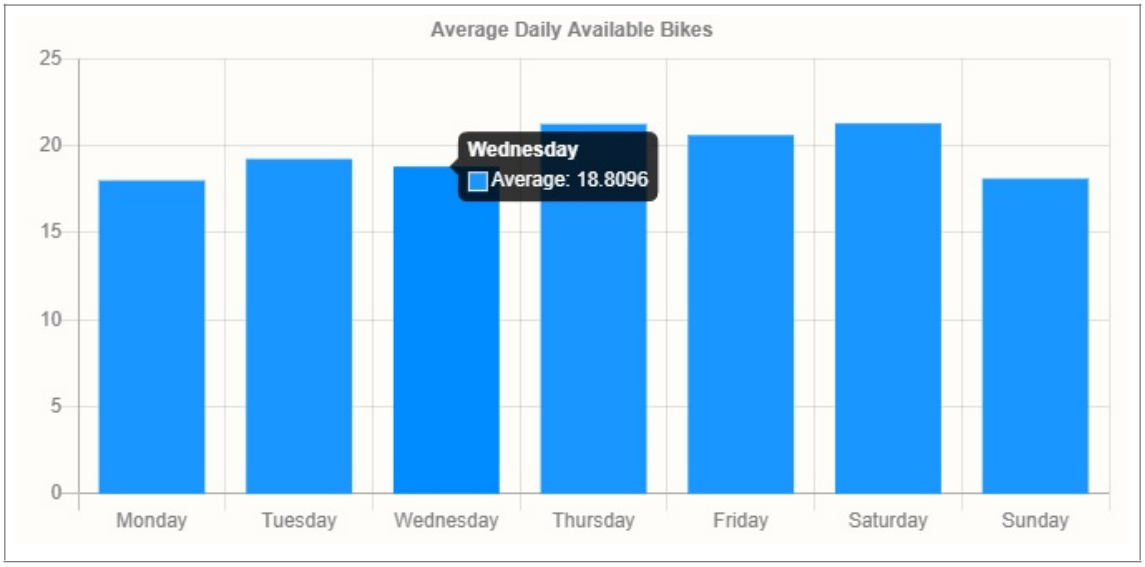


Fig 19 The Average Daily Available Bikes chart.

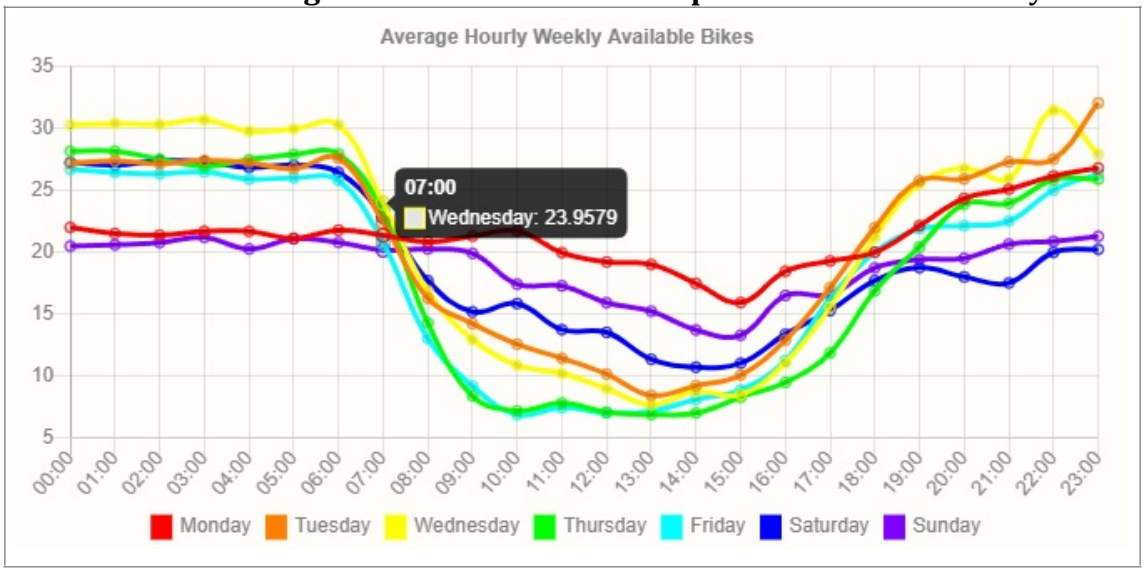
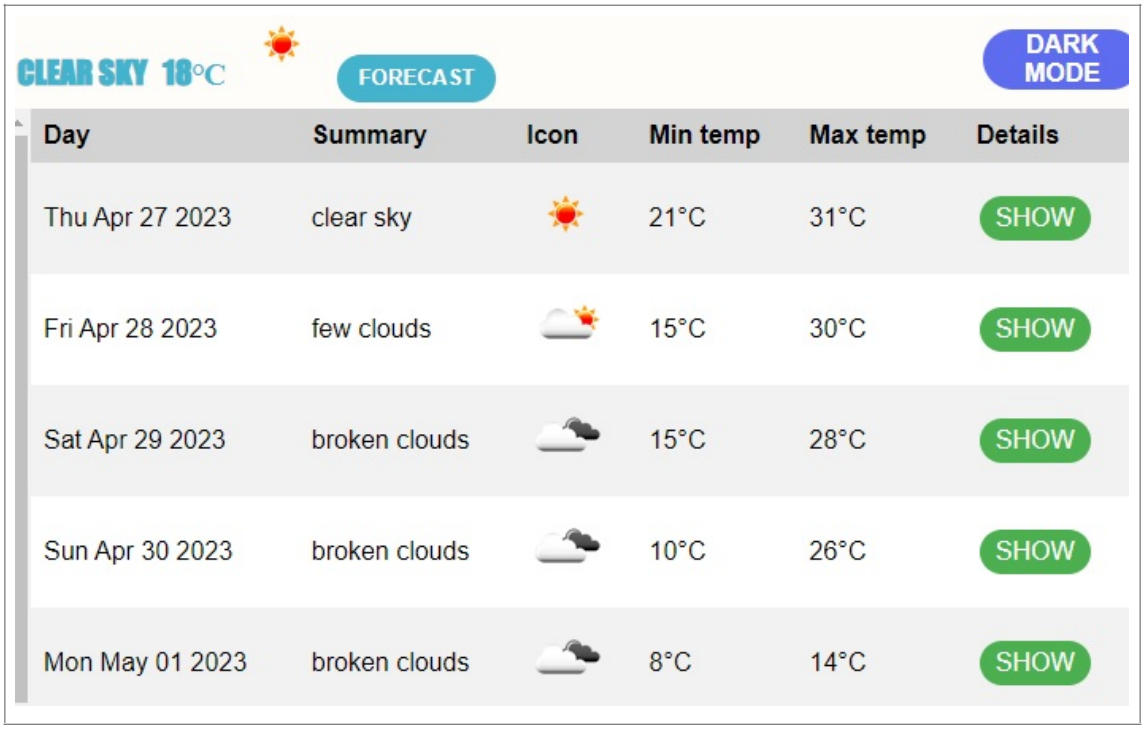


Fig 20 The Average Hourly Available Bikes chart.

These charts offer users a comprehensive view of the station's bike availability, hopefully helping them make informed decisions about the timing and starting point of their journey. On the right-hand side of the website, users can access the map and weather information. The top section displays the current weather, including the temperature and cloud coverage, depicted using an icon.

Fig 21 The current weather display.

Additionally, the website provides a forecast section, where users can access the weather forecast for the next five days. The forecast section presents the weather data in a chart format, displaying it row-wise. Users can click on the 'show' button at the end of every row to see the weather and details for every three hours of the chosen day.



Day	Summary	Icon	Min temp	Max temp	Details
Thu Apr 27 2023	clear sky		21°C	31°C	SHOW
Fri Apr 28 2023	few clouds		15°C	30°C	SHOW
Sat Apr 29 2023	broken clouds		15°C	28°C	SHOW
Sun Apr 30 2023	broken clouds		10°C	26°C	SHOW
Mon May 01 2023	broken clouds		8°C	14°C	SHOW

Fig 22 The weather forecast daily section.

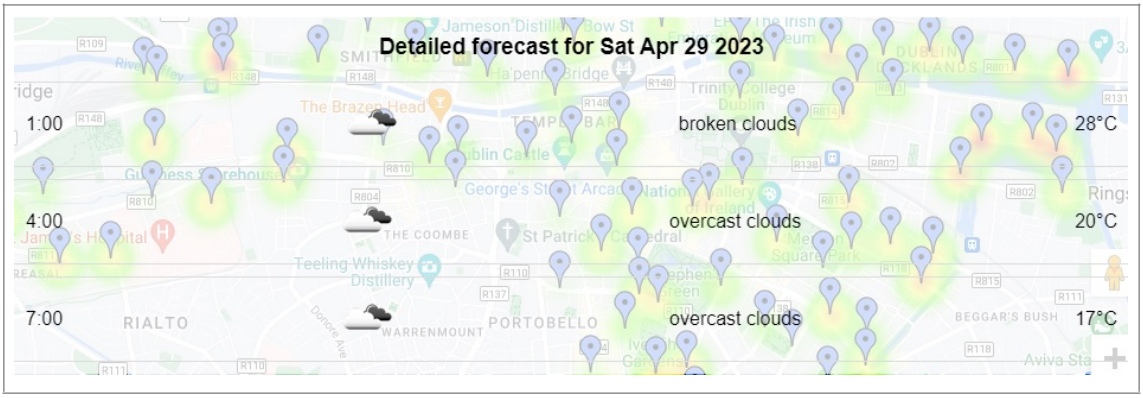


Fig 23 The weather forecast hourly section.

The website also has a 'DARK MODE' button that allows users to switch to a different website mode, enhancing their viewing experience. This useful feature makes the application comfortable to the user's eyes at all times of the day or night by providing a bright screen option in the sun and a dim screen option at night when it is not needed.

Fig 24 The Dark mode version.

Below the weather bar is a map that displays the blue icons representing every station. The website uses a heat map to show the number of available bikes in each station, making it easy for users to identify the stations with the highest availability.

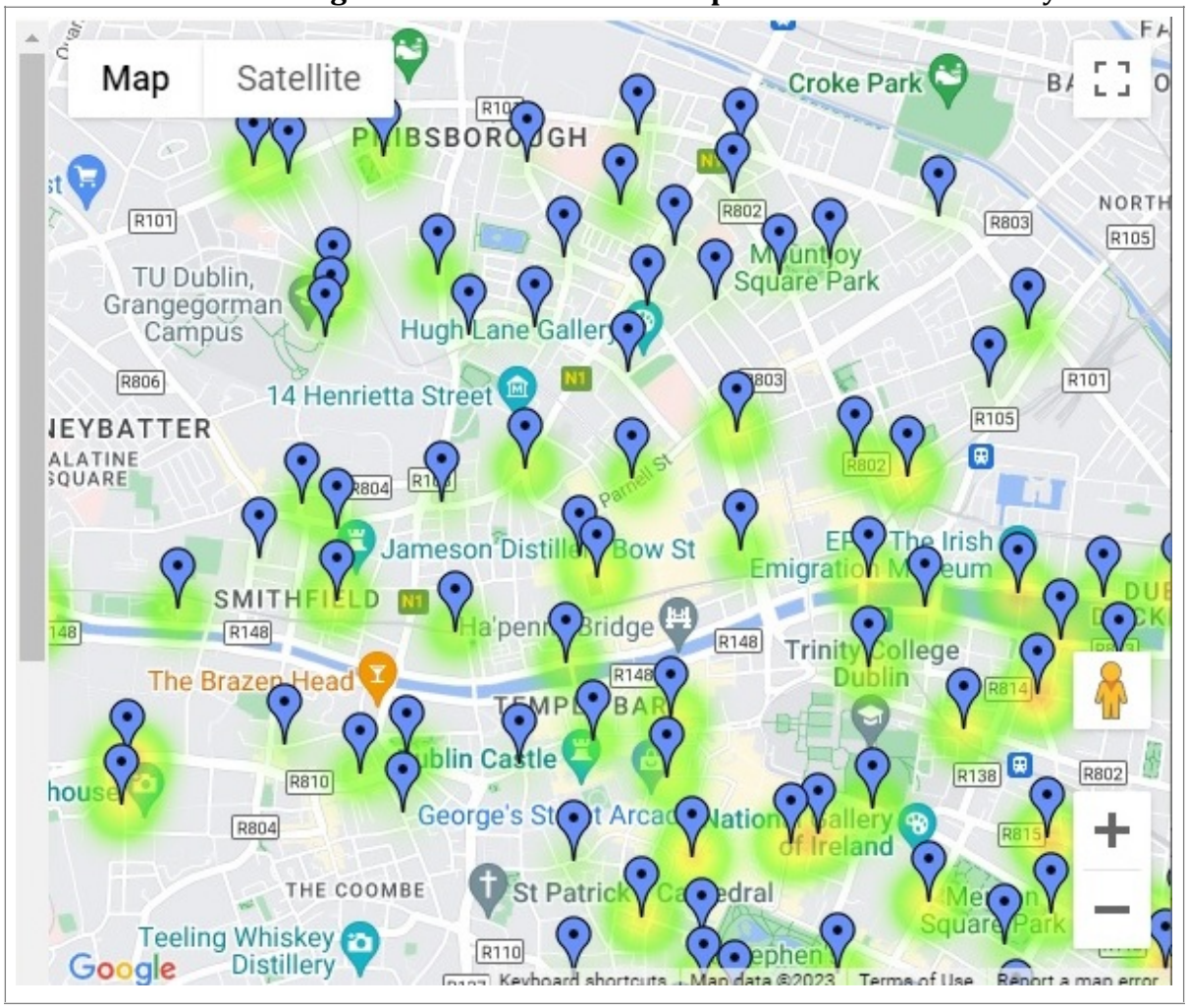


Fig 25 Illustration of the heat map. Colors symbolize availability and go from green for low availability to red for high availability.

The combination of these features provides users with a comprehensive view of the bike sharing situation in the area, making it easier for them to plan their journeys efficiently.

Data Analytics Element

Dublin Bikes App prediction feature operational overview.

The Dublin Bikes App prediction feature utilizes trained models for each of the bicycle stations to predict availability of bicycles and bicycle stands at some time in the next five days. The user triggers the prediction feature automatically when planning for a bicycle trip where it is not on a date in the past. JavaScript will handle the event in the Client's user interface and send the request for the necessary information to provide the user a "Predictive Hourly Daily Occupancy chart" showcasing the prediction results. The request is sent to the Logic Server's Flask web framework to route it to the functions that will retrieve the appropriate weather information from OpenWeather's API, the specific bicycle station machine learning model pickle file, process the prediction with the relevant arguments and send back the results.

It should be noted that the two other Average Daily Available Bikes and the Average Hourly Available Bikes charts are based on inception to date averages generated with MySQL queries and not the bicycle station's predictive model. These two charts are provided as an additional feature that allows the user to make a quick comparison with other days of the week's average occupancy on a daily and hourly basis in case they may want to try other bicycle trip parameters. These two charts do not involve the bicycle station's machine learning prediction model in any form.

Data Collected

To create the prediction model for the Dublin Bikes App project, we used a Python scraper on the AWS EC2 Server to automatically collect data every five minutes from JCDecaux's API (for the bicycle stations data details) and from OpenWeather's API (for the current Dublin weather data). The Python scraper set up a MySQL database in the AWS RDS instance, created tables for bicycles and stands availability, bicycle stations and for the current weather.

The data from JCDecaux's API we collected was split into two tables. The availability table would get the station data related to the current bicycle situation (like current timestamp, station number, number of available bikes, number of available bike stands). The API data was also checked to see if it contained new bicycle stations and when so, it updates the station table with any new station's attributes (like station number, location coordinates, name, street address, number of bike stands, open status).

The data from OpenWeather's current weather API we collected for the city of Dublin was placed into the weather table. This included the current time, temperature, humidity, wind speed, wind direction and a few categorical descriptions of precipitation, and visibility.

The decision was made to not preemptively collect forecast weather data as it was not viewed efficient. The assumption was made that the bulk of the prediction requests would likely come at certain times of the day. The collecting of that data 24 hours a day as done for the other APIs was seen as wasteful of processing and memory storage resources, that would give no speed benefit in not having to connect to the API server as it would still need to send requests to the database.

Model Building

To ensure that the data used for the machine learning model to train on was of good quality the following reviews and procedures were performed.

The collected data in the AWS RDS Database infrastructure was brought in to a Jupyter Notebook by using SQLAlchemy's create_engine function to manage the database connection / query. The database query request itself performed a few of the data manipulation tasks that needed to be completed. As discussed earlier, the database holds the information needed in two tables but to be able to use it we needed them in one. We also needed them to be able to connect the weather data rows somehow to the station data of bicycle and stands station availability history rows in a way that was correct like the timestamps. Both APIs supplied them and were included in the two tables. That said the timestamps did not match each other so in order to get to join both tables we needed to manipulate the rows timestamp fields as were brought in to the Jupyter Notebook to make it work. Since both APIs were collected every five minutes, both tables set of timestamps were for a frequency equal to that. The query was then used to both do the tables join and the timestamp manipulation needed for it by rounding

the timestamps down to get them to 5 minute even intervals. For example, from 11:03am and 11:04am to both being 11:00am (WHERE (dbikes.availability.last_update DIV 300 * 300) = (dbikes.weather.timestamp DIV 300 * 300).

The query results were placed in a Pandas dataframe to simplify the data analysis and manipulation before using as machine learning training data. The training features being Station Number, Time, Bikes Available, Bike Stands Available, Temperature, Humidity, and Wind. The numerical features joined from both tables. Station Number, Time, Bikes Available, Bike Stands Available, Temp, Humidity, Wind. The Time feature was actually the rounded down timestamp, not the time of day. Testing for row duplicates is easy with Pandas dataframes and as expected, there were no duplicates in the dataframe.

The timestamp besides being useful to join the tables, we also used them to create two new features. By using functions from Python's own datetime library module, we added a feature column for time of day and another for day of week where Friday, Saturday, Sunday had their own separate value and Monday through Thursday were lumped into a weekday value. The assumption was taken that Friday's early evening hours, as the start of the weekend, would differ greatly from the rest of the weekdays.

With the features available using the dataframe's methods we set the training targets to be the 'BikesAvailable' and 'BikeStandsAvailable' columns. The rest were set as training variables. From there we used scikit-learn's train_test_split algorithm to select random training and testing segments, construct regression models and run the training segments to develop the machine learning model.

Analysis (only Data Analytics))

Bikes Available **Linear Regression Model**

```python
# Linear Regression Model target


target = 'BikesAvailable'

#Setting Model's target data
y = data[target]

#Setting Model's variables data
x = data.drop(target,axis=1)

#Splitting target data and variables data attributes into train and test tuples
x_train, x_test, y_train, y_test = train_test_split(x, y)

#Construct Linear Regression Model
linModel = LinearRegression()

#Run the Model with train tuples
linModel.fit(x_train, y_train)

# Print Model Scores
print('\n' + '*' * 40)
print('Train Score:',linModel.score(x_train, y_train))
print('Test Score:',linModel.score(x_test, y_test))
```

Python

```
****************************************
Train Score: 0.9707121928185344
Test Score: 0.9699247300622096
```

Fig 26 Testing one of the prediction models.

While evaluating the categorical attributes, we tried setting them to dummy binary columns with Panda's get_dummies() method to see how they performed with the linear regression. As can be seen above they did too well, with over fitting the data. None the less, after thinking about the infinite possibilities of weather descriptions, the decision was made to pursue the continuous categories to avoid app prediction failures.

Development Process

Sprint 1 Review

Course of action and decisions

During the project's first sprint an Amazon RDS instance was created to store a MySQL database. This database would be used by the API scrapers set up in the EC2 instance that was created during our first course assignment. Using Python a web scraper was set up to pull the bicycle stations information from the JCDecaux API and the weather information from the OpenWeather API. The decision was made that these API requests would be sent every 5 minutes in order to have detailed information about the timing of bicycle stations' events. For example, at the end of the business workday, when would the bikes be rented or when would stands become available. The information was loaded into the RDS database to use later in the analytics portion of the project.

It should be mentioned that there were two instances scraping the weather and JCDecaux APIs for a limited amount of time which resulted in duplicate entries to the database tables. This was addressed through formulas in queries to make sure that the data analytics was not affected. We looked at Mapbox.com to use for our website map and were following a tutorial on how to create a simple map using the leaflet JavaScript library. (Halford, 2023)

By the end of the sprint, we finished the Mapbox /leaflet tutorial and added some features of the Dublin Bikes application only to find out in a private conversation with the professor that there were good reasons to do the Dublin Bikes App using Google instead of the Mapbox / leaflet version. We would be discussing in class Google Maps, charts and other features that might prove challenging for us on our own without that guidance. So, on the second week of the second Sprint we as a group decided that before it was any later, to drop Mapbox /leaflet version and switch over to Google Maps.

Documentation

```
+ # Define a function to create a new availability table every week
+ def create_availability_table():
+         table_name = "availability"
+         sql = """
+         CREATE TABLE IF NOT EXISTS {} (
+         number INTEGER,
+         available_bikes INTEGER,
+         available_bike_stands INTEGER,
+         last_update BIGINT
+         )
+         """.format(table_name)
+         try:
+                 res = engine.execute(sql)
+                 print (res.fetchall())
+         except Exception as e:
+                 print(e) #, traceback.format_exc())
+
+         return table_name
+
+ # Create the first availability table
+ create_availability_table()
```

Fig 27 Creates the availability table

```
+ def create_weather_table():
+         # Create the weather table
+         sql = """
+         CREATE TABLE IF NOT EXISTS weather (
+         id INTEGER,
+         timestamp BIGINT,
+         temperature REAL,
+         humidity INTEGER,
+         wind_speed REAL,
+         wind_deg REAL,
+         clouds INTEGER,
+         main VARCHAR(256),
+         description VARCHAR(256),
+         visibility REAL,
+         rain REAL,
+         snow REAL
+         )
+         """
+         try:
+                 res = engine.execute(sql)
+                 print(res.fetchall())
+         except Exception as e:
+                 print(e)
+
+ create_weather_table()
```

Fig 28 Creates the weather table

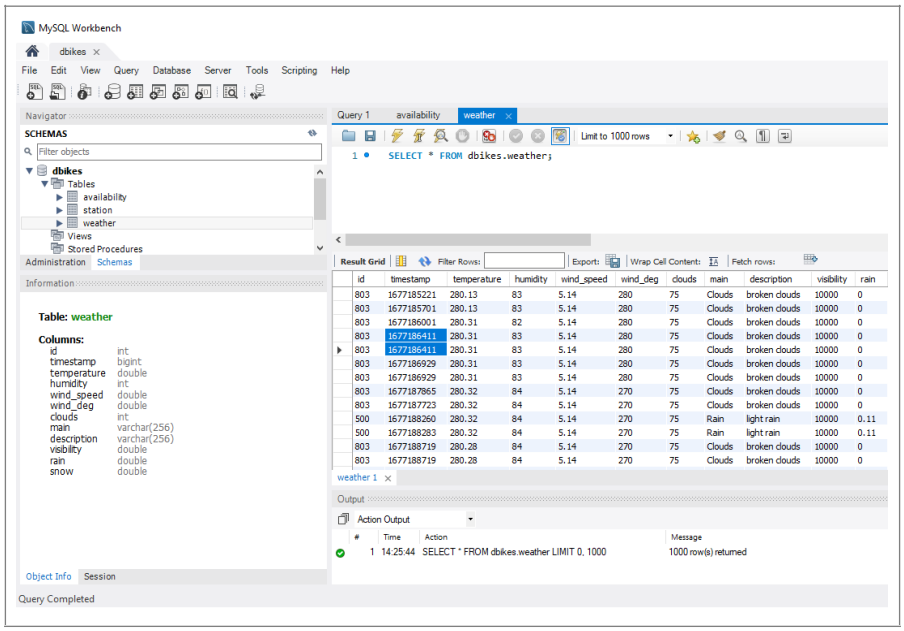


Fig 29 Table shows duplicate database entries. See timestamp value duplicates

Early Version (Non-Google Maps)

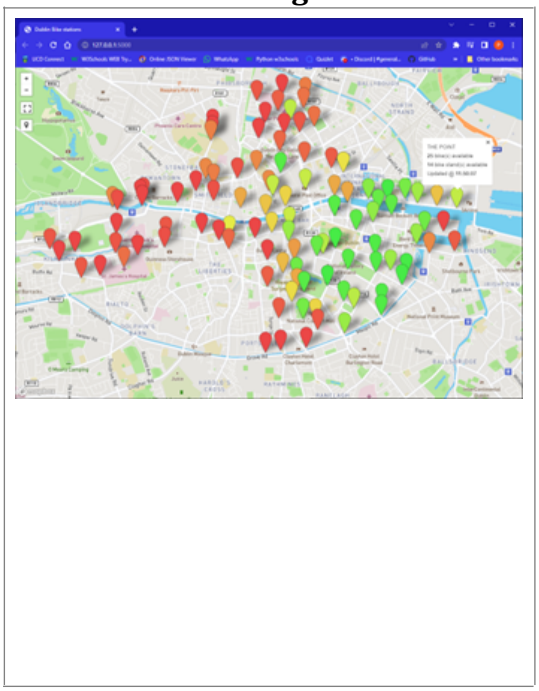


Fig 30 Group 15's Original Dublin Bike App Design using Mapbox.com
and the Leaflet library.

```
+            function getColor(value){
+                var hue=((1-value)*120).toString(10);
+                return ["hsl(",hue,",80%,60%)"].join("");
+            }
+            L.mapbox.accessToken = 'pk.eyJ1IjoiLS1wZWRyby0tIiwiYSI6ImNsZWxqNXhpaT82dmozcW5weTc0aTR1emgifQ.70YKx1ajTb17myG37fYGWQ';
+            var map = L.mapbox.map('map', 'mapbox.streets', {maxZoom: 20})
+            // initial location
+            map.setView([53.344, -6.2672], 12);
+            // add plugins
+            L.control.fullscreen().addTo(map);
+            L.control.locate().addTo(map);
+            omnivore.csv("{{ url_for('static', filename='data/dublin.csv') }}", null, L.mapbox.featureLayer()).on('ready', function(layer) {
+                this.eachLayer(function(marker) {
+                    var info = marker.toGeoJSON().properties
+                    var bikes = info.available_bikes
+                    var stands = info.available_bike_stands
```

Fig 31 Group 15's Original Dublin Bike App Map Markers using Mapbox.com

First Meeting

How are you feeling today?
An: Good
Pedro: Good
Xiuping: Good too

What are you going to do from now to the next meeting?
An: analysis requirement of this application
Pedro: Look into the API's we need to collect data from.
Xiuping: Understand the goal of the project

Is anything blocking your work?
An: No
Pedro: No
Xiuping: No

Fig 32 Group 15's 1st Sprint 1st Standup Meeting.

Second Meeting

How are you feeling today?
An: Good
Pedro: Good
Xiuping: Good too

What did you do from the last meeting to now?
An: Create AWS and RDS account and EC2 instance
Pedro: Created account for Mapbox and OpenWeather API Services
Xiuping: Learn how to connect a database using Python

What are you going to do from now to the next meeting?
An: Work on fetch data from API
Pedro: Look into creating the RDS instance and how to collect data there.
Xiuping: Create an EC2 instance and RDS instance. Write data crawlers.

Is anything blocking your work?
An: No
Pedro: No
Xiuping: No

Fig 33 Group 15's 1st Sprint 2nd Standup Meeting

Standup Meetings

When we were in the practical we would do them in person, but outside those times, our standup meetings were mostly done through electronic communications for the following reasons. English is a second language for all of us and we found communication much easier that way. No need to feel self-conscious of how you're pronouncing words, and the message is not getting misinterpreted.

The feedback from our product owner was mostly that we were on a good place and on track with the project.

Sprint 2 Review

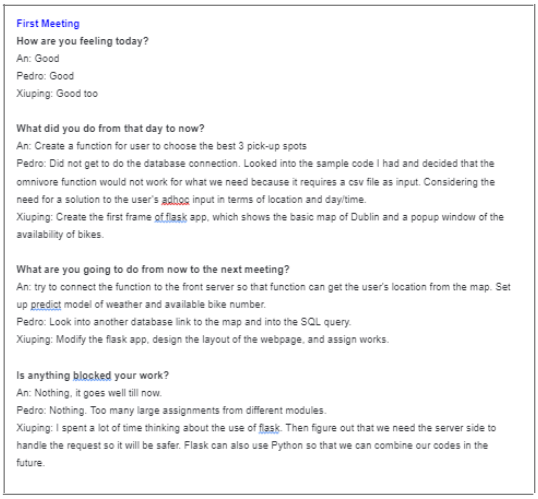
Course of action and decisions

The second sprint focused on creating a function for users to choose the best 3 pick-up spots and connecting the Server to the Client side so that it could get the user's location from the map. Also talked about the prediction model for weather and available bike numbers.

We created the first version of the Flask app, which showed the basic map of Dublin and a popup window of the availability of bikes. We also worked on removing duplicate rows in the Weather Table in MySQL and adding user features for HTML/JavaScript.

This was a tough sprint for our group as we had too many large assignments or tests from different modules right before the study break. Another issue we encountered that slowed down progress on the App was the large amounts of time that was spent thinking about and trying to understand the use of Flask.

Documentation



First Meeting

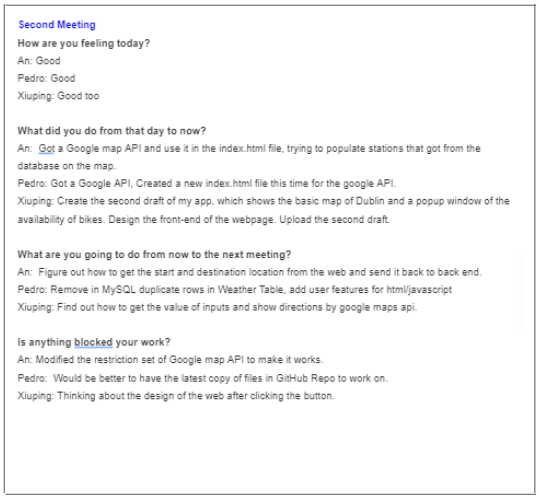
How are you feeling today?
An: Good
Pedro: Good
Xiuping: Good too

What did you do from that day to now?
An: Create a function for user to choose the best 3 pick-up spots
Pedro: Did not get to do the database connection. Looked into the sample code I had and decided that the omnivore function would not work for what we need because it requires a csv file as input. Considering the need for a solution to the user's adhoc input in terms of location and day/time.
Xiuping: Create the first frame of flask app, which shows the basic map of Dublin and a popup window of the availability of bikes.

What are you going to do from now to the next meeting?
An: try to connect the function to the front server so that function can get the user's location from the map. Set up predict model of weather and available bike number.
Pedro: Look into another database link to the map and into the SQL query.
Xiuping: Modify the flask app, design the layout of the webpage, and assign works.

Is anything blocked your work?
An: Nothing, it goes well till now.
Pedro: Nothing. Too many large assignments from different modules.
Xiuping: I spent a lot of time thinking about the use of flask. Then figure out that we need the server side to handle the request so it will be safer. Flask can also use Python so that we can combine our codes in the future.

Fig 34 Group 15's 2nd Sprint 1st Standup Meeting.



Second Meeting

How are you feeling today?
An: Good
Pedro: Good
Xiuping: Good too

What did you do from that day to now?
An: Got a Google map API and use it in the index.html file, trying to populate stations that got from the database on the map.
Pedro: Got a Google API, Created a new index.html file this time for the google API.
Xiuping: Create the second draft of my app, which shows the basic map of Dublin and a popup window of the availability of bikes. Design the front-end of the webpage. Upload the second draft.

What are you going to do from now to the next meeting?
An: Figure out how to get the start and destination location from the web and send it back to back end.
Pedro: Remove in MySQL duplicate rows in Weather Table, add user features for html/javascript
Xiuping: Find out how to get the value of inputs and show directions by google maps api.

Is anything blocked your work?
An: Modified the restriction set of Google map API to make it works.
Pedro: Would be better to have the latest copy of files in GitHub Repo to work on.
Xiuping: Thinking about the design of the web after clicking the button.

Fig 35 Group 15's 2nd Sprint 2nd Standup Meeting

Burndown Chart

Date	Planned	Remaining	Events
2023-02-26	10	10	Build Map
2023-02-27	9	10	
2023-02-28	8	9	Change Map API
2023-03-01	7	8	
2023-03-02	6	8	
2023-03-03	5	7	
2023-03-04	4	7	
2023-03-05	3	5	Create the web
2023-03-06	2	2	Add front-ends
2023-03-07	1	1	Add availability table

Fig 36 Group 15 spent a lot of time on switching the Map API, which delayed our progress.

Sprint 3 Review

Course of action and decisions

During the project's third sprint our goal was for completely implementing the Google maps and server functionalities with the possibility of trip directions, creating the prediction models and adding front end features to the Dublin Bikes App. Among these front end features, we discussed adding a hint bar to the input section and displaying detailed weather information for the next five days.

We wanted to add a hint bar because the previous bicycle station search box didn't feel very intuitive to use as it required the user to enter word for word the start and ending bicycle station's name in full. We also added a bicycle station dropdown menu below the date input. In the weather bar, we improved our weather prediction style from one row to a table, which made it clearer.

Pedro had issues connecting to the RDS database for about a week, which was needed to do his work for the prediction model. He also had issues getting the website to work on his laptop. The website issue turned out that he needed a previous version of the SQLAlchemy library(1.4.46).

Documentation

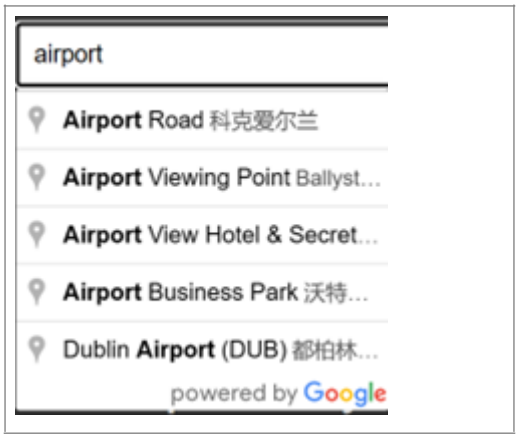


Fig 37 Hint bar(apologies for the Chinese word on it)

```
    // Add event listeners to details buttons
    document.querySelector("#weather-pane").addEventListener("click", (event) => {
        if (event.target.matches(".details-btn")) {
            const date = event.target.getAttribute("data-date");
            displayDetailedWeather(date);
        }
    });
}

function displayDetailedWeather(date) {
    // Fetch the weather data again and filter for the specific date
    $.get('https://api.openweathermap.org/data/2.5/forecast?q=Dublin&appid=ae15fcd8aa527f306b
        const detailsHTML = `<h3>Detailed forecast for ${date}</h3><div id="detailed-weather"
        $('#detailed-weather-pane').html(detailsHTML).show();

        let detailedWeatherHTML = '';
        data.list.forEach((forecast) => {
            const timestamp = new Date(forecast.dt * 1000);
            const forecastDate = timestamp.toDateString();
            if (date === forecastDate) {
                const hours = timestamp.getHours();
                const summary = forecast.weather[0].description;
                const icon = forecast.weather[0].icon;
                const temperature = Math.round(forecast.main.temp - 273.15);
                detailedWeatherHTML += `
            <div class="detail">
              <div class="time">${hours}:00</div>
              <img src="http://openweathermap.org/img/w/${icon}.png" alt="${summary}" width="
              <div class="desc">${summary}</div>
              <div class="temp">${temperature}°C</div>
            </div>`;
            }
        });
```

Fig 38 Creates the detail weather forecast

```
@app.route('/daily_avg_availability/<int:station_id>')
def get_daily_avg_availability(station_id):
    try:
        conn = engine.connect()
        results = conn.execute('''SELECT number, DAYOFWEEK(FROM_UNIXTIME(last_update)) a
                                  FROM availability
                                  WHERE number = %s
                                  GROUP BY number, day_of_week''', (station_id))
        rows = results.fetchall()
        return jsonify([row._asdict() for row in rows])
    except:
        print(traceback.format_exc())
        return "error in get_daily_avg_availability", 404


@app.route('/daily_avg_stands_availability/<int:station_id>')
def get_daily_avg_stands_availability(station_id):
    try:
        conn = engine.connect()
        results = conn.execute('''SELECT number, DAYOFWEEK(FROM_UNIXTIME(last_update)) a
                                  FROM availability
                                  WHERE number = %s
                                  GROUP BY number, day_of_week''', (station_id))
        rows = results.fetchall()
        return jsonify([row._asdict() for row in rows])
    except:
        print(traceback.format_exc())
        return "error in get_daily_avg_stands_availability", 404
```

Fig 39 Creates in the server the charts data response

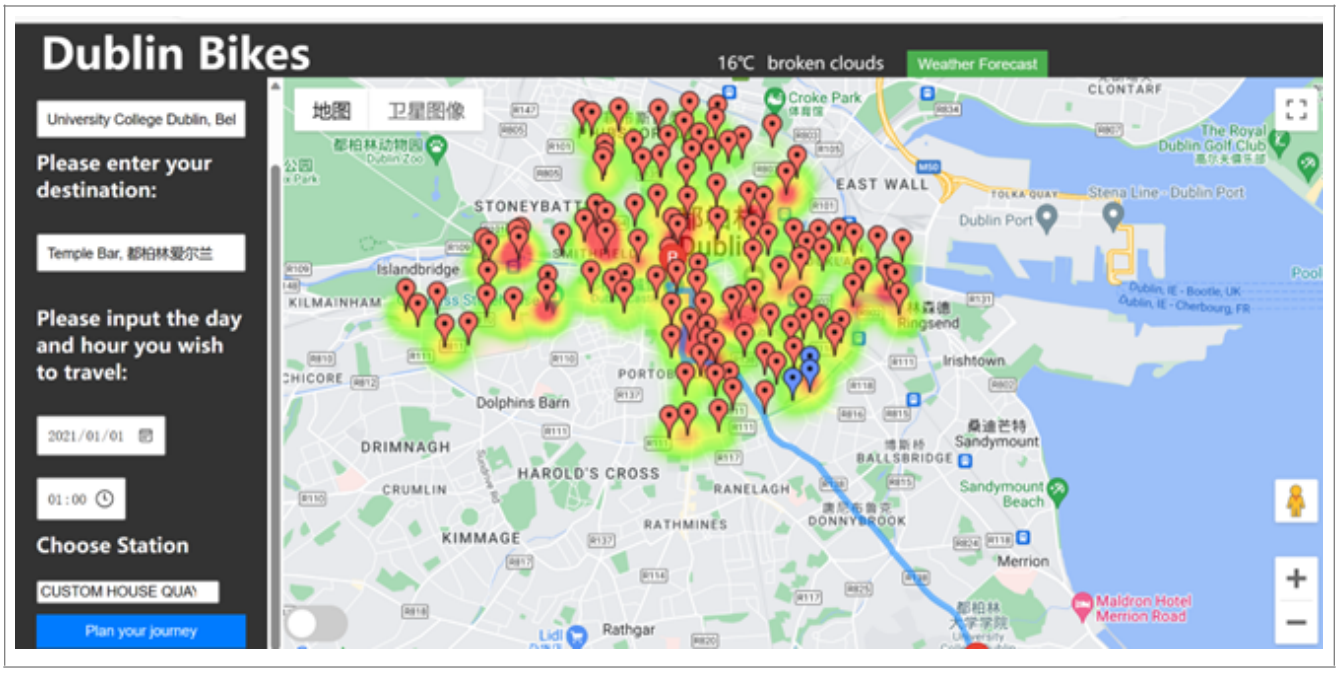
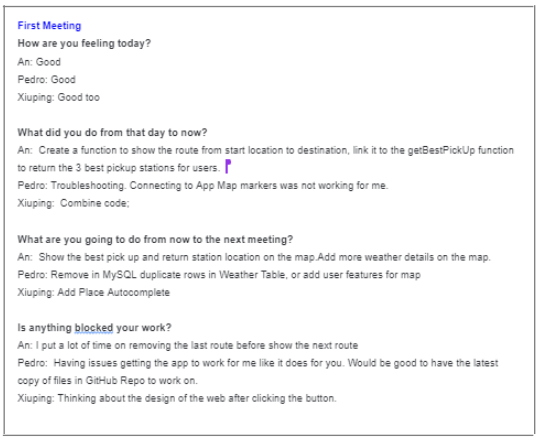


Fig 40 Group 15's trip route map

```
# Loop to do one for each Station
for station in stations[0]:

    # Filter by Target Station
    data = df.query('StationNumber == ' + str(station))

    #Setting Model's target data
    y = data[variable_targets]

    #Setting Model's variables data
    x = data[variables_training]

    #Splitting target data and variables data attributes into train and test tuples
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_st

    # Initialize the MLPRegressor
    stationModels[station] = MLPRegressor(hidden_layer_sizes=(100, 100), activation='r

    #Run the Model with train tuples
    stationModels[station].fit(x_train, y_train)

    # Serialize model object into a file called model.pkl on disk using pickle
    pkl_name = 'station' + str(station) + '.pkl'
    with open(pkl_name, 'wb') as handle:
        pickle.dump(stationModels[station], handle, pickle.HIGHEST_PROTOCOL)
```

Fig 41 Creates in the server the pickle files for data prediction



First Meeting

How are you feeling today?

An: Good

Pedro: Good

Xiuping: Good too

What did you do from that day to now?

An: Create a function to show the route from start location to destination, link it to the getBestPickUp function to return the 3 best pickup stations for users.

Pedro: Troubleshooting. Connecting to App Map markers was not working for me.

Xiuping: Combine code;

What are you going to do from now to the next meeting?

An: Show the best pick up and return station location on the map.Add more weather details on the map.

Pedro: Remove in MySQL duplicate rows in Weather Table, or add user features for map

Xiuping: Add Place Autocomplete

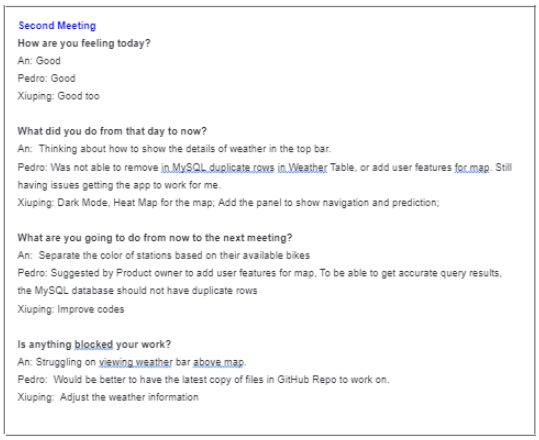
Is anything blocked your work?

An: I put a lot of time on removing the last route before show the next route

Pedro: Having issues getting the app to work for me like it does for you. Would be good to have the latest copy of files in GitHub Repo to work on.

Xiuping: Thinking about the design of the web after clicking the button.

Fig 42 Group 15's 3rd Sprint 1st Standup Meeting.



Second Meeting

How are you feeling today?

An: Good

Pedro: Good

Xiuping: Good too

What did you do from that day to now?

An: Thinking about how to show the details of weather in the top bar.

Pedro: Was not able to remove in MySQL duplicate rows in Weather Table, or add user features for map. Still having issues getting the app to work for me.

Xiuping: Dark Mode, Heat Map for the map; Add the panel to show navigation and prediction;

What are you going to do from now to the next meeting?

An: Separate the color of stations based on their available bikes

Pedro: Suggested by Product owner to add user features for map, To be able to get accurate query results, the MySQL database should not have duplicate rows

Xiuping: Improve codes

Is anything blocked your work?

An: Struggling on viewing weather bar above map.

Pedro: Would be better to have the latest copy of files in GitHub Repo to work on.

Xiuping: Adjust the weather information

Fig 43 Group 15's 3rd Sprint 2nd Standup Meeting

Burndown Chart

Date	Planed	Remaining	Events
2023-03-27	10	10	function of spot
2023-03-28	9	9	hint in search bar
2023-03-29	8	9	
2023-03-30	7	8	show root
2023-03-31	6	8	
2023-04-03	5	7	
2023-04-04	4	7	
2023-04-05	3	6	show weather details
2023-04-06	2	4	
2023-04-07	1	2	

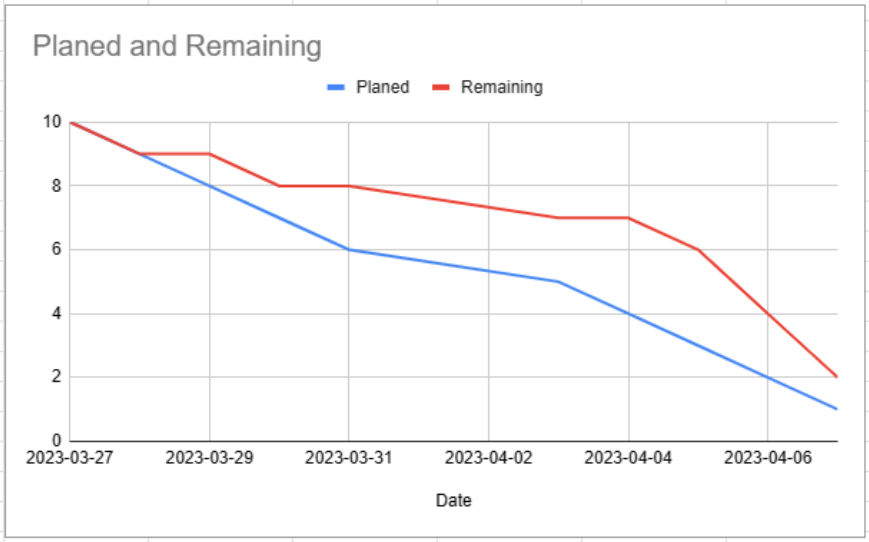


Fig 44 we spent too much time on showing route and display weather details

Sprint 4 Review

Course of action and decisions

The final sprint focused on determining the ultimate functionality of the website. Initially, users could input the time, date, and station to generate predictions, which were displayed in the navigation panel upon clicking the "show results" button. However, after some thought, we decided this design would be confusing for users, and switched to a format featuring three charts for improved clarity on the station's status levels which would translate to improved ease of use.

We decided to give up the navigation function, as it was not the primary objective of the application. Furthermore, Google does not offer a completely free version of their navigation API that we could utilize in the App(See Fig 42). Thus, we removed the navigation panel and replaced it with three charts. The first chart displays predictions if the user selects a date no earlier than today. We decided to use the chart.js library, which required some time to set up. Finally, we had to remove in Python the portion related to navigation for the prediction features to accommodate the changes.

Documentation

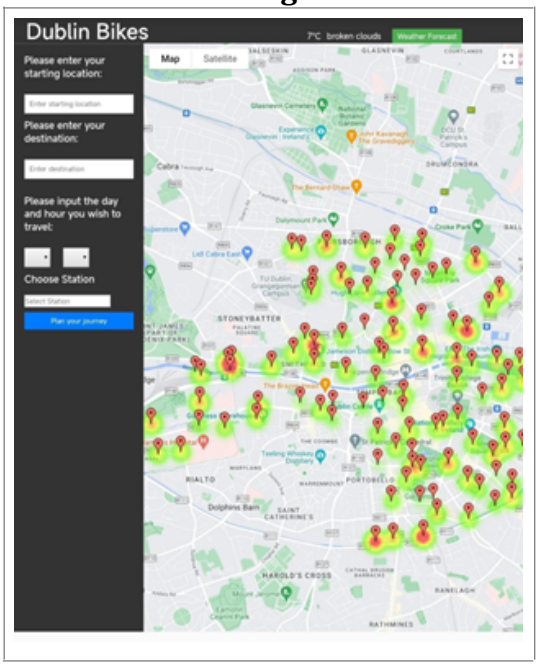


Fig 45 App before change

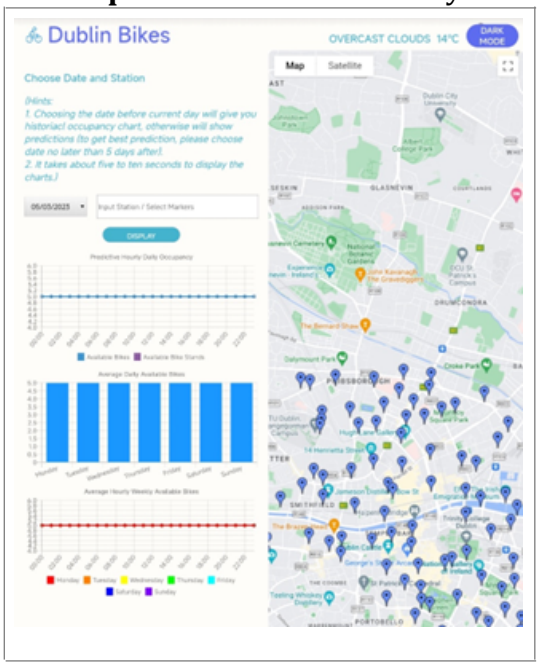


Fig 46 Fig. App after change

Pricing for the Directions API

SKU: Directions

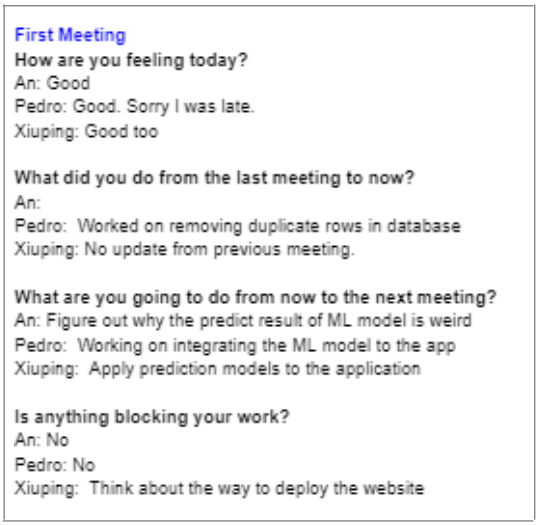
A request to the Maps JavaScript API's Directions Service (excludes requests triggering the Directions Advanced billing SKU) or the Directions API.

MONTHLY VOLUME RANGE (Price per QUERY)		
0–100,000	100,001–500,000	500,000+
0.005 USD per each (5.00 USD per 1000)	0.004 USD per each (4.00 USD per 1000)	Contact Sales for volume pricing

Fig 47 Google Navigation API is not free to use.

```
@app.route('/prediction', methods=['POST'])
def get_prediction():
    try:
        # Get the input data from the request
        data = request.get_json()
        print("Received data:", data)
        station_id = data.get('station_id')
        date = data.get('date')
        time = data.get('time')

        # Load the corresponding model for the station
        model_filename = f'station{station_id}.pkl'
        model_filepath = os.path.join(os.path.dirname(os.path.abspath(__file__)), "model", model_filename)

        print("Loading model file:", model_filepath)

        with open(model_filepath, 'rb') as f:
            model = pickle.load(f)

        input_data = preprocess_input(date, time)
        prediction = model.predict([input_data])
        print("Prediction:", prediction)

        return jsonify({'bikes': prediction[0][0], 'stands': prediction[0][1]})
    except:
        print(traceback.format_exc())
        return "error in get_prediction", 500

@app.route('/hourly_avg_availability/<date>/<int:station_id>')
def get_hourly_avg_availability(date, station_id):
    print(date)
    try:
        conn = engine.connect()
        results = conn.execute('''SELECT number, HOUR(FROM_UNIXTIME(last_update)) as hour_of_day,
                                    AVG(available_bikes) as avg_bikes, AVG(available_bike_stands) as avg_stands
                                FROM availability
                                WHERE DATE(FROM_UNIXTIME(last_update)) = %s and number = %s
                                GROUP BY number, hour_of_day;
                                ''', (date, station_id))
        rows = results.fetchall()
        result_data = [[row.avg_bikes for row in rows], [row.avg_stands for row in rows]]
        return jsonify(result_data)
    except:
        print(traceback.format_exc())
        return "error in get_hourly_avg_availability", 404

@app.route('/hourly_weekly_avg_availability/<int:station_id>')
def hourly_weekly_avg_availability(station_id):
    try:
        conn = engine.connect()
        results = conn.execute('''SELECT number, DAYOFWEEK(FROM_UNIXTIME(last_update)) as day_of_week,
HOUR(FROM_UNIXTIME(last_update)) as hour_of_day, AVG(available_bikes) as avg_bikes
                                    FROM availability
                                    WHERE number = %s
                                    GROUP BY number, day_of_week, hour_of_day;
                                    ''', (station_id,))
        rows = results.fetchall()
        return jsonify([row._asdict() for row in rows])
    except:
        print(traceback.format_exc())
        return "error in hourly_weekly_avg_availability", 404
```

Fig 48 Dublin Bikes App sample of Flask implementation after removing navigation.



First Meeting

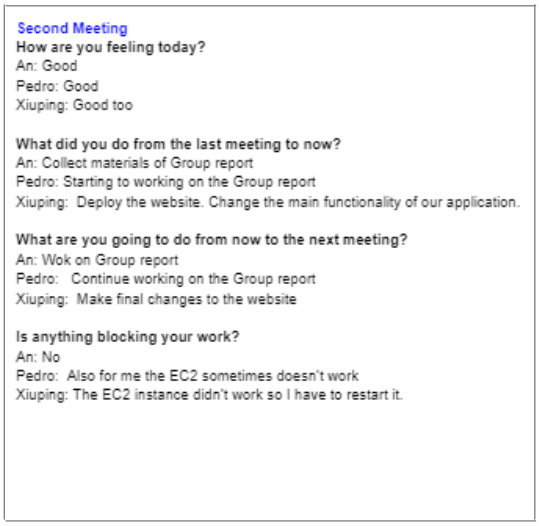
How are you feeling today?
An: Good
Pedro: Good. Sorry I was late.
Xiuping: Good too

What did you do from the last meeting to now?
An:
Pedro: Worked on removing duplicate rows in database
Xiuping: No update from previous meeting.

What are you going to do from now to the next meeting?
An: Figure out why the predict result of ML model is weird
Pedro: Working on integrating the ML model to the app
Xiuping: Apply prediction models to the application

Is anything blocking your work?
An: No
Pedro: No
Xiuping: Think about the way to deploy the website

Fig 49 Group 15's 4th Sprint 1st Standup Meeting.



Second Meeting

How are you feeling today?
An: Good
Pedro: Good
Xiuping: Good too

What did you do from the last meeting to now?
An: Collect materials of Group report
Pedro: Starting to working on the Group report
Xiuping: Deploy the website. Change the main functionality of our application.

What are you going to do from now to the next meeting?
An: Wok on Group report
Pedro: Continue working on the Group report
Xiuping: Make final changes to the website

Is anything blocking your work?
An: No
Pedro: Also for me the EC2 sometimes doesn't work
Xiuping: The EC2 instance didn't work so I have to restart it.

Fig 50 Group 15's 4th Sprint 2nd Standup Meeting

Burndown Chart

Date	Planned	Remaining	Events
2023-04-12	10	10	Change Layout
2023-04-13	9	9	
2023-04-14	8	9	Deploy site on EC2
2023-04-15	7	8	Complete Models
2023-04-16	6	7	Add route for prediction
2023-04-17	5	7	
2023-04-18	4	4	Add three charts
2023-04-19	3	3	
2023-04-20	2	2	Complete site
2023-04-21	1	1	

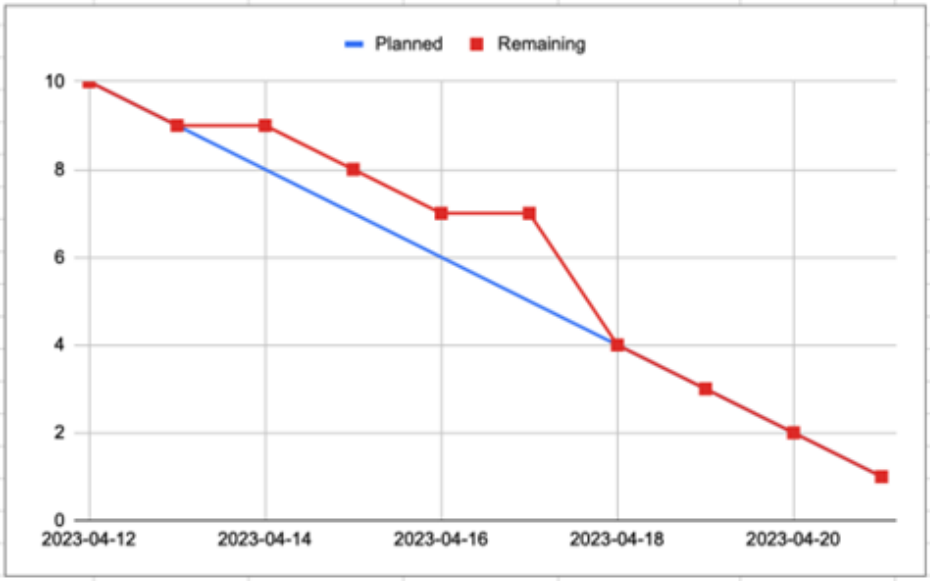


Fig 51 We finished. Hooray!

Retrospective

Overall Project Assessment

This project was very difficult and required mental and emotional effort. At the beginning of the project, none of us had used AWS, Flask, APIs and all the development tools that this simple project required. It was like we were all incoming student wizards at Hogwarts, except we were all Harry Potter because none of us came with a background in programming. Harry would be dead without Hermione. Gone. However, without Hermione we got it done and we are proud of the final Dublin Bikes App! The App does a great job on the server client communications, managing the APIs and going from MySQL database tables to predictions ready for the user.

As mentioned, we are new at this so there were some challenges. Teams in the workforce face a real challenge when putting together a new product, so for us, learning the tools required all the while there is an expectation to deliver the product was very demanding and this is without taking into account the other five modules on our graduate course. We met the challenge by learning about web frameworks and the other technologies with a smile on our faces because we were not giving up! We did not give up when other situations presented themselves. In challenges like the moment we were made aware that we should probably switch from Mapbox to Google Maps, or when we realized that the navigation feature would cost money and needed to be replaced with other features. These were learning moments that emphasized the need to not ignore any obstacles, address them as soon as it is feasible to do so and be transparent, communicate with your teammates. Pedro was having trouble getting the app to work on his PC and so he was open about it, and working together they were able to decipher that the App needed an earlier version of SQLAlchemy. It was a vulnerable moment that became an opportunity for team bonding!

Future Work

One idea to implement in the future was to hide the three status charts for the bike station selected until the user clicks display button and then for the idle time before the charts load, show a loading icon first.

Another idea was having a separate input for the destination bike station and displaying instead a prediction bike availability chart for the starting station and a prediction bike stands chart for the destination station (both separate from each other).

Bibliography

Halford, M. (2023, February 20). *Visualizing bike stations live data*. Retrieved from https://maxhalford.github.io/blog/bike-stations/

IBM. (2023, 5 5). *What is three-tier architecture?* Retrieved from https://www.ibm.com/topics/three-tier-architecture

JCDecaux Ireland Ltd. (2023, April). *dublinbikes > Maps*. Retrieved from dublinbikes.ie: https://www.dublinbikes.ie/en/mapping

The National Transport Authority. (2022). *Canal Cordon Report 2021.* Dublin: Dublin City Council.