# DUBLIN BIKES
# GROUP 14 REPORT

# for

# COMP30830 SWE
# Dublin Bikes Web App

**Victoria Keane**
**Rhys O'Dowd**
**Zaur Gouliev**[1]

**Eoin O'hAnnagain**
**Product Owner**

**April 15, 2022**

[1]Author of report

# Contents

# 1 Introduction

This document is a report, it provides a detailed account of the requirements to build a online web app for Dublin bikes tracking, simply called Dublin Bikes App. This report style document is based on the UCD module Software Engineering (COMP30830). It involves information on the design, technologies, workflow involved to create the application and this document also provides an account into the structure of the development/implementation since it was done by 3 individuals it is also used as a way of documenting the stepts taken to achieve the desired goals. It is noteworthy to mention this application is not intended to be used by the wider public and is only meant to be used as a means of assessing the quality of work. The files for this web app can be accessed on our GitHub repo on the clickable link below:

Project 14 - https://github.com/gouliev/comp30830

Our offical website hosted on the Ec2 instance:

ec2-34-243-114-11.eu-west-1.compute.amazonaws.com

This application in its primary purpose and intent is to be a web application that displays the occupancy of bikes, alongside weather information for Dublin, and consequently it allows for predictive and smarter planning for picking up a bike in a given location. This app, although not intended to be published, does have practical use in that Dublin City is a heavily populated urban area and travel via bike is not only convenient but also a regular occurrence. According to a 2019 CSO survey, 15 percent of over 18s take a journey by bicycle with many different reasons for this. So undoubtedly an application such as our Dublin Bikes makes commercial sense as a utility application.

In the successful happy path, a user user should be able to find an availability of the closest bike based on their current location (geo-location) and this gives them the closest and occupancy of bikes based on proximity. Without going into detail in the introductory section, briefly our location and map itself is provided by Google Maps API. This should also display the weather, which is taken from our Weather API. This gives the user a chance to determine if they want to book the bike or not based on this information. The occupancy itself is pulled from a JCD API which keeps log of these things.

The project consists of 3 people.

- Victoria Keane - 16395531
- Rhys O'Dowd - 17465146
- Zaur Gouliev - 18718545

Our group is project 14, and we are all in the UCD CS Conversion programme. Victoria and Zaur are studying for their M.SC in Computer Science, while Rhys is doing his HDip in Computer Science. The team dynamic was entirely remote and we operated entirely remotely with no face-to-face contact. This was an exceptional experience to be able to work in a very freelance way and still be able to communicate with each other at any moment in time. We used a multitude of communication tools for this. It included Discord, Trello, Zoom, Google Meet, WhatsApp and our UCD emails.

Our product manager is Eoin O'hAnnagain, a PhD Student in the UCD School of Computer Science. He guided us throughout the project and we did weekly check-ins (Thursday at 10:00am weekly) to determine our progress, our workflow, updates and we were given tasks to follow also that was expected to be complete in the following weeks meet. We decided to take a fair approach regarding the managerial role of the team, meaning that a scrum master position was shared every 2 weeks as per our sprints, with Zaur being the scrum master twice, both at the start of the project and for the final part of the project. We followed a scrum methodology, as we had 4 sprints. The following was the structure:

- Zaur Gouliev - Scrum Master for Sprint 1 (First Sprint)
- Victoria Keane - Scrum Master for Sprint 2 (Second Sprint)
- Rhys O'Dowd - Scrum Master for Sprint 3 (Third Sprint)
- Zaur Gouliev - Scrum Master for Sprint 4 (Final Sprint)

There were a number of reasons we decided to take this approach, but the main one was so that each member of the group had some experience of what it meant to be a scrum master and was faced with the difficult task of having to monitor, delegate and divide up the tasks amongst the group in a way that was fair for everyone considering we had other tasks for other modules also to do.

We also found that each person had different styles, times and methods of working. One member would prefer to work deeply at night, while others early morning or perhaps one member was much more comfortable with coding while the other was much better at setting up environments, connecting pieces, working on databases or even drafting up documents, scrum managing, taking meeting notes and etc.

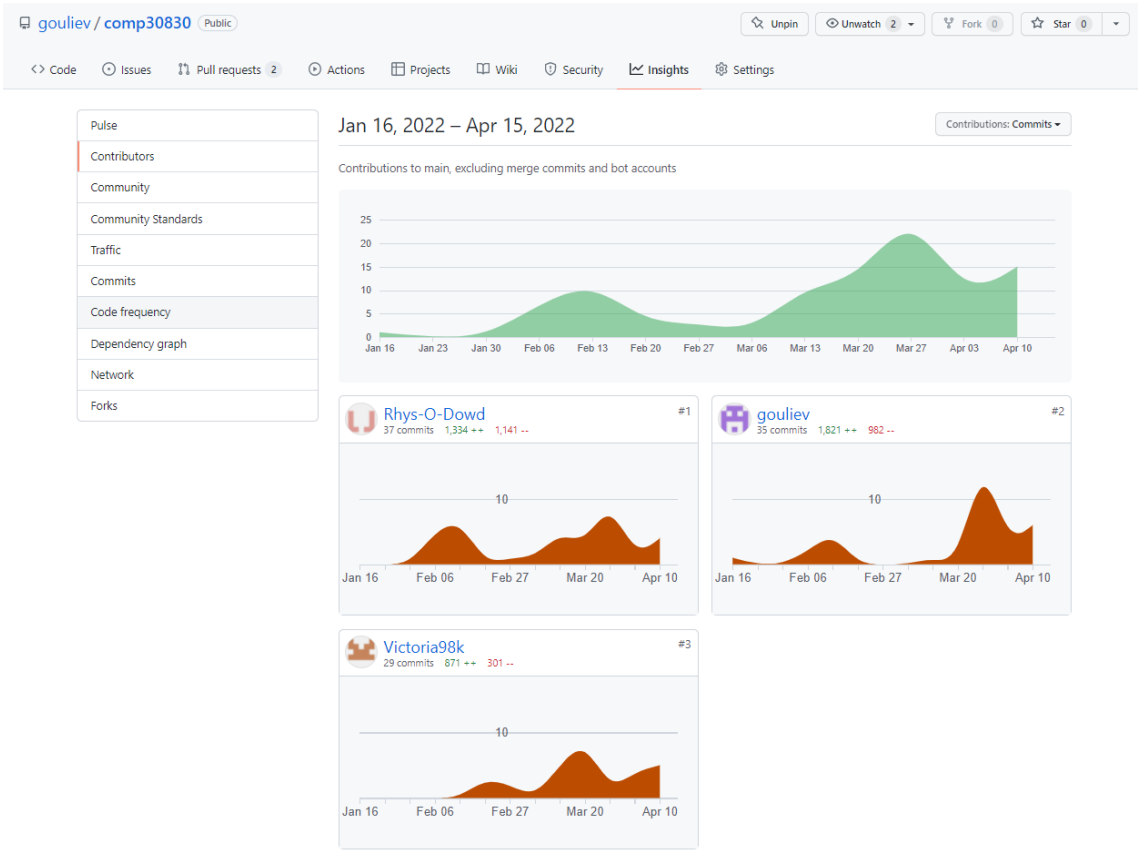



Figure 1.1: Final GitHub as of 15/04/2022

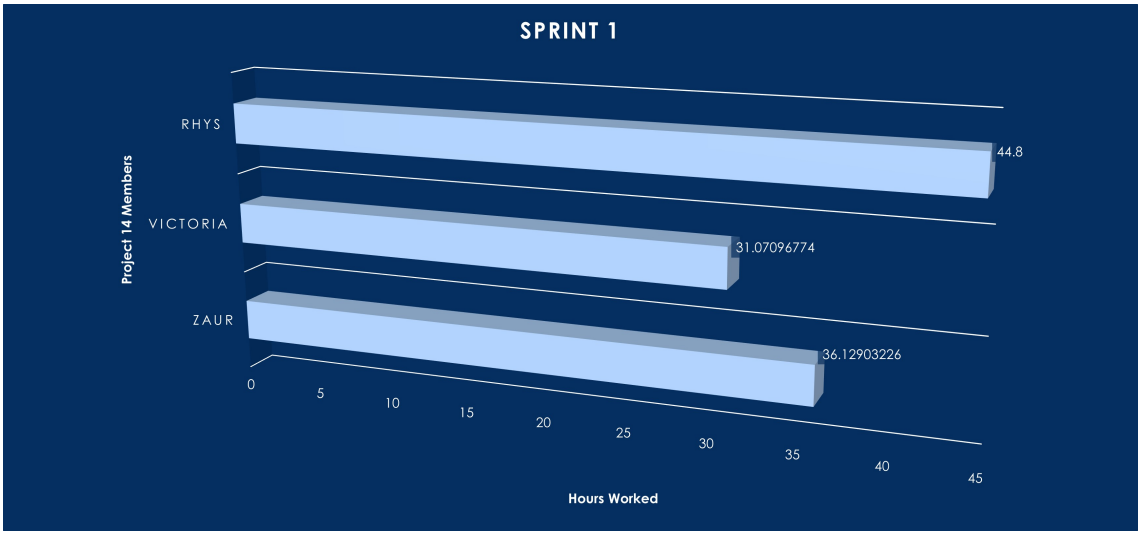



Figure 1.2: Example of Sprint 1 contributions, more details can be found in the accompanying files for burn down charts

## 1.1 Our workflow (SCRUM)

The team met weekly to discuss the division of tasks, and we took meeting notes to document it in case we needed clarification. These meeting notes document an entire timeline of our team, project 14, and show discussions, themes, topics and conversations of what we were planning. All of us were involved in the process of development and familiarized ourselves with each and every task required and what was needed to build it. This is important, meaning that even if something was done such as a python scraper, a connection between GitHub and the EC2 instance, or even connecting the flask app to the EC2 instance, we ensured that we all tried to do it at least once and looked at the code, understood it and were able to describe, tinker and explain the parts needed to make the app run. In this way, we ensured we understood the technology stack and the tools in themselves, one thing we were all forced to do was to take a turn in setting up an EC2 server and an RDS database since we ran into the complication of running out of Free Tier data on AWS.

Delegation of tasks was set each week, some were told to work on the html/css, while others on the python/javascript. As a general example of how we did a delegation run. **Zaur** took the task of doing out the entire SRS while documenting the entire process that was occurring, this was useful since he was the scrum master twice, and he had the unique task of being able to monitor and ensure everyone was adhering to the timeline by checking in daily on the discord channel we had set up strictly for our project and questions related to our project. He also contributed to various parts of code, things like connecting up instances or connecting EC2s to flask and so on. **Victoria** did quite a lot of front end work, designing the html/css and connecting it to the map to give a nice

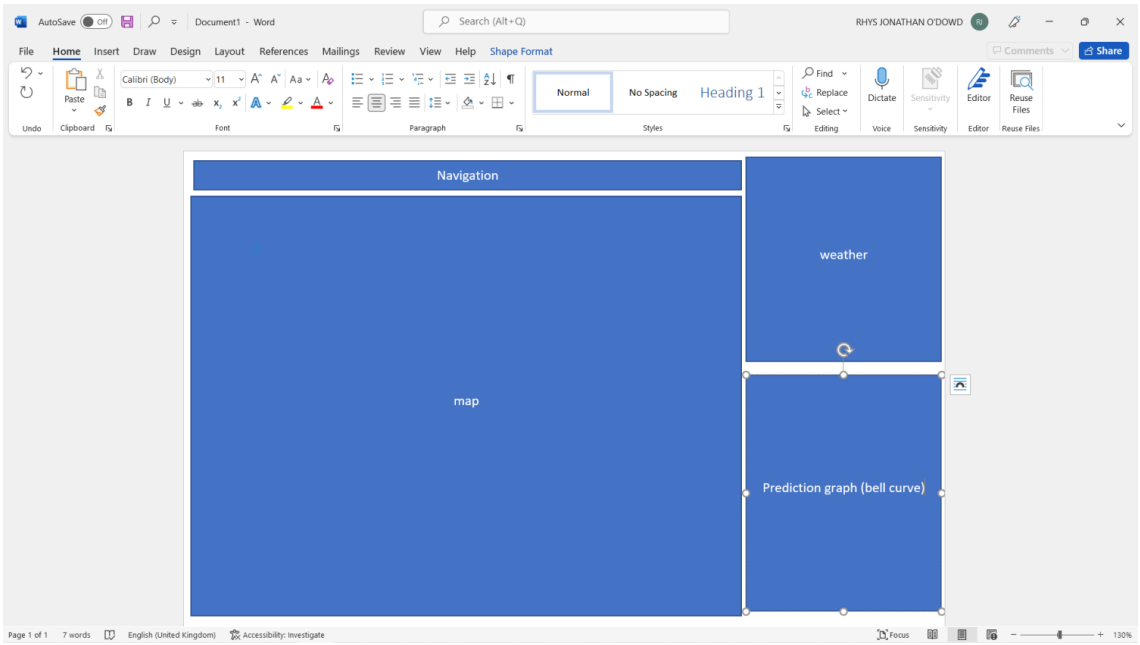



Figure 1.3: Example of design work

front end design. She had contributed greatly and was very creative in her approach.

**Rhys** did coding work on the scrapers, this included the weather scraper and the bike scraper. We all contributed to helping and discussion on various parts, often times when errors would occur we would be willingly there available to try fix it.

As aforementioned, the methodology we took for our workflow was one that involved a comprehensive understanding of the stack of technologies used to create, debug and implement something. The general sentiment and agreement amongst us towards the end of the project was that we all had agreed we were all evenly and fairly contributed to the project so nobody felt like they were doing more than others.

We worked very closely with GitHub and followed a Git based workflow, this included having a master origin where we only committed files once we vetted it through the branches. Each person had a branch. This was done in Sprint 1 when Zaur created the original repository to work in and asked if everyone could make a branch, commit a file or two to it to ensure it is working. The reviews and pull requests were looked at regularly through the weeks and discussions occurred on them where we would share screens in Zoom calls to discuss possible errors, which often when found, we discussed as a group how to solve. This concept of controlling for each version is one which we all understood very well by the end of the project. We got a sense that this system keeps track and ensures that versions do not get lost, damaged or fail because it is separated from the master file. The branch we developed all 3 of us mainly worked was called dev

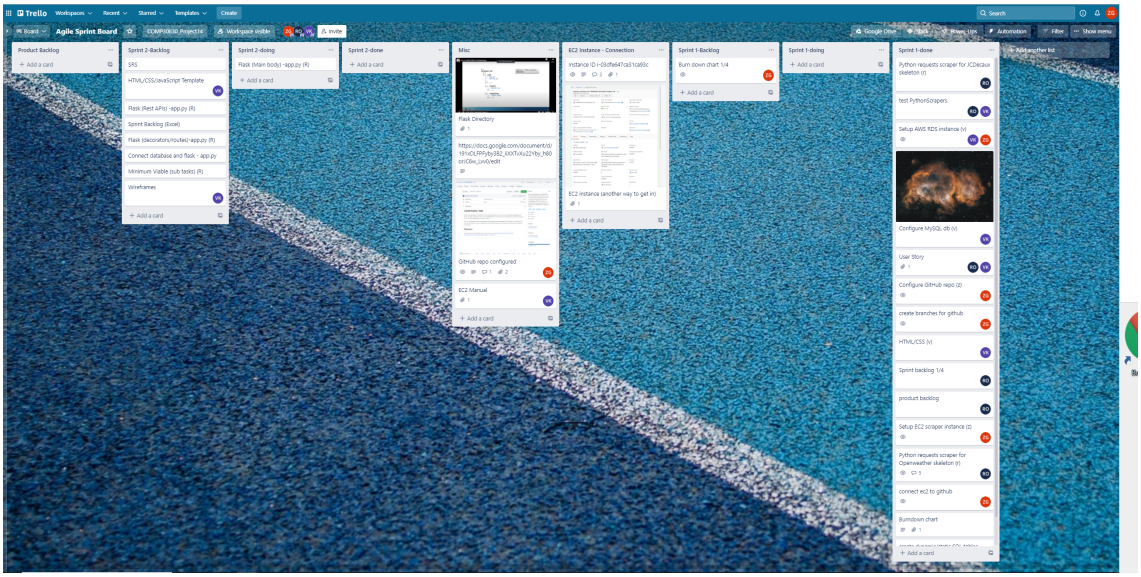



Figure 1.4: Example of design work

(short for developer) and this is where most of our pushes went to. An example of a conversation that occurred in these meetings about a project is as follows:

## 1.2 Tools/Technology

We used many tools in this project and a wide variety of them were first time. The following tools used to create this Dublin Bikes Web application were as follows:

**Trello** was a popular tool that we used to manage our scrum tasks and other information. Zaur set it up during the first week of scrum meeting and kept manage of it to ensure only useful info (API keys, links to EC2, .pem files and so on) was put in there, as we did not want to over clog it with information that was not useful to us.

**Discord** was our means of communication and we used this not only for sharing info, but transferring files, data, or any pieces of info that helped (such as YouTube tutorials and so on). We also used Discord to go on video calls to share our screen whenever we ran into problems.

**Microsoft Excel** was used for tracking our sprints, and we used to make burn down charts, due to the easy nature of generating charts.

**Amazon Web Server** was used to host our application, we used an Amazon EC2 instance (a virtual Linux machine) and this instance had scripts running on it to scrape

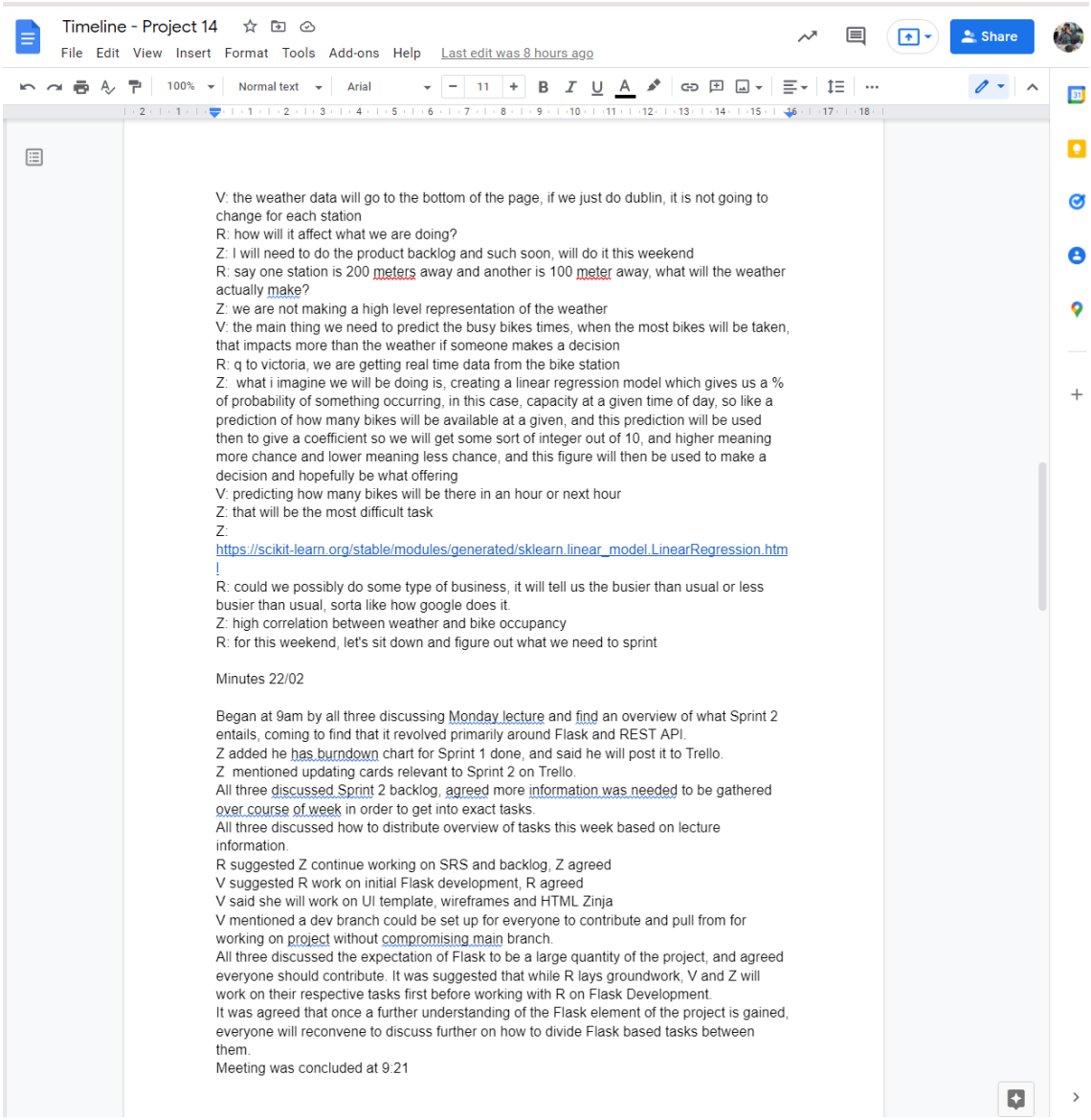



Figure 1.5: Example of design work

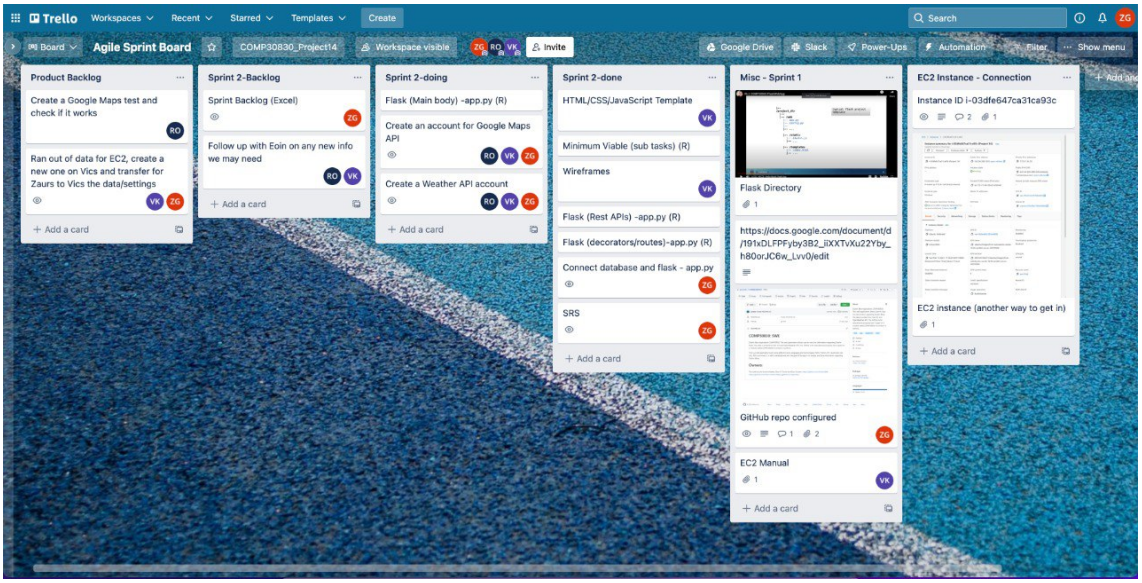



Figure 1.6: E.g. Trello Sprint 2 Information

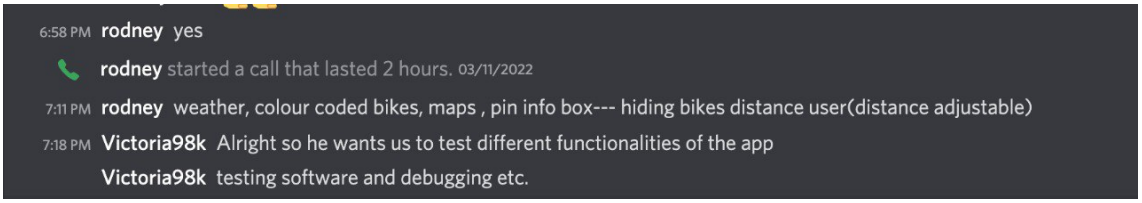



Figure 1.7: E.g. Discord call - SCRUM log

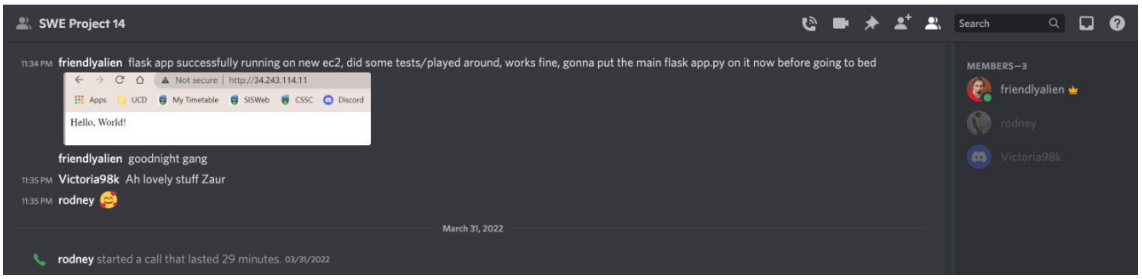



Figure 1.8: E.g. General updates via Discord regarding project

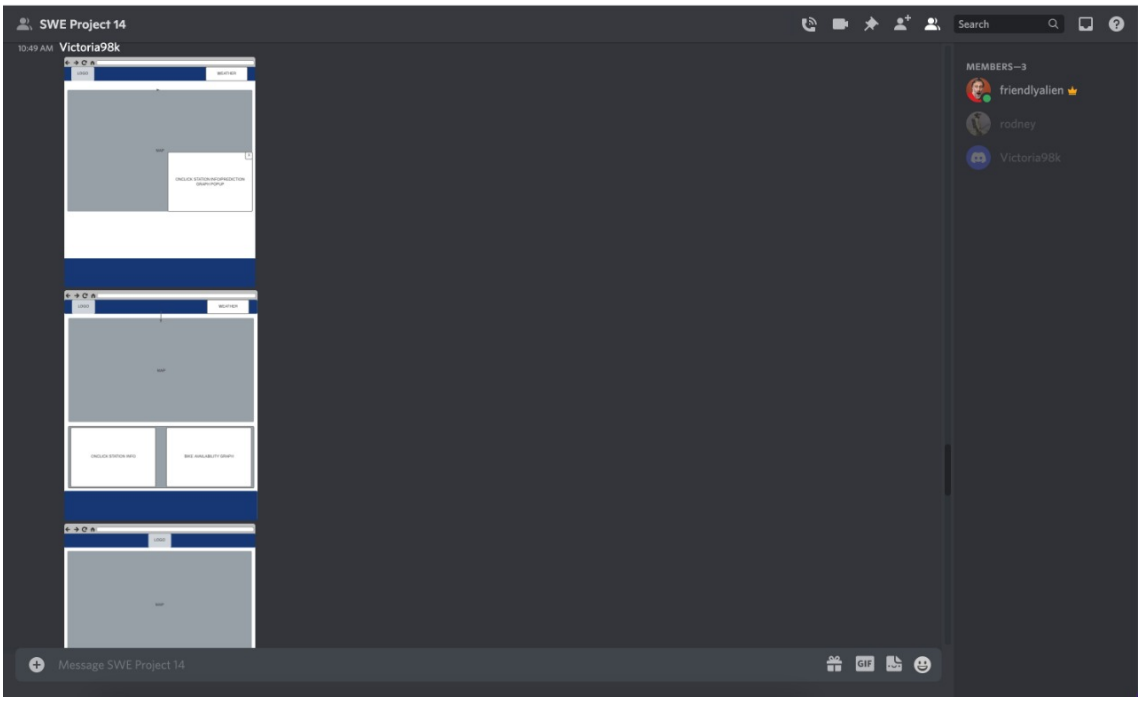



Figure 1.9: E.g. Discord chats allowing for updates to be sent while immediately notifying

**31/03/2022 - Sprint 3 second half meeting**

Z; hey how is everything going
E; weather widget is very centre, by putting 2 divs, icon in 1 and it takes up half that div space. The other half would be the written info like temp and everything like that, just as much to utilise the screen space.
V;
R; when you load the page the first time, it asks you where your location is
R: we have routing functionality too
E: that is perfect, what happens when you click one of the icons
E; that is really impressive, good job
E; everything looks good, no extra features, lots of features implemented, try something out and do a radius around the user, google provides a code dto do this, it is a simple function, decreasing or increase the radius for the user, if we have extra time,
E; anything that is difficult , every feature we implement, routing data and stuff like that, extremely obvious that we may not have to, it can be unfortunate, make sure everything else is on that. Burndown charts, meeting scrum info and screenshotting the github, tidying everything up and writing the report, the srs is a sales pitch (design pattern), report is teamwork and etc
E; product owner asked for new features product
V: working on the data model, showing her screen, vic is doing a linear regression and is experimenting with adding and removing some features, but eoin recommends trying a different type of model and in the report we can mention it.

Figure 1.10: E.g. Trello Sprint 2 Information

data from our APIs and pull this data into our RDS database. The flask application itself contains a main file called app.py and this file was essentially communicating with the sever, i.e., a flask based application server.

**HTML/CSS/Javascript** were used primarily for frontend development. The HTML and CSS for the front website which contained links with the Javascript that would shape the website and ensure all the details were viewable, clickable and so forth.

**Flask** was used, and flask is a Python web framework, the flask when connected with the EC2 acts like our server and is the main connection point from front end to back end.

**Google Docs** was used, to record all of our meetings while we were in calls.

**Python** was used throughout the application also, primarily for our weather and bike data scrapers and also for the data analysis files. this part which was used to determine

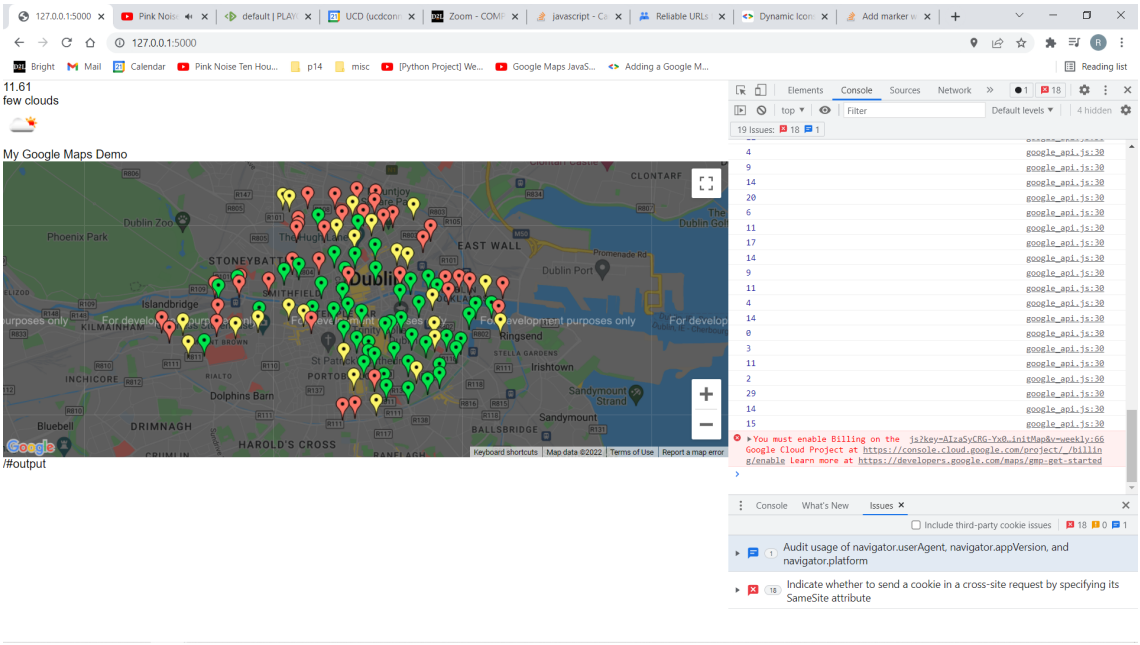



Figure 1.11: Maps API calls

the occupancy and weather prediction.

**Google & Direction Service Maps APIs** was utilized to get the information of directions and the actual map interface which we all connected to and had our own individual accounts (with 300$ student credit)

## 1.3 Intended Audiences/Expectations

As we hinted on earlier, the use of this application is one which we consider to be of a utility. This has a lot to do with the fact that taking a bike to work, school or anywhere for travel or leisure is common practice in Dublin. If we look for the Dublinbikes scheme, we can see that there is about 1600 bikes and 116 stations, we can ascertain then that this app should be built with the intent that if it reaches max capacity, we should be able to cater for these users. These users and target audience ranges from professionals, students and tourists, the stations on its own vary in how busy they are depending on how much sprawl of people there is around them. We can expect, for example, stations around Trinity College to be busy more often due to the amount of university students in the area and places that have less offices or footfall having more bike occupancy.

The user of our web application will have the following user experience when first opening the application:

- The user will open up the web application without the need to log in or create an

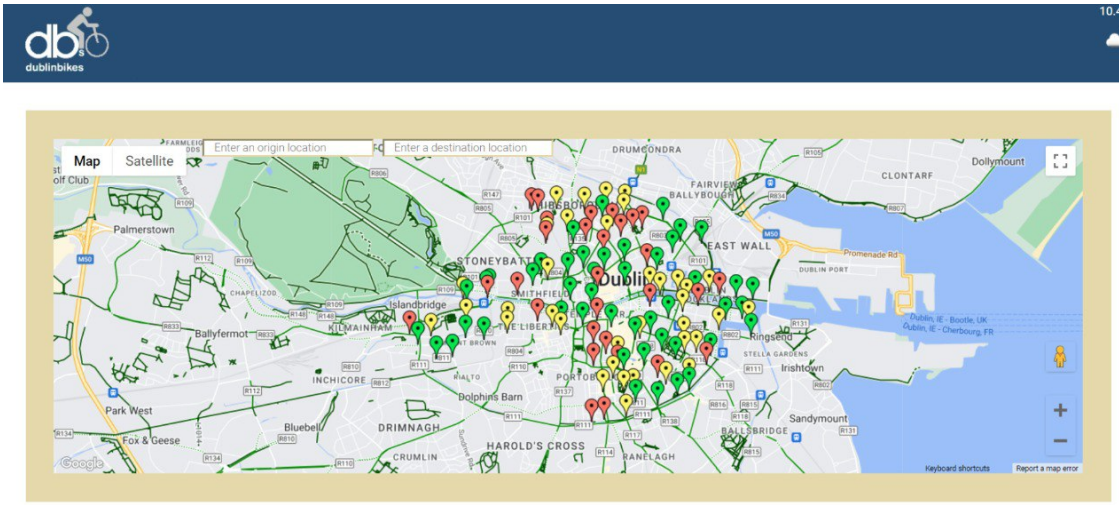



Figure 1.12: Example of design formatting with logo

account, they will be able to view and search for different biking stations. The details used for searching will be the name of the station and the address of the station. This map will be an ordinary map of Dublin. The map itself is intuitive and the label colours are standardized in a legend to display what each mean.

- The user will also be presented with weather data that contains information about the current weather on the current time, this is, for example, the likelihood if it is going to rain or how it feels right now in terms of temperature rather than what the temperature actually is. This was a conscious design decision that was made as most people rely on 'feels like' when looking at the weather
- A standard user should be able to open the app, view these details and make a decision on if he/she intends to go to the station to take out the bike. This comes with a reasonable level of due diligence as the weather can changed unexpectedly and predictions on occupancy can change if an unexpected or extraordinary situation occurs such as a bus breakdown and an influx of bike users.

## 1.4 Understanding/Examining the Map

If we look at the green lines on the map, these are are the bike lanes. The red and yellow colour pins indicate the severity of business on the actual stations. On the top 2 bars on the right hand side is the location of your own address and the second box is the location of where you want to go. As a hypothetical user, I will take my own situation, Zaur and display as if I was a first time user. Take a look in this current situation where I will

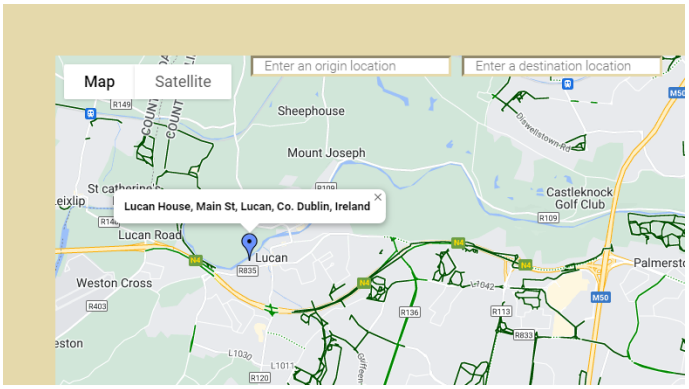



Figure 1.13: My current location

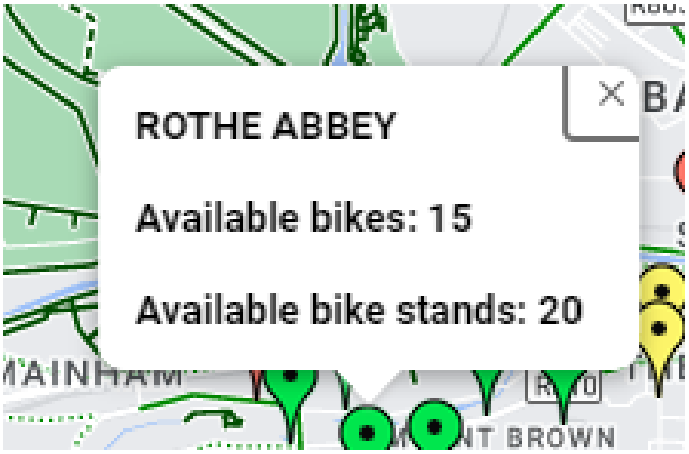



Figure 1.14: Clicking a station, green one available

find my location and route from travelling from UCD to Portobello Dublin. This gives a display when clicking at the red pins and green pins the number of available bikes and the number of available bike stands. You can examine the screenshots below to get a better idea of this and the functionality when clicking.

## 1.5 Application - The Dublin Bikes App

This section details out the application and it aims at explaining how the application works and what the user sees on first use. The application is called Dublin Bikes, we chose to keep a simple name and not focus so much on branding and designing it in terms of the user experience (UX), so we kept it simple since it was our first time designing anything of the sort and working on a project in this nature. The logo below is our app display photo, and QR code on our logo also links directly to our GitHub repository and to our application server (EC2 instance which is hosting it live by the time of reading

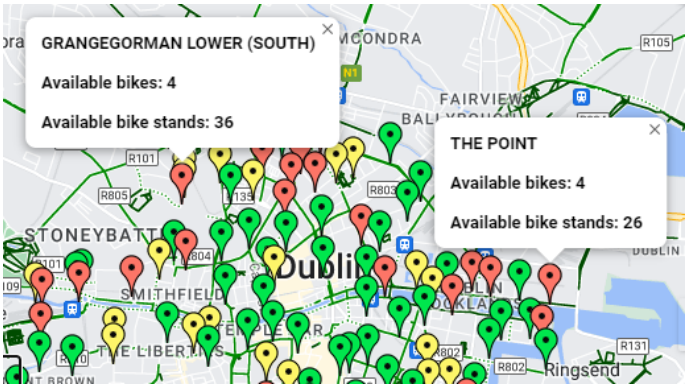



Figure 1.15: Example of a busy red pin station

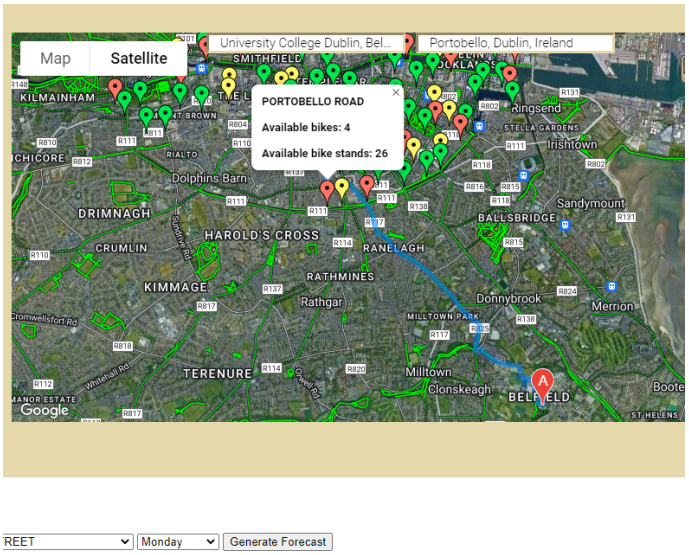



Figure 1.16: Travelling from UCD to Portobello

this).

To get a full copy of it, you can clone the repo and install the requirements (shown below in Operating Environment section). First clone the repo

```
git clone https://github.com/gouliev/comp30830
```

Once done, you can install the requirements by:

```
pip install -r requirements.txt
```

## 1.6 Product Functions

## 1.7 Operating Environment

This application is intended to work on any modern browser both on mobile phone and computer. It also works on any operating system. No special settings or equipment and so on is required to use this web application.

The application itself besides being built with Python 3, Flask, Google APIs and JavaScript, it was built on 3 devices who used the following coding/working environments:

- Victoria Keane - Macbook Pro using Apple's iOS System, primarily using VSCode, GitBash and Ubuntu Terminal
- Rhys O'Dowd - Dell XPS13 using Windows 10, primarily using VSCode and GitHub Desktop
- Zaur Gouliev - Lenova Yoga 13 using Windows 10 Premium, primarily using VSCode, Sublime Text and GitHub Desktop

Requirements to build the application are as follows (also in our requirements.txt file)

```
certifi==2021.10.8
charset-normalizer==2.0.11
click==8.0.4
colorama==0.4.4
Flask==2.0.3
Flask-SQLAlchemy==2.5.1
greenlet==1.1.2
```

```
idna==3.3
itsdangerous==2.1.0
Jinja2==3.0.3
MarkupSafe==2.1.0
requests==2.27.1
SQLAlchemy==1.4.32
urllib3==1.26.8
Werkzeug==2.0.3
wincertstore==0.2
```

## 1.8 Design

The following page shows our design sketches, and the overall design we ended up choosing and how it was implemented

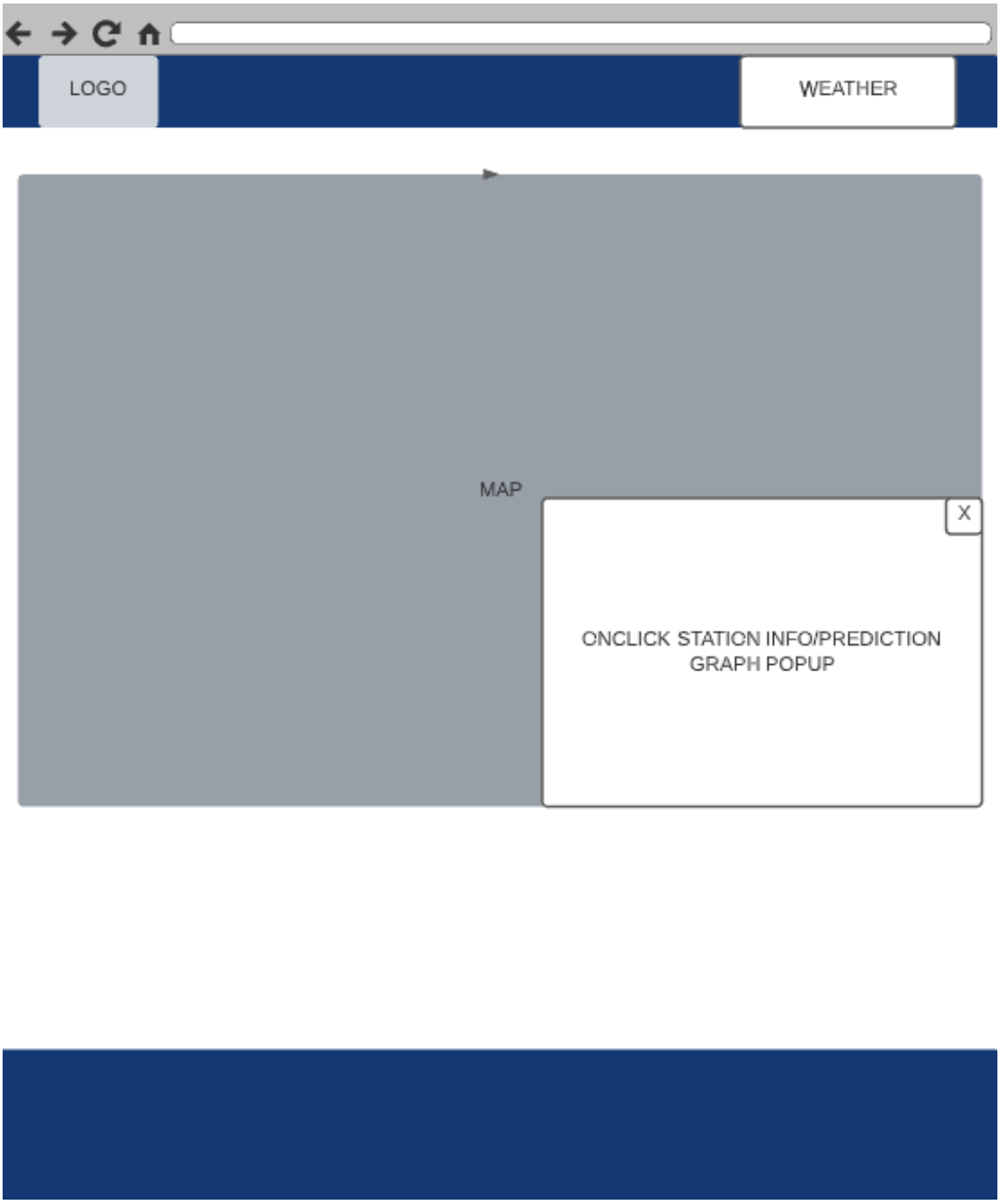



Figure 1.17: Sketches of Design

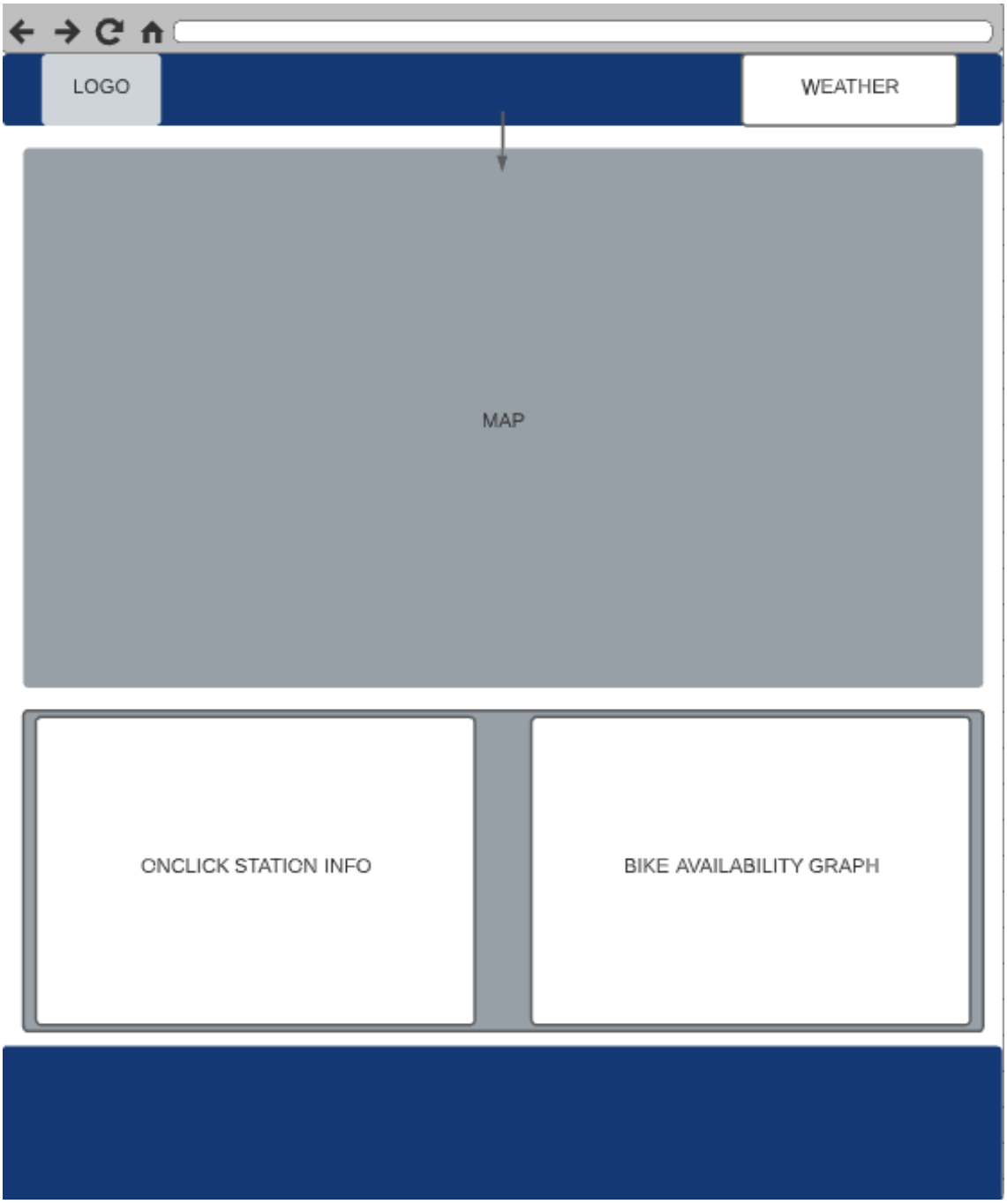



Figure 1.18: Sketches of Design

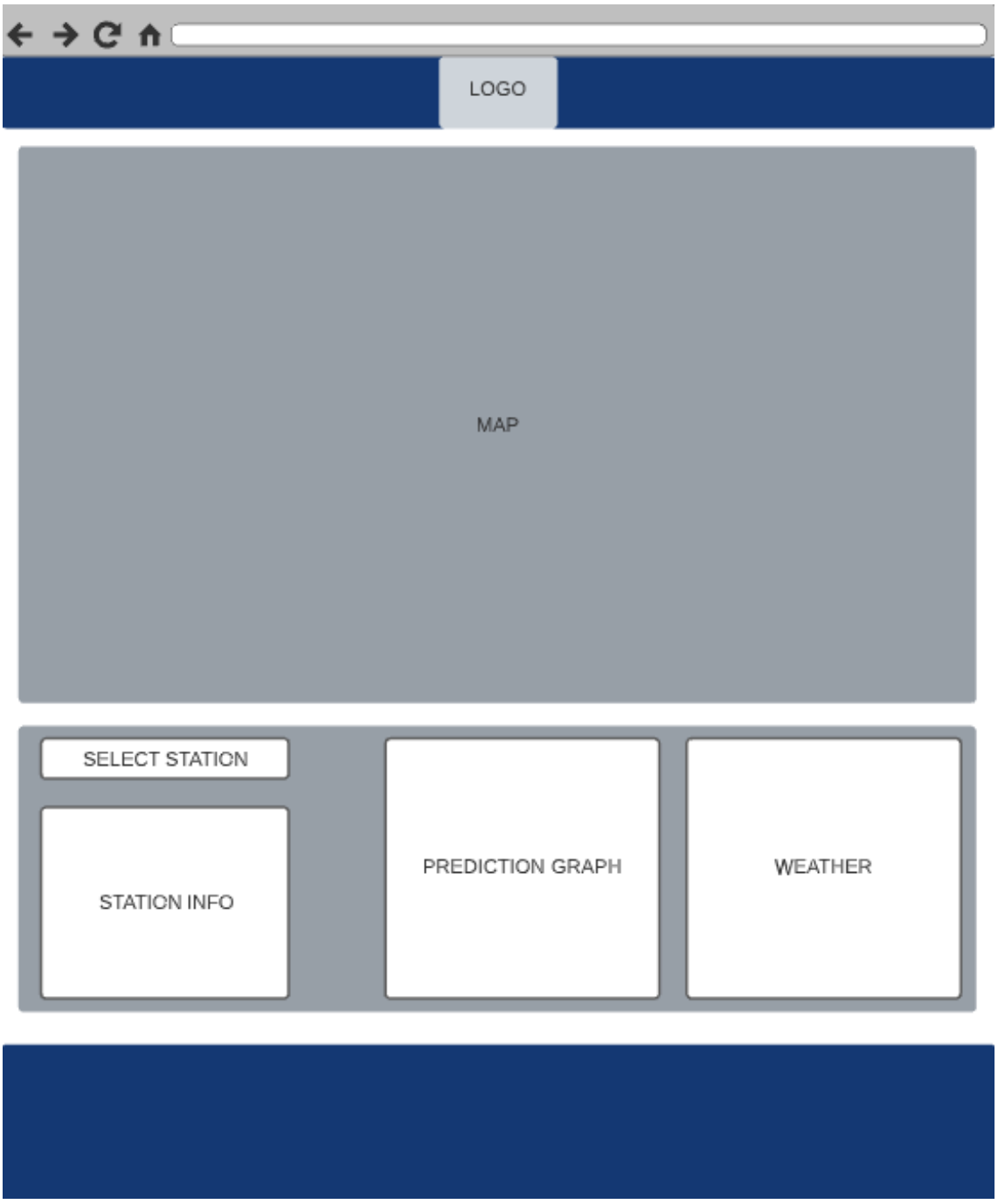



Figure 1.19: Sketches of Design

# 2 System Features

This section gives a better view and outline of what the user can see and sort of features are to be expected and observed.

The first thing that can be observed is the actual map itself, it is loaded when opening the website, as can be shown from the screenshot below, this top option includes an option of calculating where you are to where you want to go. Giving both info on how to get there on the map and also duration of time it takes.

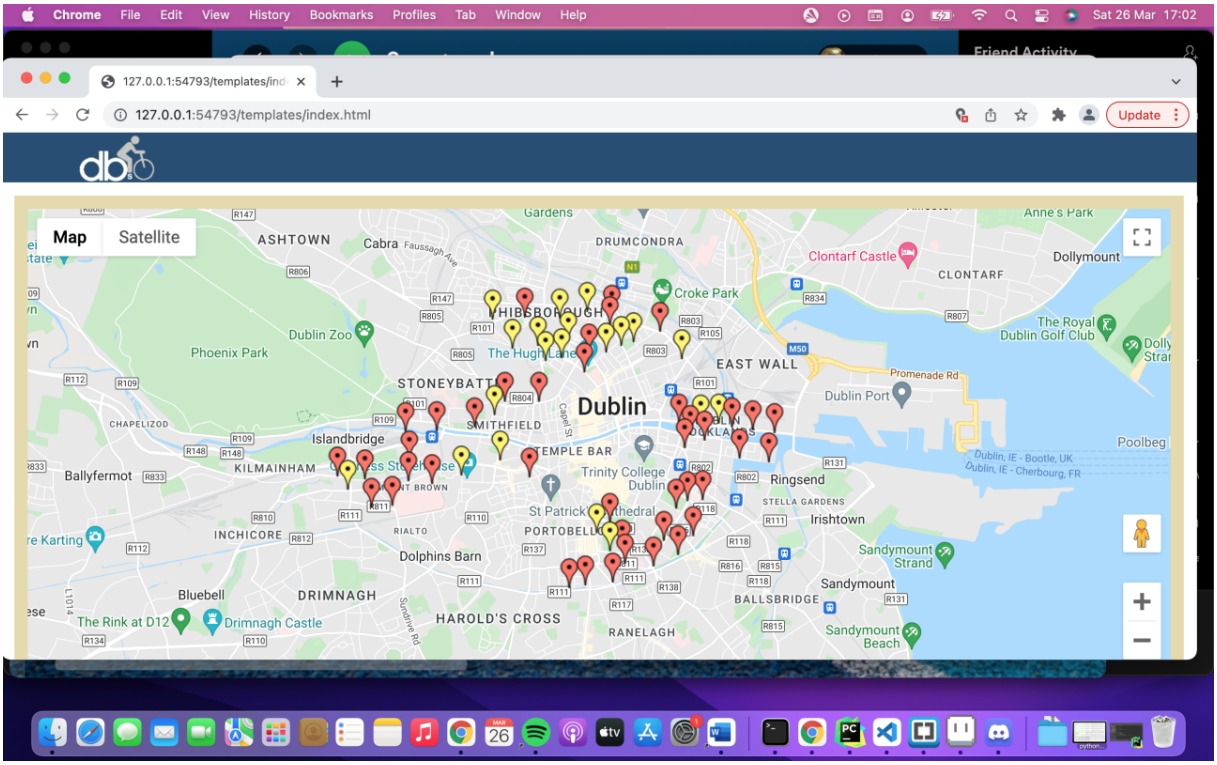



Figure 2.1: Front of Map

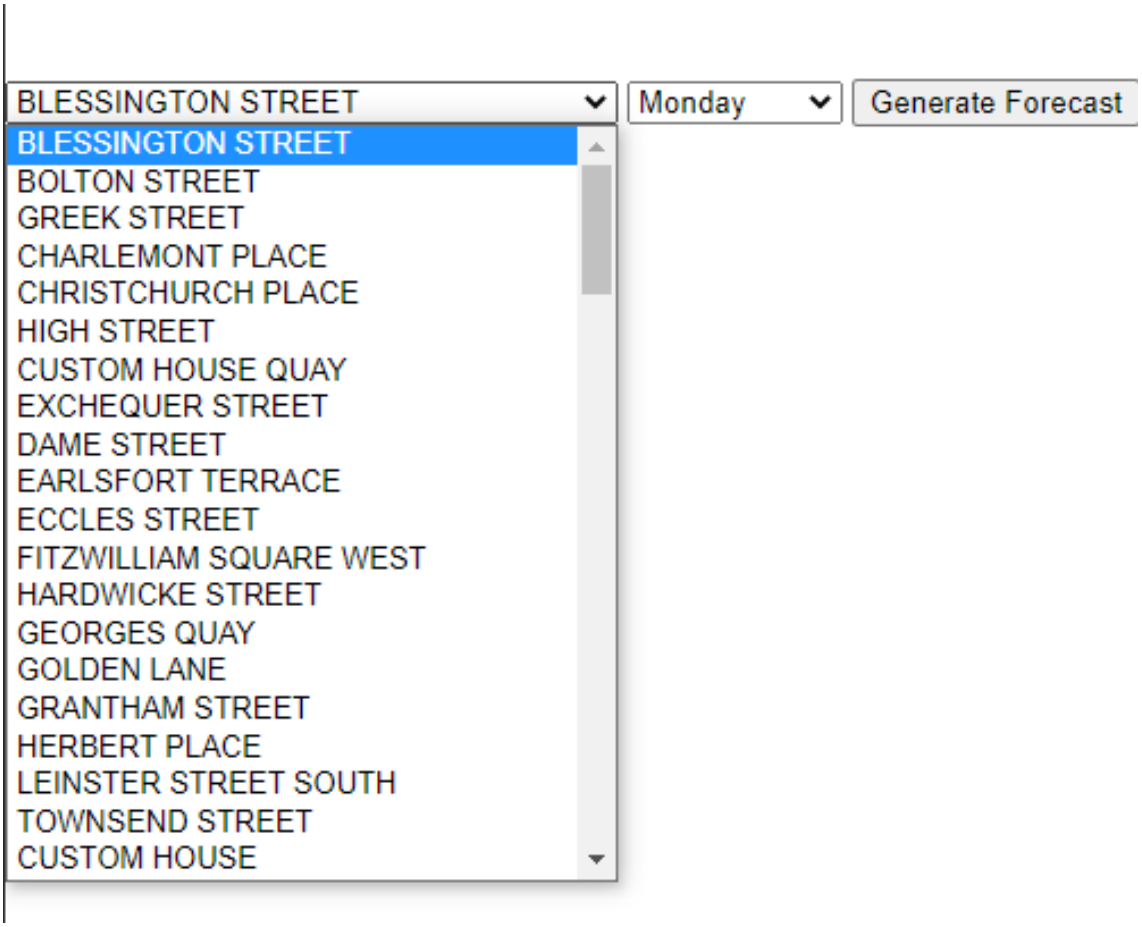



Figure 2.2: Select a street to show real time analytics. In the next option you can select a street/station which will show you real time analysis by the hour on the actual street

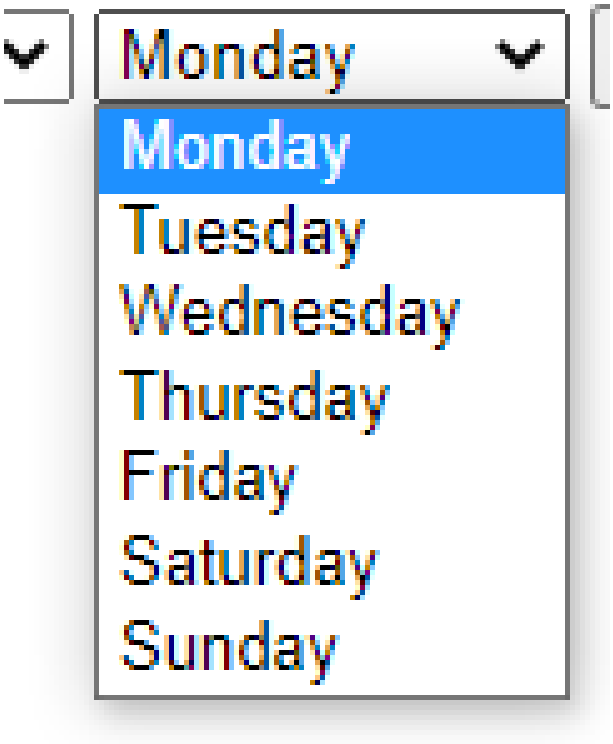



Figure 2.3: Option to select a particular day of the week. In the option to select a particular day this gives you an option from Monday to Sunday.

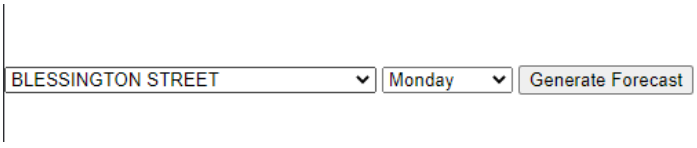



Figure 2.4: Full view of select option

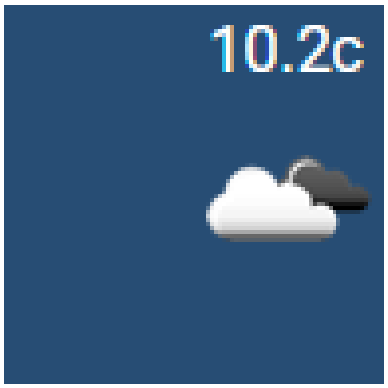



Figure 2.5: Top right screen display weather option

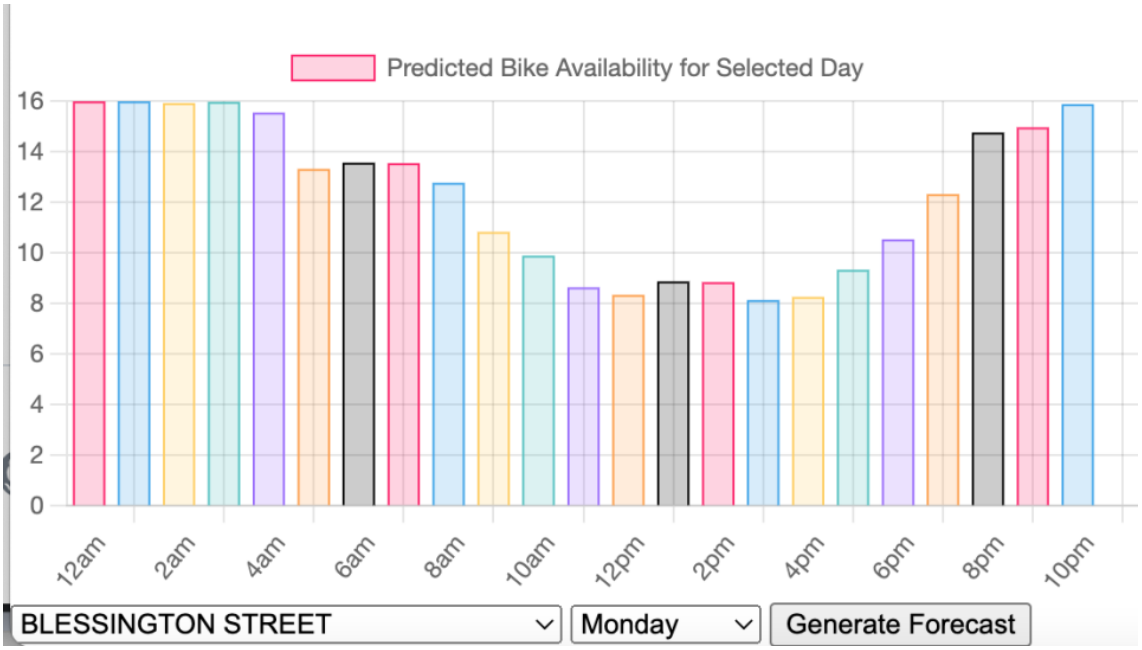



Figure 2.6: Finally a full graph and analytical view should be shown like this:

# 3 Final Product from User Side

## 3.1 MVP

This section contains the final look and product from a user side. It was taken on the final day of submission and tested both from a coding side and user side by Zaur. I looked at every aspect of the project and functionality and can confirm everything is in working order as described in this report and in the accompanying SRS.

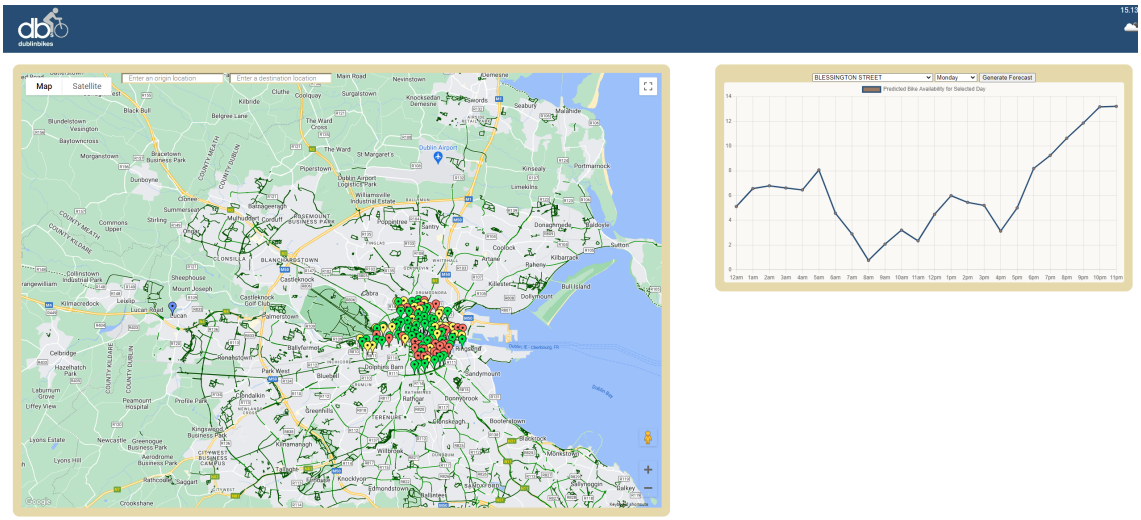



Figure 3.1: Final working product - 15/04/2022: Zaur Test on 13:14

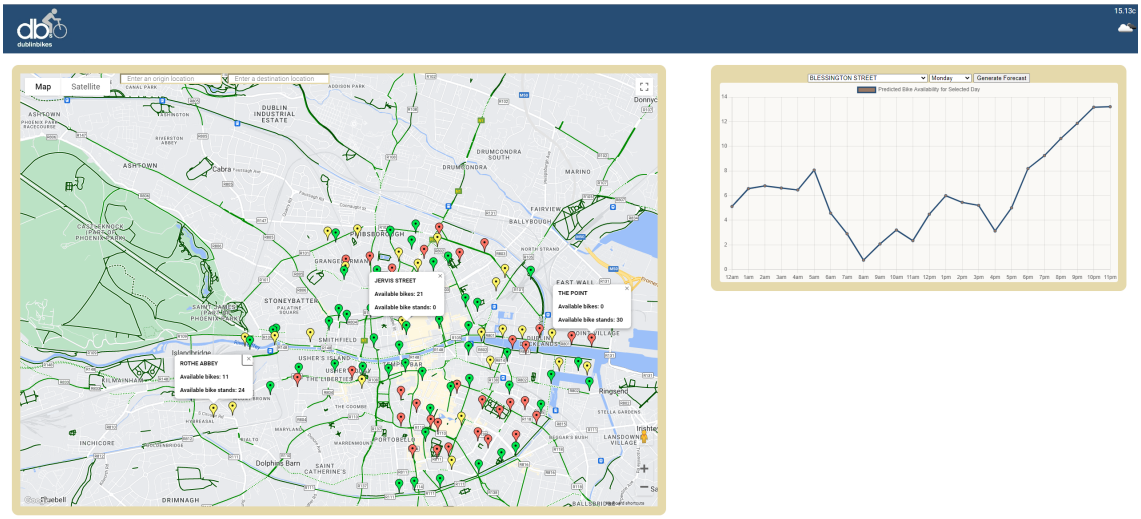



Figure 3.2: Final working product - 15/04/2022: Zaur Test on 13:14

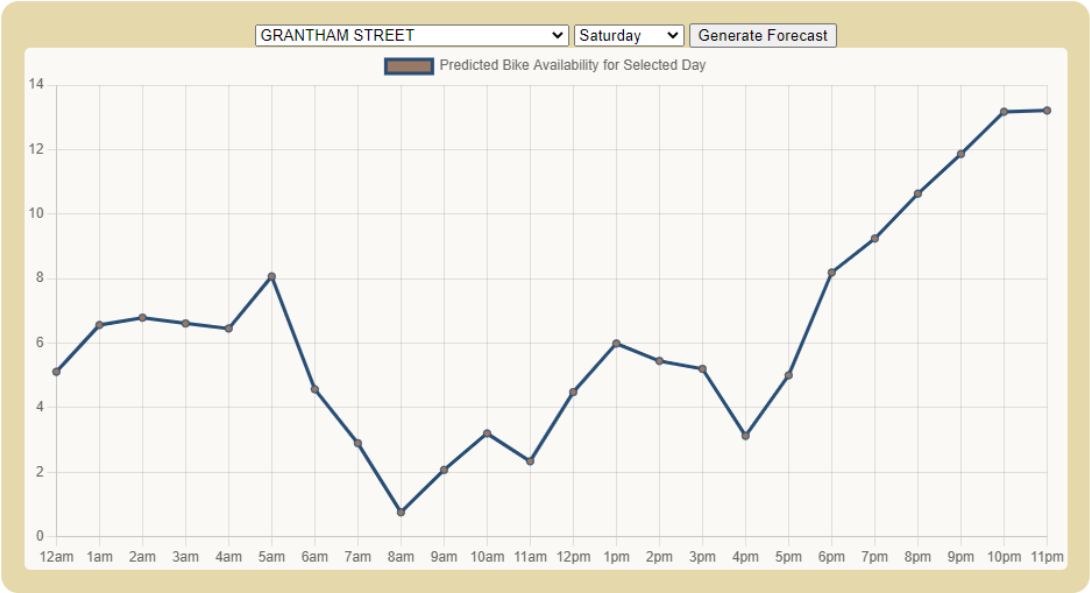



Figure 3.3: Final working product - 15/04/2022: Zaur Test on 13:14

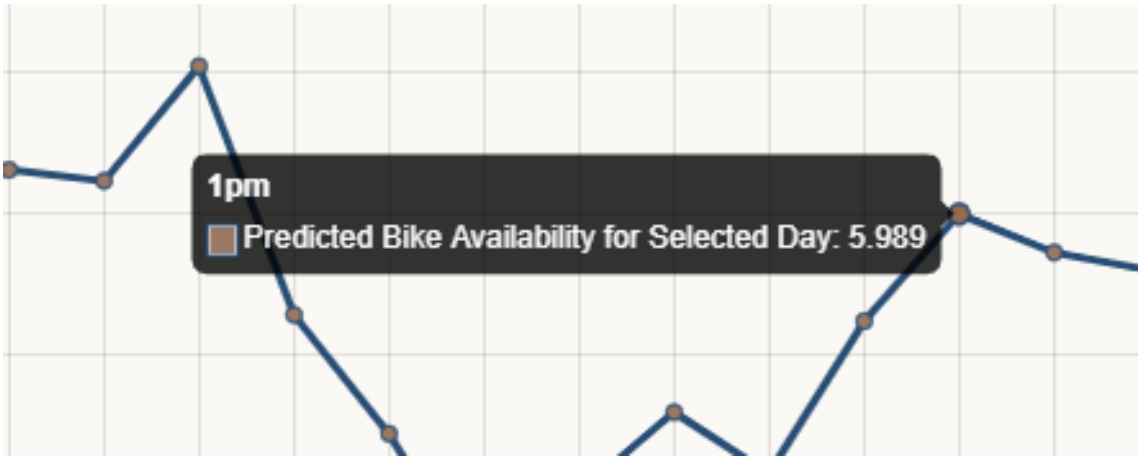



Figure 3.4: Final working product - 15/04/2022: Zaur Test on 13:14

# 4 Data Analytics

This section outlines the data analysis done and how the approach was taken. It goes into detail on the methods and the approach. We used pickle files to project our graphs and data. An example of a pickle file is below in Fig 3.1.

## 4.1 Jupyter Notebooks

For the model that we used, we focused on the historical data we had that was being pulled and scraped regarding the weather and the bike occupancy from the APIs. These APIs were being scraped using the 2 scrapers we built. This data then was transformed, prepared and used in the trained model. The model we used was based on the number of bikes, the status of the bikes, the bike stands, the time of day, and the current day.

We used 2 models as a prediction at first, this was linear regression and random forest regression. We ended up choosing random forest regression. We first began by cleaning the data, dropping unnecessary columns like address as the same values are represented by number, and converting columns with values of only two strings to binary in order to make it applicable for model training.

In our linear regression, we analysed the remainder of results in the cleaned dataset and prepped for testing. This model produced an R2 accuracy score of .62. We came to the conclusion and got an idea of the results variability and have them graphed so we were able to use them for comparison with the trained results.

We eventually did not like the linear regression model and decided that we would score better on a random forest model, so there we went on to utilise two data training

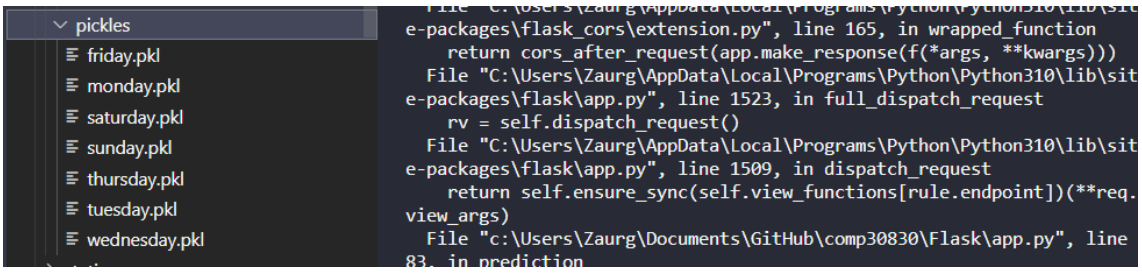



Figure 4.1: Pickle files:

## Section 1

In this section I work on cleaning the data, dropping unnecessary columns like address as the same values are represented by number, and converting columns with values of only two strings to binary in order to make it applicable for model training.

In [31]:
```
df.head()
```

Out[31]:

| | number | address | status | position_lat | position_lng | bikes | bike_stands | banking | last_update |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 42 | SMITHFIELD NORTH | OPEN | 53.349562 | -6.278198 | 16 | 14 | False | 2147483647 |
| **1** | 30 | PARNELL SQUARE NORTH | OPEN | 53.353462 | -6.265305 | 3 | 17 | False | 2147483647 |
| **2** | 54 | CLONMEL STREET | OPEN | 53.336021 | -6.262980 | 4 | 29 | False | 2147483647 |
| **3** | 108 | AVONDALE ROAD | OPEN | 53.359405 | -6.276142 | 0 | 35 | False | 2147483647 |
| **4** | 56 | MOUNT STREET LOWER | OPEN | 53.337960 | -6.241530 | 6 | 34 | False | 2147483647 |

In [32]:
```
df.tail()
```

Out[32]:

| | number | address | status | position_lat | position_lng | bikes | bike_stands | banking | last_update |
|---|---|---|---|---|---|---|---|---|---|
| **446457** | 39 | WILTON TERRACE | OPEN | 53.332383 | -6.252717 | 11 | 9 | False | 1649171493000 |
| **446458** | 83 | EMMET ROAD | OPEN | 53.340714 | -6.308191 | 8 | 32 | False | 1649171288000 |
| **446459** | 92 | HEUSTON BRIDGE (NORTH) | OPEN | 53.347802 | -6.292432 | 0 | 40 | False | 1649171418000 |
| **446460** | 21 | LEINSTER STREET SOUTH | OPEN | 53.342180 | -6.254485 | 6 | 24 | False | 1649171164000 |
| **446461** | 88 | BLACKHALL PLACE | OPEN | 53.348800 | -6.281637 | 10 | 20 | False | 1649171486000 |

In [33]:
```
df["number"] = df ["number"] #assigning each column to a name
df["last_update"] = df["last_update"]
df["bikes"] = df["bikes"]
df["address"] = df["address"]
df["status"] = df["status"]
df["position_lat"] = df["position_lat"]
df["position_lng"] = df["position_lng"]
df["bike_stands"] = df["bike_stands"]
df["banking"] = df["banking"]
df["last_update"] = df["last_update"]
```

In [34]:
```
df.dtypes #check dtypes for model training incompatibilities
```

Out[34]:
```
number           int64
address         object
status          object
position_lat   float64
position_lng   float64
bikes            int64
bike_stands      int64
banking           bool
last_update      int64
dtype: object
```

In [35]:
```
df.drop(df.index[df["last_update"] == 2147483647], inplace=True) #dropped set of duplicate values
```

In [36]:
```
df['last_update'] = df['last_update'].astype(str).str[:-3].astype(np.int64) #shortened the last update to seconds
```

In [37]:
```
df.head()
```

Figure 4.2: Cleaning Data

Out[47]: <AxesSubplot:>

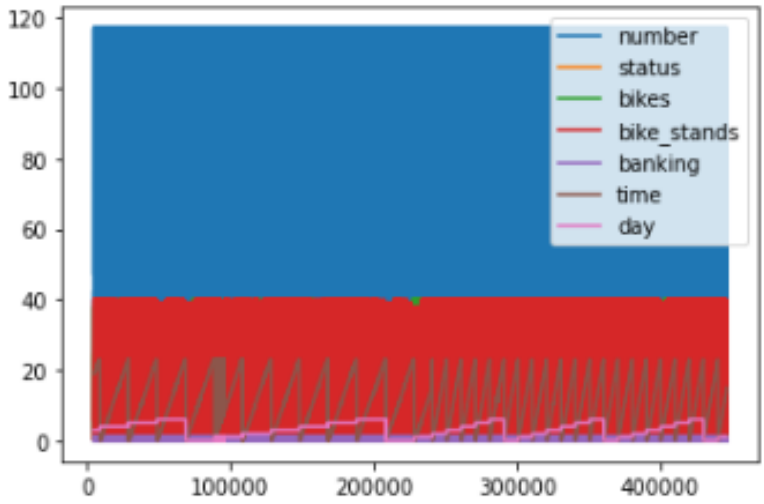



Figure 4.3: Results

models and compared results. We felt that this result was quite poor and decided to attempt a different model. So we went and tried to train our model and learned about the Forest Regression model We learned this in a different module called Data Analytics. The difference between the two is that in Forest Regression, expanding decision trees are constructed to account for data variability. Unlike Linear Regression, Forest Regression supports non linearity when searching the data set, this seems to be ideal for a dataset that has a considerable amount of noise and a large search area. Some observations

made in terms of running is that I noticed interestingly that Forest Regression took longer than Linear Regression (which in nature as it searches an entire dataset would typically take longer to run). I put that down perhaps to the Forest Regressor having to sort data into branches.

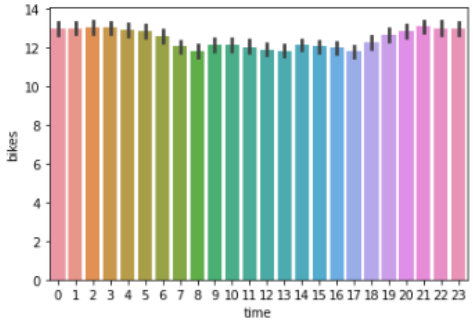
In terms of the rationale, it was very straightforward, the plan was to predict the number of bikes and slots available and use an estimator like OLS regression to decide on this. After much deliberation, we ended up predicting bike availability against just the day and time. We didn't incorporate the whether as we felt the temperature variability for the month of March was too slim, and because at night the temperatures could fall quite low yet the number of bikes available to use would be almost at full capacity which would negatively impact accuracy as it wouldn't reflect the overall tendency of usage. One thing to mention is that if we had data like whether it was raining or overcast etc. there would've been a broader scope of variables to predict

In [221…

```
#plot of the overarching availability predictions of bikes at all stations on a monday
sns.barplot(data=monday,x='time',y='bikes')
```

Out[221…

```
<AxesSubplot:xlabel='time', ylabel='bikes'>
```




In [103…

```
X = tuesday[features]
y = tuesday.bikes

X_train, X_test,y_train,y_test = train_test_split(X, y, test_size=0.3,random_state=0)

regressor = RandomForestRegressor(n_estimators=30)
regressor.fit(X_train, y_train)

with open('tuesday.pkl', 'wb') as handle:
    pickle.dump(regressor, handle, pickle.HIGHEST_PROTOCOL)


y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

print(df)
```

```
        Actual  Predicted
377908      27  11.923180
93835        7  12.760427
101275       9  12.149855
92094       20  13.172037
380262      10  13.528273
...        ...        ...
96383       20  12.115923
373870       5  12.086023
307342       9  12.149855
308103       7  11.923180
236653      10  12.112823

[17573 rows x 2 columns]
```

In [201…

```
sns.barplot(data=tuesday,x='time',y='bikes')
```

Out[201…

```
<AxesSubplot:xlabel='time', ylabel='bikes'>
```

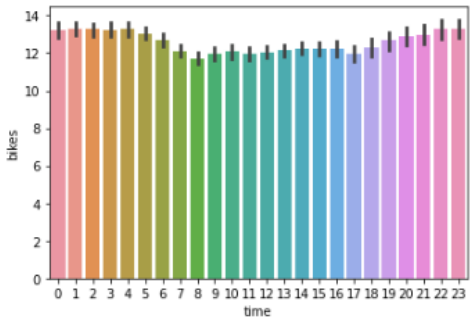



Figure 4.4: Results from random forest regression

# 5 Issues/Errors/Problems

## 5.1 Did anything go wrong?

In this project there were some issues that arose, and we had always mentioned and documented these. As myself, Zaur, being the note-taker and document creator, I felt the need to ensure everything we ran into (road bumps) were recorded and saved photos so that we could communicate this to Eoin, our product owner. He had recommended, in our final product meet that I make note and write it down in the report in a final remarks section with details of the issues we faced so that it shows we were aware of the problem and can make sense of it. These problems are documented in this section.

One of the problems we faced was the first EC2, where we ran out of data, and I was charged 6 dollars on my debit card. This was a pain and migration onto a new server, and full set up of RDS and etc took a few hours. It took me 2-3 hours setting up the flask and the server on the new EC2. The charge/issue can be seen in Figure 4.2

One other issue that was faced was the OpenWeather API, during testing phases and setting up, we ran into an issue where the API key was terminated as we went over 674 requests per minute, we had no idea how this occurred. It was especially strange as it happened 1 day before submission when everything was being finalized. In figure 4.5 you can see this. A workaround for this is Zaur created another API key on his account, and we updated the key with this log. The old key was e857655954f34ae188982244bbb23b21 and the new key is 57c745b64650ac91012e9b0215ac7b1e. Figure 4.3 and Figure 4.4 show a bit more detail into this problem with limitations (reason for being banned/blocked and the generation of a new key).

The final issue we had came with the EC2 flask instance, where it would not allow us to host our latest app, and issues ran in where we could not get the geolocation to pop up, we believe this was because of the https ssl protocol that does not allow it, and since our site runs on the http protocol instead of https, the geolocation was not working as intended, but this still can be accessed and used via the local machine.

We ended up writing the Javascript to chart the data within the .html file as it became

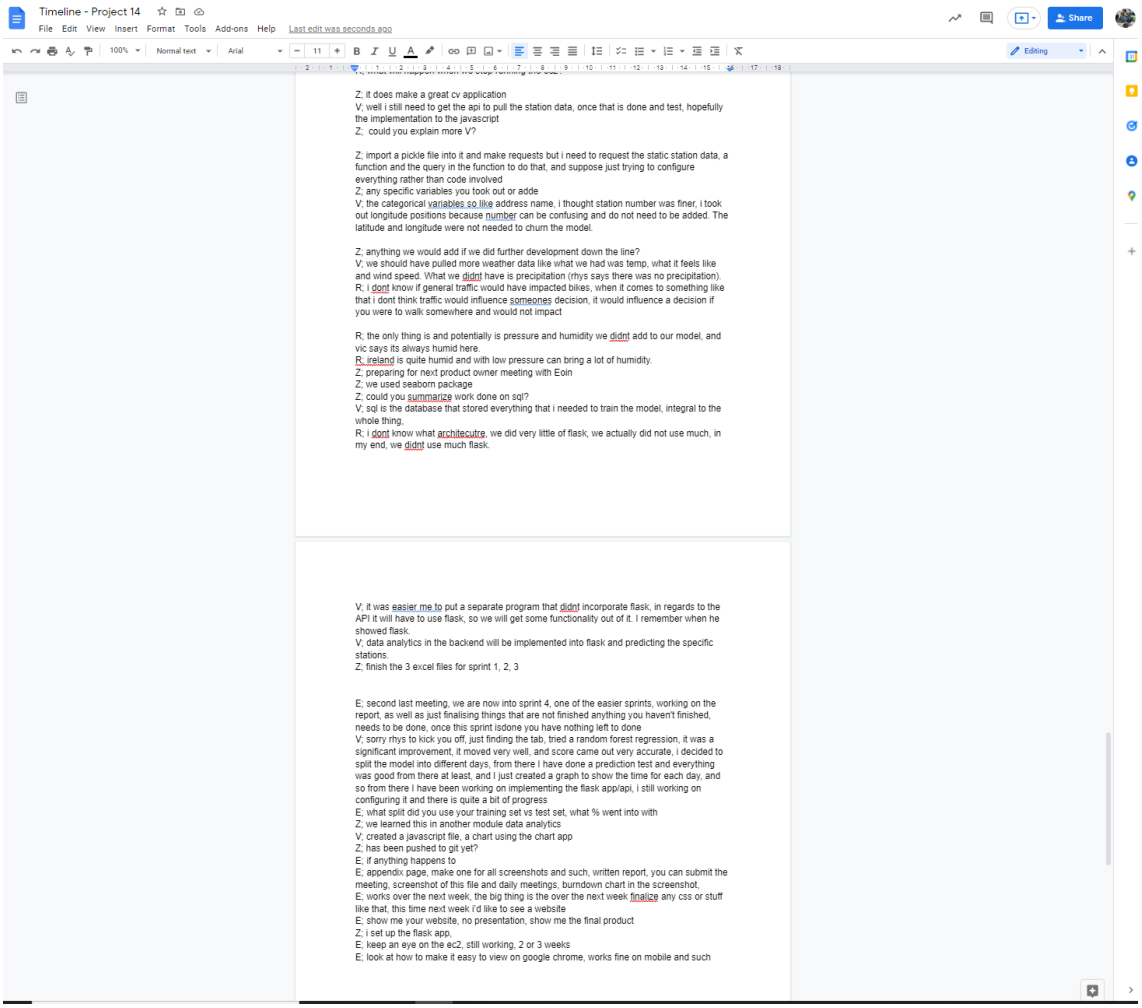



Figure 5.1: Last Meeting - Scrum

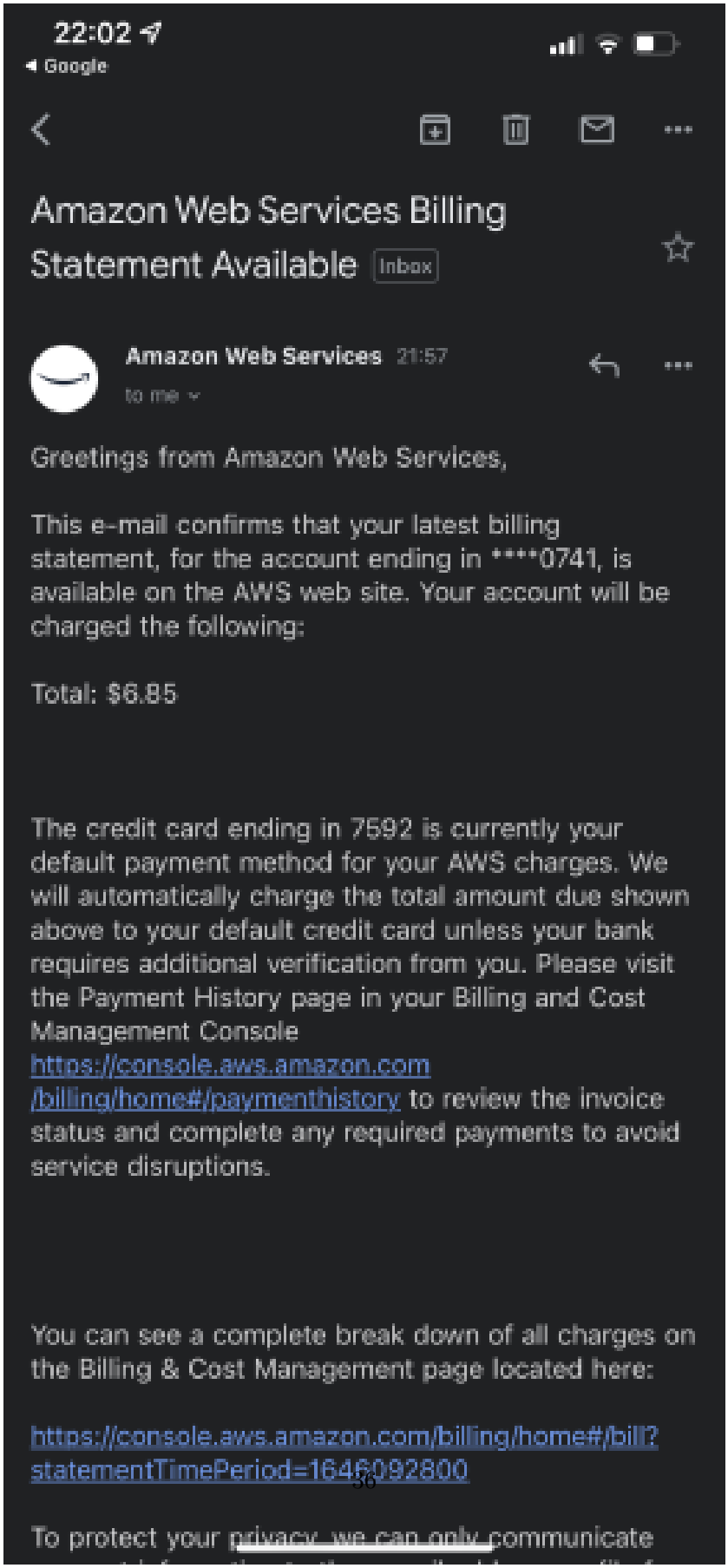



Figure 5.2: Charged for running out of free tier - Zaur


| First name | Last name | Email | Duration | Time joined | Time exited |
|---|---|---|---|---|---|
| Zaur | Gouliev | zaur.gouliev@ucdconnect.ie | 3 hrs 49 mins | 19:02 | 22:52 |
| Victoria | Keane | victoria.keane@ucdconnect.ie | 3 hrs 45 mins | 19:06 | 22:52 |
| Rhys | O Dowd | rhys.odowd@ucdconnect.ie | 2 hrs 7 mins | 20:39 | 22:46 |

Figure 5.3: Final scrum meet - 3/4 hour discussion on problems/fixes/reviewing code/discussion on report and how to submit/present

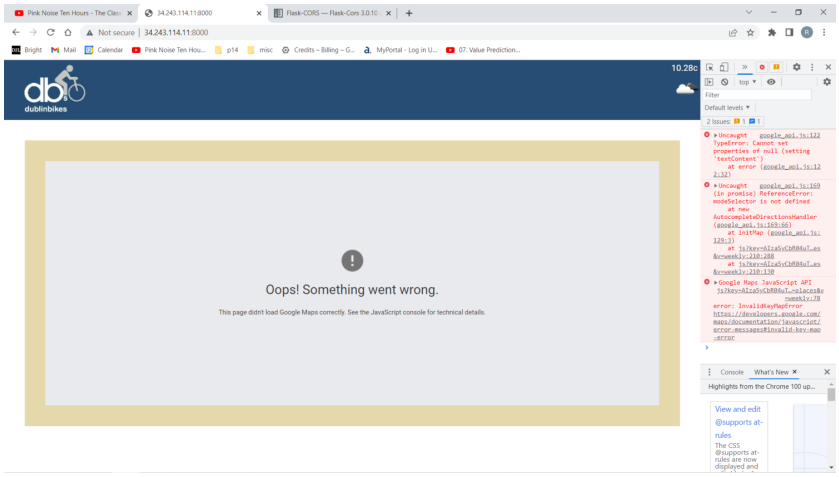
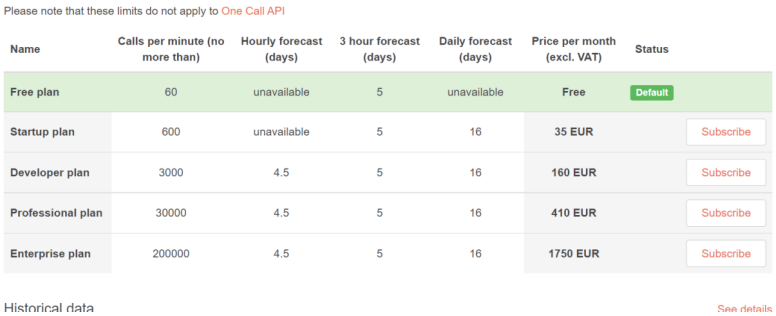



Figure 5.4: Error on API Key, would not allow the maps to load



Please note that these limits do not apply to One Call API

| Name | Calls per minute (no more than) | Hourly forecast (days) | 3 hour forecast (days) | Daily forecast (days) | Price per month (excl. VAT) | Status | |
|---|---|---|---|---|---|---|---|
| Free plan | 60 | unavailable | 5 | unavailable | Free | Default | |
| Startup plan | 600 | unavailable | 5 | 16 | 35 EUR | | Subscribe |
| Developer plan | 3000 | 4.5 | 5 | 16 | 160 EUR | | Subscribe |
| Professional plan | 30000 | 4.5 | 5 | 16 | 410 EUR | | Subscribe |
| Enterprise plan | 200000 | 4.5 | 5 | 16 | 1750 EUR | | Subscribe |

Historical data See details

Figure 5.5: Limits on API key

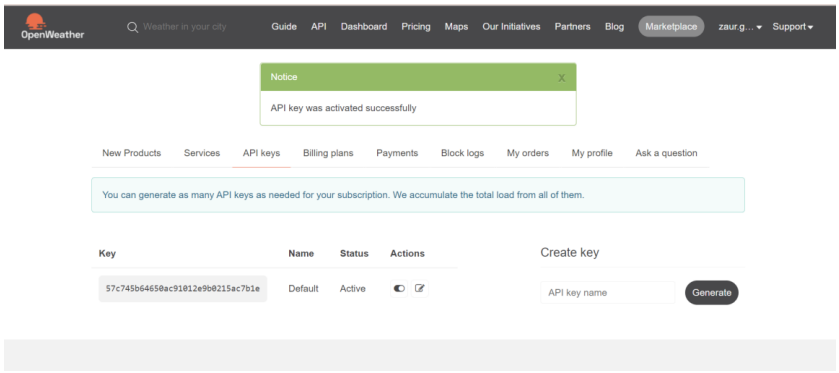



Figure 5.6: New OpenWeather APIkey - Zaur

apparent that the chart module heavily dislikes the separation and we kept receiving errors. This was not idea for a clean code perspective, but it was a workaround that worked.

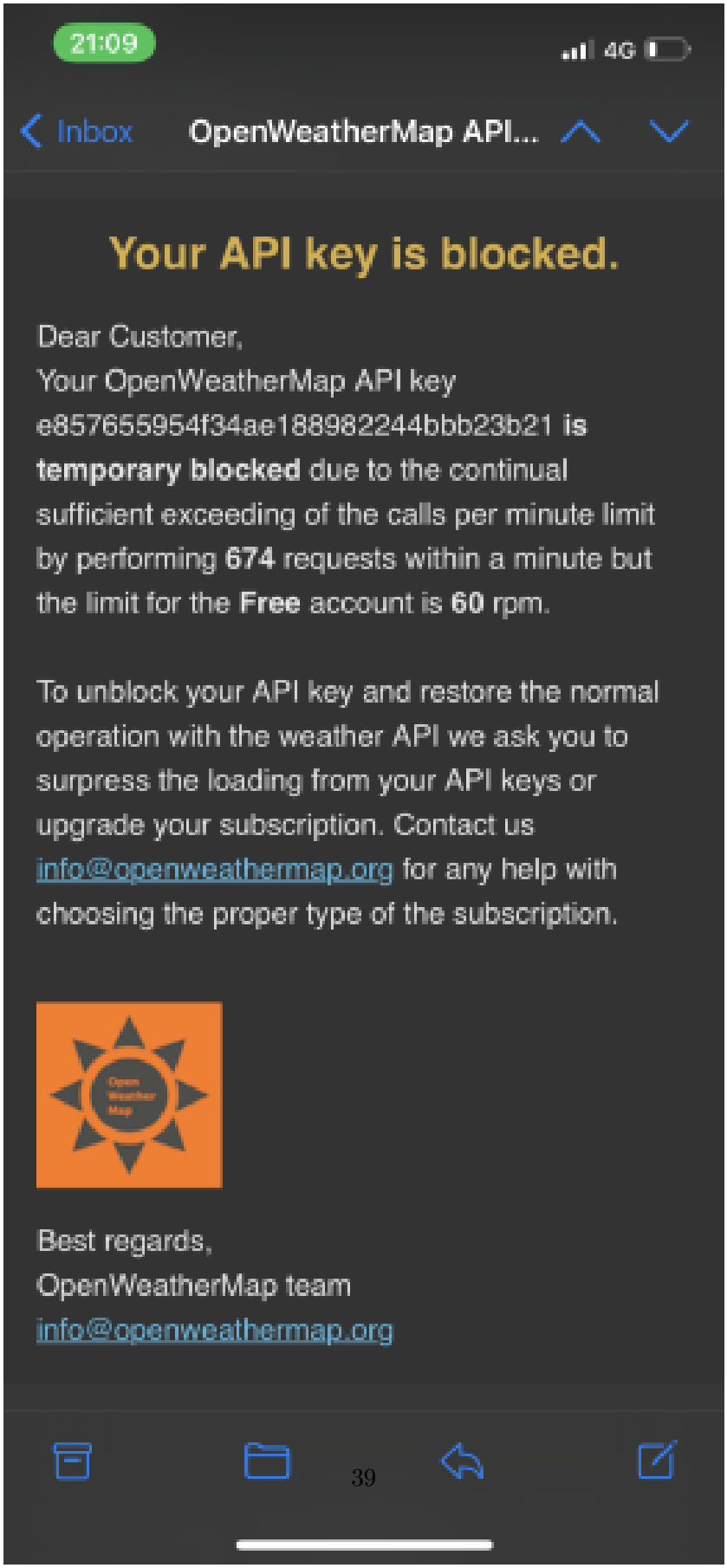





Figure 5.7: OpenWeather API banned - Victoria

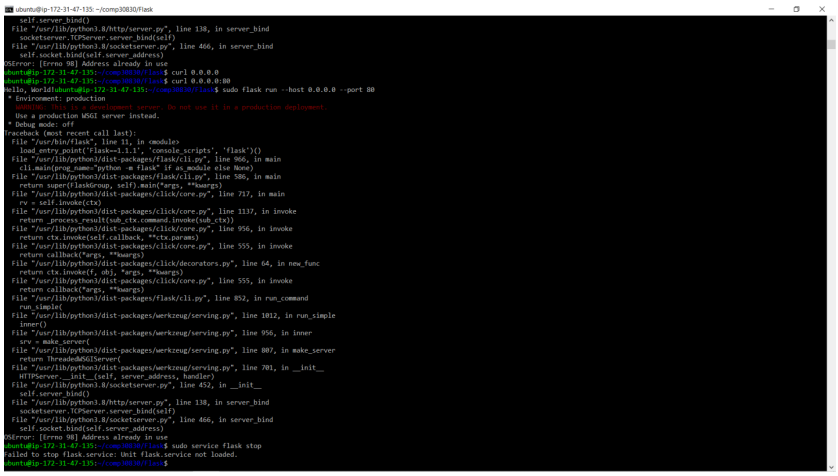



Figure 5.8: Issues with getting app.py on the EC2 (it kept showing an older version)


Figure 5.9: Issues with getting app.py on the EC2 (it kept showing an older version)

# 6 Appendix/References

## 6.1 Appendix

.

.

.

.

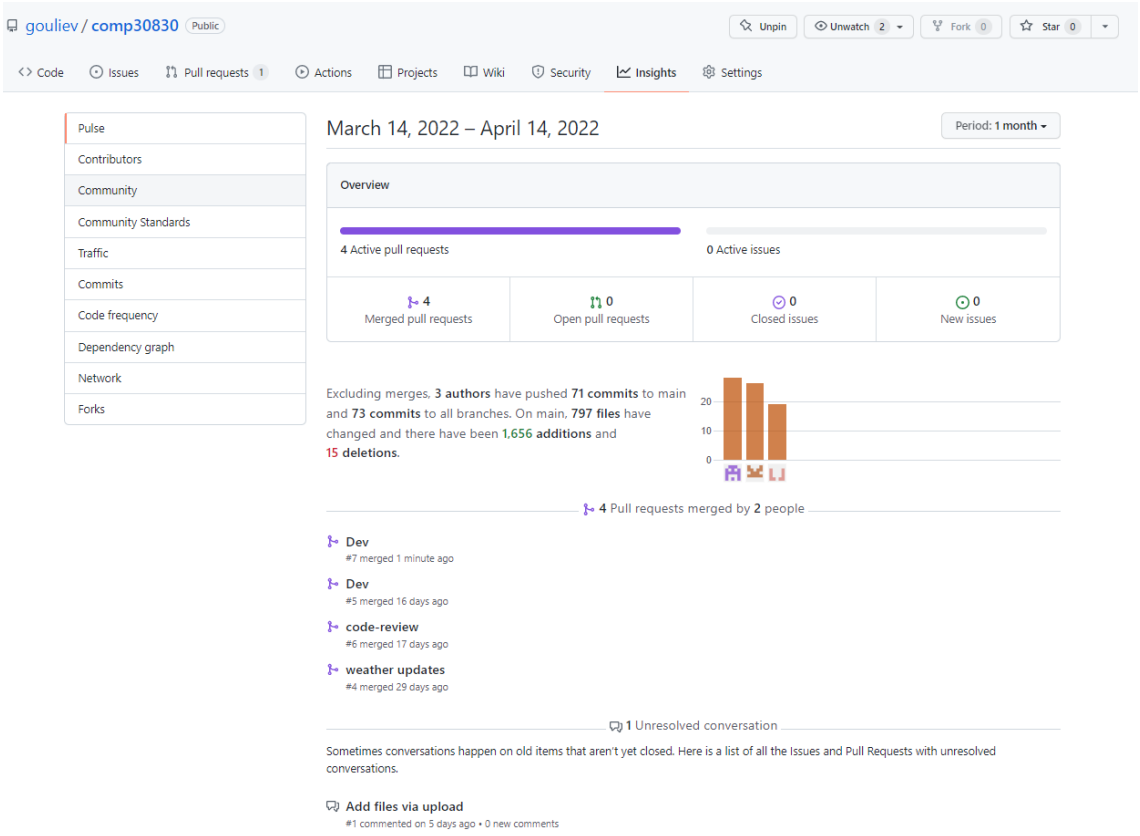



Figure 6.1: Github Appendix

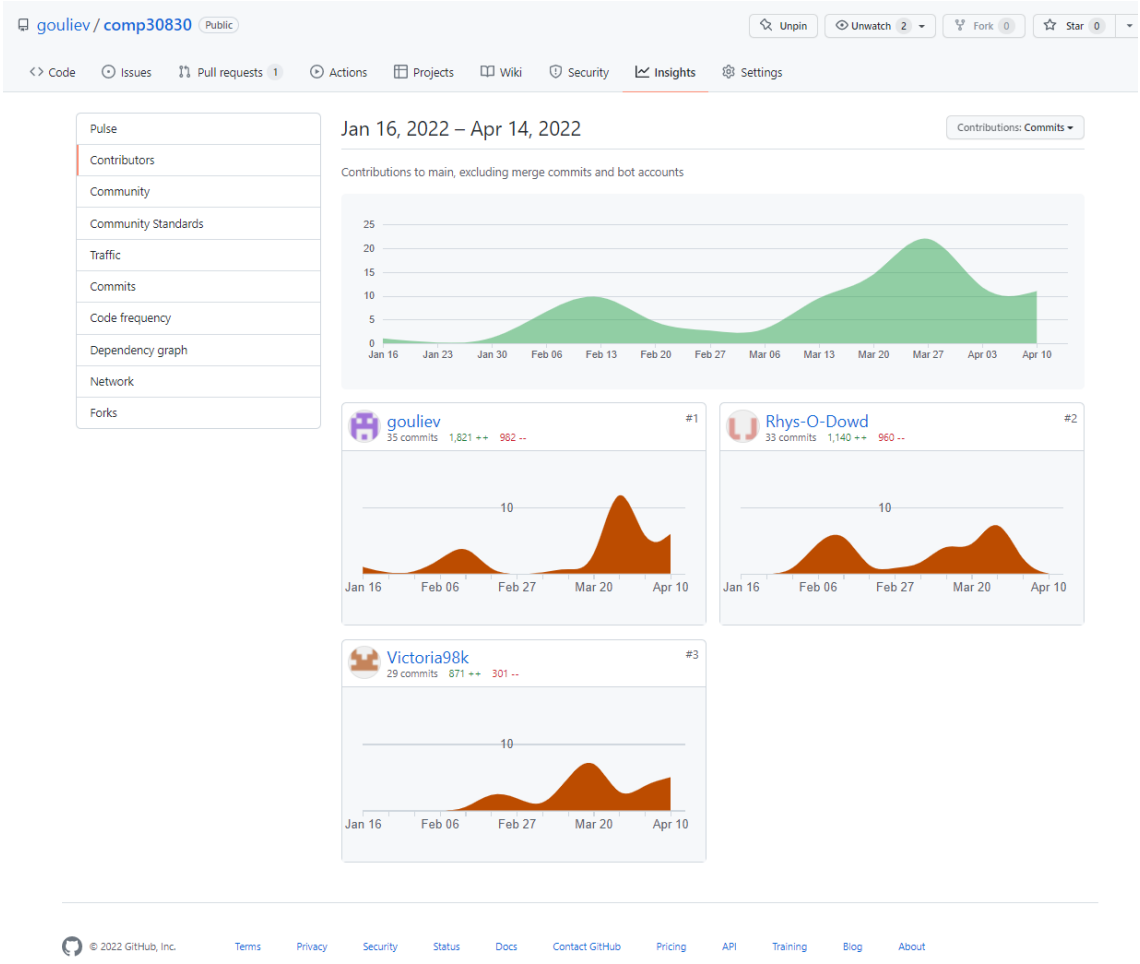



Figure 6.2: Github Appendix

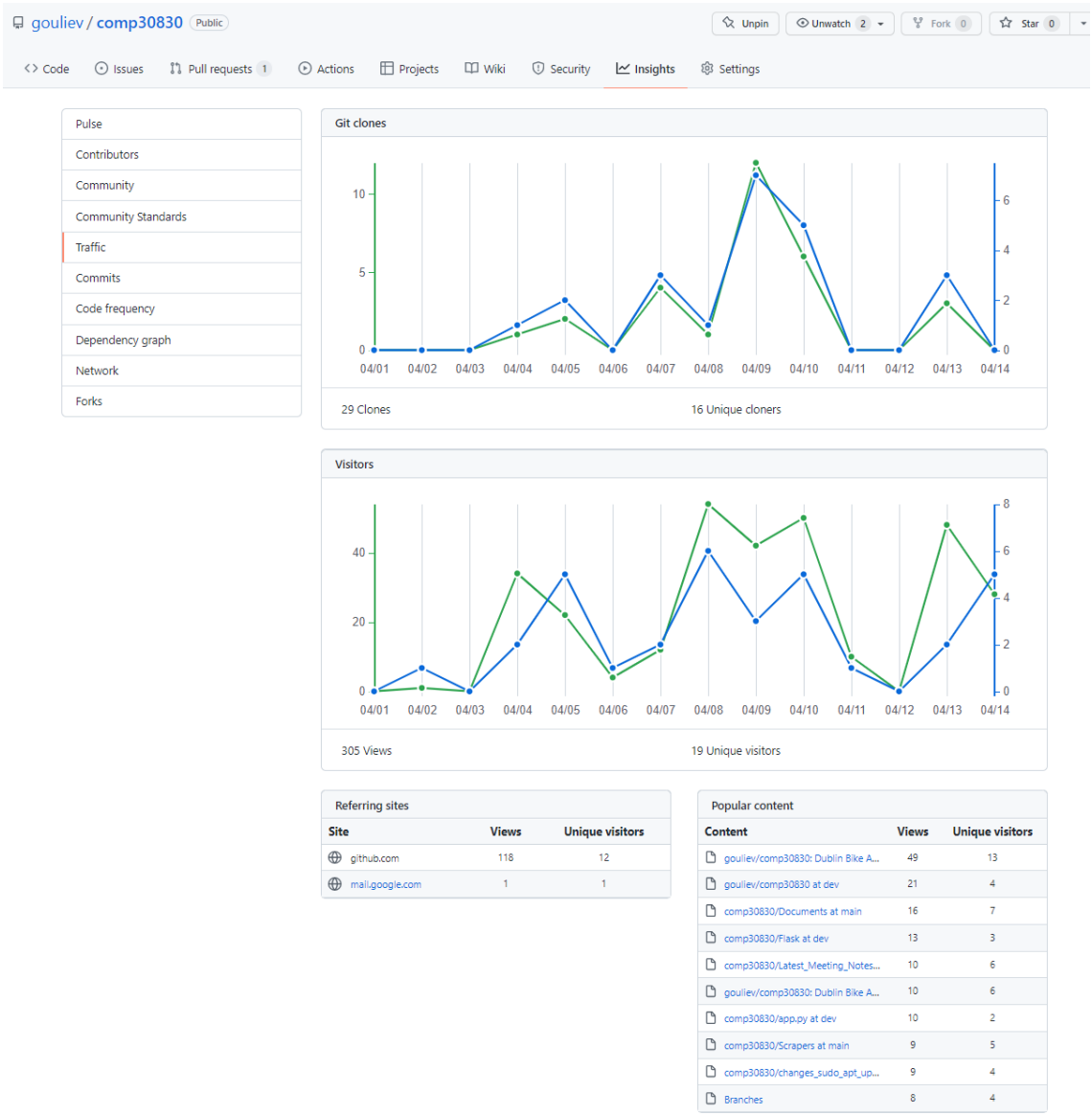



Figure 6.3: Github Appendix

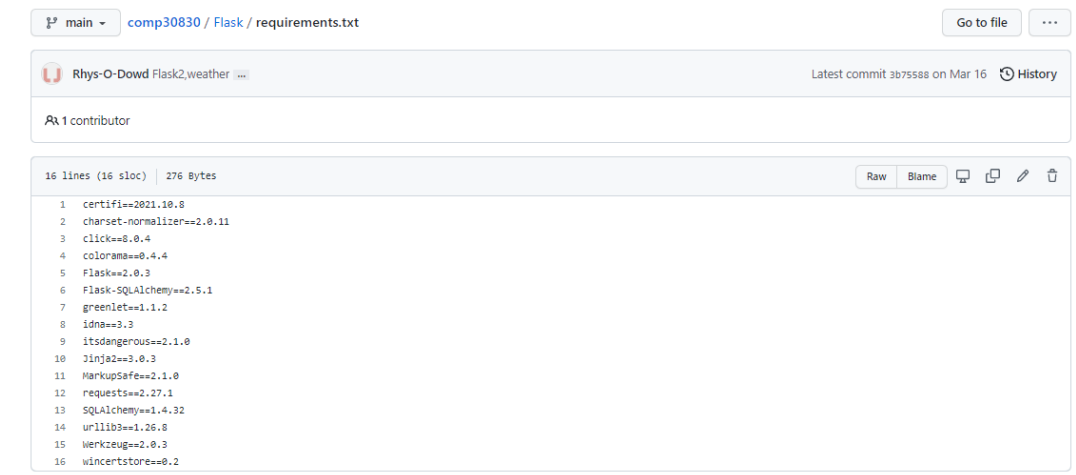



Figure 6.4: Github Appendix

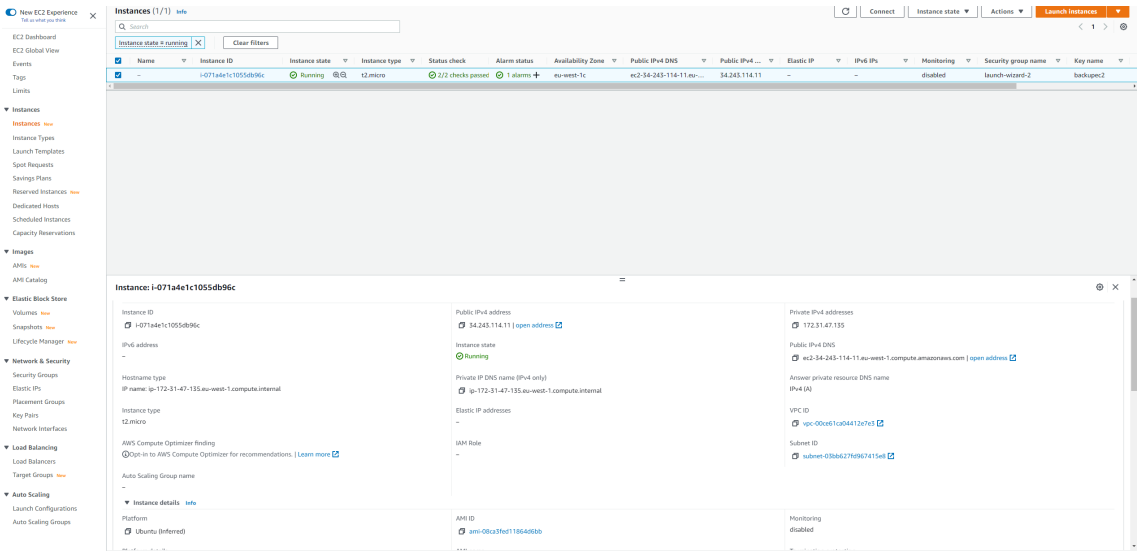



Figure 6.5: EC2 appendix

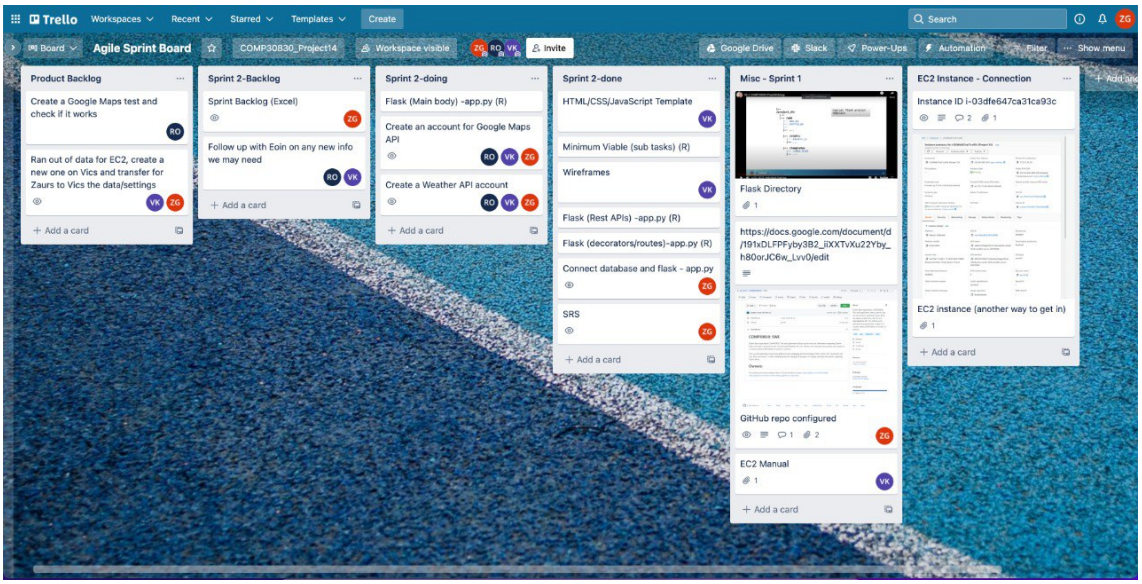



Figure 6.6: E.g. Sprint 3

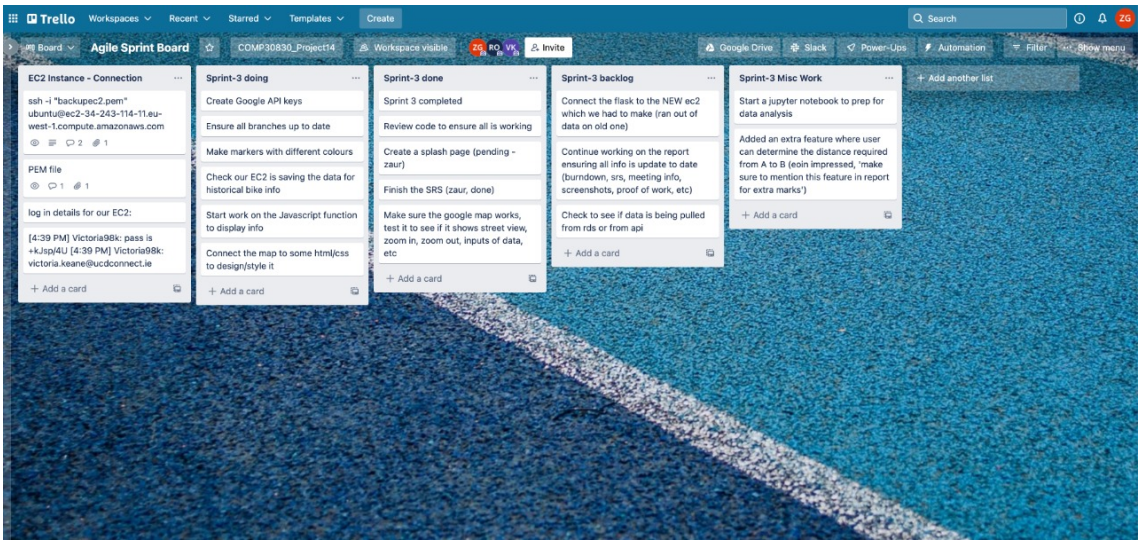



Figure 6.7: E.g. Sprint 3

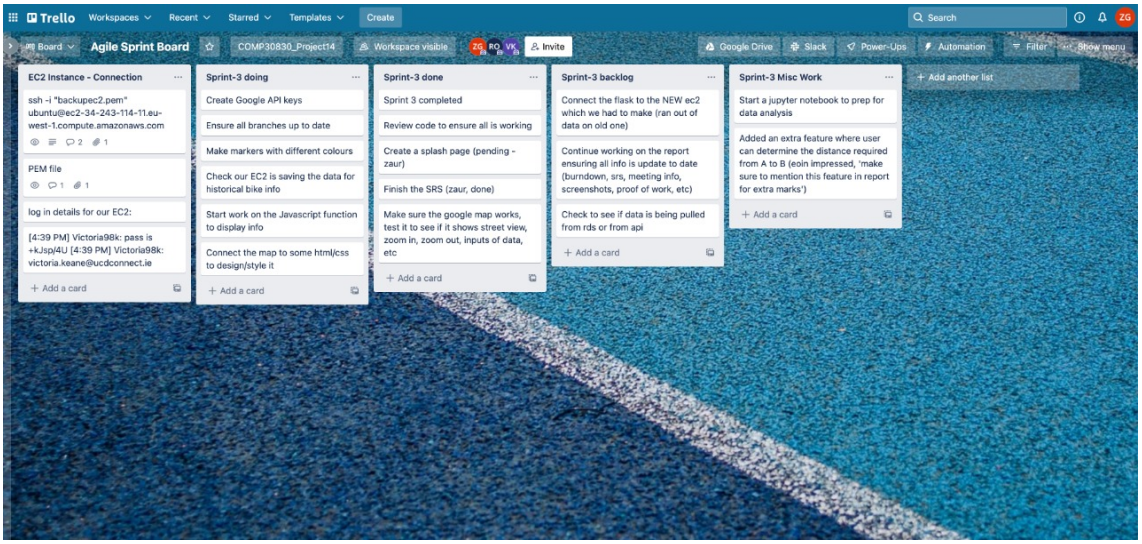



Figure 6.8: E.g. Sprint 3

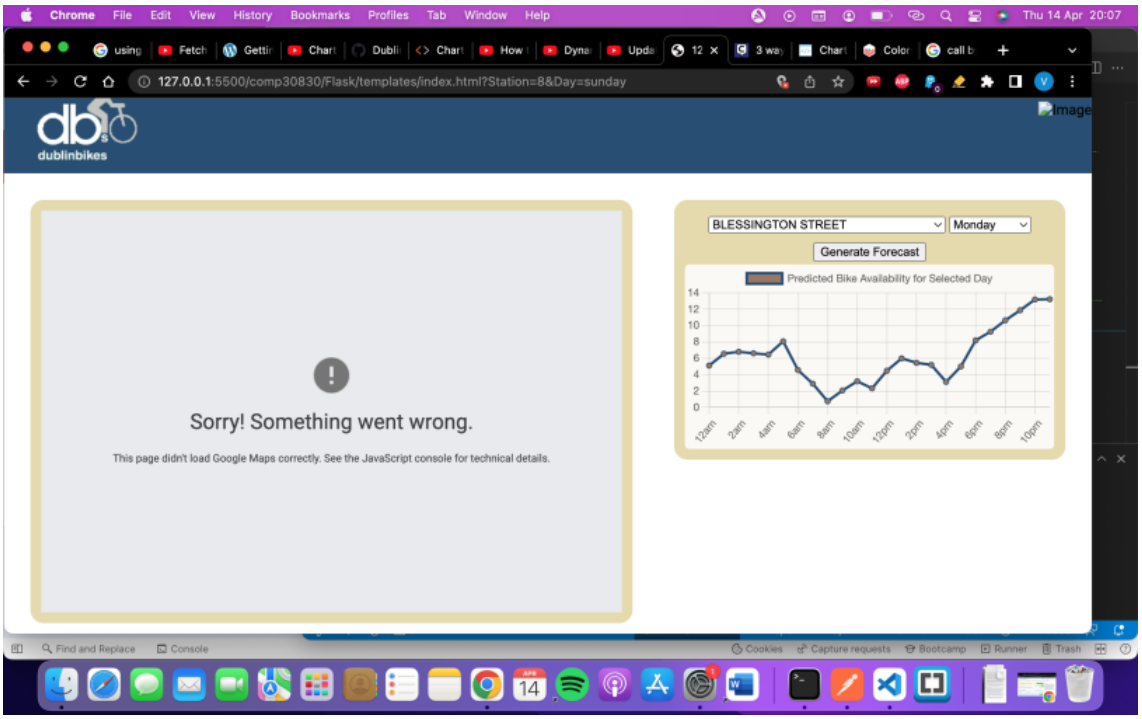



Figure 6.9: Final design proposal

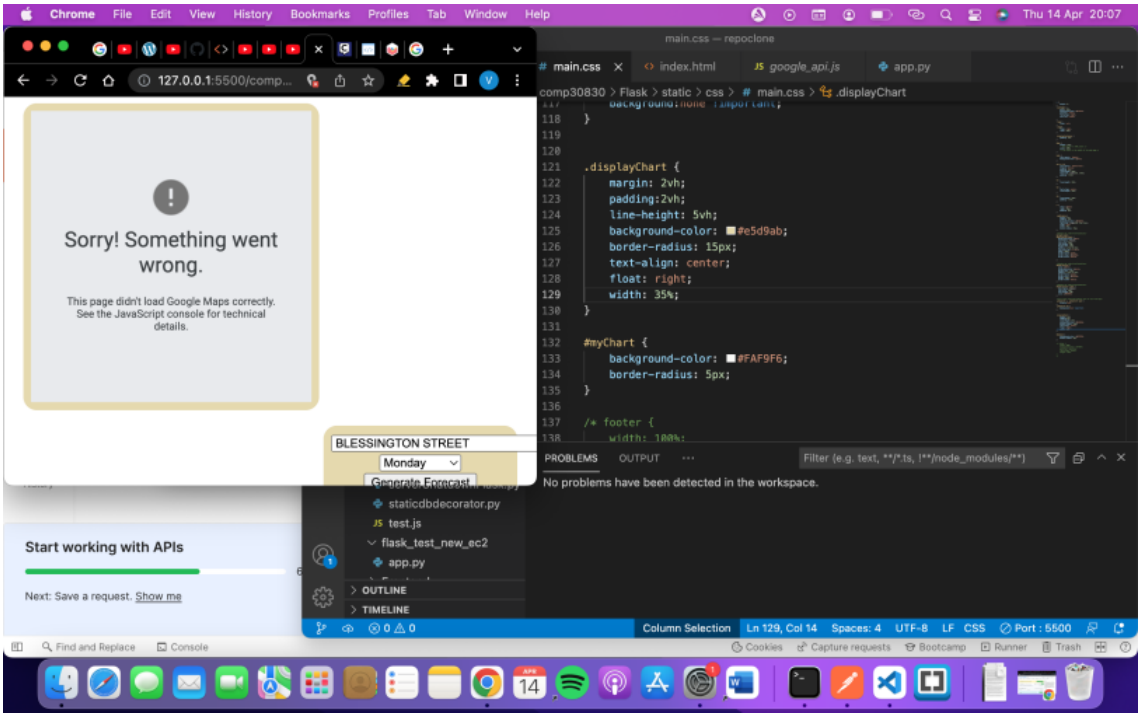



Figure 6.10: Final design proposal