

A Project Report  
on  
**“WEB APPLICATION OF DUBLIN BIKES”**



**COMP30830: SOFTWARE ENGINEERING (CONV)**  
**UCD SCHOOL OF COMPUTER SCIENCE**

By  
Team Pedal Predictors

Sarvesh Gadgil	19200380
Chaitali Keni	19200385
Swetha Bondhikrishnamurthy	19200493

Product Owner: Ms. Shreya Tadas  
Under the Guidance of Dr. Aonghus Lawlor

Date: 19/04/2020

# SUBMISSION NOTES AND OVERALL CONTRIBUTION

This report summarizes project overview, process, architecture, analytics, design and various efforts taken with an attempt to fully satisfy the requirements regarding the Group Project for COMP30830 – Software Engineering (Conv) module offered at UCD School of Computer Science for the Spring Semester 2020.

Overall contribution signed off by all the team members are listed below:

- Sarvesh Gadgil (19200380) – 46%
- Chaitali Keni (19200385) – 27%
- Swetha Bondhikrishnamurthy (19200493) – 27%

# Contents

<b>1. PROJECT OVERVIEW .....</b>	<b>1</b>
<b>1.1. INTRODUCTION .....</b>	<b>1</b>
<b>1.2. OBJECTIVES .....</b>	<b>1</b>
<b>1.3. TARGET USERS .....</b>	<b>1</b>
<b>1.4. STRUCTURE .....</b>	<b>1</b>
<b>1.5. FEATURES .....</b>	<b>1</b>
<b>2. PROCESS.....</b>	<b>3</b>
<b>2.1. MEMBERSHIP .....</b>	<b>4</b>
<b>2.2. SPRINT 1 [ 10<sup>TH</sup> FEBRUARY 2020 – 21<sup>ST</sup> FEBRUARY 2020 ] .....</b>	<b>4</b>
<b>2.2.1. SPRINT PLANNING MEETING 11<sup>th</sup> February 2020 09:00 – 11:00 .....</b>	<b>4</b>
<b>2.2.2. SCRUM MEETING 13<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>5</b>
<b>2.2.3. SCRUM MEETING 14<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>5</b>
<b>2.2.4. SCRUM MEETING 17<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>5</b>
<b>2.2.5. SCRUM MEETING 18<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>5</b>
<b>2.2.6. SCRUM MEETING 19<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>5</b>
<b>2.2.7. PROBLEMS FACED: .....</b>	<b>5</b>
<b>2.2.8. SPRINT REVIEW 20<sup>th</sup> February 2020 09:00 – 11:00 .....</b>	<b>6</b>
<b>2.3. SPRINT 2 [ 24<sup>TH</sup> FEBRUARY 2020 – 6<sup>TH</sup> MARCH 2020 ] .....</b>	<b>7</b>
<b>2.3.1. SPRINT PLANNING MEETING 25<sup>th</sup> February 2020 09:00 – 11:00 .....</b>	<b>7</b>
<b>2.3.2. SCRUM MEETING 27<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>8</b>
<b>2.3.3. SCRUM MEETING 3<sup>rd</sup> February 2020 09:00 – 11:00.....</b>	<b>8</b>
<b>2.3.4. PROBLEMS FACED: .....</b>	<b>8</b>
<b>2.3.5. SPRINT REVIEW 20<sup>th</sup> February 2020 09:00 – 11:00.....</b>	<b>9</b>
<b>2.4. SPRINT 3 [ 23<sup>RD</sup> MARCH 2020 – 3<sup>RD</sup> APRIL 2020 ] .....</b>	<b>10</b>
<b>2.4.1. SPRINT PLANNING MEETING 24<sup>th</sup> March 2020 09:00 – 11:00.....</b>	<b>10</b>
<b>2.4.2. SCRUM MEETING 31<sup>st</sup> March 2020 09:00 – 11:00 .....</b>	<b>10</b>
<b>2.4.3. SCRUM MEETING 1<sup>st</sup> April 2020 09:00 – 11:00.....</b>	<b>10</b>
<b>2.4.4. PROBLEMS FACED: .....</b>	<b>10</b>
<b>2.4.5. SPRINT REVIEW 2<sup>nd</sup> April 2020 09:00 – 11:00 .....</b>	<b>10</b>
<b>2.5. SPRINT 4 [ 6<sup>TH</sup> APRIL 2020 – 17<sup>TH</sup> APRIL 2020 ] .....</b>	<b>12</b>
<b>2.5.1. SPRINT PLANNING MEETING 7<sup>th</sup> April 2020 09:00 – 11:00 .....</b>	<b>12</b>
<b>2.5.2. SCRUM MEETING 9<sup>th</sup> April 2020 09:00 – 11:00 .....</b>	<b>12</b>
<b>2.5.3. SCRUM MEETING 14<sup>th</sup> April 2020 09:00 – 11:00 .....</b>	<b>12</b>
<b>2.5.3. PROBLEMS FACED: .....</b>	<b>12</b>
<b>2.5.4. SPRINT REVIEW 16<sup>th</sup> April 2020 09:00 – 11:00 .....</b>	<b>13</b>

2.6. BURN DOWN CHART .....	14
3. ARCHITECTURE.....	17
3.1. BLOCK DIAGRAM .....	17
3.2. TECHNOLOGIES .....	17
3.2.1. GITHUB REPOSITORY .....	17
3.2.2. AMAZON AWS SERVER: .....	20
3.2.3. Virtual Environment:.....	22
3.2.4. FRONT END .....	23
3.2.5. BACKEND: .....	24
3.2.5.1. SCRAPERS: .....	24
3.2.5.2. FLASK APPLICATION: .....	24
3.2.5.3. Database Connection .....	24
3.2.5.4. Web service .....	25
3.2.5.5. AMAZON RDS MYSQL DATABASE.....	25
3.2.6. API:.....	26
4. ANALYTICS.....	28
4.1. MODEL TRAINING AND BUILDING .....	28
4.2. ML INTEGRATION WITH FRONT-END AND BACK-END .....	28
5. DESIGN.....	30
5.1. MOCKUPS .....	30
5.2. WORKING .....	32
6. FUTURE WORK:.....	34
6.1. IMPROVEMENTS:.....	34
6.2. REFINEMENTS: .....	34
6.3.OBSERVATIONS: .....	34
7. LINKS .....	34



## FIGURES

Figure 1: Sprint Planning for Sprint 1 .....	4
Figure 2: Doubts Card .....	4
Figure 3: Trello Board for Sprint 1.....	5
Figure 4: Trello Board after Completion of Sprint 1 .....	6
Figure 5: Trello Board for Sprint 2.....	7
Figure 6: Scrum Meeting.....	8
Figure 7: Completion of Web Application after Sprint 2 .....	9
Figure 8: Trello Board after Completion of Sprint 2 .....	9
Figure 9: Trello Board for Sprint 3.....	10
Figure 10: Trello Board after Completion of Sprint 3 .....	11
Figure 11: Trello Board for Sprint 4.....	12
Figure 12: Trello Board after Completion of Sprint 4 .....	13
Figure 13: Burn Down of entire Project .....	15
Figure 14: Burn Down Chart of Project.....	16
Figure 15: Block Diagram of Web Application .....	17
Figure 16: Issues created in GitHub .....	18
Figure 17: Commits on Branch.....	18
Figure 18: Commits on Branch.....	19
Figure 19: Commits on Master .....	19
Figure 20: Sh file for bike scraper .....	20
Figure 21: Sh file for weather scraper.....	21
Figure 22: Log file for bike scraper.....	21
Figure 23: Log file for weather scraper .....	22
Figure 24: Virtual Environment.....	22
Figure 25: HTML file .....	23
Figure 26: ER Diagram of Database.....	25
Figure 27: Screenshot of data obtained from Bikes API .....	26
Figure 28: Screenshot of data obtained from Open Weather API.....	27
Figure 29: ML Model.....	28
Figure 30: Mockup 1 .....	30
Figure 31: Final Mockup.....	31
Figure 32: Web Application.....	33
Figure 33: Web Application with Visualization of Hourly Data.....	33

## 1. PROJECT OVERVIEW

### 1.1. INTRODUCTION

To be fit and healthy, physical activity is very essential. Physical activity can protect you from serious diseases like obesity, mental illness etc. Cycling is one of the best ways to reduce the risk of health problems. Moreover, cycling is enjoyed by people of all ages ranging from young children to old adults. Riding to work or other places using a bicycle is fun, good for the environment and allows a hassle-free commute.

Dublin Bikes, a public bicycle rental scheme which became operational from 2009 is used by many of Dubliners every day. Its expansion was also announced recently as the city of Dublin is expanding. Dublin Bikes is a web application which provides real time prediction regarding the availability of bikes and bike stands based on a particular date. It also provides the weather forecast along with displaying the routes. Using this application, the users are able to look at available bikes/available bike stands and plan their journey accordingly.

### 1.2. OBJECTIVES

The primary objective of this project was to create a website which was very simple and easy to use but also very intuitive so that a first-time user should not have any difficulty in using the website. The users shall be able to select a start station, destination station and start date. Using these fields, the website shall display the weather forecast for that particular day along with available bikes and available bike stands for each of the station based on the weather. Moreover, the website shall also display a bicycle route from start station to destination station along with total distance and duration of the journey. The users shall also be able to see a graphical view of the hourly and weekly bike availability for planning in advance.

### 1.3. TARGET USERS

The target audience for this web application are the citizens of Dublin along with the tourists i.e. users which are new to the city of Dublin. These types of users may regularly be using the Dublin Bikes and knows their required stations or may completely be new to the concept of rental bikes.

### 1.4. STRUCTURE

The website comprises of single page application with the search fields and the map placed adjacent to one another. This type of structure makes the application very informative as the user has a birds-eye view on the weather forecast, bikes and station availability, the route, the total distance and duration of the journey. Moreover, it also has an about section which gives some extra information to the user regarding stations, bikes, and the procedure to rent and use the bikes.

### 1.5. FEATURES

- Ability to pick any station from the marker and specify it as a source or a destination station
- Auto suggestions for places and station names whenever searching is done by user via search fields

- Displaying the multiple nearest stations in form of a list for a place searched by user via search fields
- Ability to pick any station from the above mentioned multiple nearest stations list
- Displaying a bicycle route from source station to destination station
- Displaying total distance and duration of the journey
- Predicting available bikes and stands for future dates (on the basis of weather)
- Predicting weather for future dates
- Display hourly and weekly availability of bikes in a graphical format

## 2. PROCESS

We adopted Agile methodology for this project in order to obtain high productivity and individual growth. The Agile methodology focuses on all aspects of an application like development, testing and completion of an application in small iterations. It also helps in improvement of the finished product as it has checks after completion of small parts of the project. This approach is client oriented as product owner has a chance to examine the product and make refinements. It eases the fixing of errors in the middle of the project.

The various methods associated with Agile are as follows:

- 1. Sprint Planning Meetings:**

This meeting takes place at the beginning of the sprint. This meeting is attended by product owner, ScrumMaster, and the entire Scrum Team. In this meeting, the product owner checks the work done in the previous sprint and informs the scrum team about the story points that are to be completed in the sprint along with the refinements to the product delivered in the previous sprint.

- 2. Sprints:**

Sprints consists of duration ranging from 2 to 4 weeks. In this project, we have four sprints of two weeks each. In this period a scrum team works on the story points explained in the sprint planning meeting. Each sprint has daily scrum meetings to keep the team updated about the progress of the week. The team has a burndown chart for each sprint in order to graphically represent the rate at which work is completed and how much work remains to be done.

- 3. Sprint Reviews:**

This meeting takes place at the end of the sprint and is attended by the product owner, ScrumMaster and the Team. The team gives a demo on the product and specifies the tasks that are completed in that sprint.

This is a summary of all the meetings that we had for this project. It consists of four sprints. It includes details about the work done, the work that is to be done in the sprints along with the bug fixes and changes made in the project. We used Trello Board to keep a track of the Scrum meetings, Sprint Plans and to check the progress of the project.



## Group formation 11<sup>th</sup> February 2020 09:00 – 11:00

### 2.1. MEMBERSHIP

Our team was finalized. The team members were Sarvesh Gadgil, Chaitali Keni and Swetha Bondhikrishnamurthy. Our group no was 21 and product owner assigned to us was Ms. Shreya Tadas (Demonstrator).

### 2.2. SPRINT 1 [ 10<sup>th</sup> February 2020 – 21<sup>st</sup> February 2020 ]

#### 2.2.1. SPRINT PLANNING MEETING 11<sup>th</sup> February 2020 09:00 – 11:00

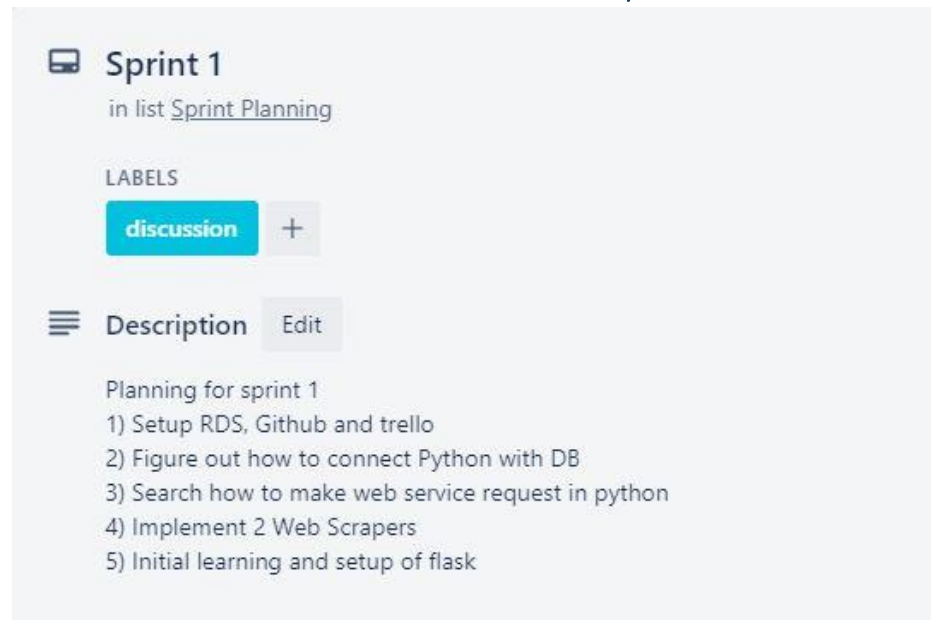


Figure 1: Sprint Planning for Sprint 1

We also had a card for Doubts on the Trello board.

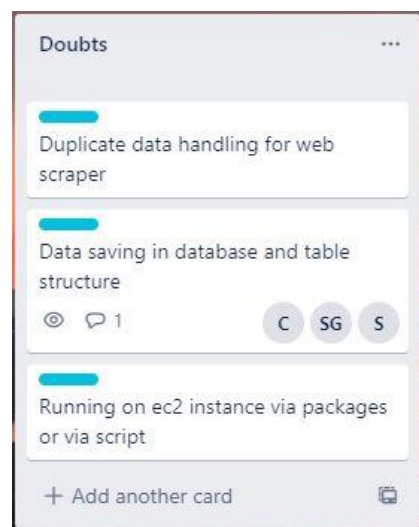


Figure 2: Doubts Card

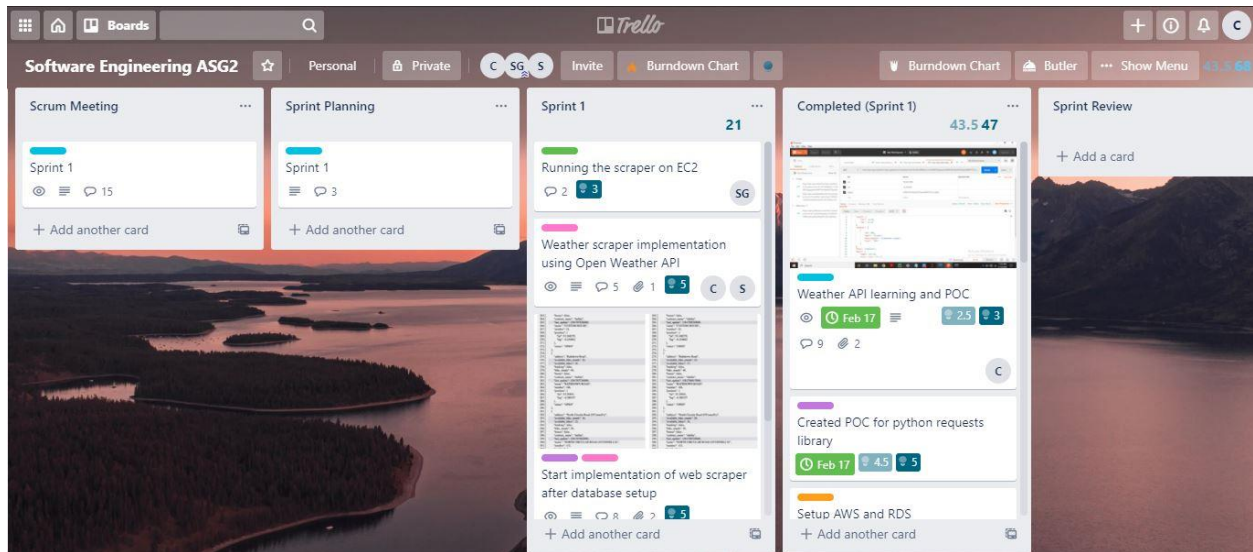


Figure 3: Trello Board for Sprint 1

### 2.2.2. SCRUM MEETING 13<sup>th</sup> February 2020 09:00 – 11:00

As this was our first meeting, we discussed regarding the main aim of the project. We also examined the Dublin Bikes website and figured out some of the limitations. We also discussed whether to write an SRS document or not.

### 2.2.3. SCRUM MEETING 14<sup>th</sup> February 2020 09:00 – 11:00

We discussed which software to use for SCRUM. We also distributed the work for researching about GitHub, AWS (EC2 and RDS) and JCDcaux.

### 2.2.4. SCRUM MEETING 17<sup>th</sup> February 2020 09:00 – 11:00

We discussed the points which we researched regarding the GitHub, AWS RDS and JCDcaux. We also figured out some of the issues which can arise during the setup of AWS RDS (like unable to connect remotely due to security group configuration). Finally, it was decided that Sarvesh will setup the GitHub repository and AWS instances (EC2 and RDS), Chaitali will setup the Trello Board and Swetha will look into JCDcaux.

### 2.2.5. SCRUM MEETING 18<sup>th</sup> February 2020 09:00 – 11:00

All the team members discussed their progress. GitHub repository, AWS instances and Trello Board were set up successfully. JCDcaux API response was also fetched with the help of key. We also discussed which weather API to use and started researching about different APIs.

### 2.2.6. SCRUM MEETING 19<sup>th</sup> February 2020 09:00 – 11:00

We ended up using the OpenWeather API. We distributed the tasks amongst ourselves and started working on weather scraper and bike scraper.

### 2.2.7. PROBLEMS FACED:

**Bike scraper:** Initially, we were inserting duplicate entries for static data in the database. We also faced some primary-foreign key relationship issues designing database.

**Weather scraper:** Initially, we had only one file for bikes data and weather. So, for each station we were inserting weather data every 5 minutes. But due to this, as we had free key, our requests were getting failed. This was because free key users can make small number requests at a given time. Thus, we separated our code and made two files: one for weather and one for bikes data.

**Scraper Hosting:** We used a while loop and sleep method to make the scraper wait for specific time. But the scraper process was stopping after some amount of time on server. To fix this, we used super user (sudo) to start the process.

## 2.2.8. SPRINT REVIEW 20<sup>th</sup> February 2020 09:00 – 11:00

Completed all the tasks allocated for Sprint 1.

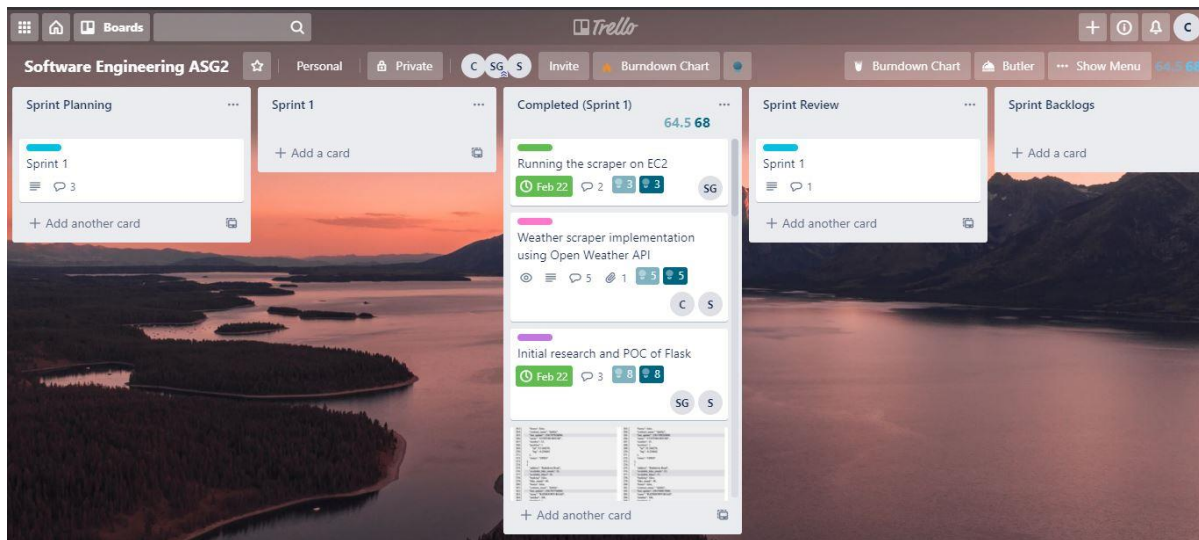


Figure 4: Trello Board after Completion of Sprint 1

## 2.3. SPRINT 2 [ 24<sup>th</sup> February 2020 – 6<sup>th</sup> March 2020 ]

### 2.3.1. SPRINT PLANNING MEETING 25<sup>th</sup> February 2020 09:00 – 11:00

Planning for Sprint 2

- 1) Learning about google maps API
- 2) Plotting maps on frontend
- 3) Connecting flask application with the Database
- 4) Creating essential APIs
- 5) Returning response in JSON
- 6) Skeleton working of website

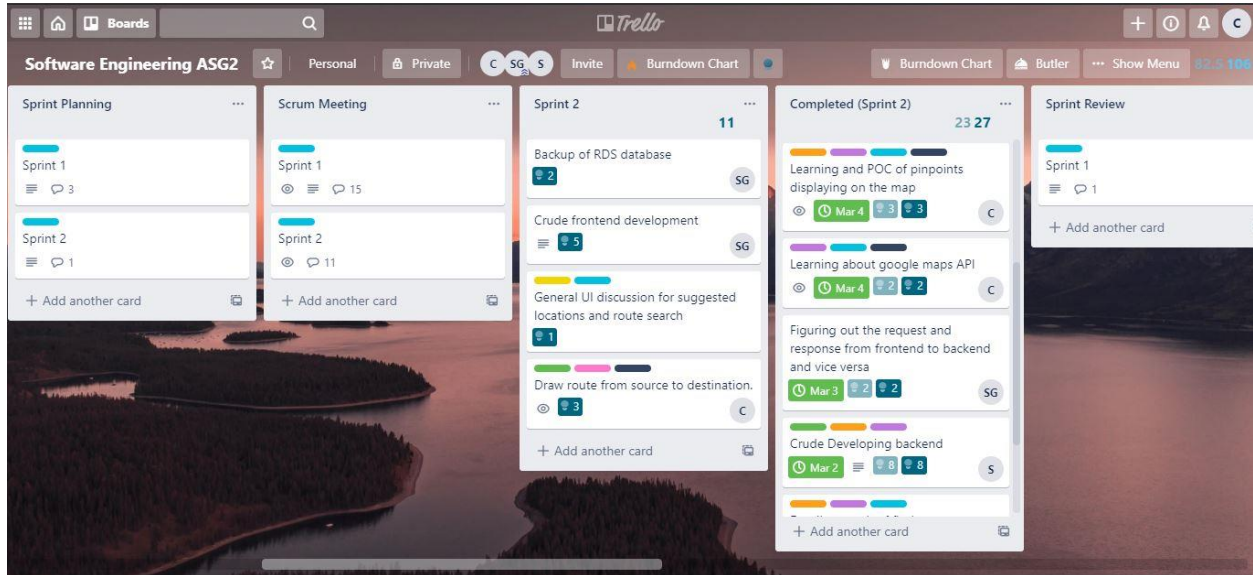


Figure 5: Trello Board for Sprint 2

### 2.3.2. SCRUM MEETING 27<sup>th</sup> February 2020 09:00 – 11:00

Completed research about Flask with HTML and Google Maps Platform API. Also discussed bit regarding the design and UX of the website.

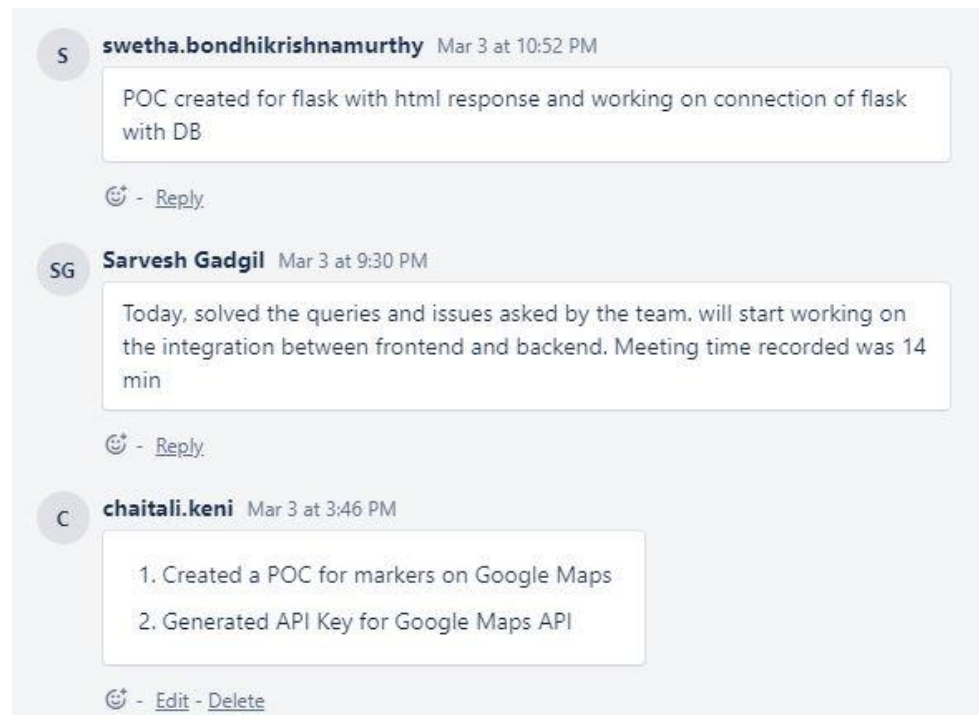


Figure 6: Scrum Meeting

### 2.3.3. SCRUM MEETING 3<sup>rd</sup> February 2020 09:00 – 11:00

We discussed the issues which were present in case of Flask framework and the front-end. We also discussed the progress of every team member which included creating a POC for markers on Google Maps, generating API Key for Google Maps API. We also started working on initial setup of Flask along with integration of front-end and back-end.

### 2.3.4. PROBLEMS FACED:

#### For Skeleton website:

1. **Markers:** We were having problems regarding loading of markers. All the markers were not loading. The problem was while creating a json response and returning to frontend.
2. **Popover window:** While picking a station from the markers or via search, the popup window over marker was not showing up. We fixed it by getting the marker object and using that object to load popup.
3. **Nearest stations:** We initially tried a SQL query to return station based on searched place. But then as the result was not proper, we handled this on the JavaScript end.
4. We had issue while sending station names through function consisting of ' in names. The content after ' was getting truncated. So, we replaced ' with ### before sending and then at receiving end we reverted back to '.



- We were unable to call the Google Maps webservices from the front-end using AJAX. We were getting CORS policy error. This was solved by calling the Google Maps webservices from Flask using requests package.

### 2.3.5. SPRINT REVIEW 20<sup>th</sup> February 2020 09:00 – 11:00

We were not able to complete routing of Source to Destination in this Sprint.

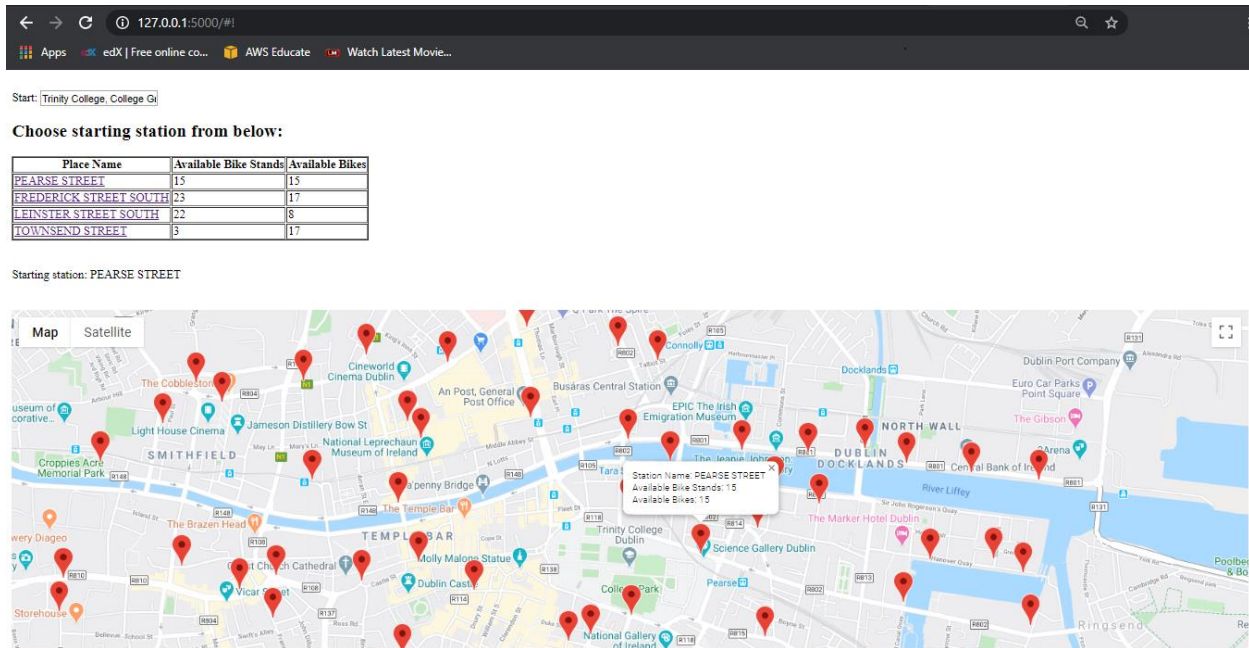


Figure 7: Completion of Web Application after Sprint 2

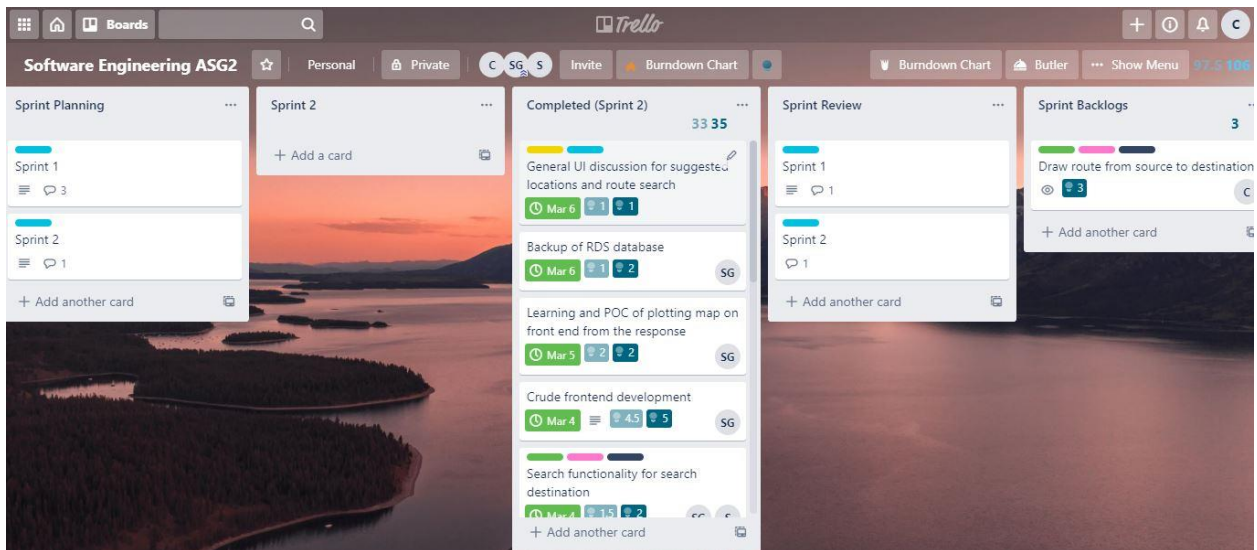


Figure 8: Trello Board after Completion of Sprint 2

## 2.4. SPRINT 3 [ 23<sup>rd</sup> March 2020 – 3<sup>rd</sup> April 2020 ]

### 2.4.1. SPRINT PLANNING MEETING 24<sup>th</sup> March 2020 09:00 – 11:00

Things to be covered in sprint 3 are:

- 1) ML model research and implementation
- 2) ML model integration
- 3) Visualization (weekly and hourly)
- 4) Solid UI design

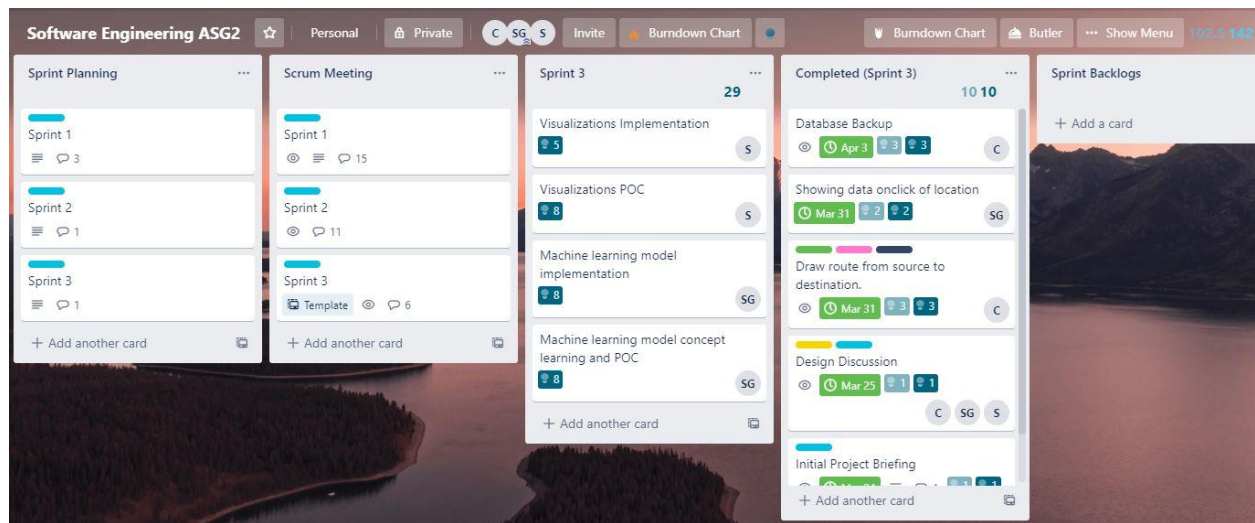


Figure 9: Trello Board for Sprint 3

### 2.4.2. SCRUM MEETING 31<sup>st</sup> March 2020 09:00 – 11:00

Completed the Destination search functionality and POC for Routing. Started working on the Visualization of the graphs.

### 2.4.3. SCRUM MEETING 1<sup>st</sup> April 2020 09:00 – 11:00

Finished ML creation and integration.

### 2.4.4. PROBLEMS FACED:

1. When we modularized our code into different files, some of the JS code was not working properly. Importing the JS files in head tag solved the problem.
2. Even though we used Bootstrap3 our website was not responsive. So, we studied what col-md, col-xs etc. meant in Bootstrap3 and then used it properly.
3. We were having caching problem on Chrome. So, we had to continuously clear the cache data.
4. ML: For categorical feature like weather description, we initially trained Naïve Bayes model. But the model was not accurate. So, then we tried RandomForestClassifier which improved the accuracy.

### 2.4.5. SPRINT REVIEW 2<sup>nd</sup> April 2020 09:00 – 11:00

We were not able to complete visualization POC and implementation.

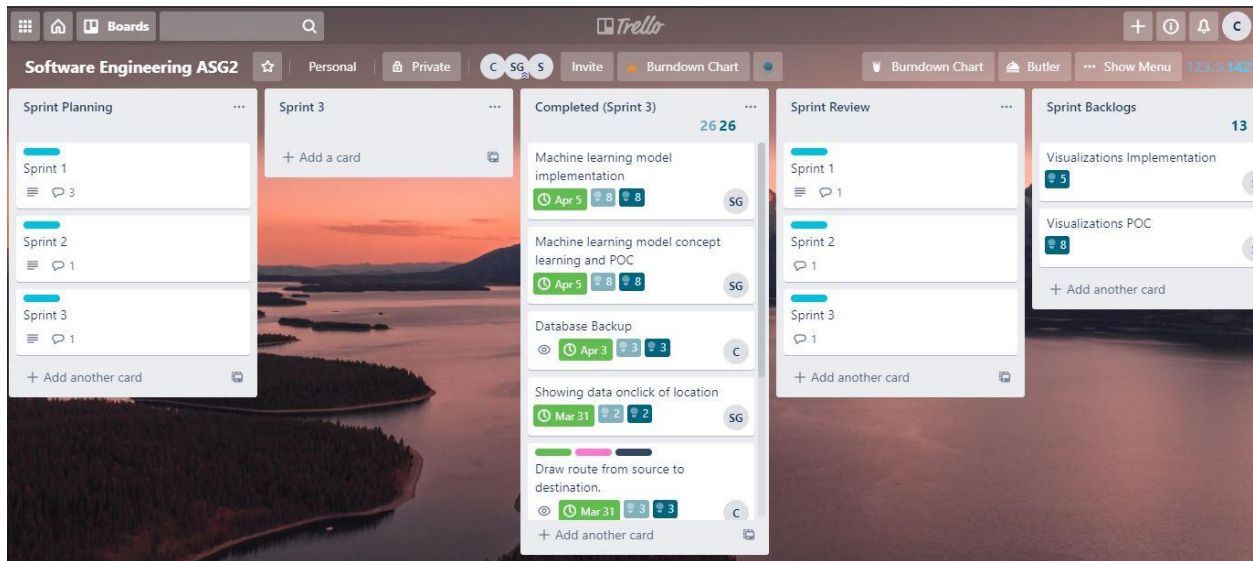


Figure 10: Trello Board after Completion of Sprint 3



## 2.5. SPRINT 4 [ 6<sup>th</sup> April 2020 – 17<sup>th</sup> April 2020 ]

### 2.5.1. SPRINT PLANNING MEETING 7<sup>th</sup> April 2020 09:00 – 11:00

Plan for Sprint 4:

1. Deployment of Flask App on EC2.
2. Completed UI
3. Optimize Performance
4. Clean up Code
5. Additional features

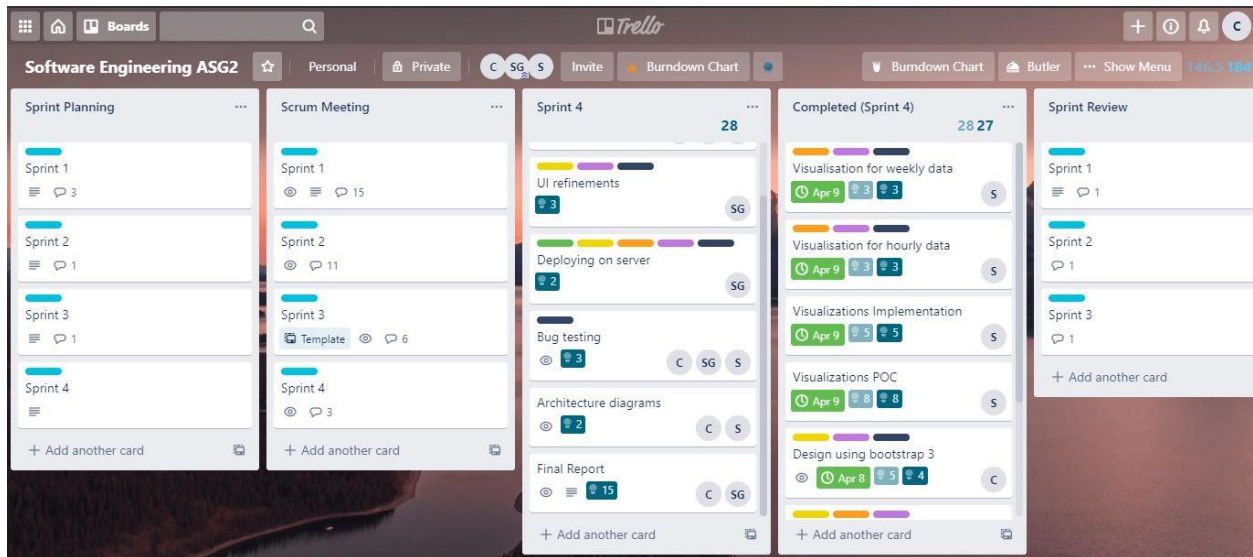


Figure 11: Trello Board for Sprint 4

### 2.5.2. SCRUM MEETING 9<sup>th</sup> April 2020 09:00 – 11:00

Completed the integration of Visualization of weekly and hourly data. Also made the website responsive.

### 2.5.3. SCRUM MEETING 14<sup>th</sup> April 2020 09:00 – 11:00

Hosted the website on EC2 instance and made some refinements.

### 2.5.3. PROBLEMS FACED:

#### Hosting

1. We initially tried to host website on our scraper server. But while installing packages we encountered “no space left” error. Then we created totally new instance on AWS (Ubuntu) for hosting the website.
2. After installing all the dependencies, we installed Apache2 webserver. The website was hosted but “Module not found” error was coming even though all the modules were installed globally. So, we then reverted to create virtual environment.
3. We fixed some problems regarding linking of virtual environment to webserver i.e. running project using virtual environment on webserver.
4. After hosting, we had problems with sklearn version. So, we had to install old version of sklearn (0.21.3) to fix issue as we created our ML models on somewhat old version.

5. If multiple users were to access the site, then we had some process management issues. The website used to hang. To fix this, we made some changes in the config file of the Apache2 webserver.

#### 2.5.4. SPRINT REVIEW 16<sup>th</sup> April 2020 09:00 – 11:00

We completed the project along with the bug fixes and refinements.

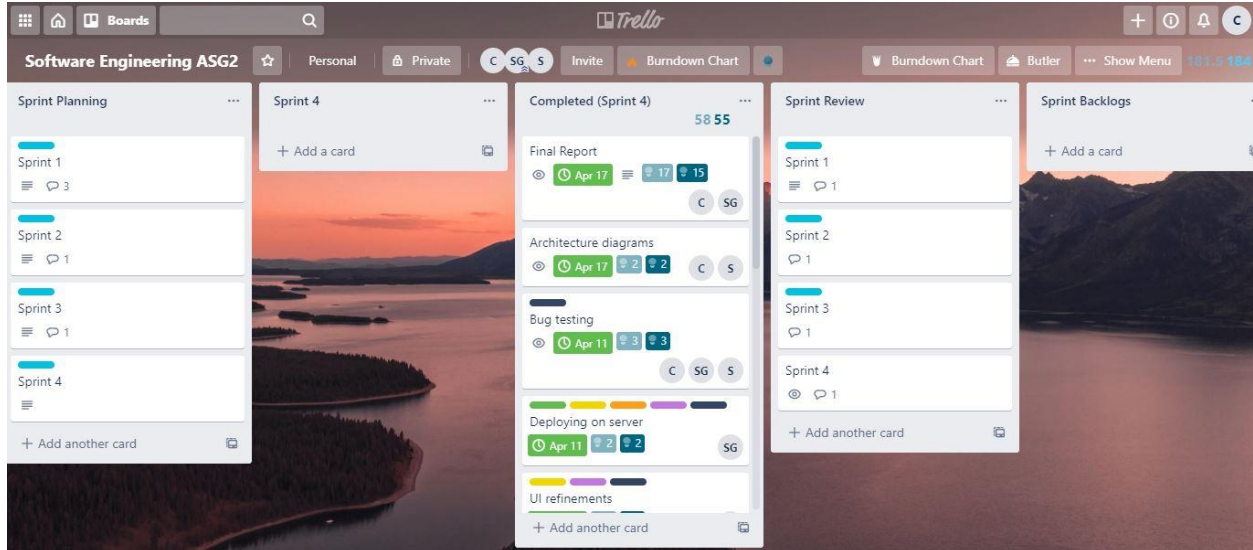


Figure 12: Trello Board after Completion of Sprint 4

## 2.6. BURN DOWN CHART

The burndown chart gives a graphical representation of how quickly a team is working for completing different features of a project divided into multiple sprints. In our case, the burndown chart regarding all of the four sprints gives a rough estimate on time spent on different modules of the project. From the chart, it is clear to notice that for the first two sprints, the team finished most of the features as planned. But in the remaining two sprints, as there was lockdown due to COVID-19, we had some issues with the communication, and we could not meet face-to-face and do the standup meetings. Due to this, the development of some of the features were finished after their end timeline. This is clearly reflected in the burndown charts. This definitely taught us the importance of constant communication and how important the role of ScrumMaster really is.

Day	Burned down		Balance		Daily Completed
	Planned	Actual	Planned	Actual	
0			184	184	#N/A
1	6.8	6	177.2	178	6
2	6.8	7	170.4	171	7
3	6.8	6	163.6	165	6
4	6.8	5	156.8	160	5
5	6.8	7	150	153	7
6	6.8	7	143.2	146	7
7	6.8	6	136.4	140	6
8	6.8	7	129.6	133	7
9	6.8	6	122.8	127	6
10	6.8	7.5	116	119.5	7.5
11	3.8	3	112.2	116.5	3
12	3.8	4	108.4	112.5	4
13	3.8	3	104.6	109.5	3
14	3.8	3	100.8	106.5	3
15	3.8	3	97	103.5	3
16	3.8	4	93.2	99.5	4
17	3.8	3	89.4	96.5	3
18	3.8	3	85.6	93.5	3
19	3.8	4	81.8	89.5	4
20	3.8	3	78	86.5	3
21	3.6	2	74.4	84.5	2
22	3.6	3	70.8	81.5	3
23	3.6	3	67.2	78.5	3
24	3.6	2	63.6	76.5	2
25	3.6	3	60	73.5	3
26	3.6	2	56.4	71.5	2
27	3.6	3	52.8	68.5	3
28	3.6	3	49.2	65.5	3
29	3.6	3	45.6	62.5	3
30	3.6	2	42	60.5	2
31	4.2	5	37.8	55.5	5
32	4.2	6	33.6	49.5	6
33	4.2	6	29.4	43.5	6
34	4.2	6	25.2	37.5	6
35	4.2	6	21	31.5	6
36	4.2	6	16.8	25.5	6
37	4.2	6	12.6	19.5	6
38	4.2	6	8.4	13.5	6
39	4.2	6	4.2	7.5	6
40	4.2	5	0	2.5	5

Figure 13: Burn Down of entire Project

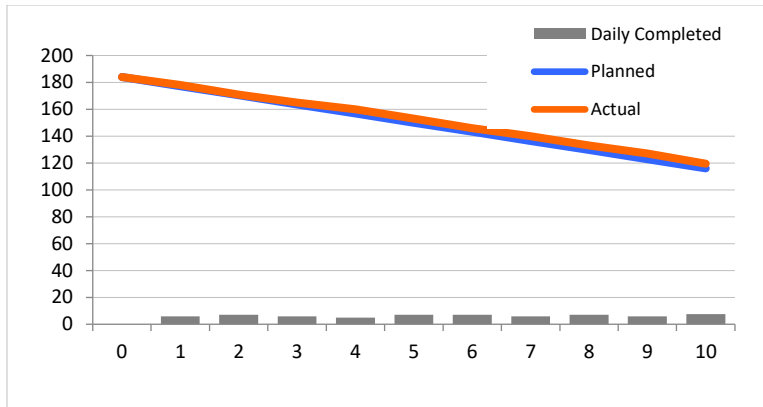


Figure 14: Burn Down Chart of Project

### 3. ARCHITECTURE

#### 3.1. BLOCK DIAGRAM

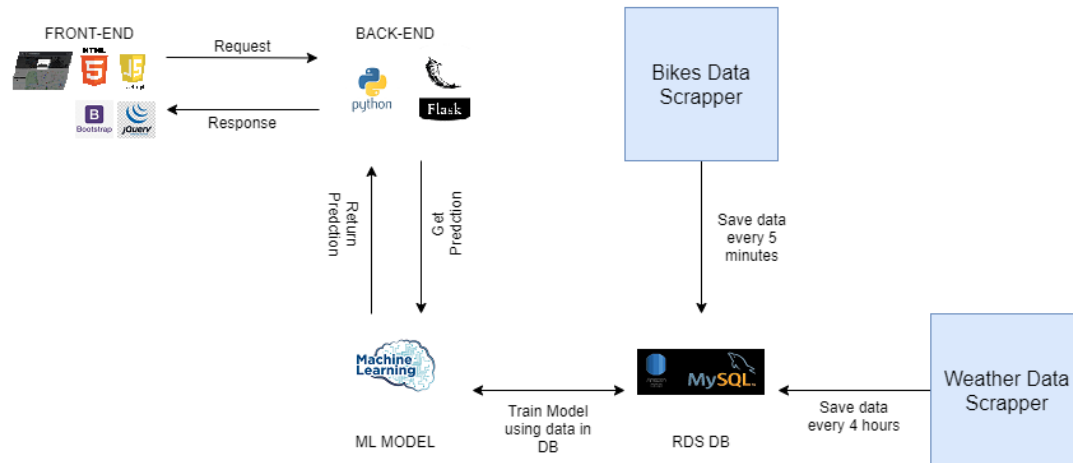


Figure 15: Block Diagram of Web Application

#### 3.2. TECHNOLOGIES

##### 3.2.1. GITHUB REPOSITORY

A GitHub Repository was used in order to store all the data related to the project. All team members were contributors in this repository.

For every development module, we created an issue/task related to that. Then we created a branch related to that issue/task. After creating branch, we started working on our respective branches. We created pull requests at the time of code merging into the master branch. Along with this, before creating any new branch for new task, each developer used to take pull of master branch using *git pull origin master*. This reduced the risk of conflicts.

Among the team members, Chaitali and Swetha were mostly responsible for doing research and development along with building proof of concept (POC) and Sarvesh was responsible for integration of the POC with the project along with UI refinements.

## Issues/Tasks:

<input type="checkbox"/>	<input type="radio"/> 0 Open <input checked="" type="radio"/> 18 Closed	Author ▾	Label ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Upload analytics jupyter notebook</b> #31 by sarvesh-gadgil was closed yesterday						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Weather animation</b> #28 by sarvesh-gadgil was closed 7 days ago						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Implementing Bootstrap</b> #27 by chaitalivkeni was closed 7 days ago						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>ML Model Integration</b> #24 by sarvesh-gadgil was closed 13 days ago						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>POC for using directions service of Google Maps API</b> #22 by chaitalivkeni was closed 17 days ago						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Implementation of searching for start location</b> #20 by sarvesh-gadgil was closed on Mar 11						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Integration Frontend and Backend Flask</b> #18 by sarvesh-gadgil was closed on Mar 10						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>POC of pinpoints displaying on the map</b> #17 by chaitalivkeni was closed 17 days ago						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>POC for flask connection with DB and returning a HTML as response n request</b> #16 by swe18 was closed 17 days ago						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Separate bike scraper from weather</b> #14 by sarvesh-gadgil was closed on Feb 21						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>POC for Flask</b> #12 by swe18 was closed on Mar 3						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>POC for requests library</b> #11 by swe18 was closed on Feb 18						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Get weather information and scrape data</b> #10 by chaitalivkeni was closed on Feb 19						
<input type="checkbox"/>	<input checked="" type="radio"/> <b>Web scraper for bikes data</b> #8 by sarvesh-gadgil was closed on Feb 15						

Figure 16: Issues created in GitHub

## Commits on various branch and the master:

Branch: #22_POC_for_us...	
Commits on Apr 2, 2020	<div>Extracting important parameters from JSON file</div> <div>chaitalivkeni committed 17 days ago</div> <div>65e3746</div>
Commits on Mar 11, 2020	<div>Merge pull request #21 from sarvesh-gadgil/#20_Implementation_of_sear...</div> <div>sarvesh-gadgil committed on Mar 11</div> <div>Verified</div> <div>2634662</div> <div>Changes:</div> <div>sarvesh-gadgil committed on Mar 11</div> <div>ff7c3ca</div> <div>Changes:</div> <div>sarvesh-gadgil committed on Mar 11</div> <div>834d6a1</div>
Commits on Mar 10, 2020	<div>Merge pull request #19 from sarvesh-gadgil/#18_Integration_Backend_an...</div> <div>sarvesh-gadgil committed on Mar 10</div> <div>Verified</div> <div>42fd740</div> <div>Changes:</div> <div>sarvesh-gadgil committed on Mar 10</div> <div>a9f0fcc</div>
Commits on Mar 5, 2020	<div>Changes:</div> <div>sarvesh-gadgil committed on Mar 5</div> <div>a81763d</div>
Commits on Mar 4, 2020	<div>Changes:</div> <div>sarvesh-gadgil committed on Mar 4</div> <div>ac89fc0</div> <div>Added git ignore</div> <div>sarvesh-gadgil committed on Mar 4</div> <div>269632b</div>
Commits on Feb 21, 2020	

Figure 17: Commits on Branch

Branch: **station\_charts** ▾

Commits on Apr 10, 2020

added api swe18 committed 8 days ago	a6d67d5	
---	---------	--

Commits on Apr 6, 2020

merge swe18 committed 12 days ago	98b16f8	
Merge branch 'master' of https://github.com/sarvesh-gadgil/software-e... swe18 committed 12 days ago	59d2190	
null check swe18 committed 12 days ago	b4203ec	
Start and End Data Values Validate swe18 committed 12 days ago	319c79e	
Merge pull request #26 from sarvesh-gadgil/ML_Weather_model sarvesh-gadgil committed 13 days ago	Verified  cd311f5	
ML model for weather: sarvesh-gadgil committed 13 days ago	d63d082	
ML model for weather: feels_like sarvesh-gadgil committed 13 days ago	08912aa	
Merge pull request #25 from sarvesh-gadgil/#24_ML_Model_Integration sarvesh-gadgil committed 13 days ago	Verified  6abc5dd	
ML models for available bikes sarvesh-gadgil committed 13 days ago	35cc0cc	
ML models for available bike stands sarvesh-gadgil committed 13 days ago	d0a3715	
Changes: sarvesh-gadgil committed 13 days ago	192a8f4	

Figure 18: Commits on Branch

Commits on Feb 21, 2020

Update README.md chaitalivkeni committed on Feb 21	Verified  9608133	
Extracting important parameters from JSON file chaitalivkeni committed on Feb 21	7e12ed4	
Merge remote-tracking branch 'origin/master' chaitalivkeni committed on Feb 21	f3f1a27	
Merge branch 'master' of C:\Users\DELL\Desktop\UCD\Semester 2\Softwar... chaitalivkeni committed on Feb 21	78b47e1	
Changed the db config sarvesh-gadgil committed on Feb 21	e914ca5	
Merge pull request #15 from sarvesh-gadgil/#14_Separate_bike_scraper_... sarvesh-gadgil committed on Feb 21	Verified  588a9a6	
Changes: sarvesh-gadgil committed on Feb 21	7e79142	

Commits on Feb 19, 2020

Merge pull request #13 from sarvesh-gadgil/#10_Get_weather_informatio... sarvesh-gadgil committed on Feb 19	Verified  5644b4a	
Changes: sarvesh-gadgil committed on Feb 19	9b52d37	

Commits on Feb 18, 2020

Extracting important parameters from JSON file chaitalivkeni committed on Feb 18	e154580	
---	---------	--

Commits on Feb 17, 2020

Merge remote-tracking branch 'origin/master' into #7_Linking_RDS_with... chaitalivkeni committed on Feb 17	7a8459e	
---	---------	--

Figure 19: Commits on Master

Link: <https://github.com/sarvesh-gadgil/software-engg-asg2>







## Log file for weather scraper

```
(base) [root@ip-172-31-33-189 ec2-user]# cat weather_data_scraper.log
Starting at: 2020-02-20 19:13:06.898746
Sleeping at: 2020-02-20 19:13:06.974143
Starting at: 2020-02-20 23:13:07.009629
Sleeping at: 2020-02-20 23:13:07.075074
Starting at: 2020-02-21 03:13:07.175247
Sleeping at: 2020-02-21 03:13:07.316122
Starting at: 2020-02-21 07:13:07.333732
Sleeping at: 2020-02-21 07:13:07.385533
Starting at: 2020-02-21 11:13:07.393646
Sleeping at: 2020-02-21 11:13:07.457158
Starting at: 2020-02-21 15:13:07.497649
Sleeping at: 2020-02-21 15:13:07.566318
Starting at: 2020-02-21 19:13:07.666495
Sleeping at: 2020-02-21 19:13:07.725240
```

Figure 23: Log file for weather scraper

### 3.2.3. Virtual Environment:

All of the team members had Anaconda virtual environment installed on their local machine. Inside the virtual environment, all the required packages were installed for development and testing of new features.

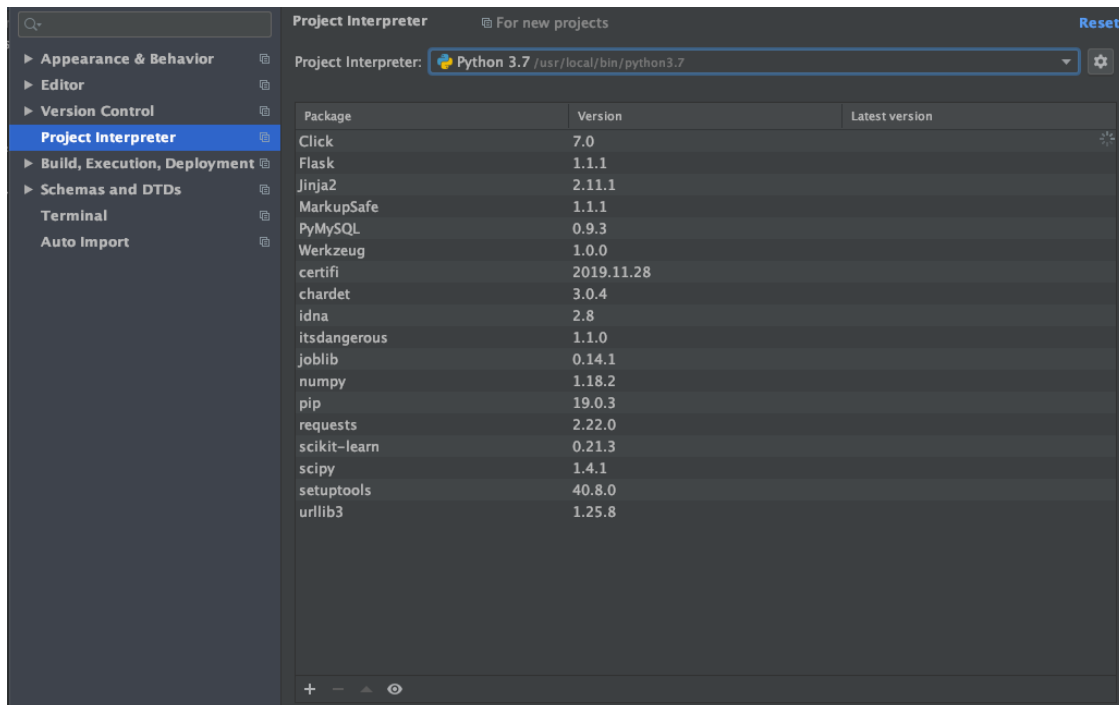


Figure 24: Virtual Environment

### 3.2.4. FRONT END

The main language used for front-end was JavaScript for HTML elements manipulation. We also explored the new features of the new version of **JavaScript (ES6)**. We used functionalities like `let`, `const` and arrow functions. We also extensively used **jQuery** which is a library of JavaScript. In jQuery, we mainly used **AJAX** along with some built-in functions like `foreach` etc.

For the design part, we used **HTML5, CSS and Bootstrap3**. We initially learned about Bootstrap3 on W3Schools. Bootstrap3 enabled us to design simple website in small amount of time. This is how we learned the importance of design framework as opposed to designing website from scratch in previous modules of the course. In Bootstrap3, we made use of `well` and `row` classes to display search box and map side by side. We also used panels to display stations data from the searched place. For search box we used `form-group` classes. We also used Bootstrap modals to display analytics information. We also included a hero image which made the website elegant and modern.

For choosing date-time, we used a **Bootstrap date-time picker**. This also had some jQuery code with it for formatting date, disabling old dates etc.

For displaying error messages, we also used **sweetalert.js**. It displays messages in a stylish format as opposed to simple `alert()` method of JavaScript.

For displaying analytics, we used **chart.js**. Chart.js really made it simple to load charts and display analytics information in an elegant way.

```

9      <!-- For jquery -->
10     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
11
12     <!-- For jquery ui autocomplete -->
13     <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
14     <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
15     <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
16
17     <!-- Bootstrap links -->
18     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
19     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
20
21     <!-- Bootstrap datetime picker links -->
22     <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.24.0/moment.min.js"></script>
23     <script
24       | src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datetimepicker/4.7.14/js/bootstrap-datetimepicker.min.js">
25     </script>
26     <link rel="stylesheet"
27       | href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datetimepicker/4.7.14/css/bootstrap-datetimepicker.min.css">
28     <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
29
30     <!-- Sweetalert link -->
31     <script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
32
33     <!-- Chart JS link -->
34     <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.min.js"></script>
35
36     <!-- Custom JS code below -->
37     <script type="text/javascript" src="static/js/googlemaps.js"></script>
38     <script type="text/javascript" src="static/js/initialstaticdataload.js"></script>
39     <script type="text/javascript" src="static/js/stationsearch.js"></script>
40     <script type="text/javascript" src="static/js/routes.js"></script>
41     <script type="text/javascript" src="static/js/validation.js"></script>
42     <script type="text/javascript" src="static/js/stationcharts.js"></script>
43     <link rel="stylesheet" href="static/css/style.css">
44     <link rel="stylesheet" href="static/css/weather.css">
45     <link rel="stylesheet" href="static/css/mappopup.css">
46     <link rel="stylesheet" href="static/css/heroimage.css">
47     <link rel="shortcut icon" href="static/image/bike-icon.png">

```

Figure 25: HTML file

### 3.2.5. BACKEND:

#### 3.2.5.1. SCRAPERS:

Scrapers were Python3 programs used to retrieve some information for data collection. This data was later used to display on front-end and to train ML models. We created two separate files for bike and weather scraping viz. `bikes_data_scraper.py` and `weather_data_scraper.py`.

1. **`bikes_data_scraper.py`**: The data for the bikes was retrieved using the JCDecaux API. A loop used to run every 5 minutes. After retrieving the data, it was split into static and dynamic data from the JSON response and then saved into the database.
2. **`weather_data_scraper.py`**: The weather data was retrieved using the OpenWeather API. A loop used to run every 4 hours. The response of the OpenWeather API was large and varied. So only the important parameters like temperature, weather description etc. was extracted and saved into the database.

#### 3.2.5.2. FLASK APPLICATION:

To serve all the requests from the front-end, we used Python3 as our backend programming language. Specifically, in Python3, we used Flask framework as it is easy to create complex web applications using Flask. The entire back-end was in `app.py` and served as the starting point of our web application. Furthermore, some functions which were present in `app.py` are:

1. **`get_bike_data_by_station_id(station_id)`**: It returned the dynamic bike data on the basis of station id.
2. **`get_all_static_bikes_data()`**: It returned all the static details of all the stations. This was called initially from front-end to setup the markers on the map.
3. **`get_places_by_query()`**: It returned the list of similar places as user typed a place to search on front-end. It internally called the Google Maps API with user's query and key as the request parameters.
4. **`get_place_coordinates()`**: It returned the latitude and longitude on the basis of place\_id. It internally called the Google Maps API with place\_id and key as the request parameters.
5. **`get_bike_and_weather_prediction()`**: It returned the weather; bike availability and available bikes stand prediction for start and end station. Using the start date, initially temperature and weather description were predicted by loading the appropriate ML model. On the basis of this, the bike availability and available bikes stand values were predicted and returned as JSON response.
6. **`calculate_average_bike_availability_weekly(station_id)`**: It calculated average bike availability for all the days of week and returned a JSON response containing day of week as key and the averages as values.
7. **`calculate_average_bike_availability_hourly(station_id)`**: It calculated average bike availability for all the hours for the selected date (by user) and returned a JSON response containing all hours as key and the averages as values.
8. **`load_ml_model(path)`**: As we had a total of 229 models, it was not feasible to load every model on start of the application. So, we implemented this function which accepted path as input argument and returned the model object.

#### 3.2.5.3. Database Connection

For database connection, we searched for numerous libraries. We initially tried SQLAlchemy but we had problems with interacting with the database. Also, we found it much complex. So, then we used **PyMySQL**. We also kept a centralized file called `db_connect.py`. This consisted of the database credentials along with

functions to return the database connection and to return the Webservice API request (converted in JSON format). This file was used by both scrapers and the flask application.

#### 3.2.5.4. Web service

To call JCDecaux, Google Maps webservices, we used a simple built in library in Python3 called **requests**. Moreover, to check the response of all the APIs, an application called **Postman** was used.

#### 3.2.5.5. AMAZON RDS MYSQL DATABASE

The RDS Database have been developed mostly using MySQL Workbench. All the relations of the single database were created by the developers and populated dynamically through python scripts and functions. The database consists of two tables which store the bikes data and one table which stores the weather data. The bikes data is separated in two tables i.e. table for storing static data and table for storing dynamic data and also consisted of primary-foreign key relationship.

This is the ER Diagram of Database :

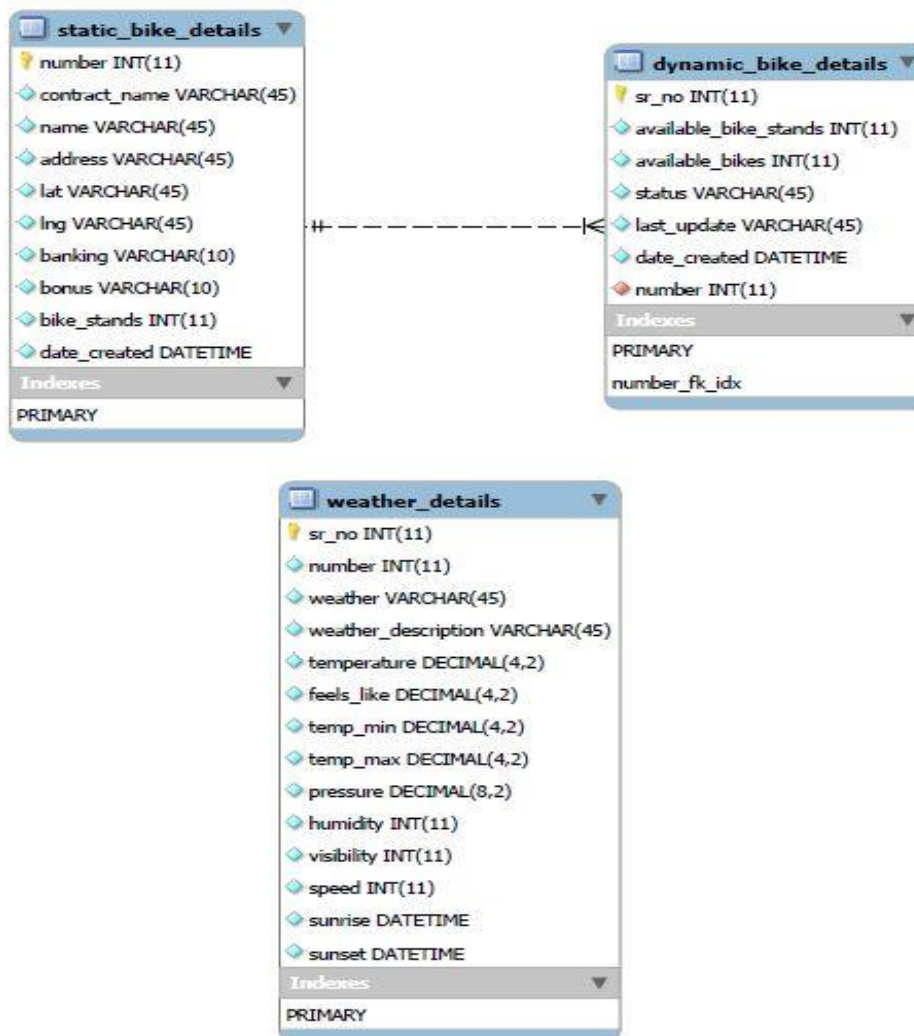


Figure 26: ER Diagram of Database

### Static Data:

The static data comprises of the information related to the location of bike station, name of bike station etc. number is the primary key in this table.

### Dynamic Data:

The dynamic data stores all the real time data related to bikes; the web scrapper checks for new data every 5 minutes.

### Weather Data:

The weather data also stores all the real time data related to weather; the web scrapper checks for new data once in 4 hours.

### 3.2.6. API:

We have used various API's in this project in order to gather necessary information. We used **Postman** application to check the API and view its data.

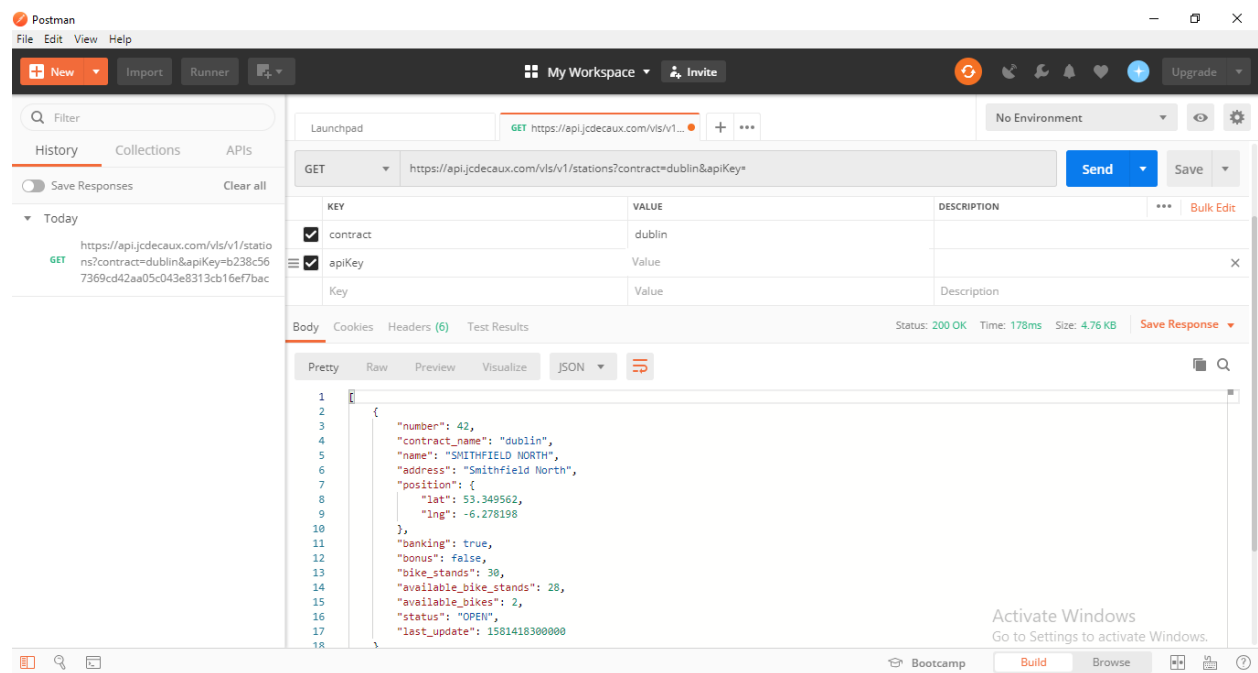


Figure 27: Screenshot of data obtained from Bikes API

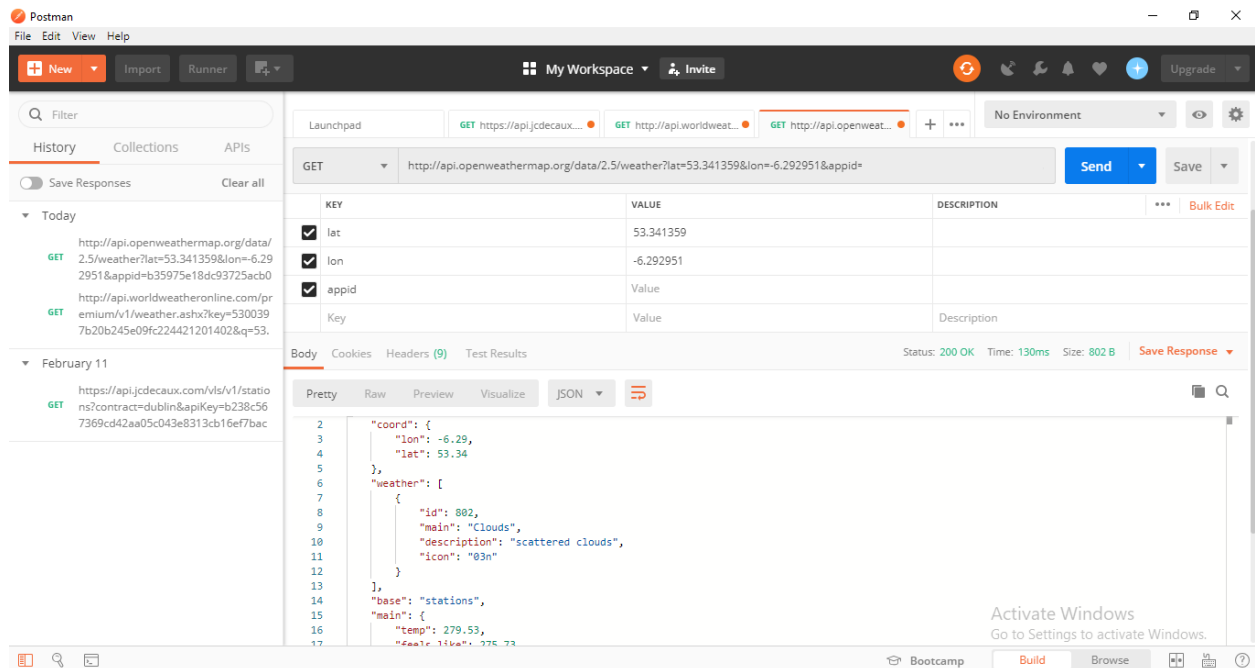


Figure 28: Screenshot of data obtained from Open Weather API

The API's used in this project are as below:

- **JCDecaux API:** This API was used to gather real time information of the bikes and the bike stand.
- **OpenWeatherAPI:** We opted for this API as it was chargeless and gave correct information about the weather.
- **Google Maps Platform API:** This was used for displaying the maps, routes, and various other features. Various Google Maps API used in this project are Places API, Maps JavaScript API and Directions API.



## 4. ANALYTICS

### 4.1. MODEL TRAINING AND BUILDING

We had to predict bikes on the basis weather for particular date. But even for that, we needed the weather prediction first. In the database we had date\_created column for every entry (both bikes table and weather table). So, using this parameter, we extracted numerous features to be feed to ML model. These includes time of day, date, month, minutes, day of week etc. We had some categorical features like day of week etc. which were **one hot encoded** using the built-in `get_dummies()` method.

Moreover, for bike availability and bike stands prediction, we also needed to join bikes data with the weather for training. So, for a particular bike entry, we got the weather close to that time period and merged it with the bikes data. This was all done using pandas dataframe in Jupyter notebook. The packages which we used for analytics includes pandas, numpy, sklearn etc.

We predicted numerous features for weather viz. temperature, weather description, feels like, max temperature and min temperature. For bike availability and bike stands prediction, we experimented with linear regression. But after printing the evaluation metrics like RMSE (Root Mean Square Error) and R2 (Rsquare), we found out that the accuracy was not good as RMSE was around 4 and R2 was around 0.3. This was expected as there was no linear relationship amongst the input features and target outcome. So, then we decided to use RandomForestRegressor as linear relationship is not required for this model. We experimented with the decision trees (kept to 10) and then printed the metrics which had a good accuracy.

```
X = features
y = df.available_bikes

train_features, test_features, train_labels, test_labels = train_test_split(X, y, test_size=0.3, random_state=0)

# Instantiate model with 10 decision trees
rf = RandomForestRegressor(n_estimators = 10)
# Train the model on training data
rf.fit(train_features, train_labels);

predictions = rf.predict(test_features)
printMetrics(test_labels, predictions)
print("\n\n")

# Serialize model object into a file called model.pkl on disk using pickle
file_name = "available_bikes_" + str(num) + ".pkl"
with open(file_name, 'wb') as handle:
    pickle.dump(rf, handle, pickle.HIGHEST_PROTOCOL)
# break
```

```
=====
MAE: 0.45439560439560434
RMSE: 0.915900445667337
R2: 0.9686902712988777
```

Figure 29: ML Model

We also decided to have a model for each of the station. This improved the accuracy as compared to having only one model.

### 4.2. ML INTEGRATION WITH FRONT-END AND BACK-END

Our model returned predictions whenever the user clicked on the Show Route button. An AJAX call was sent to the back-end from front-end consisting of request parameters like start\_station\_id, end\_station\_id and start date. In the back-end, using the start date, all the necessary features were extracted, and an array of these features were created as input. By loading the appropriate ML model, temperature and weather

description were predicted based upon the input. Using temperature and weather description values, the bike availability and available bikes stand values were predicted and returned as JSON response.

**Link to the Jupyter notebook:** [https://github.com/sarvesh-gadgil/software-engg-asg2/tree/master/modelling\\_notebooks](https://github.com/sarvesh-gadgil/software-engg-asg2/tree/master/modelling_notebooks)

## 5. DESIGN

### 5.1. MOCKUPS

We have used **MockPlus** and **Moqups** to design the mockups for this application. The initial design of the application is as below:

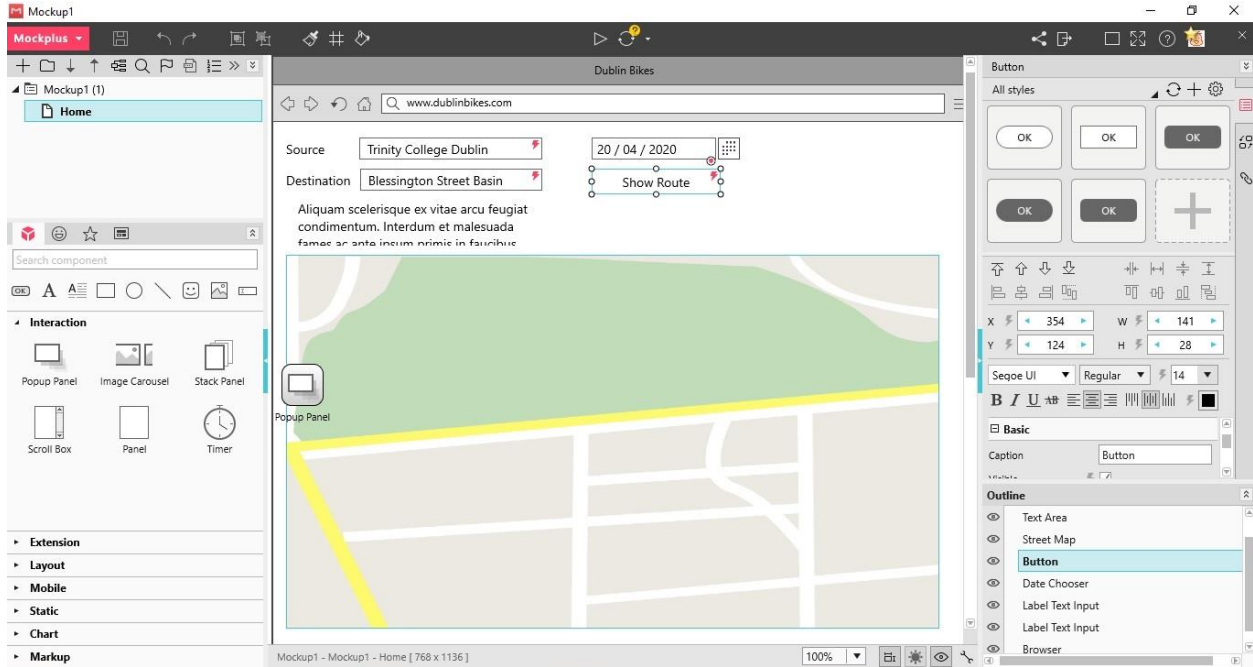


Figure 30: Mockup 1

The final design of mockup is as follows:


Chrome

← → ↻ <http://dublinbikes.com>

## Welcome to Dublin Bikes

Enter Start Station

Enter Destination Station

Select Start time  
4/22/2012 

Show Route

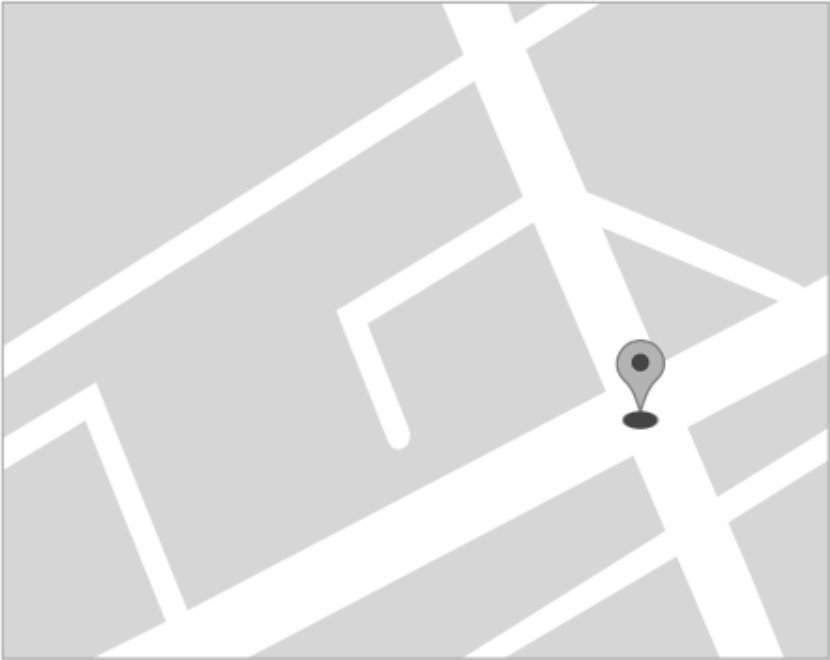


Figure 31: Final Mockup

## 5.2. WORKING

Our main aim was to have a UI which was simple and easy to use. Thus, we designed a side-by-side view of the search box and map as it would have been troublesome to continuously go back and forth between the map and search box. We also have a separate colored marker for start station (black) and destination station (green). To accompany this, we also implemented legend on the map consisting for different types of information related to markers and bike availability.

Moreover, we imagined two types of users who will use the website. The first one being an experienced user who is a resident of Dublin City and largely makes use of Dublin Bikes. So, if this user knows the stations, he/she can directly pick stations by clicking on any marker on the map and specifying it as source or destination station.

The second type of user could be a tourist; for which the city of Dublin is completely new. This type of user can use the search box present on the website. The user can directly search for a station, provided he/she knows the station name; OR can search for a location (e.g. trinity) and the website will display all the stations nearby (done by integrating Google Maps APIs). The user can then pick any of the station.

After picking the date time and clicking on show route button, we are displaying the route to be taken via bicycle along with the distance and total duration of the journey. Moreover, we are also displaying the predicted weather forecast for the chosen date time in an animated format with dynamic weather messages. The bike predictions (available bikes and stands) are displayed right on top of the respective station marker using a popup window. Moreover, we are also highlighting the area around station based on bike availability. So if station has large number of available bikes (greater than 40%) then the area is displayed as green with large circle; else if station has medium number of available bikes (greater than 10% and less than equal to 40%) then the area is displayed as orange with medium circle; else if station has low number of available bikes (less than 10%) then the area is displayed as red with small circle. If the user clicks on view details button on this popup window, all the analytics information regarding that particular station is displayed. This includes the weekly bike availability for each day of week and hourly bike availability for the day picked by the user from the date-time field in the searched box.

The finished product of this web application is as follows:

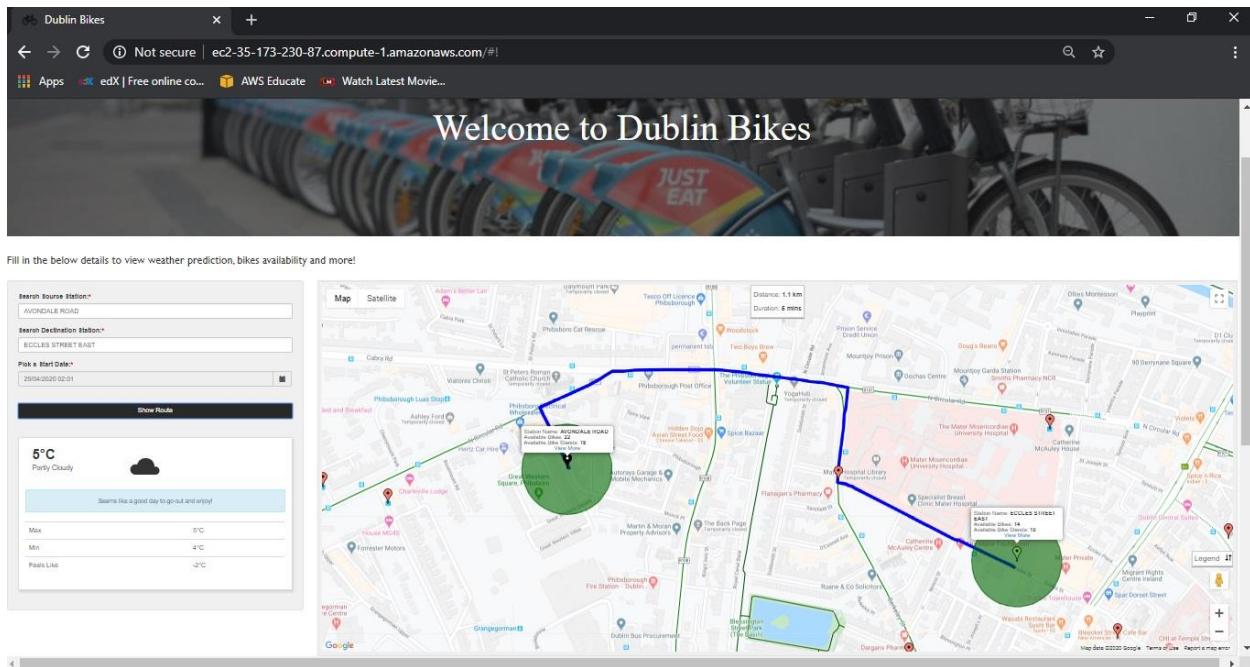


Figure 32: Web Application

With Visualization of Weekly/Hourly data:

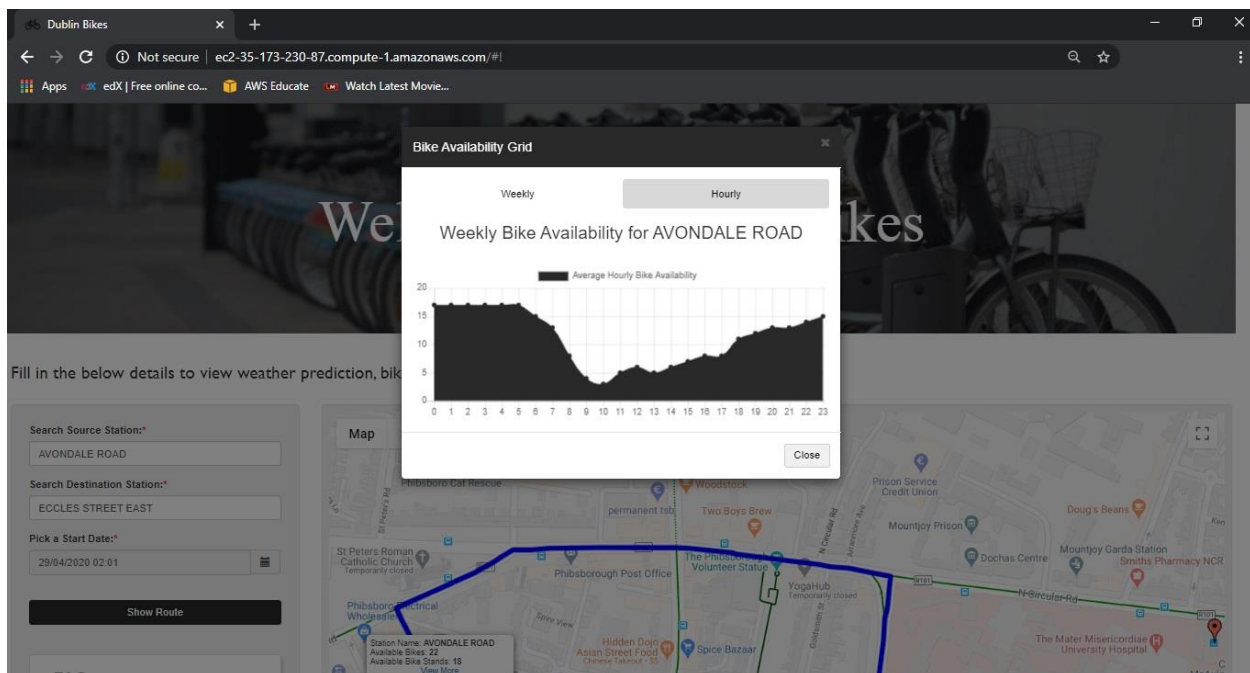


Figure 33: Web Application with Visualization of Hourly Data

## 6. FUTURE WORK:

We tried to implement a lot of features in this project but because of time constraint, we were not able to add few things and make improvements.

### 6.1. IMPROVEMENTS:

The following things can be added to the application:

- Build a user creation and user login page so that any user can be able to bookmark their favorite locations along with saving prediction history.
- Implementing OAuth 2.0 after creating login module for entire website.
- Create a new model to estimate the waiting time for a bike to arrive at every empty station across the city.
- Calculating walking distance between stations.
- Adding a domain name and SSL certificate to website to make it more secure.
- Ability to send the predictions to friends via email
- Setting alerts to get predictions via email at a particular time of day

### 6.2. REFINEMENTS:

- The UI of the application can be improved.
- Step by steps directions of the route can be displayed below the search box on the website.

### 6.3. OBSERVATIONS:

Due to COVID-19, we were forced to stop our scrapers. This was done as there was a complete lockdown and no one were using bikes. Thus, we were getting the duplicate data for the available bikes and available bike stands. In the future, after the lockdown is over, additional data can be collected to improve the accuracy of ML Models.

## 7. LINKS

**GitHub Repository:** <https://github.com/sarvesh-gadgil/software-engg-asg2>

**Web application:** <http://ec2-35-173-230-87.compute-1.amazonaws.com/>