1. Introduction

City biking has been traditional across Europe. Expanding cities like Copenhagen, Paris, Berlin, Vienna are often referred as bike friendly capitals. Bicycles are a healthy, environmental friendly way of commute. It needs simple infrastructure of a bicycle lane and bikes station network to operate. A motor road attempts to occupy garbage truck, bus, medium sized trucks, cars and motor bikes at a same time. Unlike this scenario, bicycle tracks carry single sized, single speed traffic providing congestion free commute.

Dublin bikes , which is operational since late 2006 has been a efficient commute option for Dublin citizens. It would be put on test more and more as Dublin is expanding. City has seen population growth of 79,600 merely in last 5 years since record GDP growth of 26% in 2015.

Dublin Bike web application provides a real time availability of bikes and bike stands at Dublin bike stations. Present weather condition is also available given volatile weather experienced in Dublin. Similarly, predicted weather and bike station availabilities for following 5 hours is displayed for source and destination stations.

* 1. Aim

Dublin Bike Web application is made to facilitate existing and new users with near real time bike station information. With forecasted weather conditions and availability of bikes and stands at bike stations, it helps users and subscribers in planning their commute. This application is made for use on desktops as well as on mobile phone browsers. Appropriate visualizations and easy navigation are incorporated to improve user experience.

1. Project Management

This project is an instrument to learn end to end software development life cycle using agile methodology. Entire project was divided into 4 major sprints:

* Planning and Database setup
* Flask and Web UI design
* Data Analytics
* Integration Testing , UX and Improvements

We had team of 3 engineers. A team member was appointed as scrum master and given individual expertise, a team member each lead first 3 sprints.

We used following tools for project management:

* Trello is Digital Kanban board developed by Atlassian is efficient way to plan track project progress. Sprints are broken into lists. And individual stories of sprints are represented by cards. Cards can be assigned labels, checklists and due dates. Each card can be assigned to team members and attachments can be sent.
* GitHub is industry standard version control software tool. Since parallel development cycles and CI/CD are main features of agile development, GitHub enabled us to avoid cumbersome mail chains.
  1. Stand-up meetings and progression

Scope of project was limited to only 4 sprints hence we avoided separate communication channel tool like slack. Instead we used a dedicated card on Trello to maintain stand up meeting logs. Trello provides an efficient cell phone application facilitating continuous communication.

1. Sprints and sprint analysis
   1. Sprint 1
      1. Meeting 1 Date : 13/02/2020
         * Discussion

Team was formed and outline of project was discussed. WE discussed possibility to define an SRS as a guideline function. W discussed our industrial experience about agile methodology, web development.

* + - * Work Done
        + Setup Github
        + Setup Trello scrum Kanban board
      * Work To Do
        + Setup EC2 instance
        + Setup RDS database
    1. Meeting 2 Date : 20/02/2020
       - Discussion

Team head for each sprint was finalised as sprints were divided by technologies. We discussed data features to scrape from the JCDecaux and openweathermap API.

* + - * Work Done
        + Finalized database structure
        + Literature survey about Flask application
      * Work To Do : Write a python script to process a json file, create a datastruture and update it to a local mySQL instance.
    1. Meeting 3 Date : 21/02/2020
       - Discussion

W discussed major challenges in this meeting since sprint was about to conclude soon. We froze datastructure of our database into 3 individual databases. Also, datatype of individual feature was critically analysed to conserve space avoid redundancy.

* + - * Work Done
        + Set up Amazon EC2 server instance
        + Set up amazon RDS database
      * Work To Do
        + Error checks and build cron to automate scraper
    1. Meeting 4 Date : 23/02/2020
       - Discussion

Efficiency of scraper code was discussed. Redundant loops were removed. Server instability and errors during database interaction were resolved. We finalised ToDo cards for Trello board for sprint2.

* + - * Work Done
        + Flask Setup on local also I initialize it on server
      * Work To Do
        + Finalise scraper code and commence web scraping
  1. Sprint 2
     1. Meeting 1 Date : 24/02/2020
        + Discussion

Cron cycles were discussed to decide frequency with which data was logged. We finalised frequency of data logging to be 10 minutes for both bike data and weather data.

* + - * Work Done
        + Database publisher script written
        + Database architecture
      * Work To Do
        + API scraper and error logger scripts
        + Working on the server to implement Nginx
    1. Meeting 2 Date : 26/02/2020
       - Discussion

We decided that station availability and weather prediction would be for present day only since we would have data only between February to April to train out model upon. But we decided to train models including an artificial entry each of all months. Thus, our model would be able to comprehend categorical data like months April onwards in future when those entries are scraped and logged in database.

* + - * Work Done
        + Database scrape code completed
        + Created working service for EC2 server instance
      * Work To Do
        + Complete weather scraper and cron scripts
    1. Meeting 3 Date : 29/02/2020
       - Discussion

We made pen paper UI and decided to make sample wireframes for visualisation.

* + - * Work Done
        + Scraper started
      * Work To Do
        + Start the cron service to automate scraper operation
        + Design wireframes for website UI
    1. Meeting 4 Date : 03/03/2020
       - Discussion

We finalised to experiment with both time series approach and regression approach for predicting bike station availability.

* + - * Work Done
        + Google maps API integration with flask
        + Designed few wireframe options to finalise web UI
      * Work To Do
        + Finalise Flask application and UI
  1. Sprint 3
     1. Meeting 1 Date : 25/03/2020
        + Discussion

As 2 out of 3 team members were home quarantined and hospitalised given corona outbreak in India, backlog had increased. We decided to move implementation of backup database into backlog.

* + - * Work Done
        + Google map API integration with web UI completed
        + Local instance of website working
      * Work To Do
        + Finalise Data analytics algorithm
    1. Meeting 2 Date : 27/03/2020
       - Discussion

We discussed about architecture of machine learning pipeline. GUI wireframe was finalised.

* + - * Work Done
        + Extracted csv from database
        + Preliminary data processing done, null entries resolved
      * Work To Do
        + Implement timeseries and random forest algorithms for comparative study
    1. Meeting 3 Date : 30/03/2020
       - Discussion

We discussed possibility of generating artificial data using timeseries algorithm for emulating regular traffic during Stay at home phase.

* + - * Work Done
        + Obtained satisfactory training loss for available\_bikes prediction sing time series analysis
      * Work To Do
        + Weather data processing and integration
    1. Meeting 4 Date : 02/04/2020
       - Discussion

Concept of artificial data introduction was dropped since it could not improve model.

* + - * Work Done
        + Model visualisation
        + Filtering weather data accessed from database by front end
      * Work To Do
        + Model training
        + Frontend and javascript API to python [flask]
    1. Meeting 5 Date : 04/04/2020
       - Discussion

Time series model was shown to demonstrator in last meet and he expressed satisfaction with the work. We decided t omove on with random forest approach as it would have shorter training time and comparable accuracy.

* + - * Work Done
        + Timeseries model completed
      * Work To Do
        + Start development of random forest model
        + Design weather widgets
    1. Meeting 6 Date : 06/04/2020
       - Discussion

We discussed placement of weather icons on UI and decided to replace them with icons as it would be critical to handle widgets during responsive design of website for mobile phones.

* + - * Work Done
        + Graphs, header and footers for website
        + Weather widgets
      * Work To Do

Complete accessory functions to use generated random forest models

* 1. Sprint 4
     1. Meeting 1 Date : : 11/04/2020
        + Discussion

Model for bike station prediction was ready. We discussed about backlog of weather model development and hurdles faced due to handling datetime object.

* + - * Work Done
        + Weather model integration with Bike station models
      * Work To Do
        + Model deployment
    1. Meeting 2 Date : : 13/04/2020
       - Discussion

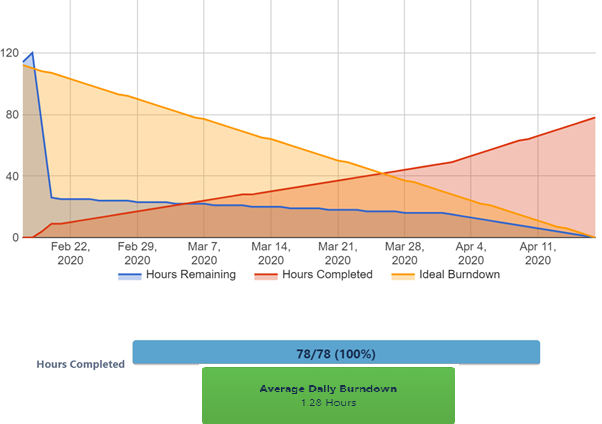
Requirements for JavaScript API were frozen. Accordingly, Accessory functions would be designed to return predictions and statistics for the station.

* + - * Work Done
        + Model deployment and accessory functions for model
      * Work To Do
        + Website integration with accessory functions of ML model
    1. Meeting 3 Date : : 14/04/2020
       - Discussion

Website was successfully hosted online. Ui improvements and responsiveness across browsers was discussed.

* + - * Work Done
        + JavaScript API fo python to access ML models
      * Work To Do
        + Debugging on live website
        + Project Report
  1. Burndown analysis

A burn down chart is a graphical representation of work left to do versus time. AS burndown charts need measurable units of work, we chose Fibonacci scale of evaluation provided in Trello burndown add on. As it is evident from chart, our intial work was completed relatively strinct to the timeline. But after sprint 2, as pressures built up, our remaining hours depleted rapidly. Due to unavailability of a hospitalised team member, our productivity in first half of sprint 3 was hampered. In spite of harsh scrum retrospectives, we could not stock up and complete our data analytics phase in sprint 3 and it had to be moved partially intp sprint 4. But our Web design was timely and we had hosted site live in sprint 2, which gave us edge in integration of ML models with web environment through flask.



1. Development Operations (Dev-ops)

Scope of project was to deploy a working public accessible website hosted on Amazon EC2 cloud and developed using python web framework. This section describe the Development operations decisions taken through the entire project development.

* 1. Local Environment

Project consisted 4 major languages to work with:

* MySQL for Database
* Python for web framework and Data analytics
* JavaScript and HTML for front end

To avoid conflicts with environment variables, all team members worked with anaconda

virtual environment. This enabled unit testing efficient as environments could be shared

and updated as yaml files

* 1. GitHub repository

Two private git repositories were created for this project : each for static development and dynamic development. Team members decided to opensource both repo post academic evaluation.

* Static development consisted of data scraper, Server and database related files, test Time series data analytics algorithms and documentation.
* dynamic development contained driver programs using flask web framework, website front end, and data analytics backend with deployed Machine learning models.

We appointed a team member as master for each repo and others worked on separate branches. This avoided merge conflicts and codes were unit tested regularly.

* + 1. Repository Structure
  1. Amazon AWS server

In this application, we have integrated one server to arrange all the applications. We have used Amazon AS EC2 free tier one with ubuntu 18 instances. This server is the latest on ubuntu from AWS. While creating the server new key pair generated and as our local system is based on windows, we have used putty to interact with the server. This server is handling our Dublin bike scrapper code, weather API and most important runs flask application. To run Dublin bike, we have created crontab to run scrapper code every 5 minutes and our weather code is running every 10 minutes. While working on crontab at the start there was some problem but later on, we got to know that it is because of some initialization issue. In the past two months, our server has crashed three times. First time it's just because of environmental issues. The second one was we were trying to connect to the server but couldn’t establish our connection then after careful understanding, we got to know that because ones you stopped the server (As we crossed our monthly limit) and trying again to start it will change the IP address so the server configuration in putty need to be changed. The latest server crash happens it's because of the installation package version. This error was tough to debug as the system is not giving exact error problems. So we need to debug all the changes, this time git commit came handly as we rollback some changes and debug each operation that helps us to solve the package problem. In the future, to avoid a problem like this we track our changes or crashes occurrence by creating a function that will generate the log file. This log file is come into handy to fix future problems. One thing we learned from this is during your production environment it is always better to have a backup server running to avoid unnecessary crashing of your website.

As main cloud server we've used Amazon AWS EC2 free tier ubuntu instance.

While in the development phase has been set to accept connections only via

SSH, this has been changed to accept all connections on deployment. The

EC2 Server runs in multiplexing (using tmux) the Dublin Bikes API scraper,

the Open Weather Map API scraper, the Accuracy evaluation script and the

Flask server application. In the last months of development we got 2 server

crashes which were most likely due to AWS: the \_rst one occurred on the 28th

of February at 16.00 but speaking with other teams, this crash occurred to

everyone else. The second crash has not been tracked correctly. After the \_rst

crash we agreed to implement a function that would have sent an email to each

one of the team members when a crash occurs. The actual function works well to

catch crashes due to API Scrapers malfunctions and Database connection errors

but is not e\_ective to catch server crashes. Maybe a further implementation on

the backup server would have been able to catch them but this feature has not

been implemented due to time restrictions.

8.4.1 All Packages installed

Python 3 and Conda is installed on the EC2 Instance.

-asn1crypto==0.24.0

attrs==17.4.0

Automat==0.6.0

blinker==1.4

certifi==2018.1.18

chardet==3.0.4

click==7.1.1

cloud-init==19.3

colorama==0.3.7

command-not-found==0.3

configobj==5.0.6

constantly==15.1.0

cryptography==2.1.4

distro-info===0.18ubuntu0.18.04.1

ec2-hibinit-agent==1.0.0

Flask==1.1.1

Flask-MySQLdb==0.2.0

gunicorn==19.7.1

hibagent==1.0.1

httplib2==0.9.2

hyperlink==17.3.1

idna==2.6

incremental==16.10.1

itsdangerous==1.1.0

Jinja2==2.11.1

jsonpatch==1.16

jsonpointer==1.10

jsonschema==2.6.0

keyring==10.6.0

keyrings.alt==3.0

language-selector==0.1

MarkupSafe==1.1.1

mysqlclient==1.4.6

netifaces==0.10.4

numpy==1.18.2

oauthlib==2.0.6

PAM==0.4.2

pandas==1.0.3

pyasn1==0.4.2

pyasn1-modules==0.2.1

pycrypto==2.6.1

pygobject==3.26.1

PyJWT==1.5.3

PyMySQL==0.9.3

pyOpenSSL==17.5.0

pyOpenSSL==17.5.0

pyserial==3.4

python-apt==1.6.5+ubuntu0.2

python-dateutil==2.8.1

python-debian==0.1.32

pytz==2019.3

pyxdg==0.25

PyYAML==3.12

requests==2.18.4

requests-unixsocket==0.1.5

SecretStorage==2.3.1

service-identity==16.0.0

six==1.14.0

ssh-import-id==5.7

systemd-python==234

Twisted==17.9.0

ufw==0.36

unattended-upgrades==0.1

urllib3==1.22

Werkzeug==1.0.0

zope.interface==4.3.2

8.4.1 Custom Packages installed

Python 3 and Conda shall be installed on the EC2 Instance.

\_ pip list

{ ask 1.0.2

{ matplotlib 3.0.2

{ mysql-connector-python 8.0.15

{ numpy 1.15.4

{ pandas 0.24.0

{ pip 18.1

{ pypyodbc 1.3.4

{ requests 2.21.0

\_ brew list

{ wget

{ mysql

{ tmux

* 1. Amazon RDS MySQL Database

The RDS database setup has configured with running EC2 instance via security group. The database structure is formed via MYSQL Workbench, After working on structure we populate data dynamically through python script which is running on crontab. The structure is planned with three tables. One table is containing all the static data from stations like station information and its position. Another table contains dynamic data coming from station which includes available bikes and stand. To make database effective and secure we are backing up our database every seven days.

* 1. Amazon RDS MySQL Database

The RDS Database have been developed mostly using MySQL Workbench. All

the relations of the single database have been created manually by the devel-

opers and populated dynamically through python scripts and functions. The

initial Database schema was composed by three tables: one for the static data

from the Dublin Bikes API, one for the dynamic ones and the last one for test-

ing. While the testing table has been left into the database till the end of the

deployment, one other table have been added to store info regarding the Open

Weather Map API scraper. We discussed regarding the opportunity to save data

with a certain frequency and which data were meaningful for our application.

This is the \_nal Database Schema:

* + 1. Static Dublin Bike Data

Dublin\_bike\_staticdata contained one time information about the stations. This information has long shelf life and ideally an entry must not change. New entry are added only when a new Dublin bike station is introduced. This table logs following key features:

Id\_entry(PK): introduced primary key

number(FK): station Id

name: station name

position\_lat: station location latitude

position\_lng: station location longitude

bonus: station bonus

banking: status of e-payment service

data\_entry\_timestamp : Time at which JCDecaux API was queried and station entry was made in Dublin\_bike\_staticdata. Irrespective of last\_update; for consumer of API, time of data entry into database remains ultimate status of station.

* + 1. Dynamic Dublin Bike Data

Table Dublin\_bike\_dynamicdata contain dynamic information regarding all the stations in dublin city. This data is accessed for making ML predictions and generating station statistics of weekly and hourly bike availability.

Id\_entry: introduced primary key

Number: station id

Status: station status open/ closed

bike\_stands: Total capacity of station

available\_bike\_stands: number of empty stands available to park

available\_bikes: Number of bikes available for pickup

data\_entry\_timestamp : Time at which JCDecaux API was queried and station entry was made in Dublin\_bike\_dynamicdata. Irrespective of last\_update; for consumer of API, time of data entry into database remains ultimate status of station.

* + 1. Weather Data

Open\_weather\_dynamicdata stores over 37 parameters of Dublin weather. Only 9 of those are used for actual ML predictions. But we have chosen to preserve this data for future use, where ML model for weather prediction takes more weather features into consideration. Key features of database used are:

Id\_entry: Introduced primary key

main\_temp: Temperature

main\_feels\_like: human perception of temperature

main\_pressure: Atmospheric pressure of Dublin sea level in hPa

main\_humidity: percentage humidity

main\_temp\_min: Minimum temperature at the moment

main\_temp\_max: Maximum temperature at the moment.

wind\_speed: Wind speed in meter/seconds

wind\_deg: Wind direction in meteorological degree

weather\_main ; Group of weather parameters (Rain, Snow, Extreme etc.)

data\_entry\_timestamp : Time at which openweather API was queried and station entry was made in Dublin\_bike\_dynamicdata. Irrespective of last\_update; for consumer of API, time of data entry into database remains ultimate status of station.

* 1. Deployment on EC2 Instance

During the deployment, we first studied all necessary deployment tools which can be feasible and can perform in a better manner. After comparing Apache and NGINX we come to the conclusion that NGINX will be good to go. We first set up all the required package and once all installation is done we use Gunicorn3 service with NGINX for deployment and setting up an environment that allows us to update the application and run using few commands. Our security group is configured to allow any connection. One thing which we learn now is we could create more security in this phase but due to lack of time, we could not implement it. The link to our running application is <http://34.243.168.221/>

* 1. Deployment on EC2 Instance

Important: Due to a technical issue with pip on the EC2 server, we had to temporarily deactivate all cache features from the python code. The issue cannot be resolved in time prior the submission of the product. This issue is due to a pip package problem since the functools.lru cache packet is now incorporated into the latest versions of python3 and shouldn't require any install through pip. Deployment has been made on the EC2 instance modifying the default ip and port: 127.0.0.1:5000 to 0.0.0.0:80 and setting the security group of the Server to allow any connection. Other solutions such as nginx and Gunicorn have been considered but discarded due to operational reason. Furthermore, we are not much concerned about eventual security breaches at this stage. the link to the running web application is : http://ec2-34-238-40-161.compute-1.amazonaws.com

1. Back-end development
   1. API Scrapers

Dublin Bike station data is made public through an webAPI by operating firm JCDecaux. Static and dynamic data is saved into database. Similarly, weather data for Dublin published by openweathermap.com data is made public through an webAPI. API web scraper written in python is used to log bike data and weather conditions into database. Error checking and data processing is done for mentioned factors:

API json response rendered false : log into logfile.txt with datetime

KeyError for json response: When Feature[column] is not returned by API call , append None entry and complete the data frame for data entry

Database connection in error : log into logfile.txt with datetime

Temperature unit conversion: Temeprature parameters published by openweathermap.com are in degree Kelvin. They are converted into Celsius scale before logging.

* 1. Cron Program
* Every 10 minute call the scrapper and update the respective table in the database.
* Keep track of log files
  1. Cron Program
* Periodically call scraper to update database
* Threading – avoid infinite loop
* Automation
* Error check
  1. Flask Application

Flask is a library free web framework written in Python. For a small scale rationed server application like this project, flask is a suitable infrastructure since it is lightweight than counterparts like nodeJS. Flask has simple task of rendering homepage of website on launch. Further, flask application serves requests from webpage, through defined functions on call.

* + 1. get\_data

Argument: None Return: jsonified Dublin Bike Static data

Operation: Fetches all entries in Dublin Bike Static data. The function returns bike station geolocation, banking, bonus and status indicating if station is operational or closed.

* + 1. get\_station\_info

Argument: Return:

Operation:

* + 1. station\_data

Argument: None Return: jsonified latest entry for a station in Dublin Bike Dynamic data

Operation: Data from JCDecaux is routinely scraped and logged into database. Hence, latest availability at the station is found by operating on latest entry for the station.

* + 1. get\_station\_history

Argument: None Return: Jsonified station availability in terms of weekdays and of 24 hours

Operation: Availability of bikes and bike stands at station varies depending on the time and day of week. It is imperative that more bikes would be available in late night and on weekends as office and university hours are over. Hence, User is provided feature to observe daily and hourly availabilities at a station.

* + 1. get\_prediction

Argument: None Return: jsonified availability, weather and temeperature at source and destination station

Operation: This web application is made to facilitate user. When a user checks availability of bikes and stands at particular stations; she/he is provided with the availability for next 5 hours by 30 minute interval. This offers use flexibility to plan pickup and drop.

* + 1. current\_weather:

Argument: None Return: jsonified current weather from Dublin weather dynamic data

Operation: Fetch weather entries order descending.

* + 1. toCSV

Argument: None Return: None

Operation: By design, machine learning models built for predictions periodically continuously trained using cron program as new data keeps coming. Hence, this helper function generates csv tables from database tables for data processing.

* 1. Data Analysis and Machine learning

City transport services are operated by authorities and hence can be scheduled. But Dublin bike service is operated by users of bikes only. Being unscheduled, users would like to know availability of bikes at pick up stations and bike stands at destination before head. But along with, if users could know about near future predictions of bike station availability; then planning pickup and drop times can be easier. Similarly, providing weather conditions at corresponding time would greatly help given swinging weather in coastal cities like Dublin.

* + 1. Weather prediction

Temperature, Wind condition and main weather state are prime factors in deciding bike hire. These parameters are predicted using a random forest regressor and classifier models trained over 4 weeks’ worth data.

* + 1. Bike station prediction

Users of Dublin bike are mostly from universities and from office staffs. Hence, usage has a time cycle as well as weekly occupancy slots. Hence, datetime parameters along with weather at the time are utilised to predict the bike station availability.

Even though traffic and occupancies at stations are correlated; developing a universal model is tedious for the scope of this project. This is because, routes taken by users are completely random unlike bus/ luas routes. Also, degree of that model would be very large as each station would have some weight over others. Hence, we designed a prediction model for each station independently.

* 1. Accuracy check

By design, cron program is supposed to continuously scrape data. But given COVID – 29 pandemic, stay at home phase was implied by government of Ireland in last week of March 2020. Hence, data forth 24 March is omitted for model training and models are trained on approximately 24 days of data. The individual model performance can be observed in following result files:

* result\_available\_bike\_stands.csv
* result\_available\_bikes.csv
* resultWeather.csv

Average performance of the models seem satisfactory for a 100 layer random forest with unnormalized features. Metrics of predicted parameters are tabulated bellow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PARAMETER** | **AVERAGE MAE** | **AVERAGE MSE** | **AVERAGE RMSE** | **AVERAGE R2** |
| **Bike Station** | | | | |
| Available Bike Stands | 0.754 | 2.285 | 1.447 | 0.966 |
| Available Bikes | 0.752 | 2.272 | 1.442 | 0.966 |
| **Weather** | | | | |
| main\_temp | 0.256538908 | 0.173240186 | 0.416221318 | 0.982769756 |
| main\_feels\_like | 0.425911731 | 0.579625668 | 0.76133151 | 0.93579148 |
| main\_pressure | 0.17960511 | 0.106446225 | 0.326260977 | 0.999453893 |
| main\_humidity | 2.555911731 | 14.00315912 | 3.742079518 | 0.873247242 |
| main\_temp\_min | 0.278315912 | 0.205021719 | 0.452793241 | 0.980531098 |
| main\_temp\_max | 0.307409988 | 0.241941463 | 0.491875455 | 0.97548472 |
| wind\_speed | 0.451118118 | 0.665969461 | 0.81606952 | 0.891950984 |
| wind\_deg | 7.286504065 | 446.1720049 | 21.12278402 | 0.901932654 |

Figure 7: Architecture structure

10 Front-End development

This section explains the technologies used for the front-end development along

with the structure of the \_les deployed. Having used Flask throughout the

testing phase, the structure required by Flask to work is as follow:

\_ bikes dublin FFSTeam22

\_ index.html

{ Static

\_ CSS

\_ images

The main HTML \_le is inside the main folder, meanwhile the css \_les and

the images used are respectively inside the CSS and images folders.

30

10.1 Front-End technologies

\_ HTML

\_ CSS

\_ JavaScipt

\_ JQuery

10.2 HTML

For the project we have used the HTML5 standard, which has made easier to

structure an entire page, relegating most of the styling to CSS. Utilizing newer

elements like "header" to create the uppermost page DIV and "footer" for the

lowermost one. But also canvas used on the charts DIVs and several other

elements like calendar and Geolocation, which can track the user position in

real time if required, making the whole process very easy to implement. The

whole page is divided into four main sections, namely the header, the map, the

information window and the footer.

\_ Header

\_ Map

\_ Information window

\_ Footer

The header is where we have decided to show the weather predictions coming

from the weather API. Below it there is the map which takes up most of the

screen estate. The map we have used is from the Google Map API. The map can

be customized to ones preference and has very well written manuals to further

customize it and implement on all of its functionalities. Following the map there

is the section where all the information is displayed. To welcome the user and

avoid cluttering the screen with much information at the start, there is a splash

screen covering the section. The idea is for the user to only see the general

sensible info and later select the extra information required. Upon selecting one

of the markers, the splash screen disappears and the actual window replaces

it. For the purpose of this application the footer is left empty. In a future

update information like the person to contact or the product owner name could

be added if wanted.

10.3 CSS

CSS3 was used to stylize in the web page the arrangement of the elements.

There are two \_les for the CSS:

/.static/css/grid.css

/.static/css/style.css

Throughout the page elements have been displayed using `Grid' layout.

While inside sections the `Flex box module layout is also used. Everything in

31

relation to modularity and display arrangement has been created and modi\_ed

within the `grid.css' \_le. For instance the header width spacing has been done

using ex grid, which makes the header responsive and keeps the ratio between

all the elements included. The second \_le 'style.css' is the one where all the

style options have been done: from the size, to the color of di\_erent elements,

including info-windows. The style that was chosen for the web-page vaguely

resembles that of an arcade video game, in term of colors and proportions.

10.4 JavaScript

JavaScript is the main engine behind the site, transforming the originally static

page into a dynamic one, increasing the client interactions. It is used to make

requests to the scraper, and stylize the various DIVs in which the information is

contained or sent to. For the project the Javascript code resides in the HTML

\_le. We have used some vanilla code to implement some functions, and some

Jquery code to do the pull requests to the server. Javascript o\_ers many frame-

works. We have used Jquery extensively (described next) and also ApexCharts

to visualize the charts on the site. These charts are already implemented to

animate and be dynamic, adapting to the div they are placed in. Furthermore

using a JSON style format is it possible to customise them to anyone's liking.

10.5 JQuery

This JS framework is used mostly for the Ajax capabilities of easily sending

pull requests to the server and for the way it simplify complex methods into few

lines of code. For the site we used two versions one for the standard framework

and one for the jQuery-UI theme implementation.

The jQuery has several functions relative to the Flask application, that pull

speci\_c data to \_ll di\_erent sections of the website.

Speci\_cally:

request static data: this function returns all the information about the stands,

like name, number, status, location and if it has banking available. These infor-

mation is then used to create all the markers on the map, with corresponding

variables.

Furthermore by knowing the location of each station using its latitude and lon-

gitude, it is also possible to calculate the distance with the user and advice

on the nearest stations. Within the request is also possible to create a div to

show the information pulled from the server to the user, in this case the nearest

stations from the user.

request info box: is used to request to the scraper the speci\_c information about

a stand. It returns the latest update on the number of bikes and stands available

and displays it in an info-window that is created within the function

32

request hourly prediction and request weekly prediction: functions are both

sending the required data to the respective chart, placed at the bottom of the

screen. The charts are respectively showing the amount of bikes and stands

available throughout the day and on a weekly prediction.

get weather inuence: function returns the predictions calculated based on

the weather for the day selected. These predictions are returned on the initial

splash screen.

In the next example we can see how a request is made and variable is passed.

The example shows how the request is made to retrieve the station data, to fur-

ther manipulate it.

11 Project Delivered

The web app interface provide sensible information about the current status

of the Dublin Bike service, with advanced functionalities to make predictions

based on weather forecast.

11.1 Key Features

\_ Display real time availability of bikes and stands.

\_ Display real time weather forecast.

\_ Predict hourly and weekly availability for bikes and stands.

\_ Display current position of the user (hard coded).

\_ Display nearest stations based on the user location.

\_ Display weather inuence on bike and stands availability.

33

11.2 Performance optimisations

To improve performance when retrieving data from the database, we imple-

mented on the Python built-in LRU-cache library. Due to the deployment oper-

ational issues (described in the dev-ops section) this feature has been temporary

deactivated.

Another issue encountered when deploying the system on the server, has

been with the HTML Geo-location feature, which requires a SSL certi\_cate to

operate. Being unable to retrieve a certi\_cate before the deadline, we have been

advised to hard-code the user location just to show proof the functionality.

11.3 Key Shortfalls and Planned Improvements

Due to unforeseen circumstances with one of the team members, some of the

features that were originally planned, have been delay or postponed to a fu-

ture update of the app. These features are: Data analysis models (e.g Linear

regression), proper deployment methods (e.g usng NGINX), Geolocation and

SSL certi\_cates, security improvments of the server and the database, proper

analisys of station that systematically fail to update using station error.csv

\_le, proper accuracy analysis of the predictions based on accuracy.csv \_le.

Other considerations that have been made during the development phase, were:

to provide the user with a registration \_eld, which would grant access to a user

area, where is possible to report issues like faulty stations, faulty bikes with a

direct line of contact in case of any other issue related to the service.

12 LINKS:

GITHUB repository for the project

https://github.com/fabiom91/bikes dublin FFSTeam22

Project Web page

http://ec2-34-238-40-161.compute-1.amazonaws.com/