COMP30830
Software Engineering
Spring 2020

Dublin Bike web application

REPORT BY GROUP 11

JOSHI, OMKAR | KANE, RASIK | KESHWAR, RATNAPRABHA

TABLE OF CONTENT

TABLE OF FIGURES

SUBMISSION NOTE AND LINKS

This report summarises process outline, technical details, processes followed and efforts taken by GROUP 11 in order to partially fulfil the requirements for Group Project of module COMP30830-Software Engineering (CONV) offered at UCD School of Computer Science in year 2020 spring semester.

Contribution to Project by group members is weighed with sheer analysis. Alphabetically as follows:

Name	Contribution	Details	Reasons for shortcomings
Joshi Omkar	40%	• Designed UI • Designed server infrastructure • API for python; integration with JS • JS implementation	• Faced challenges with ML - UI integration • Security group problem with EC2 and RDS
Kane Rasik	40%	• Designed Database architecture • Designed webAPI Scraper • Designed ML models for prediction • Project report and scrum management	• Faced challenges with scraper optimization • Faced problems regarding ML model optimization
Keshwar Ratnaprabha	20%	• Contributed to Database design • Contributed to ML model design • Designed UI Wireframes Contributed to UI design	• Returned home and was hospital quarantined for 2 week • Faced queries regarding weather ML model design

- Weblink to the project website is : Dublin Bike web application [1]
- Weblink to the dynamic development GitHub repository: gitRepo1 [2]
    - Jupyter Notebooks [3]
- Weblink to the static development GitHub repository: gitRepo2[4]
    - Jupyter Notebooks [5]

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, environmentally friendly way of commute. It needs simple material infrastructure of a bicycle lane and bikes station network to operate; unlike cabs, car rentals and city transport. 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 an 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 [6] merely in last 5 years since record GDP growth of 26% was recorded in 2015.

Dublin Bike web application provides information about 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.

1.1. Objective

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.

1.2. Structure

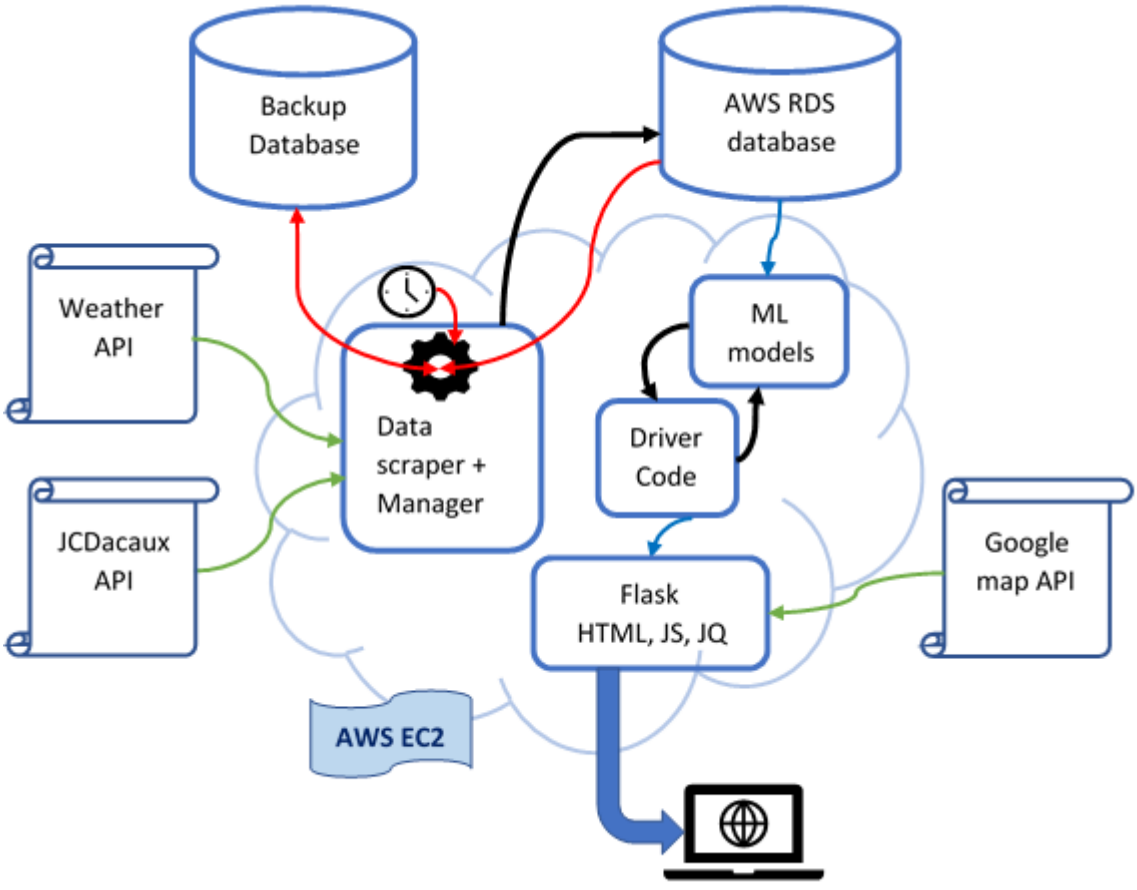

Figure 1 Structure of Dublin Bike Web Application

2. 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 a digital Kanban Board developed by Atlassian. It is efficient way to plan and 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.

2.1. 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.

2.2. Sprint 1

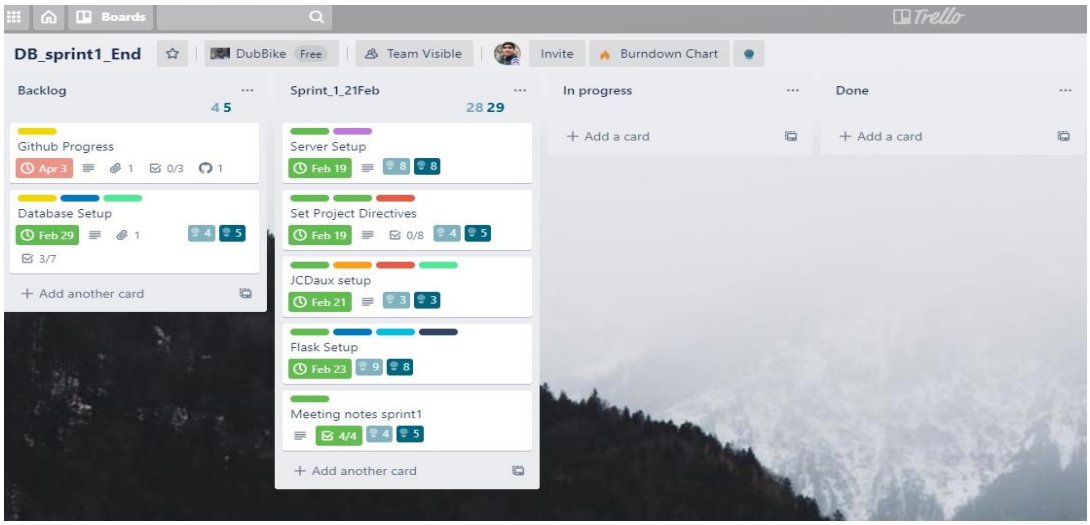

Figure 2 Sprint 1 Kanban Board [7]

2.2.1. Meeting 1 Date : 13/02/2020

- Discussion
  Team was formed and outline of project was discussed. Team discussed possibility to define an SRS as a guideline function. Team 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

2.2.2. 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 data structure and update it to a local MySQL instance.

2.2.3. Meeting 3 Date : 21/02/2020

- Discussion
  Team discussed major challenges in this meeting since sprint was about to conclude soon. We froze data structure 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

2.2.4. 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 To-do 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

2.2.5. Sprint 1 Retrospective

- Work was lagging for web scraper
- Team was still not using Trello effectively

2.3. Sprint 2

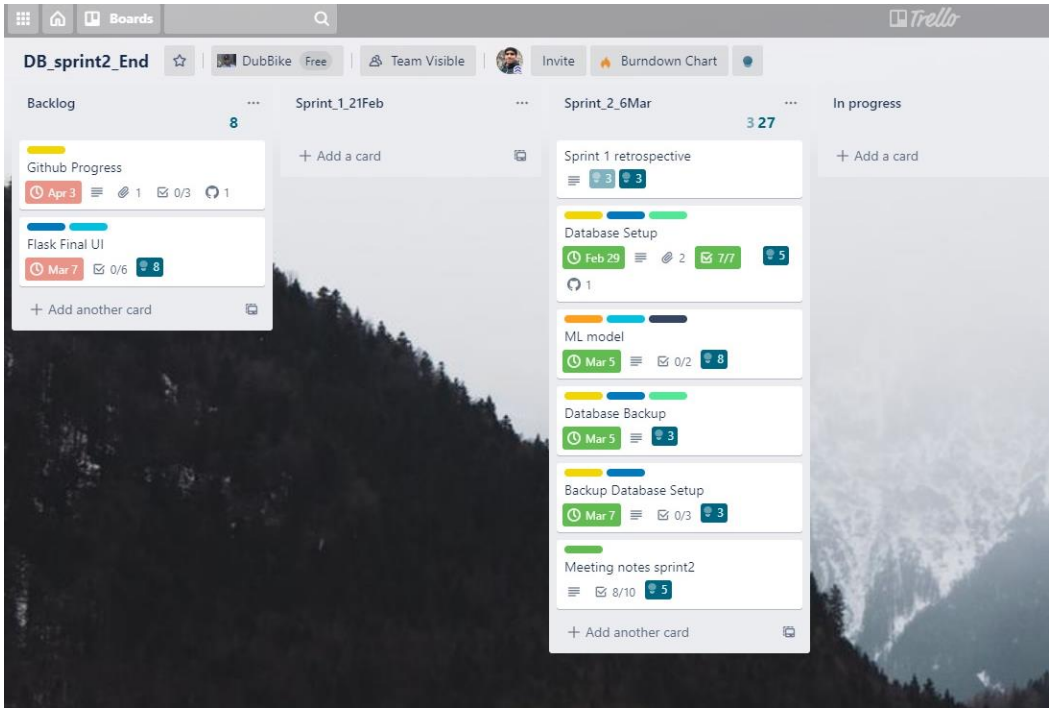

Figure 3 Sprint 2 Kanban Board [8]

2.3.1. 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

2.3.2. 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

2.3.3. 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

2.3.4. 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

2.3.5. Sprint 2 Retrospective

- Scraper code is modular and maintainable
- Cron is working and team members took good efforts
- Google maps integration took time and support resources are required in integration testing during delivery

2.4. Sprint 3

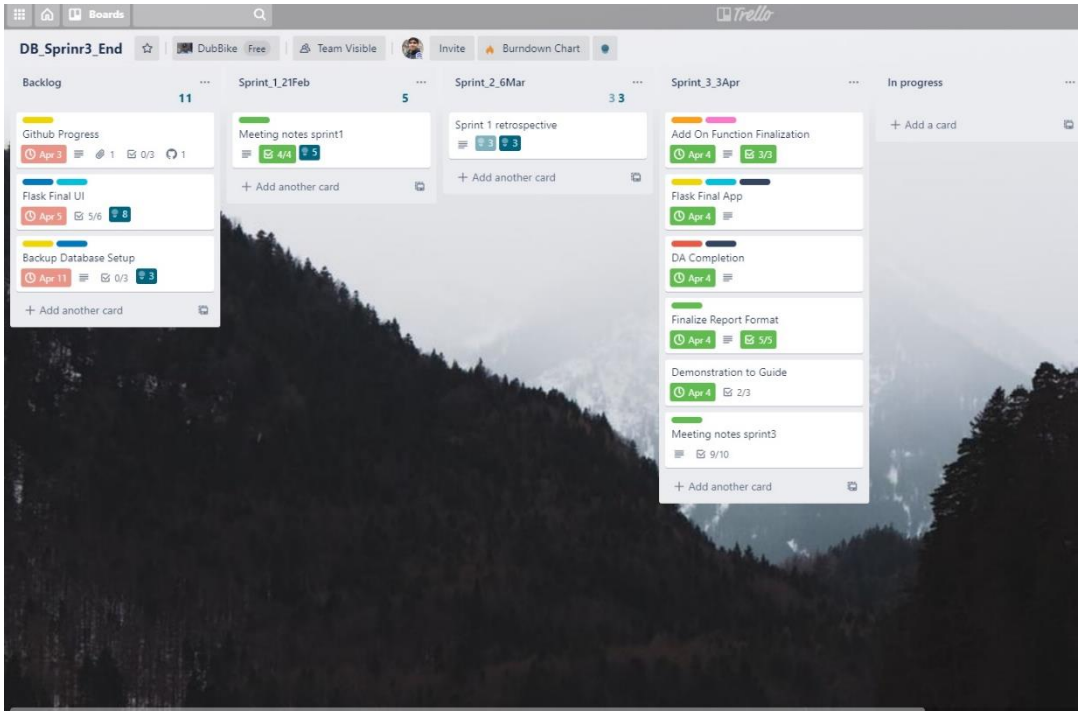

Figure 4 Sprint 3 Kanban Board [9]

2.4.1. 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

2.4.2. 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

2.4.3. 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

2.4.4. 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]

2.4.5. 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 to move 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

2.4.6. 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

2.4.7. Sprint 3 Retrospective

- Half time of sprint not utilised given pandemic relocation
- Need to capitalise on trained models and deploy as quickly as possible
- Weather model integration with bike model is primary task

2.5. Sprint 4

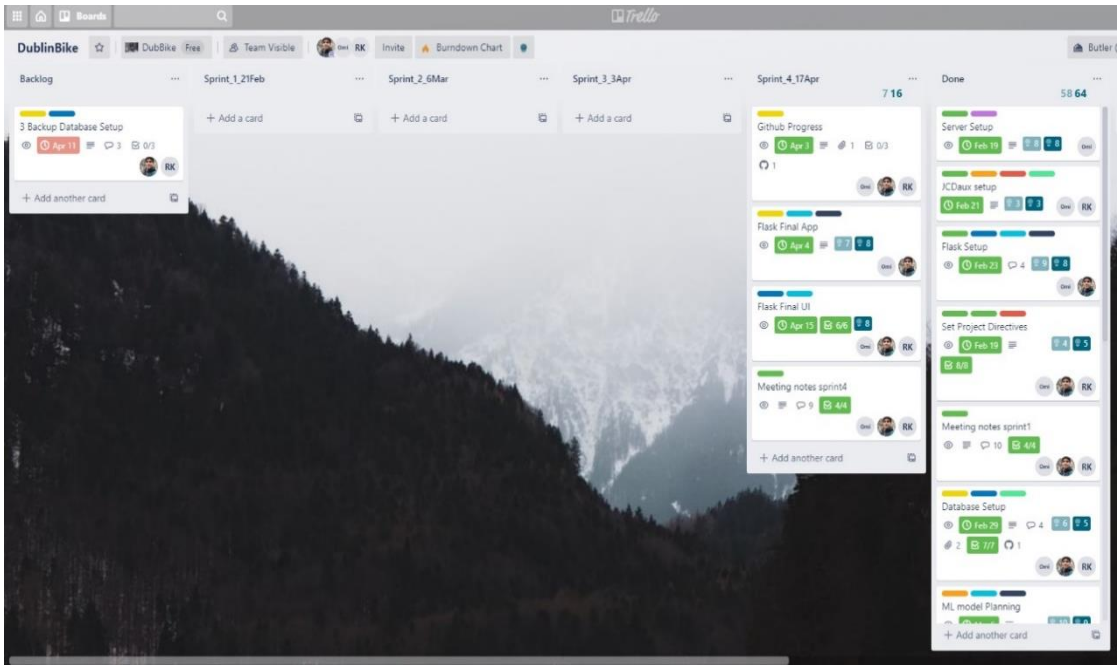

Figure 5 Sprint 4 Kanban Board [10]

2.5.1. 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

2.5.2. 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

2.5.3. Meeting 3 Date : 14/04/2020

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

- Work Done
  - JavaScript API for python to access ML models

- Work To Do
  - Debugging on live website and completing project report

2.5.4. Sprint 4 Retrospective

- Project is delivered in time despite of tight schedule
- Designed test cases are covered

2.6. 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 initial work was completed relatively strict 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 into 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.

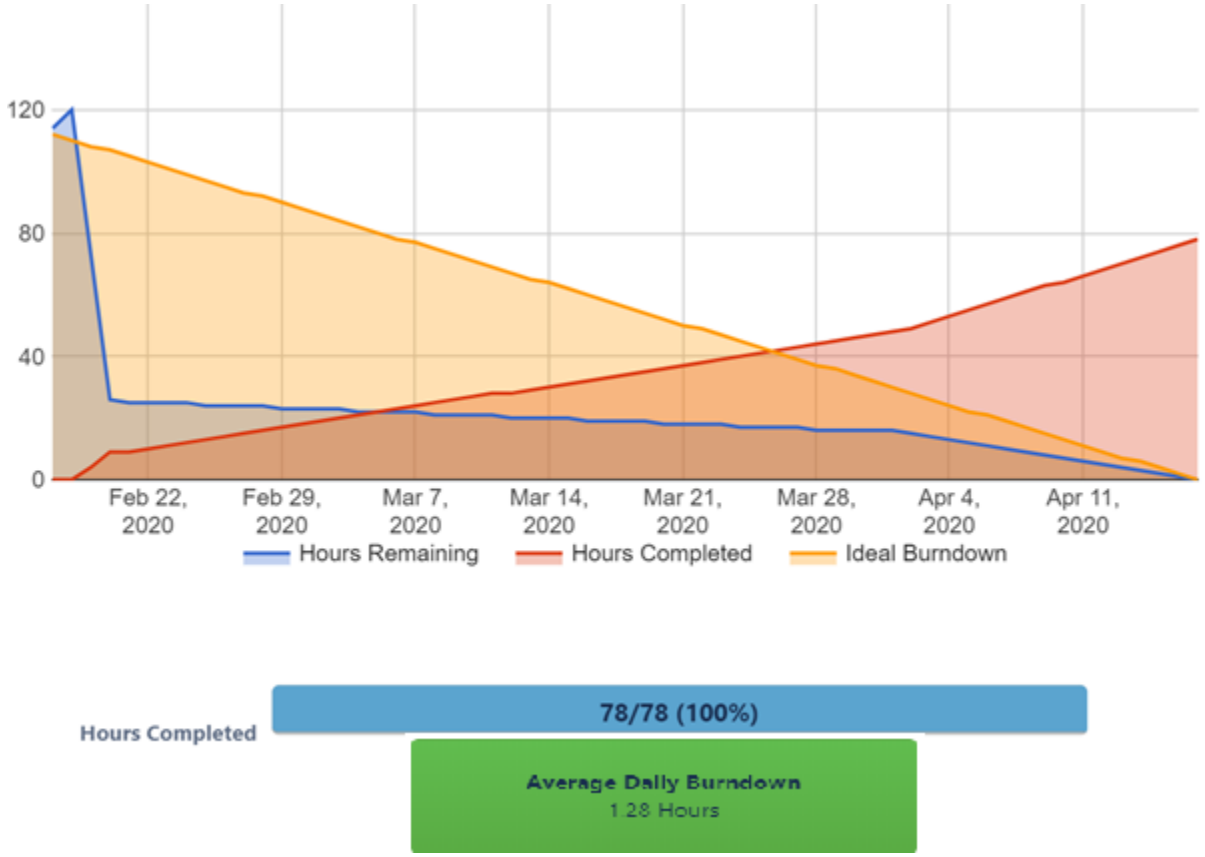

Figure 6 Burndown Chart

3. DEVELOPMENT OPERATIONS (DEV-OPS)

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

3.1. 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

3.2. GitHub repository

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

- Static development consisted of data scraper, Server and database related files, testing of Time series LSTM algorithms and documentation. Please find repository [11].
- 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. Please find repository [12].

3.3. 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 handy 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. List of Custom Packages installed can be found in EC2_packageList.txt [13]

3.4. 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.

Figure 7 Database architecture

3.4.1. 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 is 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.

3.4.2. 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.

3.4.3. 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.

3.5. 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. Follow the link to our running application [14].

4. BACK-END DEVELOPMENT

4.1. API Scrapers

Dublin Bike station data is made public through a 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 a 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**: Temperature parameters published by openweathermap.com are in degree Kelvin. They are converted into Celsius scale before logging.

4.2. Cron Program

Cron program periodically calls scraper to update database. Respective tables of database are updated every 10 minutes. Logs are maintained for data entries. Cron serves following purpose:

- **Threading** : avoid infinite loop to run scraper
- **Automation** : more object-oriented way to call scraper scripts

4.3. 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 . Flaks has very little boilerplate code hence it is lightweight than counterparts like Django. Flask has simple root task of rendering homepage of website on launch. Further, flask application serves requests from webpage, through defined functions on call.

4.3.1. 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.

4.3.2. 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.

4.3.3. 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.
Please refer notebook [15] for details.

4.3.4. get_prediction

Argument: None Return: jsonified availability, weather and temperature 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. Please refer notebook [16] for details.

4.3.5. current_weather:

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

Operation: Fetch weather entries order descending.

4.3.6. 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.

4.4. 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.

4.4.1. Timeseries Approach

Dublin bike users were mainly office going people and students. Hence, Bike use had definite periodicity governed by office time. Our analysis displayed the same when bike stand availability was plotted against time over 2 week data. As can be observed, 5 weekdays show high traffic from morning till evening and weekends are relatively diminished. Refer figure 7 (a). As can be seen in figure 7 (b); we obtained acceptable training loss and RMSE with this approach. Please refer notebook [17].

But, as can be seen in last section of fig 7 (a), traffic activity started to show unlikely disturbance. This happened because of “Stay at home” phase announced in Ireland in response of COVID-19 pandemic in late March, 2020. We had only 3 weeks’ worth ‘normal’ data and Time series needs valid pattern for prediction. Hence, we shifted from timeseries approach to random forest approach.

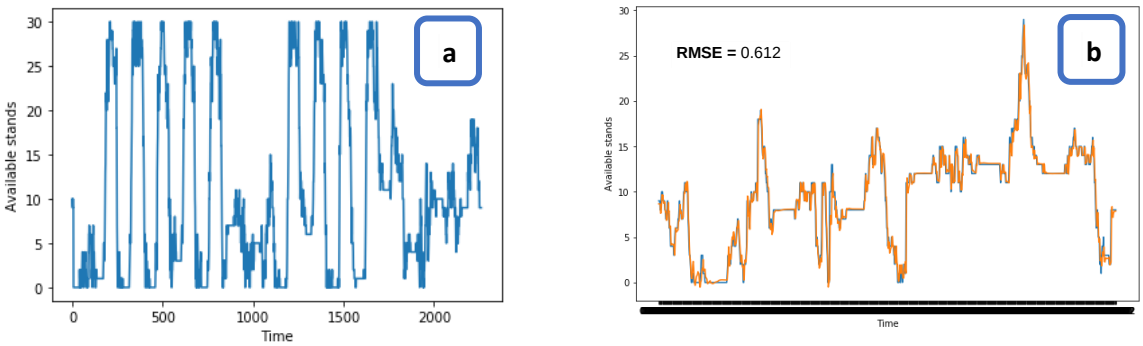

Figure 8 (a) Train data [Notice weekly pattern] (b) Train data validation[Test data Prediction]

4.4.2. Random Forest approach

Bikes and Bike stand availability is predicted using a 100 layered random forest regressor. While, main weather is predicted using a classifier. Architecture and data flow can be seen with help of block diagram shown in figure 8.

4.4.3. 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. Please refer notebook [18].

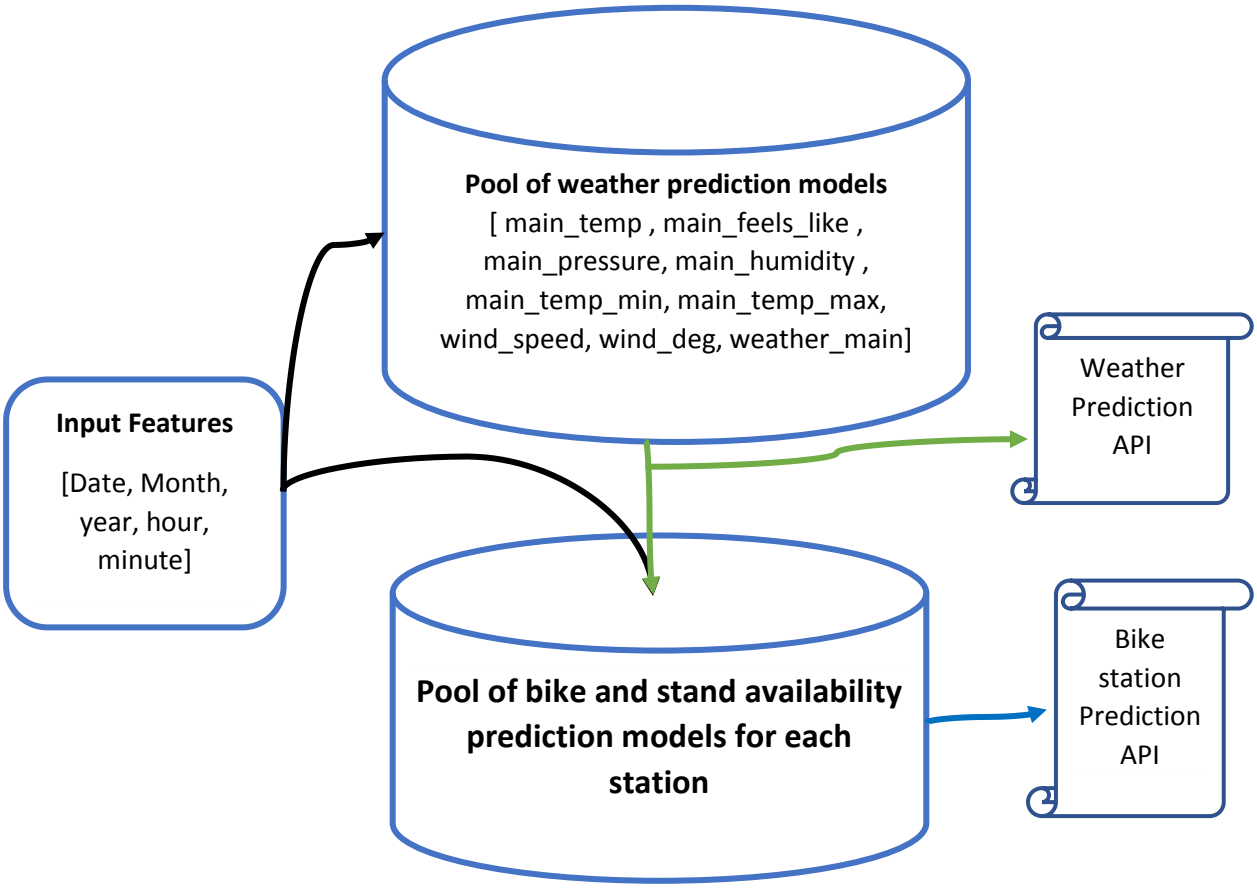

Figure 9 Data analytics model training process

4.4.4. 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. Please refer notebook [19].

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.

4.5. Accuracy check

By design, cron program is supposed to continuously scrape data. But given COVID – 19 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 [20]
- result_available_bikes.csv [21]
- resultWeather.csv [22]

Average performance of the models was found 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	0.256	0.173	0.416	0.982
main_feels_like	0.425	0.57	0.761	0.935
main_pressure	0.179	0.106	0.326	0.999
main_humidity	2.555	14.003	3.742	0.873
main_temp_min	0.278	0.205	0.452	0.980
main_temp_max	0.307	0.241	0.491	0.975
wind_speed	0.451	0.665	0.816	0.891
wind_deg	7.286	446.172	21.122	0.901

Figure 10 Metrics for trained Random Forest ML models

5. FRONT END DEVELOPMENT

5.1. HTML

This project is a single page application. Apart from navigation bar, individual functionalities work as distinct HTML5 divisions. Webpage is structured in 4 sections viz.

- **Header** displays Bike stations, about us and book a bike menu. In app navigation is used to traverse the page. Real time weather, temperature and Timing are also shown besides weather.
- **Station Map** shows all Dublin bike stations. Once user selects the station, availability for that station is displayed. Also, if user is interested in station traffic tendency over week and per hour, more info tab is provided.
- **Station selection** allows user to state source station to pick up bike and destination station to park bike in a bike stand. Time at both locations can be selected too. Search renders bike and stand availability at source and destination station respectively along with weather. Google maps directions API is used to show directions between stations on map.
  Additionally, station availability along with corresponding weather for next 5 hours by 30-minute interval is also displayed in form of bar charts. These charts are available to be downloaded as image.
- **Footer** is part of structure and does not have any valid links in deployment.

5.2. CSS

Webpage is stylized using CSS3 language. Website is single page application without using third party libraries like bootstrap. Hence, display property Flex is used for search boxes and loaders to make website responsive. Important CSS files are discussed below:

- Error_toaster.css : modal displays error encountered during rendering prediction results.
- Loading.css : Bouncing loaderball object is used to indicate that site is busy calculating and processing user data with ML model.
- Weather.css : Keyframes are used to create dynamic logo of weather. 5 major weather logos are used : sunny, moon, rainy, thunder, cloudy for expressing weather category.
- Main.css: Styling of drop-down menu for bike search is designed. Also, design of station history graphs is coded in main.css. Navigation within webpage is also controlled.

5.3. JavaScript

JavaScript plays important role in this application development as it seamlessly integrate with html. Moreover it makes web page interactive and respond the user instaneously using AJAX request. In this application app.js acts as system heart, It will get data from flask app.py file which acts like API interface for the application. The javascript file resides in folder static/js [23]. In this application we have used vanilla javscript with Jquery also in some cases we have used ES6 arrow function and we have used ApexChart in terms of visualization.

5.4. jQuery

jQuery makes easier use of javascript as it is fast and has many feature contained javascript library. One feature will make JQuery enrich is use of AJAX call that wil make API call in smoother manner. The code formatting and structure with the help of JQuery makes javascript more powerful. In our application we have use many features related to Dublin bike like in terms of getting all station data, search operation, showing history and chart about bike, It makes our application powerful.

6. PROJECT DELIVERY AND FUTURE SCOPE

Dublin Bike web application was a project delivered in form of a working open source website hosted on cloud infrastructure. All source codes for the site are in public domain through GitHub repositories. Project was completed and evaluated using Agile methodology metrics.

6.1. Salient Features

- Web application shows bikes and bike stands available at a Dublin bike stations at selected time.
- Prediction of availability for next 5 hours from chosen time is shown.
- Each predicted result is supplemented by real time weather forecast so that users can plan their commute.
- Weekly and hourly availability statistics for each station are available to view traffic tendency at a station.
- Bicycle route from source to destination is displayed.

6.2. Website Monitoring

We were concerned about website crashes. Implementing e-mail alerts for all errors is a tedious and clumsy task. Hence we resorted to google analytics API. It provides site analysis dashboard for performance monitoring. Our website is constantly monitored for:

- Website crashes
- usage across globe, devices on which is website is accessed
- Latency experienced by users
- Usage pattern and timing of website

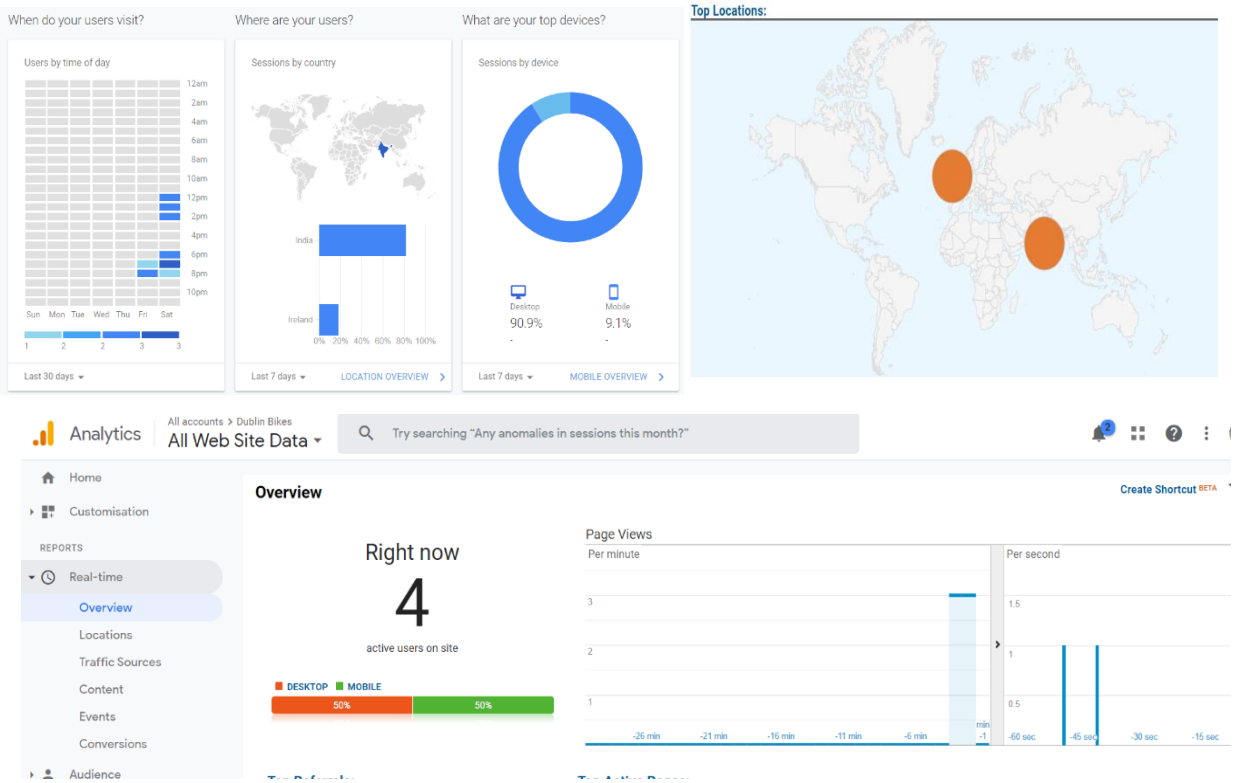

Figure 11 Google Analytics insights about traffic pattern, locations and site crashes [24]

6.3. Key Shortfalls and Planned Improvements

Due to COVID-19 situation, we missed nearly half of a sprint. And covering the backlog was challenging. Given the fact, we could not incorporate following implementation in our final submission:

- Display distance and bicycling time between 2 stations using google maps JS API
- Scrape data for more than 4 weeks and optimise machine learning prediction using time series prediction model
- Improve machine learning prediction model using third party factors like real time city traffic
- Provide user account to register feedback, and use their commute data within GDPR guidelines to improve prediction model
- Provide widgets and subscription service to mark stations favourite and get alerts about availability over notification/SMS/e-mail using a CPaaS like Telestax [25], Twilio [26], Amazon Connect [27]
- Implement threading in web infrastructure to process static data in background