Table of Contents

[Introduction 2](#_Toc6302150)

[Overview 3](#_Toc6302151)

[Objectives of the App 3](#_Toc6302152)

[Target Customer 3](#_Toc6302153)

[App Functionality 3](#_Toc6302154)

[Structure 4](#_Toc6302155)

[Features TO-DO 5](#_Toc6302156)

[Analytics: 5](#_Toc6302157)

[Architecture 6](#_Toc6302158)

[Execution 8](#_Toc6302159)

[Reflection 8](#_Toc6302160)

[Design Changes 8](#_Toc6302161)

[Forward 8](#_Toc6302162)

# Introduction

The following document outlines how we developed an application that was deployed to customers of the Dublin Bike network check for current and future occupancy. The application is a web application that helps users optimise their experience with Dublin Bikes.

The project had a 8 week timeline. In order to get this product developed in such a short timeline, we decided to use the Scrum methodology. We divided up the workload into 4, two week sprints. Using the principles of scrum, Stand ups, product backlog and review meetings, allowed us to achieve a considerable amount of work give the time frame.

Please read the following to get a better understanding as to how this was achieved. For demonstration purposes, screenshots of our various tools, drawings and minutes taken during meetings are attached when discussed. This is enough to get an understanding as to our team cohesion, but I encourage you to read explore the supporting documentation in their respective folders – versions are also provided to get a understanding into our progress.

**Website:** [www.mydublinbikes.com/home](http://www.mydublinbikes.com/home)

**Github:**

**Analytics:**

# Overview

## **Objectives of the App**

The objective of the app is to provide the Dublin bikes customers with a service that can: 1) Provide real-time data on the stations on the network, 2) Predict the occupancy of the station at a day and time specified the user 3) Provide information on how users can use the network.

One of the striking things about the current offering is the lack of attention to the users experience with it. We set out with a very clear objective – ‘to build a better, quicker and more user friendly alternative’. Because of this we placed a lot of weight to the UI/UX.

The applications predictive model is built on previous occupancy and other relevant features - this delivers an estimation on the number of bikes that will be available at that time. Given the nature of the service, weather is going to play an important role in the use. Based on that, we have also incorporated it into the model.

## **Target Customer**

At the beginning of the project, we set out to see who the demographic is that uses the Dublin Bike network. Out requirements gathering (Please see Pg.3 of SRS) was brief but it gave us an indication on who uses the network.

1. **Young Professionals (23-32)**: This segment used the network the most frequently. The most common uses cases was:
   1. Cross city community
   2. Transport to meetings
   3. Lunch gatherings

This group commonly replaced the Dublin Bikes with Taxi’s or the Dublin Bus network. The segment are very useful for helping with the

1. **Leisure (20-55):** This group spanned across the all age groups and times of the week. These are arguable the hardest to group to predict and target as the uses is often sporadic and unpredictable.
2. **Students (18-22):** A small group but still a user profile. There use is often irregular and only used for short hops (Most students seemed to have their own bikes).

## **App Functionality**

Please refer to the SRS for a more comprehensive outline

The application has three primary functions – current status, prediction, information. We originally want to

The user should is to allow user to see the current status of the Dublin bike stands network in Dublin. They will also be able to requests a time and day that they wish to travel and be returned the likely number of bikes in that stand.

The product will give the user the weather for the day. If a prediction is made, the weather for that day will also be provided. The product will also display an ‘availability graph’. This graph will show the typical number of bikes in that stand a time interval throughout the day.

The product will also provide the user with information on how to use the DublinBikes and a contact page. There will also be a page that allows the user to sign-up to the network so that they can ‘Get Ridin’.

Finally, the product will be functional and lightweight, meaning that the user will experience near full interactivity.

## **Structure**

Below is a representation of the structure of our application. It contains four pages titled – *Station*, *How it Works, Subscription, and Contact.*

A screenshot of a cell phone

Description automatically generated

***Page Layout:***

As outlined in our SRS, we set out to create an application that reduced time spent on navigating/clicking etc. Our research concluded that most Dublin bike user that use the existing site do so to see the current status of the bike network. This insight drove us to design the website so that the user immediately lands on the page that is needed.

***User Input:***

As stated, we wanted a high click: value ratio.

*Home Page:*

This was a driving factor for the inclusion of only 4 inputs fields on the websites home page – These are made up of:

1. *Address*: Zooms in to the area in which you have stated (autocomplete functionality)
2. *Station Click:* Clicking on a stations displays information on the station and also auto-completes the predictive element on the ‘predictive form’
3. *Time:* Enter a time
4. *Day:* *Enter a day ( four day limit implemented – Year and Month are already inputted)*

*Contact Page:*

This page consists of a simple ‘contact form’ that allows user to input: email, name, subject and message. Disclaimer: This form does not submit to anything.

***Links:***

The ‘subscription’ page includes links to the app store and to the partners –‘Just Eat’- pages. We have also include the Google, Facebook and email sign up options for new users to the network.

## **Features** TO-DO

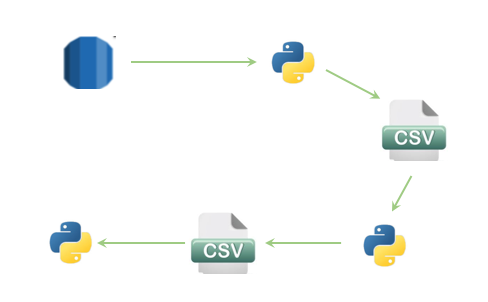
Please refer to the SRS for a more comprehensive outline

The features that we wanted to include in our

## **Analytics:**

***Github Notebook:***

The analytics model is created using a Random Forest Regression model. Our database, containing all the weather and Dublin bike information that has been scraped from the API is queried. The result is stored in a csv file. Once cleaned, we use the sklearn Random forest regression module. This creates a random forest for each station on each day – giving us a accurate model for predicting the availability of bikes



Example of a tree – Grand Canal Station (Thursday):

A picture containing photo

Description automatically generated

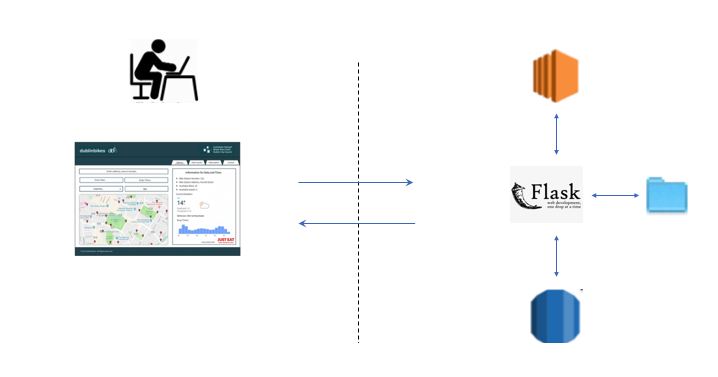
We had built the predictive model using Linear Regression – by encoding the categorical features – but found this option to be very inaccurate. Our decision to use the Random Forest was based off: 1) The good predictions that we were getting, and 2) the abundancy of literature that support this type of modelling for a problem such as the one we faced.

I encourage you to read through our notebook. It is short and is accompanied by comprehensive markdowns explaining our decision and some analysis.

## **Architecture**

Below is a basic graphical representation of the architecture of our application.

**Client Side** **Server Side**

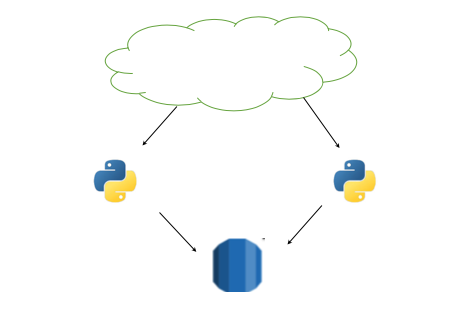


***Client Side:***

Our client side is built using *HTML, CSS and Javascript (JS + JQuery).* This is a static page that receives data from the server side. The page is rendered using this data.

***Server Side:***

The server is where all of the heavy lifting is carried out. The flask application sits on an EC2 instance. Our database is ran on a AWS RDS instance. The RDS is populated through 2 separate python scripts. These scripts scrap the weather and Dublin bike’s API and insert the information in the database.



Our predictive models are stored on the server. Before the models are created, we must first query the weather and bikes database for all the data. When these queries are stored, we then run a python script that creates a csv containing all the information needed to make a predictive model.

Using another script, we clean the data and produce another csv with the adjustments needed to run the model. Finally, we run a final scrip that creates a model for each of the stations on a particular day. These models are pickled and stored.



A close up of a clock

Description automatically generated

The flask application receives requests from the web pages and process the requests based on the route called. These requests are made through the browser and hit our uWSGI application server and then is configured using Nginx that acts as a front-end reverse proxy. This is required because the web cannot talk to python. Therefore, uWSGI is implemented to get around this. The Nginx is used as a load balancer.

# Execution

* Talk about the scrum methodology, attack screenshots of conversations / meetings / Product backlog across the sprints

# Reflection

## **Design Changes**

* Refer to slack & Stephen Notes
* Random Forest over Linear
* **ONLY CHROME**

## **Forward**

General reflection on the project