A close up of a sign

Description automatically generated

4/20/2019

Client: JCDecaux

Team: TheVillageBikes

Dublin Bikes Application

Table of Contents

[Introduction 2](#_Toc6472634)

[Overview 3](#_Toc6472635)

[Objectives of the App 3](#_Toc6472636)

[Target Customer 3](#_Toc6472637)

[App Functionality 3](#_Toc6472638)

[Structure 4](#_Toc6472639)

[Features 5](#_Toc6472640)

[Analytics: 6](#_Toc6472641)

[Architecture (GAFFNEY) 7](#_Toc6472642)

[***Client Side*** 7](#_Toc6472643)

[***Server Side*** 7](#_Toc6472644)

[Execution 8](#_Toc6472645)

[Planning Sprints 8](#_Toc6472646)

[Managing Workload 9](#_Toc6472647)

[Product Backlog 10](#_Toc6472648)

[Ensuring the Magic Happens 11](#_Toc6472649)

[***Stand-ups*** 11](#_Toc6472650)

[***Sprint Reviews*** 12](#_Toc6472651)

[Summary 13](#_Toc6472652)

[Design 14](#_Toc6472653)

[Intro 14](#_Toc6472654)

[Requirements 14](#_Toc6472655)

[Changes 14](#_Toc6472656)

[***UI/UX*** 14](#_Toc6472657)

[***Model*** 15](#_Toc6472658)

[***Functionality – Technical Choices*** *(Add and change this around)* 15](#_Toc6472659)

[Forward 15](#_Toc6472660)

# **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:** [mydublinbikes.com](https://www.mydublinbikes.com/home)

**Github:** [Repository](https://github.com/SteGaff7/TheVillageBikes)

**Analytics:** [Notebook](https://github.com/SteGaff7/TheVillageBikes/blob/master/Random_Forest_Final.ipynb)

# **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**

Please refer to the SRS for a more comprehensive outline

The application is lean, but it packs a punch. When deciding on what to include in the application, we looked at the current offering as a baseline. This current offering offered to user the most up-to-date information on each stand. The UX/UI was poor. This, as already mentioned, was something we really wanted to improve on.

Powered with this, and the brief, we set out to build an application that would give the user the most up-to-date information for each station – including weather. We targeted a predictive model that had an accuracy of +/- 3 bikes. Finally, we endeavoured to put a lot of emphasis on the UX/UI. User often came to the application in a rush, so it only made sense that the application was built with this in mind.

This was what we intended, and that is what we built. Mydublinbikes has design at the heart of the application. The current information gives the user the most up-to-date data on each station, the current weather across the local, and gives an estimation, in the form of a graph, on the availability for the station throughout that day.

Our predictive model has a cross-validation MAE of between 3-4 bikes. We had targeted 3, so this was a good outcome. We expect this figure to only improve over time as the model is trained a larger dataset.

Some auxiliary features we implemented include :

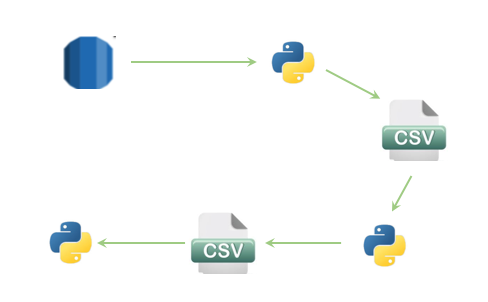
1. Buttons to toggle between stations for dropping off and collecting bikes
2. Graphical representation of the predictive times (Modelled of the ‘busy’ graphs that are commonly seen when Google searching a public location)

## **Analytics:**

***Github:*** [Notebook](https://github.com/SteGaff7/TheVillageBikes/blob/master/Random_Forest_Final.ipynb)

All of our modelling was originally done in Jupyter notebook. In order to work with the models on our EC2, we transferred the finished script to a .py file and ran on our EC2 instance. I will now describe what model we used and the reasoning behind it – I do encourage you to refer to the notebook attached above, it is short and is accompanied by comprehensive markdowns explaining our decision and some analysis.

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 using SQL statements. 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

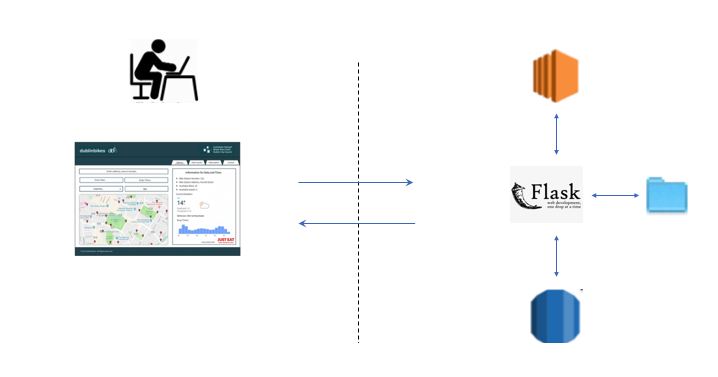
We had built a predictive model using Linear Regression – by encoding the categorical features – but found this option to be very inaccurate. Our decision to use the Random Forest was based off the superior results that we were getting form the Random Forest. We used cross validation on both the Linear and the Random Forest, particularly paying attention to the R2 score and the Mean Absolute Error – ensuring to use a train\_test split. The Random Forest Model was returning greater accuracy in both the train and test results. For these reasons we decided to use the Random Forest.

Another thing that we did consider was the abundance of literature that support this type of modelling for a problem such as the one we faced. We also talked to some experts in this field and they confirmed our results and research.

## **Architecture (GAFFNEY)**

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

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

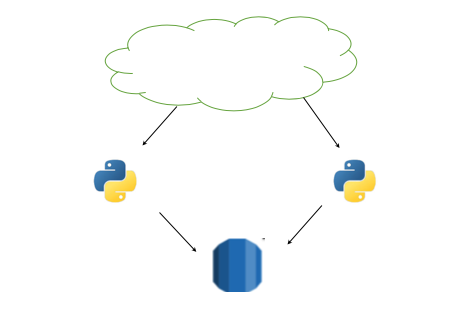


### ***Client Side***

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

The project followed the Scrum methodology. This meant that we used timelines called Sprints. These were divided into two week intervals. Before each sprint we would plan out the features that we wanted to complete. We assigned and estimated time that it would take for these features to complete. Using the ‘divide and conquer’ approach requires well placed communication channels. To bring all of this together, we used a number of tools including – Slack, Trello, Excel, Google Docs and WhatsApp.

Below is demonstrates how we used Gantts Charts, Burndown Charts, Product Backlogs, Stand-ups and Sprint Reviews. These are only screenshots of our actual documentation. All versions and final documentation is accompanied in their respective folders and I encourage you to explore them.

## **Planning Sprints**

Below is a demonstration of our Gantt chart. This was used for the most high-level objectives for each sprint. It also assigned a lead to each of the features (workload was shared but there was always someone who was responsible for finishing the feature). As you can see there a 4 sprints with a two week gap in the middle (reading weeks). Looking at this, we can see that all of the features were complete on time – excluding the requirements which was always needed contributed.

A screenshot of a cell phone

Description automatically generated

## **Managing Workload**

Below is an example of the burndown charts for two of the sprints. These are graphed with the Expected Hours vs. the Actual Hours. As this was the first time that we had used the Scrum methodology, we can see that our estimations were off for the first Sprint - surprisingly overestimating the time taken for the features to be complete.

However, as you can see began to improve on our accuracy. Our thirds sprint demonstrates this. In our opinion we think that this was to be expected. Our key takeaway here is that we continued to learn each other’s capabilities and strengths.

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

## **Product Backlog**

**Trello:** [**Our Boards**](https://trello.com/thevillagebikes)

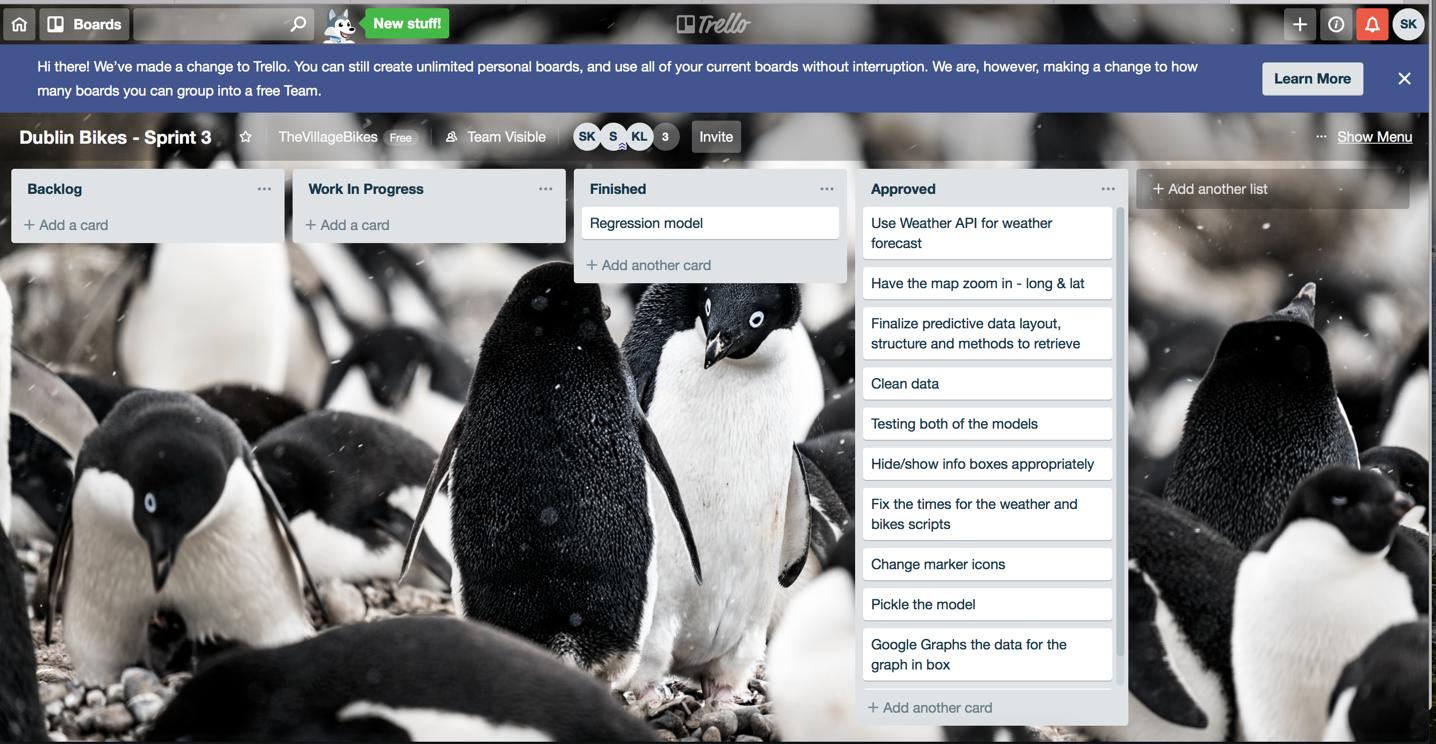
We decided to use a Trello board for each sprint. The purpose of this was to track all of the features that needed to be completed. This served as a granular version of the Burndown and Gantt charts. We took the features and broke them down into sub-problems that needed to be built. Each feature was either in the Backlog -> Work in Progress -> Finished or Approved. Below are screenshots from our Trello Boards.

We found this very useful. Between all the different methods of communication it can be hard to see what has been built and what is still in WIP. Frequently we would be working on inter-related features. We found slack very useful for trying to get these interdependent features working.

A screenshot of a social media post

Description automatically generated

***Sprint 2 – Beginning***



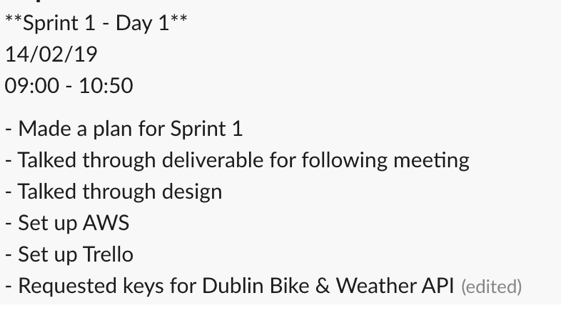
***Sprint 3 – End of Sprint***

## **Ensuring the Magic Happens**

Although the above tools help with breaking our problems down and then dividing, there would simple not be the product today if we had not done stand-ups and sprint reviews. These shaped the continuous improvement we saw throughout the project. Whatapps and Slack were both incredibly convenient, and brought a lot of value, but it was the stand-ups and reviews that allowed the team know exactly what people were working on, re-assess benchmarks, allocated more resources and help develop the ethos within the team.

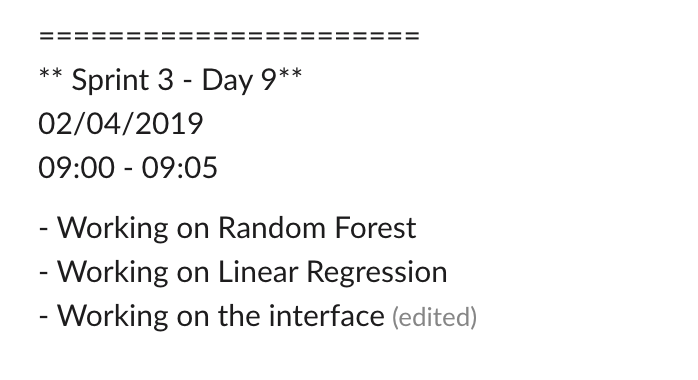
### ***Stand-ups***

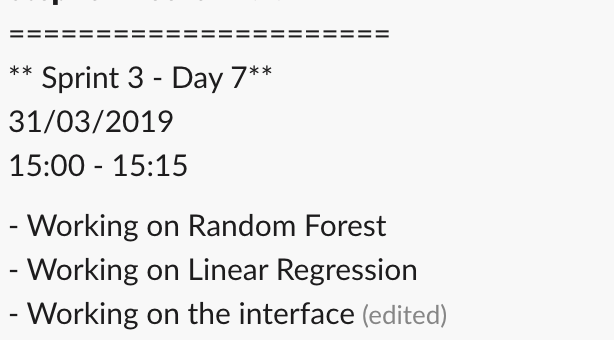
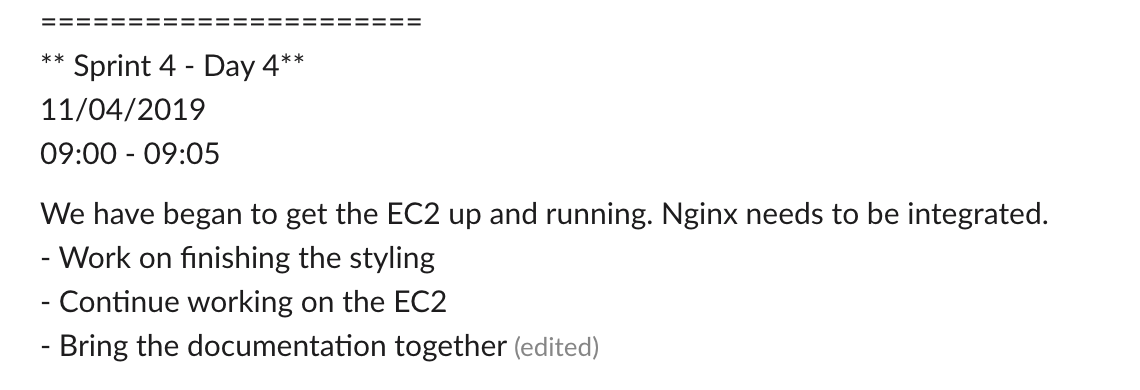
The effectiveness of these cannot be underestimated. From the beginning we adopted our stand-ups as a feature assessment and delegation exercise. Each day we would say how the last days features were coming along. If complete, features were assigned from the product backlog. If feature was taking longer, more resources were appointed. This worked incredible well. We also believe that this was the driving factor for us finishing our backlogs in advance – at the beginning. We did not have sprint meetings every day. However, they tended to occur on Monday, Tuesday, Thursday and Friday. On days were we would have a gap between sprint meetings, an adequate workload was assigned. Please see below for an illustration on some of the logs that we took.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

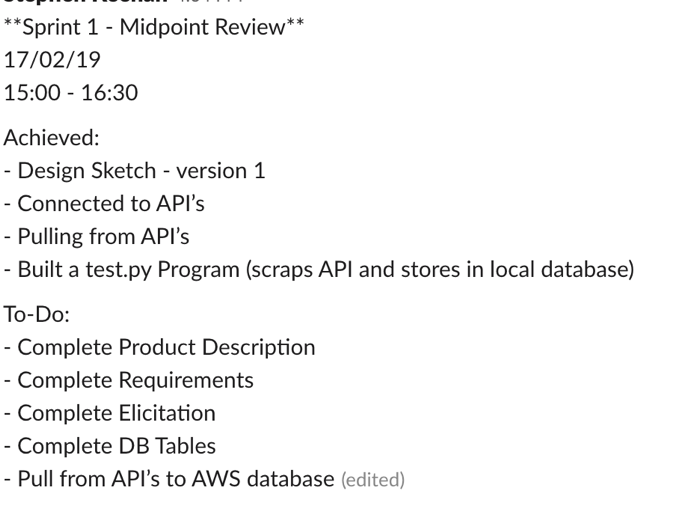
Description automatically generated



### ***Sprint Reviews***

Sprint reviews were valuable. They would often take the form of – Review, Thoughts and Plan. At this point the features were decided on. Gantt and Burndown estimates were made and a note was taken on what was discussed. The most value add part of this was definitely in the Review/Thoughts. This shaped how to approach the next sprint. As we have already mentioned, at the beginning there was issues with backlog timing estimations and communication breakdown. We were honest, with what we thought, needed to be improved upon. Our aim then was trying to rectify this in the next sprint – for the most part succeeding.

Below are a few screenshots from our reviews.

***A screenshot of a cell phone

Description automatically generated***

***A screenshot of a social media post

Description automatically generatedA screenshot of a cell phone

Description automatically generated***

## **Summary**

In summary, I think we all agree that the scrum methodology was a success. At the beginning it was difficult to remember to put the time into all of the documentation etc. when all you could think about was coding. However, in hindsight we are satisfied of the returns from it. Who knows what product we would be shipping without it.

# Design

## **Intro**

We have already touched on how the application is structure. However, this is a good point to explain our decision around some of the design. As previously mentioned, user experience was a big element in this application – partially because the current model is so poor.

## **Requirements**

The objective of the app was for the user to have little interaction with the app. This allowed us to control the input while also keeping the features simple and functional for the user.

As per our SRS, we aimed to produce an application that had design at the core. It aimed to:

1. **Disability**: The application should adhere to best practiced standards to accommodate those with a disability (WCAG, DDA)
2. **Prediction**: Create predictions to for the user up to 4 days in advance
3. **Minimal**: Produce data that is valuable to the user, with as little navigation as possible.

We believe that we have achieved this. The user, when he lands on the page, is immediately confronted with the map. They can:

* Immediately click on a station to get up-to-date information.
* Fill in the form – date, time and location – to get a prediction on the number of stations at that time

Weather is also provided as part of the current and predictive display.

Our SRS outlined that we would build a ‘rout planning’ and a ‘entertainment box’ feature. Unfortunately, these were not possible in the time frame of the product. As these were only ‘value-adding features’ it was hypothesised that they would not have been able to be completed in time.

## **Changes**

There were a number of design changes made throughout the project. Below are some examples:

### ***UI/UX***

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Once the prediction model was built, we quickly realised that our current design may have to undergo a re-design. When the user lands on the page we did not foresee that the prediction (right-side) would be empty. For this reason we have placed a diagram illustrating on how to use the network. Once a predication is made, the page displays out original design.

### ***Model***

As previously mentioned, we opted to use a random forest model, rather than a linear regression. Initially we did not understand which model to use. We initially thought that the linear regression would work. It functioned well under tests. However, we delved further into the problem and discovered that a random forest may be a better option.

As described above, we build both predictive models and tested. The random forest preformed superiorly to the linear regression.

### ***Functionality – Technical Choices*** *(Add and change this around)*

Below are a list of design choices that were made throughout the project:

1. jQuery rather than POST/GET
2. pymySQL rather than mySQLAlchemy: The latter often became temperamental when moving between devices
3. nohup – logs messages created by processes. This was needed when our scripts often were killed. The reason on which was not known.

# Forward

Despite the application been functional and containing extensive features, there are still aspects that we could like to build out and improve. In the following I will be describing some of these aspects:

**Weather API**

The current weather API only gives information for the next 5 days. If we had more time we would have further investigated a API that would allow for further predictions. This would be built into our model.

WE have discussed this in depth, and hypothesis that we would also have to build in a ‘weather accuracy metric’. This would be an index on the accuracy of a weather forecast. Inherently, the further away the prediction, the less likely it is to be correct. Building this into the model would have been both interesting and challenging.

**Sign Up**

We would have liked to set up a function that would allow for the user to sign up to the Dublin bikes network, or at a minimum, download the app.

**Incorporate Advertising**

Going forward we would incorporate advertisement within the page, reaching out to ‘JustEat’ to sponsor some of the space.

**Entertainment**

There are a number of large events that occur around Dublin. Many people use the dublinbikes network to access these events. If we were to continue building the platform, we would build these events into our model.

Further investigation would have to be carried out, however, techniques such as web scaping and using certain API’s could be used to identify events that effect our model.

**Server Maintenance – Automation**

Currently our model is running on a EC2 server that has only 1GB Memory and 30GB Disk space. The implication of this means that our models and csv’s are all created locally and then pushed to the EC2 instance. For forward we would like to carry this operation out on the instance – earmarking for the whole process to be done on a Sunday at 8pm. Simple bash script would automate this.

ADD SCRIPT SYNTAX

# Summary

The individual reflections will tie up our experience of the project, but as a team we have learnt to much. Coming into the semester we were ignorant to how an application such as this would be built. ‘Learning on the fly’ is a phrase that comes to mind when we think about this. Some of the things we have been exposed to:

* API integration
* Database Management
* Cloud Management
* Flask
* Predicative modelling
* Hosting
* Scrum methodology