

|  |  |
| --- | --- |
| **Name** | **Student Number** |
| Stephen Keenan | 14398016 |

COMP30830

Software Engineering

**Software Requirements Specification:**

To Develop an Automated Assessment Plug-in that is Compatible with the CSMoodle Page.

**Client**:

UCD Computer Science Department

Table of Contents

[Introduction TO-DO 3](#_Toc1503545)

[Purpose 3](#_Toc1503546)

[Scope 3](#_Toc1503547)

[Definitions 3](#_Toc1503548)

[Overview 3](#_Toc1503549)

[Requirements Analysis Summary 3](#_Toc1503550)

[Data Collecting 3](#_Toc1503551)

[**System Observation** 3](#_Toc1503552)

[**Customer Interview** 3](#_Toc1503553)

[**Brainstorming** 3](#_Toc1503554)

[Requirements Analysis 3](#_Toc1503555)

[Selecting Data 3](#_Toc1503556)

[Analysing Data 4](#_Toc1503557)

[Overall Description 5](#_Toc1503558)

[Product Perspective 5](#_Toc1503559)

[**System interfaces** 5](#_Toc1503560)

[**User Interfaces** 6](#_Toc1503561)

[**Hardware Interfaces** 6](#_Toc1503562)

[**Software Interfaces** 7](#_Toc1503563)

[**Communications Interfaces** 7](#_Toc1503564)

[**Memory Constraints** 7](#_Toc1503565)

[Product Functions 7](#_Toc1503566)

[User Characteristics 7](#_Toc1503567)

[Constraints 7](#_Toc1503568)

[Assumptions & Dependencies 8](#_Toc1503569)

[Appointment of Requirements 8](#_Toc1503570)

[Requirements Statement 8](#_Toc1503571)

[User Requirements 8](#_Toc1503572)

[Functional Requirements 10](#_Toc1503573)

[Non-Functional Requirements 12](#_Toc1503574)

[Performance Requirements TO-DO 13](#_Toc1503575)

[Domain Requirements 13](#_Toc1503576)

[Design Requirements TO-DO 13](#_Toc1503577)

[Change Management Process 13](#_Toc1503578)

[Approval 14](#_Toc1503579)

[Appendix 15](#_Toc1503580)

­­

# **Introduction TO-DO**

## Purpose

## Scope

## Definitions

## Overview

# **Requirements Analysis Summary**

## Data Collecting

The following approach was taken during the requirements elicitation process. Requirements were elicited through (1) System Observation, (2) Customer Interview (3) and Brainstorming.

### **System Observation**

The first method of elicitation was the observation of the current system. Through this, we aimed to have a greater understanding of who the system users were and how they interacted with the legacy systems. This ensured that we were not missing any pinch points.

Our research was focused on two main observations: general observation and the current DublinBikes web application. General observation was constructed from observations that we could make including: general experience; accuracy of information; and product offering. The DublinBikes application was reviewed by the team. We decided to borrow a common business practice called ‘SWOT analysis’.

### **Customer Interview**

The second method of elicitation was interviewing customers of the DublinBikes network and web application. We used people who we knew have used the service. The customers ranged in age and usage.

### **Brainstorming**

We believed that the technology that was required to produce this application was already been successfully implemented in a number of other industries. For this reason, we decided to research what was currently being offered individually and concluded by continuing the process as a team.

## Requirements Analysis

## **Selecting Data**

We were aware that there was a considerable amount of information collected as a result of the various elicitation techniques deployed. This required us to select the most relevant information that was applicable when determining the products requirements.

## **Analysing Data**

We chose to divide up the data that we gathered into clusters. We broke down each of the operations of the product into its own parts: (1) User Interaction / Information Display (2) Analytics (3) Display (4) Accuracy

The segregation allowed us to accurately identify what was needed. By dividing up the elements of the product, we see the features that should take priority. Using the information from the system observation, customer interview and brainstorming session, we outlined the features that were pinnacle to the success of the product.

* User Interaction / Information Display
* Analytics
* Display
* Accuracy

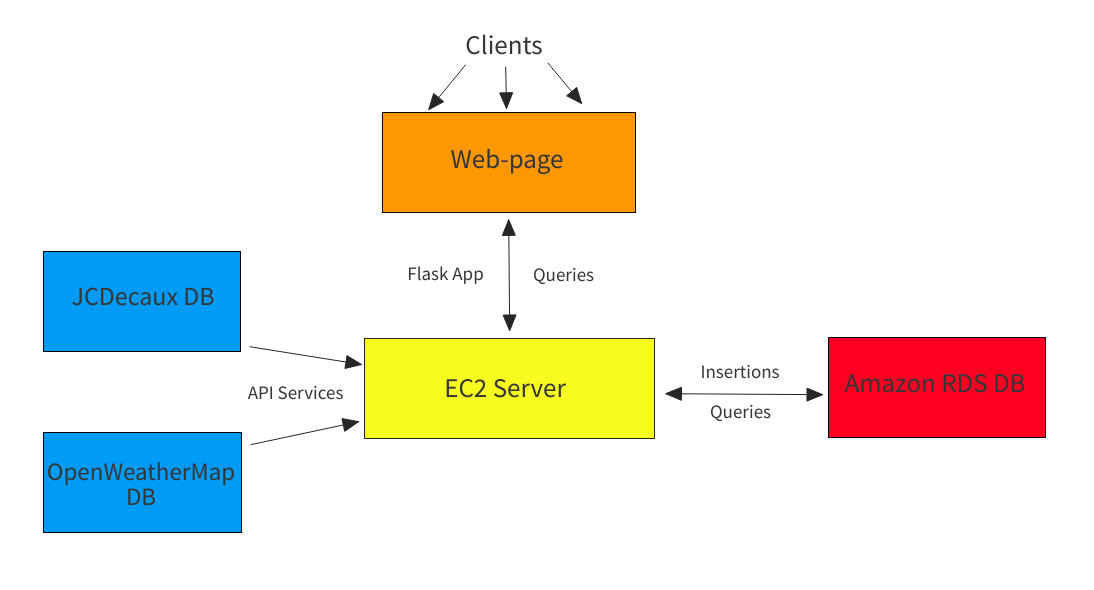
***Our requirements analysis highlighted the shortcomings of the current system. We believe that we have equipped ourselves well for producing a product that understands the needs of our customers. It is envisaged that the product will be deployed fully to the Dublin Bikes domain.***

# **Overall Description**

## Product Perspective

The VillageBikes will be a web application for Dublin Bikes. It will not operate independently and will require a web-browser to run. Dublin Bikes website offers a similar service however this system will simply be a web-page page that will provide a clearer more interactive service to the user. This system will be similar to an existing mobile application called “Just Eat dublinbikes”, however this app is currently only available on mobile.

An overview of the system interfaces can be seen below:



### **System interfaces**

*JCDecaux:*

* System must have an interface with JCDecaux which allows it to scrape static and dynamic data from the database via an API
* System must not exceed daily request limit resulting in system becoming banned

*OpenWeatherMap:*

* System must have an interface with OpenWeatherMap which allows it to scrape dynamic weather data from the database via an API
* System must not exceed daily request limit resulting in system becoming banned

*MySQL Database:*

* *System must have access to MySQL database to create databases and tables*
* System must have an interface with the MySQL database that allows insertion of scraped data
* System must have an interface with the MySQL database that allows querying of data from the tables in the database

Web-page:

* The client web-page will be the graphical user interface between the end users and the flask application running on the EC2 host server

*Flask Web-page:*

* *The flask app will sit on the server. It will do the ‘heavy’ lifting for the web page. The data will then be sent, by the flask app, to the HTML static page and render the static HTML and CSS as well as the data sent.*

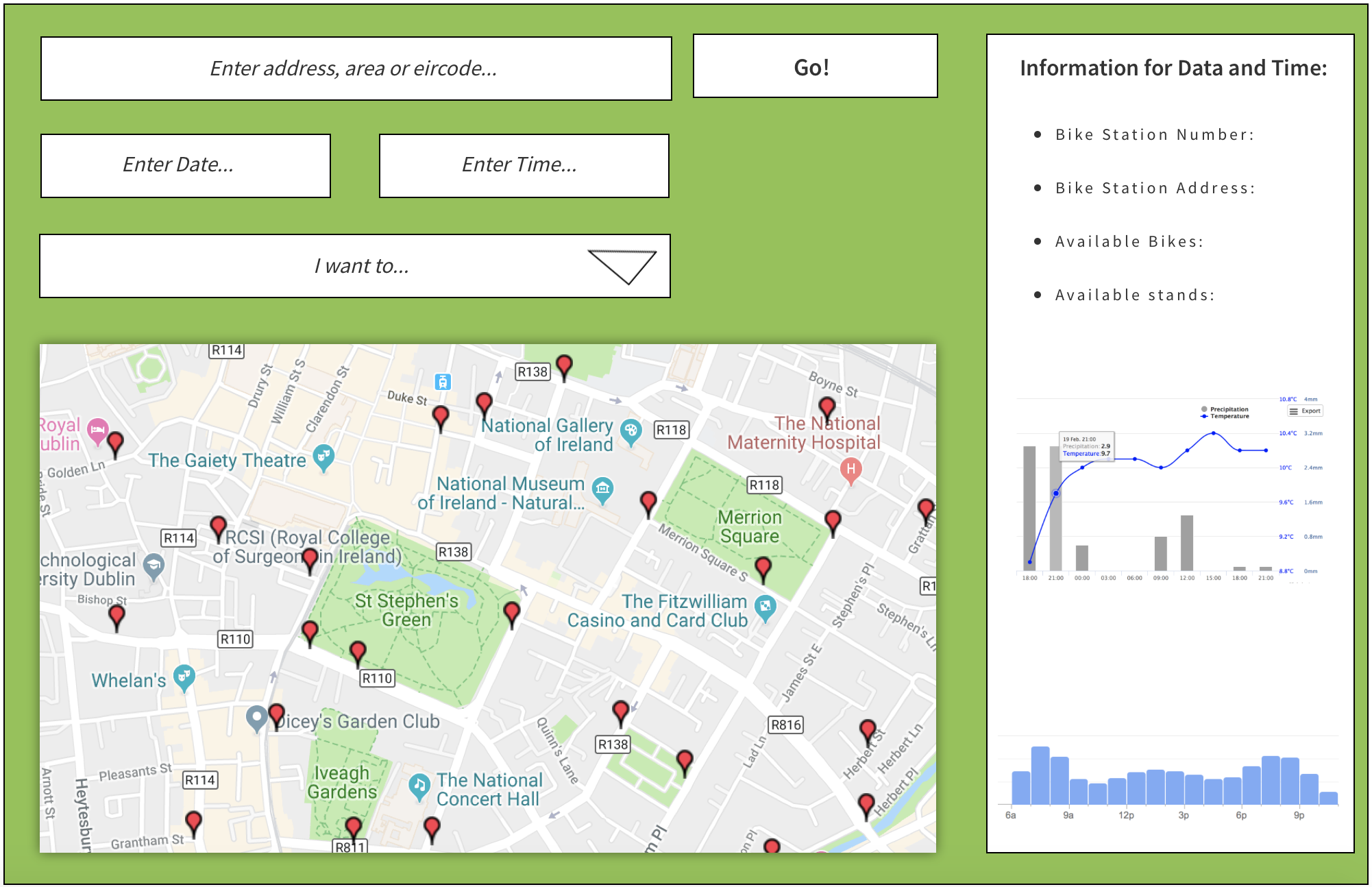
### **User Interfaces**

VillageBikes will use a GUI between the user and the application. The application will be a web-page accessible via a web browser. The GUI will be interactive with filters, pins, pop-up displays and submission buttons to filter specific parameters.

For a list of specific functions within the GUI see user and functional requirements.

*GUI:*

* System must have a GUI that will operate with web browsers
* There will be one page (Index)
* There will be an option to access real-time data or future data estimates
* There will be a map with pins representing bike stations
* Each pin will be interactive and will display all relevant information of that particular bike station including the weather
* Some simple statistical analysis through graphs and charts will display representing bike/stand availability and busy periods



### **Hardware Interfaces**

*VillageBikes will interact with a Linux host machine on EC2 to run the scraping and flask applications. The MySQL database will interact with the Amazon RDS server through the built in Amazon RDS protocols and services.*

### **Software Interfaces**

* *Operating System of host (EC2) server: Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1021-aws x86\_64)*
* *MySQL (Version 5.6.40) of Amazon RDS database*
* *API of JCDecaux Developer to scrape static and dynamic data from the database*
* *API of OpenWeatherMap to scrape dynamic weather data*
* *?Software web-page/flask app? EC2 interacting with web-page through the flask application*

### **Communications Interfaces**

* *Communication between EC2, the APIs of JCDecaux and OpenWeatherMap and the Amazon RDS database will be needed.*
* The system must use MySQL Connector to connect with JCDecaux and OpenWeatherMap databases to retrieve the data.
* The system must use MySQL protocols to connect with the Amazon RDS database to query database.
* *The flask app will sit in the EC2. Using uWSGI and nGINX to deal with the routing when user request the page URL.*

### **Memory Constraints**

* The RDS database must have enough storage to store scraping of data from the APIs for 10 weeks.
* The host server must have enough CPU utilization and RAM to run the API scraping scripts continuously and support the flask application hosting the website.
* Users must be able to run a web browser to access this application hence their machine must meet the following requirements:
  + Google Chrome requires 100MB of free hard drive space and 128MB of RAM
  + Mozilla Firefox requires 512MB of RAM / 2GB of RAM for the 64-bit version and 200MB of hard drive space

## Product Functions

The should be able 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.

## User Characteristics

The user of TheVillageBikes is to use cycling as a form of commuting. The user is to be familiar with Dublin bikes whether commuting by cycling alone around Dublin City Centre or combining the use of Dublin Bikes with public transport or car.

A low-level of technical expertise in computers and website navigation is required as the web application shall have a simple, easy to use interface.

## Constraints

There shall be no hardware interfaces for TheVillageBikes. Thus, there shall be no hardware constraints that will limit the developer’s options. The bike/stand availability prediction of each station shall be dependent on the data collected from OpenWeather and JCDeaux. Therefore, if the collection of data is corrupted, it shall create inaccurate predictions.

## Assumptions & Dependencies

It shall be assumed that:

· The user has a functioning device that has access to internet.

· The data collected from OpenWeather and JCDeaux is correct.

· The appearance of the web application shall be the same on all devices.

· The web application is responsive on all mobile phone devices.

## Appointment of Requirements

The plug-in is being built with a very specific purpose. It is important that the offering stays lean and to the specifications of this document. As seen in the following section, the requirements are divided up in priority – core functionality taking precedence.

# **Requirements Statement**

## User Requirements

The following are the features that the application will perform. This is an overall of an existing system, and as such is a considerable task to undertake. Therefore, and with the deadline in mind, we have decided to prioritise the features to be built. These features are listed under the headings: (1) Essential Features (2) Value-adding features (3) Non-Essential Features.

## 

**Essential Features**

1. Display all Dublin Bike stations:

* By default, the web application shall display all bike stations in Dublin on a map
* The user shall have the opportunity to zoom in / zoom out on the map
* Each station shall have a pinpoint icon where the user has the ability to find out more information about a specific bike station, including; station number, station address, available bikes and available stands
* This information shall appear in a large box on the right of the user’s screen

1. Display stations for collecting a Dublin Bike:

* The user is to click the “Collect” button within the drop-down menu that displays “I want to…” to gain information about collecting a Dublin Bike
* The user shall have ability to view bike stations on a map that possess bikes that are available to collect
* The user shall have the opportunity to zoom in / zoom out on the map
* The colour of the pinpoint icon on each station shall indicate whether a station has available bikes or not.
* Red shall indicate that a station has no available bikes to collect
* Green shall indicate that a station has available bikes to collect
* The user shall have the ability to gain specific station information by clicking on a pinpoint. This information shall include; station number, station address and the number of available bikes

1. Display stations for dropping off a Dublin Bike:

* The user is to click the “Drop-off” button within the drop-down menu that displays “I want to…” to gain information about dropping off a Dublin Bike
* The user shall have ability to view bike stations on a map that possess free bike stands
* The user shall have the opportunity to zoom in / zoom out on the map
* The colour of the pinpoint icon on each station shall indicate whether a station has bike available stands or not
* Red shall indicate that a station has no available stands to drop off a Dublin bike
* Green shall indicate that a station has available stands to drop off a Dublin bike
* The user shall have the ability to gain specific station information by clicking on a pinpoint. This information shall include; station number, station address, and the number of available bikes stands

1. Display stations that accept card payment:

* The user shall have access to information regarding the bike stations that accept card payment
* By clicking the “Card payment” button, the map shall display the Dublin Bike stations that accept card payment

1. Filter map location:

* The user shall have the ability to filter the location shown on the map
* When the user enters a street address, area or Eircode, the map shall display an area within a 500-meter radius of that input
* The user shall have the opportunity to zoom in further or zoom out on the map

1. Predict the number of available bikes on a specific date and time:

* By entering a specific date and time, the user shall gain information on available bikes or available stands of stations
* The user shall not be able to enter a date that exceeds five days after the current date
* Once a specific date and time has been entered, the map shall show a prediction of available bikes or stands (depending on which option the user has chosen) for that date and time
* The user shall have the ability to gain information of each station by clicking on a pinpoint icon

**Value-add Features**

1. To allow provide a route planning:

* The user will be allowed to plan a route on the website
* The plan will predict the station availability at the start and end of the journey.
* It will also recommend an alternative station to collect / drop off in the event of unfavourable station information

1. Entertainment box:

* The website will contain a box containing activities in the area
* This information will be collected from Dublin’s most trusted entertainment websites

## Functional Requirements

Based on the user requirements, the following actions are necessary to provide the required functionality for the web application. Each of the following requirements correspond to the above essential user requirements.

1. **Displaying all the Dublin Bikes Stations that are available across Dublin City**
   1. Display the Dublin map, using the Google Maps API
   2. Run a script, located on an EC2, that will call the JCDecaux API
   3. The results of the call will be stored in a table [Appendix 1]
   4. Place a location pin on each station - as represented on the map
   5. For each location, on click, a query will be sent to the RDS on our EC2 instance. It will return:
      1. *station number*
      2. *station address*
      3. *available bikes*
      4. *available stands*
      5. *current weather*
   6. This information will be displayed in a box at the side of the map
   7. If the user clicks on another station, another query will be sent, and the information in the box will be updated accordingly

## 

1. **Display stations for collecting a Dublin Bike**
   1. Above the map, there will be a drop-down box that contains the “I want to…”
   2. Onclick, the drop-down menu will provide the user with the option “Collect”
   3. Onclick, a query will be sent to the RDS and requests all stations that have bikes available
   4. The requests will update the map.
      1. Each station that contains bikes will be highlighted Green
      2. Each station that does not have available bikes will be highlighted Red
   5. For each location, on click, a query will be sent to the RDS on our EC2 instance. It will return:
      1. *station number*
      2. *station address*
      3. *available bikes*
      4. *available stands*
      5. *current weather*
   6. This information will be displayed in a box at the side of the map
2. **Display stations for dropping off a Dublin Bike**
   1. Above the map, there will be a drop-down box that contains the “I want to…”
   2. Onclick, the drop-down menu will provide the user with the option “Drop off”
   3. Onclick, a query will be sent to the RDS and requests all stations that have bikes available
   4. The requests will update the map.
      1. Each station that contains free stations will be highlighted Green
      2. Each station that does has all stations occupied will be highlighted Red
   5. For each location, on click, a query will be sent to the RDS on our EC2 instance. It will return
      1. *station number*
      2. *station address*
      3. *available bikes*
      4. *available stands*
      5. *current weather*
   6. This information will be displayed in a box at the side of the map

## 

1. **Display stations that accept card payment**
   1. Above the map, there will be a box containing ‘Card Payment’
   2. Onclick, a query will be sent to the RDS and request all stations that accept ‘Bank’.
   3. The requests will update the map.
      1. Each station that contains bank payment will be highlighted Green
      2. Each station that does not support bank payment will be highlighted Red
   4. For each location, on click, a query will be sent to the RDS on our EC2 instance. It will return:
      1. *station number*
      2. *station address*
      3. *available bikes*
      4. *available stands*
      5. *current weather*
   5. This information will be displayed in a box at the side of the map

## 

1. **Filter map location**
   1. Above the map, there will be a box containing ‘Enter Address, area code or Eircode’
   2. The box will be connected to the Eircode API
   3. When the user begins to type, the box will prompt the user for ‘*suggested addresses*’
   4. Onclick, the longitude and latitude will be sent to a script on our EC2 instance
   5. The script will then request all information longitude and latitude information from the RDS request will be sent to the longitude and latitude of the stations
   6. All stations are within this radius (500m), will be return to the web application.
   7. The requests will update the map.
      1. Each station that is within the radius will be highlighted Green
      2. Each station that is not within the radius will be highlighted Red
   8. For each location, on click, a query will be sent to the RDS on our EC2 instance. It will return:
      1. *station number*
      2. *station address*
      3. *available bikes*
      4. *available stands*
      5. *current weather*
   9. This information will be displayed in a box at the side of the map

## 

1. **Predict the number of available bikes on a specific date and time**
   1. Above the map, there will be a series of boxes containing ‘*Enter Time’ & ‘Enter Date’*
   2. When the user enters the time and date, the information will be sent to a script running on our EC2 instance - Note: the user can only look at 5 days in advance (See design constraint for information)
   3. The script will carry out regression on our existing dataset and predict the station information for the inputted time and date. Factors included in the model:
      1. Time - Historic: RDS
      2. Day - Historic: RDS
      3. Weather predicted - Future: OpenWeather API
   4. For each location, on click, a query will be sent to the RDS on our EC2 instance. It will return:
      1. *station number*
      2. *station address*
      3. *available bikes*
      4. *available stands*
      5. *Expected weather*
   5. This information will be displayed in a box at the side of the map

**Value-Adding Features:**

1. **To allow provide a route planning**
   1. There will be a ‘*Plan my route’* to the side of the map
   2. Onclick, the field will expand and display Departure station‘ and ‘*Destination station’*
   3. The user can drag two stations into their respective boxes
   4. The user will then input his expected departure and arrival time and date
   5. A request will be sent to the EC2 and return
      1. the availability of the bikes at the departure station
      2. the available stands at the destination station
   6. The route will then be displayed on the map. Information will be displayed to the right of the map. It will contain:
      1. *station numbers*
      2. *station addresses*
      3. *expected available bikes at the departure station*
      4. *expected available stands at the destination station*
      5. *Expected weather*

1. **Entertainment box**
   1. We will display information on the top entertainment attractions across Dublin.
   2. This will be done through web scraping social platforms and other known websites

## Non-Functional Requirements

1. **Maintainability**
   1. The system should be ‘built to last’. The robust design should mean the errors and bugs will make up a large portion of the time spent on maintaining
2. **Ease of Use**
   1. The system should be user friendly. The installation process of the plug-in should be quick and largely autonomous.
3. **Reliability**
   1. The product will be used 7 days a week and can be accessed 24 hours a day
   2. In the event of an update to the website, the website will display a message informing the users of the work being carried out
   3. The system should be able to process up to 500 requests simultaneously – we anticipate that this load will occur regularly
4. **Portability**
   1. The website should be portable across all browsers
5. **Security**
   1. The website should encrypt all data transfers between the browsers and the EC2 instance
   2. It should also encrypt request and data transfers across the various API’s.
   3. All other transfers and data handling should also be encrypted

## Performance Requirements

Performance Requirements are important in the development of any system as they provide a clear benchmark that the is expected from the performance of the product. This section will focus on how the system will perform under (1) *Response Time* (2) *Workload* (3) *Security* (4) *Accuracy.*

1. **Response Time:** We have set a benchmark to have the webpage responsive. This has been set to a load time of less than 1 sec. We think that this is very important given the type of interaction the user will have with the site.
2. **Workload:** The website will be lightweight. The heavy lifting will be done on the server-side. This will include model calculation, database configuration etc. The front-end is only being rendered with static information – this is in line with the python flask convention.
3. **Security:** The website will have standard HTTPS when trasnfering data. Best practices will be adhered to when requesting information from the database and returning it to the frontend web page.
4. **Accuracy:** We aim to provide current information as up-to-date as can be provided by from the API from the weather and dublinbikes. Our prediction model has been built so as to provide accuracy. However, we are cognitive of the fact that this will improve with time.

## Domain Requirements

1. **Security and Privacy**
   1. We assume that the browser is not compromised
   2. The database will be in-line with GDPR Regulation (EU) (2016/679)
2. **Storage**
   1. AWS storage will be adjusted as necessary
   2. This will be continuously monitored, and certain alerts will be built in the to prevent failure

## Design Requirements

The following are the design requirements that must be adhered to:

1. **Disability**
   1. The plug-in should adhere to best practiced standards to accommodate those with a disability (WCAG, DDA)
2. **Prediction:**
   1. Due to our API, predictions will only be allowed to be made 5 days in advance. This is because the predicition model uses weather and we can only access +5 days’ worth of weather.
   2. If the user attempts to try and see beyond 5 days, they will be prevented.
3. **Minimal**:
   1. Our users will often be in a rush to check the current state of the stands.
   2. For this reason, we have designed the website so the user can make a request in one click.
   3. We went for the ‘less than 4’ methodology - everything can be down in 4 clicks or less.

# **Change Management Process**

The SRS will be modified as the requirements change. These changes can come about from (1) customer feedback (2) feature implantation issues (3) budget (4) external events. These changes will be logged and updated in a new version – will implement the industry standard of version control. These changes will be made by the Product Owner.

We agree that the team must reach an agreement to change the requirements. It is taken that the current issue of the document is, with the information that we have available, the best ‘action plan’ for producing the product. In order to make the changes, the team will sign off on the changes (those who it affects). This will stand as an agreement.

# **Approval**

The above SRS has been approved by the Product Owner ‘*Stephen Keenan’.* Any queries should be directed at the owner. All information contained in this document is original copyright of CS Technologies. The product developed should be done so for the named client.

*Stephen Gaffney Kerrie Lowe Stephen Keenan*

*20th April 2019 20th April 2019 20th April 2019*

**Title Title Title**

# **Appendix**

**Appendix 1:** User Story

**A close up of text on a white background

Description automatically generated**

**Appendix 2:** Table Properties

|  |  |  |
| --- | --- | --- |
| **Table Name: Weather** | | |
| **Field Name** | **Caption** | **Data Type** |
| number | Number | INT(Primary Key) |
| location | Location | VarChar(30) |
| time | Time | VarChar (80) |
| day | Day | VarChar(20) |

|  |  |  |
| --- | --- | --- |
| **Table Name: StaticData** | | |
| **Field Name** | **Caption** | **Data Type** |
| number | Number | INT(Primary Key) |
| name | Name | VarChar(80) |
| address | Address | VarChar (80) |
| latitude | Latitude | Float |
| longitude | Longitude | Float |

|  |  |  |
| --- | --- | --- |
| **Table Name: DynamicData** | | |
| **Field Name** | **Caption** | **Data Type** |
| number | Number | INT(Primary Key) |
| day | Day | INT () |
| time | Time | INT() |
| name | Name | VarChar (80) |
| Status | Status | VarChar(20) |
| bike\_stand | Bike Stands | INT() |
| available\_bikes | Available Bikes | INT() |
| available\_stands | Available Stands | INT() |
| last\_update | Last Update | INT() |
| banking | Banking | Bool() |

**Appendix 4:** Burndown Chart

**A screenshot of a cell phone

Description automatically generated**