

COMP30830

Software Engineering

**Requirements:**

To develop a Web Application to Display occupancy and weather information for Dublin Bikes

***Client****:*

Dublin Bikes

**Table of Contents**

[**Introduction 3**](#_30j0zll)

[**Purpose 3**](#_1fob9te)

[**Scope 3**](#_3znysh7)

[**Definitions 3**](#_2et92p0)

[**Overview 3**](#_tyjcwt)

[**Requirements Analysis Summary 3**](#_3dy6vkm)

[**Data Collecting 3**](#_1t3h5sf)

[**Pitch / Brief Analysis** 3](#_1hmsyys)

[**System Observation** 3](#_4d34og8)

[**Brainstorming** 4](#_2s8eyo1)

[**Requirements Analysis 4**](#_17dp8vu)

[**Selecting Data 4**](#_3rdcrjn)

[**Analysing Data 4**](#_26in1rg)

[**Overall Description 5**](#_lnxbz9)

[**Product Overview 5**](#_35nkun2)

[**System Interfaces** 5](#_1ksv4uv)

[**Interfaces** 5](#_44sinio)

[**Hardware Interfaces** 5](#_2jxsxqh)

[**Software Interfaces** 5](#_z337ya)

[**Memory Constraints** 5](#_3j2qqm3)

[**Product Functions 5**](#_1y810tw)

[**User Characteristics 5**](#_4i7ojhp)

[**Constraints 5**](#_2xcytpi)

[**Assumptions & Dependencies 5**](#_1ci93xb)

[**Appointment of Requirements 5**](#_3whwml4)

[**Requirements Statement 5**](#_2bn6wsx)

[**User Requirements 5**](#_qsh70q)

[**Functional Requirements 7**](#_1pxezwc)

[**Non-Functional Requirements 9**](#_49x2ik5)

[**Performance Requirements 9**](#_2p2csry)

[**Domain Requirements 10**](#_147n2zr)

[**Design Requirements 11**](#_3o7alnk)

[**Change Management Process 11**](#_23ckvvd)

[**Approval 12**](#_ihv636)

[**Appendix 12**](#_32hioqz)

# **Introduction – Anyone**

## 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 – Stephen G**

## Product Overview

### **System Interfaces**

### **Interfaces**

### **Hardware Interfaces**

### **Software Interfaces**

### **Memory Constraints**

## 

## 

## 

## 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**

## 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.

## 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 color 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.

## 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 color 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 bike stands.

## 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.

## 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.

## 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:
   1. The user will be allowed to plan a route on the website.
   2. The plan will predict the station availability at the start and end of the journey.
   3. It will also recommend a alternative station to collect / drop off in the event of unfavourable station information
2. Entertainment box:
   1. The website will contain an box containing activities in the area.
   2. This information will be collected from Dublin’s most trusted entertainment websites.
3. Another

## 

## 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.

# **Requirements Statement**

## 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 JCDeaux 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

## 

## **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

## **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

## 

## **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

## 

## **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 in the box there address, 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

## 

## **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. Another
  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‘ & ‘*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.
   2. Once installed, the user will not need further training. It should have a native feel to the Moodle platform
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 plug-in, the user will be prompted to install the update of the plug-in. Alternatively, the user can choose auto-updates.
   3. The previous version of the plug-in should be functional for a period of 6 months
   4. In the event of a major issue arising, the user will be prompted to update immediately
   5. The system should be able to process up to 200 requests simultaneously – we anticipate that this load will only be experienced around submission deadlines
4. **Portability**
   1. The plug-in should be portable across all browsers
   2. The plug-in should not be dependent on any operating system
5. **Security**
   1. The plug-in should encrypt all data transfers between the browsers and the EC2 instance
   2. It should also encrypt request and data transfers across the Moodle API
   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.*

## Domain Requirements

1. **Security and Privacy**
   1. We assume that the browser is not compromised
   2. The Moodle login feature is functional
   3. The database will be in-line with GDPR Regulation (EU) (2016/679) and will hold only necessary information regarding members, staff and supplier contact information
   4. We assume students will be aware of the storing of information
   5. We assume professors will be aware of the storing of information
   6. We will duplicate all information in case of technical failures
2. **Storage**
   1. AWS will be flexible. Storage will be added as required.

## Design Requirements

* 1. Except for those cases where the user has been notified of data gathering, the plug-in should not gather data on the user
  2. All information that is gathered should be stored securely and deleted when deemed obsolete, or to be in compliance with GDPR standards.

# **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 – RD Tables**

|  |  |  |
| --- | --- | --- |
| **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() |

|  |  |  |
| --- | --- | --- |
| **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) |