SYSTEM REQUIREMENT SPECIFICATIONS

SIMPLIFIED SRS, VERSION 8

BikePlanner Application

PREPARED FOR

Tejas Bhat, Product Owner & Teaching Assistant

Lecturer/Assistant Professor: Dr. Aonghus Lawlor

Dublin Bikes Project

Software Engineering COMP 30830

University College Dublin

PREPARED BY

Team 9

Contributors: Scarlet Grant, Jiarui Qin, Xiang Zhao

Computer Science (Conversion)

Class of 2019

University College Dublin

APR 18, 2019

Tejas Bhat
University College Dublin

BELFIELD
DUBLIN, D4

Dear Tejas,

Re: Enclosed System Requirement Specifications, Simplified Version 8

Please find enclosed the updated System Requirement Specifications for your software engineering project DublinBikes.

Yours Truly,

Team9

Scarlet Grant
Scrum Master, Developer

Jiarui Qin
Developer

Xiang Zhao
Senior Developer

TABLE OF CONTENTS

CHANGE LOG

Versio	YY/MM/DD	Editor	Changes
1		Scarlet Grant	First concept Functional Requirements 1. Introduction 2 1.1 Purpose 3 1.2 Product Scope 3 2. Overall description 4 2.1 Product Perspective 4 2.2 Product Functions 5 2.3 User characteristics 6 2.4 Operating environment 6 2.5 Contribution to Bike-user: 6 2.6 Contribution to other stakeholders: 7 2.7 User stories 7 3. System features 8 3.1 Assumptions 8 3.2 Input 8 3.3 Output 8 3.4 Overview functionalities (needs further specification) 8 4. Wireframes 8 4.1 Mobile 8 4.2 Desktop
1	2019/03/28	Scarlet Grant	Added index and changelog
2	2019/03/28	Jiarui Qin	4.1 Mobile wireframe: Adjusted Mobile User Interface Mockup (Balsamiq): Search bar and map moved to top.
3	2019/04/04	Xiang Zhao	1.3 Definitions, Acronyms, and Abbreviations. 1.2 Product Scope 2.1 Product Perspective: backend description 2.1.1 System Interfaces 2.1.2 Interfaces 2.1.3 Hardware Interfaces 2.1.4 Software Interfaces 2.1.5 Communications Interfaces 2.1.6 Memory Constraints 2.1.7 Operations 2.1.8 Site Adaptation Requirements

			2.2 Product Functions (extra) 2.5 Constraints 3.2.1 The Backend Server
4	2019/04/11	Scarlet Grant	Change log: specify earlier changes, adjust numbering of versions 1-4 Adjust document styling, grammer. 2.5 Constraints: extra descriptions Moved 3.2.1 The Backend Server to 5. Architecture Moved extra functionalities described by XZ to System Features. Added unspecified functionality Machine learning: Applying Linear Regression model
5	2019/04/15	Jiarui Qin	1. Simple Linear Regression Model 2. ML server to process incoming request 3. Prediction Request Handler 4. Forecast Request Handler
6	2019/04/16	Jiarui Qin	Modified wireframe: Adjusted Desktop User Interface Mockup (Balsamiq): Deleted the ads and availability forecast.
7	2019/04/17	Jiarui Qin / Xiang Zhao	Added extra description Prediction Model
8	2019/04/18	Jiarui Qin	Modified wireframe: Adjusted Desktop User Interface Mockup (Balsamiq): Add some map points and a street map on the webpage and adjust the size of them. Adjusted Mobile User Interface Mockup(Balsamiq): Add some map points and a street map on the mobile page and adjust the size of them. Delete some Icons.

1. Introduction

1.1 Purpose

Software requirements (Version 4) regarding the creation of a standalone app to help Dublin Bike users in their journey. This document is created to confirm the requirements with the (technical) product owner of the Project.

1.2 Scope

The BikePlanner Application is a solution to remove all possible friction of biking with DublinBikes, the bike scheme run by Dublin City Council. It is build as example to upgrade the website www.dublinbikes.com. The Application could however be adjusted and delivered to any other bike sharing scheme provider.

The application helps users finding the nearest bike station to get and return a bike from the bike scheme. It will include a function to predict the availability. This will be delivered as beta feature in version 1 of the application.

1.3 Terminology

BikePlanner or BikePlanner Application: The application to be delivered

Core Server: The core server is the server runs on the cloud, which handles the incoming requests. It return the data to the client or forward the request to the ML server according the type of the request.

ML Server: The ML server accepts incoming requests. It uses the simple linear regression model to predict the number of available bikes in the next hour and return such data to the core server.

Web Server: The web server is responsible render the web pages and update the web pages. It send user's requests to the core server and display the updated data on the web pages.

AIOHTTP: The async framework which used in ML Server and Web Server.

API: Application Programming Interface - API's allow the Project Team to use the data and services from other sites that provide the interface to enable that. The Project Team created internal API's as well, to enable our web application to retrieve generated from our servers.

Flask: a micro web framework for Python, based on Werkzeug and Jinja 2

Jupyter Notebook: Open document format based on JSON. A notebook can contain code, narrative text, equations and rich output.

1.4 Features specified by Product Owner

The app aims to help Dublin Bike users with the following benefits:

- The application should help users find the nearest station to get and return a bike.
    - Find the best station with available bikes to start a journey
    - Find the best station with free spaces to return the bike

- The application should also give users some predictions based on the current conditions.
    - Should display relevant and useful information learned from recent occupancy trends
    - Incorporate weather information, as it affects the occupancy

- Include other sources of information
    - The application should however not provide any unrelated information to users.

- The application should be able to handle the user's search request and send back the needed information to users within a reasonable time.

2. Overall description

2.1 Product Perspective

The BikePlanner is an independent, standalone application that collects data from the API's of JCDecaux and DarkSky to show bike stations, availability and weather. A prediction module analyses the gathered data from the API's. The site will consist of one single main page.

There are many applications out there in the market that show the nearest bike stations. Yet, the quality of the user interface is limited and they do not provide prediction features.

2.1.1 System Interfaces

The application collects data from third-party providers.

- It gets the following data from the following providers:

  Current weather and weather forecasts from DarkSky;
  Bike station data from JCDecaux;
  Geocoding data from MapBox.

- All data from these service providers is returned in JSON format.

2.1.2 Interfaces

- Users interacts with the application by a Graphical User Interface (GUI). The GUI runs on the web browser, so it should be able to work correctly on different devices and web browsers.

2.1.3 Software Interfaces

- The backend of the application should run on the Linux Operating System. The version of the kernel should be later than 3.0.
- Each component of the application are packed into a single docker container. The docker version should be later than 18.0.
- The Node.JS should be the latest version, the Python should be the latest version.
- All JavaScript codes run on the web browser should be in the strict mode, and target to ES6. The HTML should be version 5 and the CSS should be version 3.

2.1.4 Hardware Interfaces

- The server should have at least 1 GB memory and 25 GB data storage.

2.1.5 Communications Interfaces

- Each component of the application communicate with each other through the HTTP 1.1 protocol.
- The body of each HTTP request and response should be in the format of JSON.

2.1.6 Memory Constraints

The application requires at least 1 GB memory to boots and run.

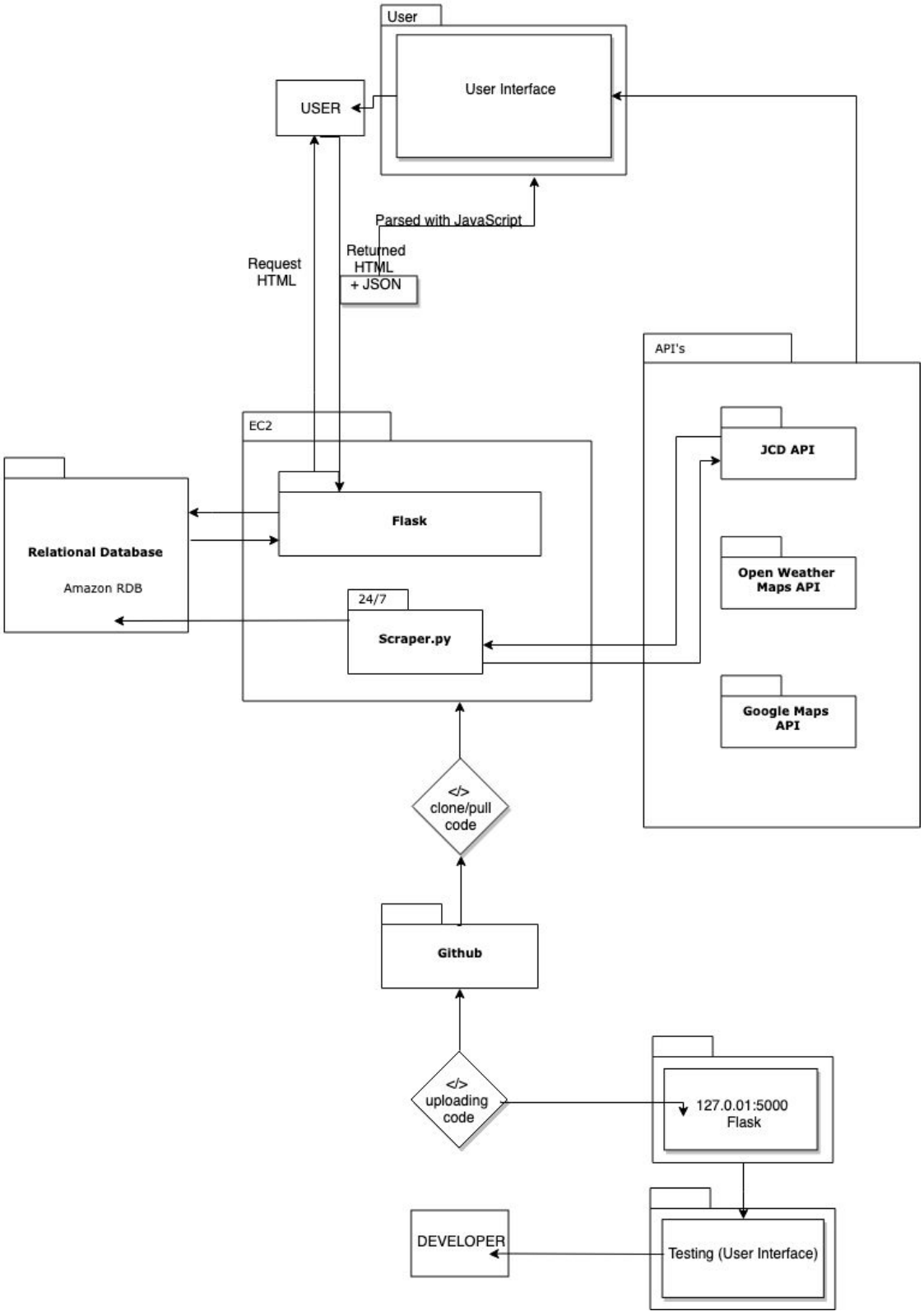
2.1.7 Operations

There are no special operations required by the user.

2.1.8 Site Adaptation Requirements

The database system and cache system should already be started prior to the data collection server.

Basic Flowchart of app



2.2 Product Functions

The application delivers the following functionalities to the user:

- Find nearest bike at desired/current location in least amount of clicks;
- Find nearest dock at desired/current destination in least amount of clicks;
- Show forecast of bike availability (at given datetime and location);
- Show availability for card payments at given location.

Approved adjustments of original product briefing:

- Show weather-forecast (to bike or not to bike); show weather directly upon opening the app, so that the user can decide whether to bike or not ot bike:

The project description of the client informs that the weather only needs to be shown after the after the station is clicked and shown. However, the team is of the opinion that the user wants to see the weather around him/her or in Dublin before it takes the effort to look up a specific Dublin station. The weather (and/or nearest station) should be visible on opening of the app. This adjustment has been proposed and approved by the Product Owner.

2.3 User Characteristics

The user is not expected to have any technical knowledge. The users might be once off users who try out Dublin Biking or they could be frequent users who will rely on the app. It is likely they will use the app mostly on mobile before starting their journey.

2.4 Operating environment

Taking the expected usage in consideration, the app will be firstly designed and build for mobile use, with a responsive scheme for larger screen usage. It also should be able to responsive under complex situation. It is supposed to work on any standard internet browser. Future applications could be developed for the Smart Watch and that would not be in the scope of this part of the project.

2.5 Constraints

1. The budget for the hardware devices / servers

The application needs to run as lean as possible. Various performance optimization will need to be done to keep cost below 10 Euro a month on Amazon Web Server (AWS)

2. The request limitation set by service providers.

The API providers such as JCDecaux allow only a maximum amount of requests of 5000 a day. To make the application scalable we need to create a solution to store data so that it can be reused for multiple requests.

2.5 Contribution to Bike-user

- Offer convenience by friction less biking;
    - Show weather-forecast (to bike or not to bike);
    - Find nearest bike and dock at destination in least amount of clicks;
    - Show predicted bike availability at a given time.

2.6 Contribution to other stakeholders

- Dublin City: stimulate use of their biking scheme by eliminating friction for the users
- JCDecaux:: another extra option for ‘green advertising’

We have chosen to deliver the first version free of Ads. We reserved a place and created an example so that the company who will eventually purchase the system, can choose start with incorporating advertising from day one if they want to.

2.7 User stories

As a user, I can:

- open the app and see immediately the weather forecast that has me decide to bike or not to bike;
- open the app and see immediately the availability at a particular (preferably the closest) bike station;
- view the predicted bike availability at a particular data/time in the future;
- open the app and see quickly where I can return my bike;
- Open the application and find the nearest bike station.

3. System features

3.1 Overview functionalities

Web Server

- The application should be able to display the map on the web browser.
- The application should be able to display current available bikes on the web browser.
- The application should be able to display the prediction of available bikes on the web browser.
- The application should be able to adjust its appearance on different devices (Responsive design)

Core Server

- The Core Server should be able to fetch weather data from service provider.
- The Core Server should be able to fetch geocoding data from service provider.
- The Core Server should be able to fetch bike station data from service provider.

Interaction between Data Collection Server and Web Application Server

- The Data Collection Server should be able to return data to Web Application Server.
- The Web Application Server should be able to send request to Data Collection Server and process the returned data from Data Collection Server .

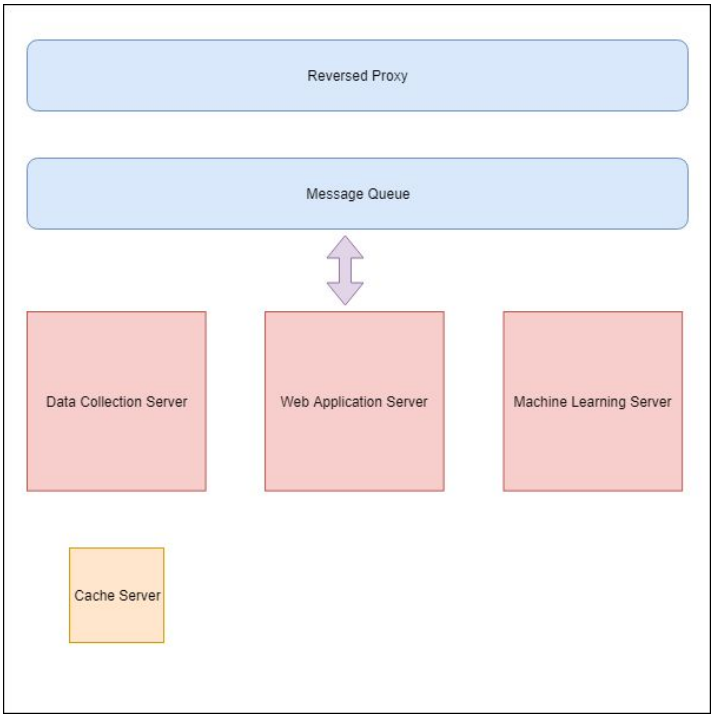
ML Server

- The Machine Learning Server should be able to read data from database.
- The Machine Learning Server should be able to process request and send required data to the client.
- *Beta:* A Linear Regression Model will be applied to predict bike availability per hour. The functionality will be in beta during Version 1 and will show an accuracy of anything higher than 1%. To be transparent to the users, the accuracy percentage will be displayed next to the predictions.

4. Architecture

4.1 The Backend Server

The system consists of several components. Here is the architecture of the different components in the application.



4.1.1 Core Server

The Core Server has several components. The first component is the request handler component. In this component, the server will either process the request or forward such request to the ML Server according the type of the request. For the weather request and geolocation request, the server will fetch data from remote server providers and return the data to the web server. For the prediction request, the server will forward the request to the ML Server and wait the data get back from ML Server, then send the data back to the web server.

The second component in the server is the data collection component. For this component, it fetches data from remote service providers and return the data to the web server.

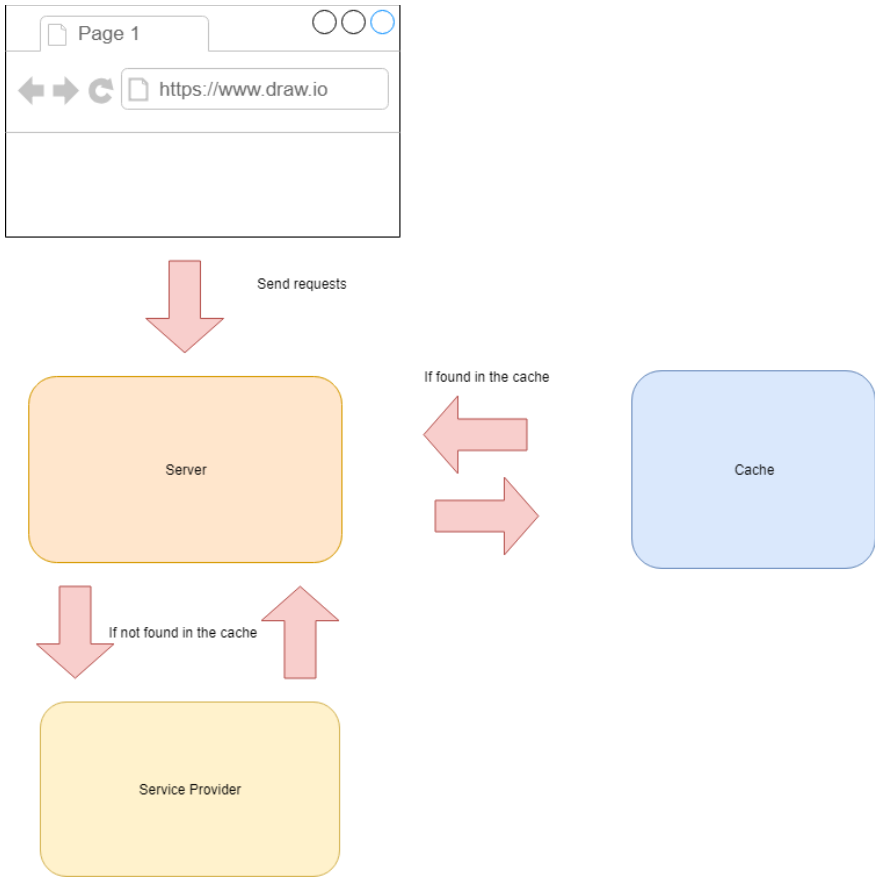
The third component in the server is the database component, This component always fetch data from the service providers and save the data into the database.

4.1.2 ML Server

The main task for the ML Server is to do some prediction on the data by using simple linear regression. When the server get a new request from the core server, it searches the database and train the data. After the training, it return the prediction to the core server.

4.1.3 Web Server

The Web Server is always listening the incoming request and forward such request to the core server.



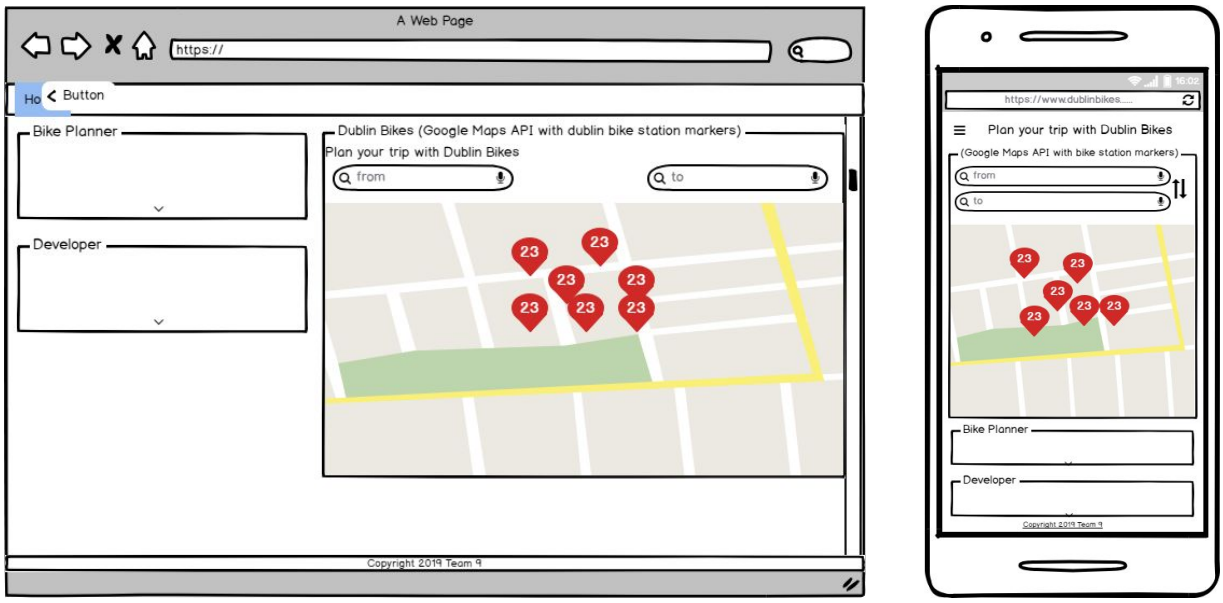
5. Prediction Model

We use simple linear regression model to predict the number of available bikes in next hour.

First, the user will provide the number of a station on the map. Then the ML Server will search all documents from the database whose number is equal to that station number. For the dataset we get from database, we use 1 / 4 as test data and 3 / 4 as training data. We train the data and return the prediction to the Core Server.

For each bike station, the ML server fetch the historical data for that station and make the prediction based on the historical data. For some station which has few available bikes, the prediction is less accurate. But for many bike station where available bikes are more than 10, the prediction is very accurate.

4. Wireframes



The red points indicate the number of current available bikes. When you click on the red pointer, you will see more information about the bike and weather, such as in the examples for UX. The team is aware that The Google maps API has some limitations and so a simplified card would be implemented.