



UNIVERSITY OF  
**LEICESTER**

**Department of Informatics**  
**University of Leicester**  
**CO7201 Individual Project**

**Final Report**

**Web Based Bike Sharing Application**

**Mahesh Srivinay Rayavarapu**

[msr31@student.le.ac.uk](mailto:msr31@student.le.ac.uk)

**199047813**

**Project Supervisor: Ms. Monasadat Zarbaf**

**Second Marker: Ms. Nicole Yap**

**Word Count: 10071**

**Date: 20-May-2022**

### **ABSTRACT**

It is evident that the Covid-19 pandemic has wreaked havoc on people's lives. As everything is going back to normal, people who are living in cities are changing their ways to commute to work and other places which are of shorter proximity. The preferences are now shifting more towards cycling and public transport. On the other hand, road congestion is evolving into a modern-day menace. Traffic with higher levels of air pollution is causing more environmental damage. Although there are several number of measures available to tackle this issue, bike sharing offers more advantages to the problem. Advantages include zero carbon emissions with zero road congestions. Bike sharing is flexible and cycling also have several health benefits. Biker's den is a bike sharing web platform where the customers will get a hassle-free experience in picking and dropping the bikes at their nearest convenience. A new student living in Leicester or a tourist visiting the city can buy a day pass at minimal price. A QR code is provided with the day pass for easy pick up and drop off operations at dock stations.

### **DECLARATION**

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date, and page(s). I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Mahesh Srivinay Rayavarapu

Date: 20-May-2022

## Contents

1	INTRODUCTION .....	4
1.1	AIM AND MOTIVATION .....	4
1.2	BACKGROUND RESEARCH .....	4
1.3	LITERATURE SURVEY .....	4
1.4	CHALLENGES.....	5
1.5	RISKS.....	6
2	REQUIREMENTS.....	6
2.1	HIGH LEVEL REQUIREMENTS.....	6
2.2	DETAILED REQUIREMENTS.....	7
3	TECHNICAL SPECIFICATIONS.....	8
4	METHODOLOGY.....	9
4.1	APPROACH.....	10
4.2	TIME PLAN AND MILESTONES .....	10
5	DESIGN.....	13
5.1	FLOW CHART DIAGRAM .....	13
5.2	USE CASE DIAGRAM .....	15
5.3	ENTITY RELATIONSHIP DIAGRAM (ER DIAGRAM) .....	16
6	PROJECT OUTCOME.....	20
6.1	URL ENDPOINTS FOR FUNCTIONALITIES AND WEBPAGES – CUSTOMER MODULE .....	20
6.2	URL ENDPOINTS FOR FUNCTIONALITIES AND WEBPAGES – MANAGER MODULE.....	22
6.3	BUILDING AND RUNNING THE SERVER .....	24
6.4	WEB PAGES AND CODE SNIPPETS.....	24
7	CONCLUSION .....	47
7.1	TESTING.....	47
7.2	FUTURE IMPROVEMENTS.....	51
8	REFERENCES.....	52

## Figures

Figure 1 - Waterfall Model.....	10
Figure 2- Flow Chart Diagram .....	14
Figure 3- Use Case Diagram .....	15
Figure 4- ER DIAGRAM .....	18
Figure 5 - Biker's Den Homepage.....	25
Figure 6- Customer Registration page logic.....	26
Figure 7- Postman Test for Register endpoint.....	26
Figure 8- Postman Test Result Preview for Register endpoint.....	27
Figure 9- Adding Dock Station from manager site.....	28
Figure 10- Google Map marked with coordinated entered by manager .....	29
Figure 11- Tom Tom map with all the Dock Stations in database .....	30
Figure 12- Sending dock station coordinates as JSON to Tom Tom maps API .....	30
Figure 13- Tom Tom Maps JavaScript code snippet .....	31
Figure 14- Postcode Search Backend Logic.....	32
Figure 15- Command to fetch data from API request .....	33
Figure 16- Postcode search output.....	34
Figure 17- Bike Availability View .....	35
Figure 18- Stripe Payment Flow.....	36
Figure 19- Stripe Checkout Session Logic .....	37
Figure 20- Stripe Checkout Session Output Snippet.....	38
Figure 21- Stripe CLI Webhook Response.....	39
Figure 22 - Webhook Logic for fetching event data and triggering actions .....	40
Figure 23- Logic for issuing day pass to the customer.....	41
Figure 24- Active Day Pass with QR code .....	44
Figure 25- Cancelling Next Day Recurring Payments Logic .....	44
Figure 26- Pick Bike and Drop Bike Logic .....	46
Figure 27- Functionality Test in Postman .....	48
Figure 28- Test Result for Postcode Search Functionality .....	48

## Tables

Table 1 - Technical Specifications .....	8
Table 2 - Time Plan.....	11
Table 3- Customer URL Endpoints .....	20
Table 4- Manager URL Endpoints .....	22
Table 5- Postman Testing - Result .....	49

# 1 INTRODUCTION

## 1.1 AIM AND MOTIVATION

The aim of this project is to develop a web application which facilitates mobility to the customers by allowing them to hire bicycles, which are carbon neutral and eco-friendly, for their everyday use. After completing register/login process, the customer will then be provided with the postcode search facility. This search shows the availability of bikes and allow customer to make booking using a payment method. The motivation is to create a bike hiring system, where the customers can hire a bike in website and can pick them at their nearest docking station, ride them and return the bike to the nearest available dock. Admin site/manager panel provides access to the superusers, who are responsible for monitoring and updating the dock stations and bikes availability.

## 1.2 BACKGROUND RESEARCH

People now a days are shifting from cars to bikes to avoid congestion. According to gov.uk, study reveals that road transport alone emits 91% of pollution in the whole transport sector in UK [3]. In cities, people are preferring bicycles for the shorter commutes, this is creating a huge demand in the market for bike sharing providers. There are numerous bike hiring providers in the market who are offering their services to the users based on Pay-as-you-go and subscription models. Biker's Den platform allows the users to hire a bike for travelling from one location to another, in a greener and healthier way. However, bike sharing is not a new concept, it started back in 1960's, when there was no advanced technology. With the increasing technological advancements, new ideas and improvements regarding the bike sharing schemes are getting better day by day.

## 1.3 LITERATURE SURVEY

### **BIKE SHARING AS TRANSPORT**

Bike sharing is one of the latest revolutionary trends in the public transportation service. Bicycles emit zero pollution and it also reduces the traffic obstruction. After doing research on the present available models in the market, there are certain things which can be changed or updated. The main agenda is to provide a hassle-free solution to the customers with a simple and easily understandable web-app for bikes booking. [1]

## **DOCK STATION BASED BIKE SHARING**

The dock station bike hiring and dock less bike hiring are the two common schemes available in the market. In Dock station-based scheme, the customer will hire a bicycle based on the availability of bikes in dock station. Dock Stations are often setup near popular landmarks like Tesco, Asda etc., After using the bike, the customer is supposed to drop the bike in the same or any other Dock Station. While picking a bike, the users should input some sort of information at the dock station to unlock the bike. Few of the popular methods are to unlock by OTP and to unlock by bar code. [2]

## **BIKE SHARING SYSTEM ADVANTAGES**

Bike sharing system offers several advantages. Riding bikes won't release any toxic pollutants to the environment. Bike sharing is considered as one of the healthy methods for travelling. Bike sharing system in metro cities gained more popularity from tourists and working people. Bike share is so economical. In some cities, the average cost of bike share per day is around £5.00 pounds. Riding bikes will also increase the physical activity of the customers [3].

## **1.4 CHALLENGES**

The challenges faced while working on this project are mentioned below:

- Code Indentation: Indentation in python programming is responsible for defining the individual blocks in a statement. Improper indentation results in serious errors.
- EMAIL Verification Using OTP: Verifying the email address of the customer using one time password during the registration process.
- Working with Geo Spatial Libraries: Creating a postcode search functionality which returns the nearest dock station involves using the geo spatial libraries. At first the author tried to use GeoDjango to solve this challenge, but it is taking more time than expected to get the work done. So, by keeping the time frame in mind, the author solved the challenge by using Python's Geopy, which is less accurate compared to GeoDjango.
- Displaying multiple marker map: It took a while to get the logic right for displaying a map with multiple markers. The map is needed to be responsive and zoomed out automatically to fit all the markers.
- Confirming payments using stripe webhooks: Payments are needed to be verified programmatically, before issuing the day pass to customers. The author used incoming webhooks to get the real time updates about the transactions.
- Creating a responsive user interface which supports all the screen sizes.

## 1.5 RISKS

- Risk-1: Complexity in working with the Model-View-Template Architecture: Since the author have a basic understanding on web frameworks, throughout the project, it is necessary for the author to learn and research more about the Django framework to develop the webapp user friendly with all the functionalities.
- Risk-2: Since there is a limited time available to complete the project, the frontend development with a good-looking user interface might take more than the expected time. So, it is necessary for the author to prioritise the tasks starting from creating the working functionalities.
- Risk-3: Real time session management and tracking cannot be achieved in this project as it requires the physical hardware equipment.
- Risk-4: Compiling Errors: Python by default uses shell script which helps the users to easily identify the errors in the code.
- Risk-5: Bicycles theft and Damage: Terms and conditions will be applied while registering for the website and booking bicycle via payment method. The company will charge the users, in the case of thefts and vandalism.
- Risk-6: Payment issues and security: Webhooks can be used to send info and messages from primer to the server so that the system will get updated only when the correct event took place.

## 2 REQUIREMENTS

### 2.1 HIGH LEVEL REQUIREMENTS

**Customer:** Once the registered customer is logged in to the web-app, the customer should be able to plan his journey starting from the post code search to find the nearby docking station. After that the customer can select the preferred docking station to get the information which includes the availability of pickup-bikes count. The customer can also pay for the booking via payment method as well as check their booking history to view the past bookings. After a successful payment, the customer should be able to access the My Pass page, which shows the pass information along with QR code and option to cancel the next day recurring payments.

**Administrator/Manager:** Site Administrator or the manager is connected to the superuser network and operates the Dock stations management. Once the registered super user is logged in to the administration panel, he should be able to add or remove the docking stations and he should also be able to add or remove the bikes count depending on the market and bike's condition. Manager should also have an option to view the user's bookings.

## 2.2 DETAILED REQUIREMENTS

### 2.2.1 ESSENTIAL FEATURES

#### **Customer:**

- Registration, Login and Logout functionalities via homepage
- Postcode Search facility to find the nearest available dock stations
- Bike's availability view for the selected dock station
- Payment gateway to manage the customer booking
- Contact-us page to get in touch with the team in case of any queries

#### **Manager/Administrator:**

- Login and Logout functionalities via homepage
- Functionality to add, remove and update the docking stations
- Functionality to change the bikes availability
- View and manage customers

### 2.2.2 RECOMMENDED FEATURES

#### **Customer:**

- Email Verification while registering for the webpage
- Booking history to view the customer's earlier bookings
- Change/Forgot password option
- Map view of all the dock stations with multiple markers which are added to the database by manager.
- Issuing a day pass, which is recurring and is valid for 24-hours.

#### **Manager/Administrator:**

- Map view to the manager to mark exact coordinates of the dock station using latitude and longitude
- Functionality to view the customer's previous bookings.
- Functionality to view the stripe customers (the details of all customers, whose transactions are verified by the webhooks)

### 2.2.3 OPTIONAL FEATURES

#### **Customer:**

- QR code generation after successful booking
- Option for the customers to cancel the next day recurring payments
- Real time session management
- Inclusion of highway code- rules for cyclists in UK



### 3 TECHNICAL SPECIFICATIONS

Table 1 provides an overview of the project's technical specifications which are needed to complete the tasks within the required time frame.

*Table 1 - Technical Specifications*

Type	Name	Summary
Programming Language	Python3 (Backend)	Python is a high-level general-purpose programming language; it facilitates modular programming by supporting number of modules and packages. Python's vital role in web development includes sending and receiving data from the servers, communicating with the database etc.
	JavaScript (Frontend)	JavaScript is used to make the websites interactive.
Markup Language	HTML5 (Frontend)	HTML is responsible for the structure and format of the content in the webpage. After receiving the HTML documents from the web server/local storage, web browsers render these documents to the multimedia webpages.
Design Language	CSS (Frontend)	CSS is used to enhance the appearance in a webpage. CSS is used in setting colours, backgrounds, spacings etc. for better presentation and for an aesthetic user experience.
Frameworks	Python- Django	Django is an open-source and server-side web framework build in python. It is based on MVT(Model-View-Template) architecture. It allows users to create better web apps in lesser time.

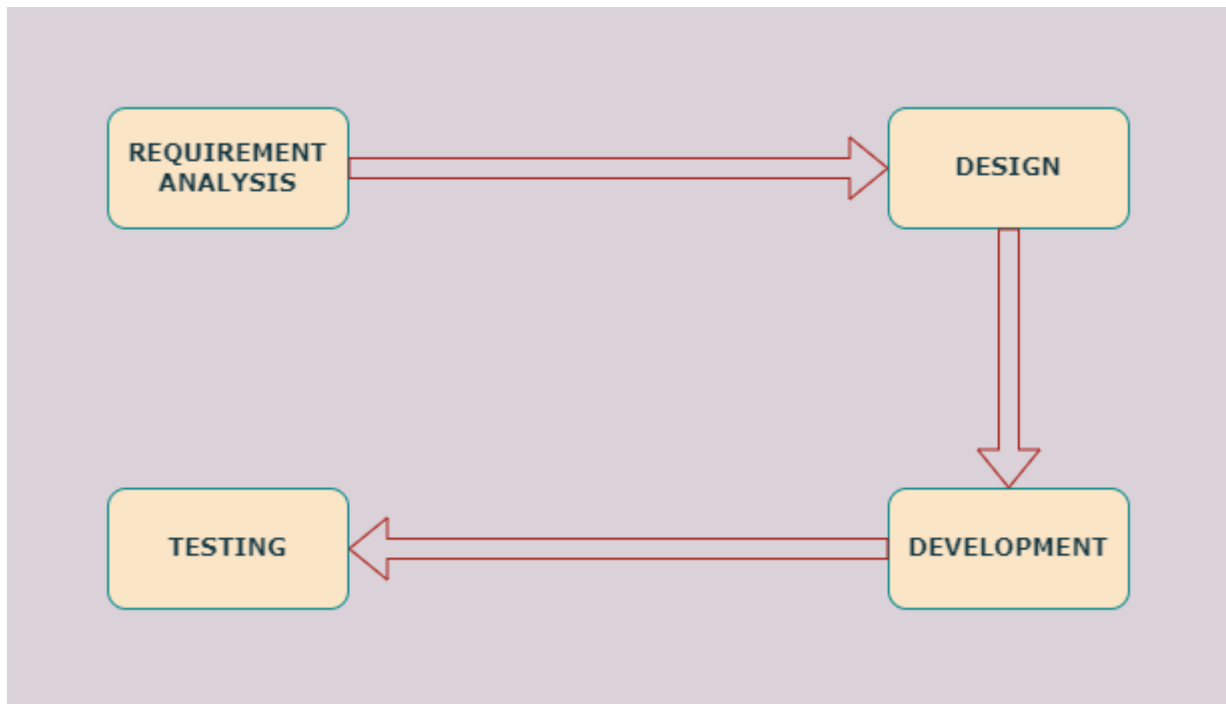
	Bootstrap	Bootstrap is a frontend CSS framework with JavaScript and CSS design templates.
	Vue Js	Vue js is a front-end java script framework used to develop single page applications and Vue Js is also used in fetching data from the APIs
Database Management	SQLite3	SQLite is an embedded database which uses the SQL query language.
	PostgreSQL	PostgreSQL is a RDBMS (Relational Database management system) which supports both relational (SQL) and nonrelational (JSON) queries. It offers support to geospatial extensions
Source Code Editor	Microsoft visual studio code	Visual studio code is a source-code editor which supports several development operations with so many tools and extensions.
Version Control System	Tortoise SVN	Tortoise SVN is a version control system that helps in work coordination and tracking files.

## 4 METHODOLOGY

The author followed the popular and simple waterfall methodology. Waterfall methodology follows 'one successful step at a time' formula where the development is carried out in the form of a stream. Waterfall model is suitable for completing the projects when the requirements are very well understood. The author also took inspiration from agile methodology to divide the requirement tasks in small pieces.

The phases involved in methodology workflow are:

- *Requirement Analysis*
- *Design*
- *Development*
- *Testing*



*Figure 1 - Waterfall Model*

The steps/phases involved in water fall model is shown in Figure-1. Initially, the requirement is analysed and a proper design is made to identify the logical flow. The development process will start only after the requirement is very well understood. The developed module will undergo testing right after the development to identify any potential bugs or fixes.

#### 4.1 APPROACH

Author started the project with initial study on the basics of Python, HTML and JavaScript. Learning phases also include gaining skills on frameworks – Django and Bootstrap. Further research was done on choosing the proper technologies and tools to get the better outcome of requirements. The requirements are then divided into smaller tasks with a time plan to follow.

#### 4.2 TIME PLAN AND MILESTONES

A time plan is created by analysing all the requirements. The requirements are split into several logical tasks and then time frames are created for each task. The time plan details with tasks and their deadlines are explained in the Table- 2.

**Table 2 - Time Plan**

<b>WEEK</b>	<b>TASKS</b>	<b>STATUS</b>
<b>Week 0</b> (February 14 <sup>th</sup> to 20 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Preliminary Meeting and project selection.</li> <li>• Project Description Submission.</li> <li>• Requirements Gathering.</li> </ul>	COMPLETED
<b>Week 1</b> (February 21 <sup>st</sup> to 27 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Project Analysis.</li> <li>• Creating Development Workflow.</li> </ul>	COMPLETED
<b>Week 2</b> (February 28 <sup>th</sup> to March 6 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Revised Python and Html basics.</li> <li>• Learned about the workflow of Django Framework.</li> <li>• Preliminary Report Submission.</li> <li>• Supervision Meeting-1</li> </ul>	COMPLETED
<b>Week 3</b> (March 7 <sup>th</sup> to 13 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Installed VS Code IDE and Python.</li> <li>• Created a virtual environment and installed all required dependencies.</li> <li>• Started developing backed for the User Registration page.</li> <li>• Group Meeting-1</li> </ul>	COMPLETED
<b>Week 4</b> (March 14 <sup>th</sup> to 20 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Created both backend and frontend for user registration and login pages with logout functionality.</li> <li>• Created forgot or change password option for users.</li> <li>• Supervision Meeting-2</li> </ul>	COMPLETED

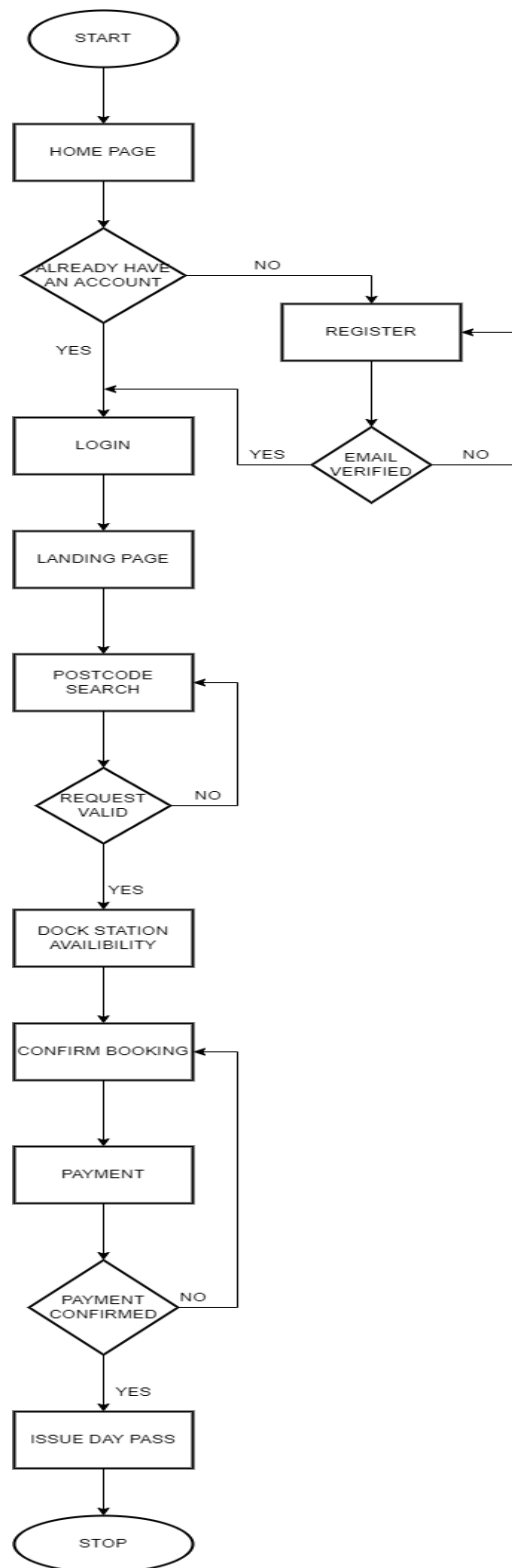
<b>Week 5</b> (March 21 <sup>st</sup> to 27 <sup>th</sup> )	<ul style="list-style-type: none"> <li>Integrating OTP based email verification for user registration page.</li> <li>Creating 'Contact Us' page.</li> <li>Creating database table for dock stations in Django models.</li> </ul>	COMPLETED
<b>Week 6</b> (March 28 <sup>th</sup> to April 3 <sup>rd</sup> )	<ul style="list-style-type: none"> <li>Integrating Google Maps API to Dock Stations model.</li> <li>Integrating Gmail SMTP setting to send emails to the user.</li> <li>Supervision Meeting-3</li> <li>Interim Report Submission.</li> </ul>	COMPLETED
<b>Week 7</b> (April 4 <sup>th</sup> to 10 <sup>th</sup> )	<ul style="list-style-type: none"> <li>Creating front end to display all dock stations in a single map dynamically.</li> <li>Second Marker Interview.</li> </ul>	COMPLETED
<b>Week 8</b> (April 11 <sup>th</sup> to 17 <sup>th</sup> )	<ul style="list-style-type: none"> <li>Creating postcode search functionality.</li> <li>Creating backend to add bikes count.</li> <li>Supervision Meeting-4</li> </ul>	COMPLETED
<b>Week 9</b> (April 18 <sup>th</sup> to 24 <sup>th</sup> )	<ul style="list-style-type: none"> <li>Displaying bikes count in frontend.</li> <li>Adding more frontend details to the search functionality.</li> <li>Preparing final report template.</li> </ul>	COMPLETED
<b>Week 10</b> (April 25 <sup>th</sup> to May 1 <sup>st</sup> )	<ul style="list-style-type: none"> <li>Creating functionality for providing booking option which based on day and weekly pass.</li> <li>Supervision meeting- 5</li> <li>Final report template submission.</li> </ul>	COMPLETED

<b>Week 11</b> (May 2 <sup>nd</sup> to 8 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Working on payment gateway integration and GUI touch-ups.</li> <li>• Creating Booking history functionality to view the user's previous bookings</li> <li>• Group Meeting-2</li> </ul>	COMPLETED
<b>Week 12</b> (May 9 <sup>th</sup> to 15 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Working on frontend GUI and optional requirements.</li> <li>• Supervision meeting- 6</li> </ul>	COMPLETED
<b>Week 13</b> (May 16 <sup>th</sup> to 22 <sup>nd</sup> )	<ul style="list-style-type: none"> <li>• Final GUI work.</li> <li>• Dissertation and code submission.</li> <li>• Viva Preparations.</li> </ul>	CURRENT WEEK
<b>Week 14</b> (May 23 <sup>rd</sup> to May 29 <sup>th</sup> )	<ul style="list-style-type: none"> <li>• Final Viva</li> </ul>	PENDING

## 5 DESIGN

### 5.1 FLOW CHART DIAGRAM

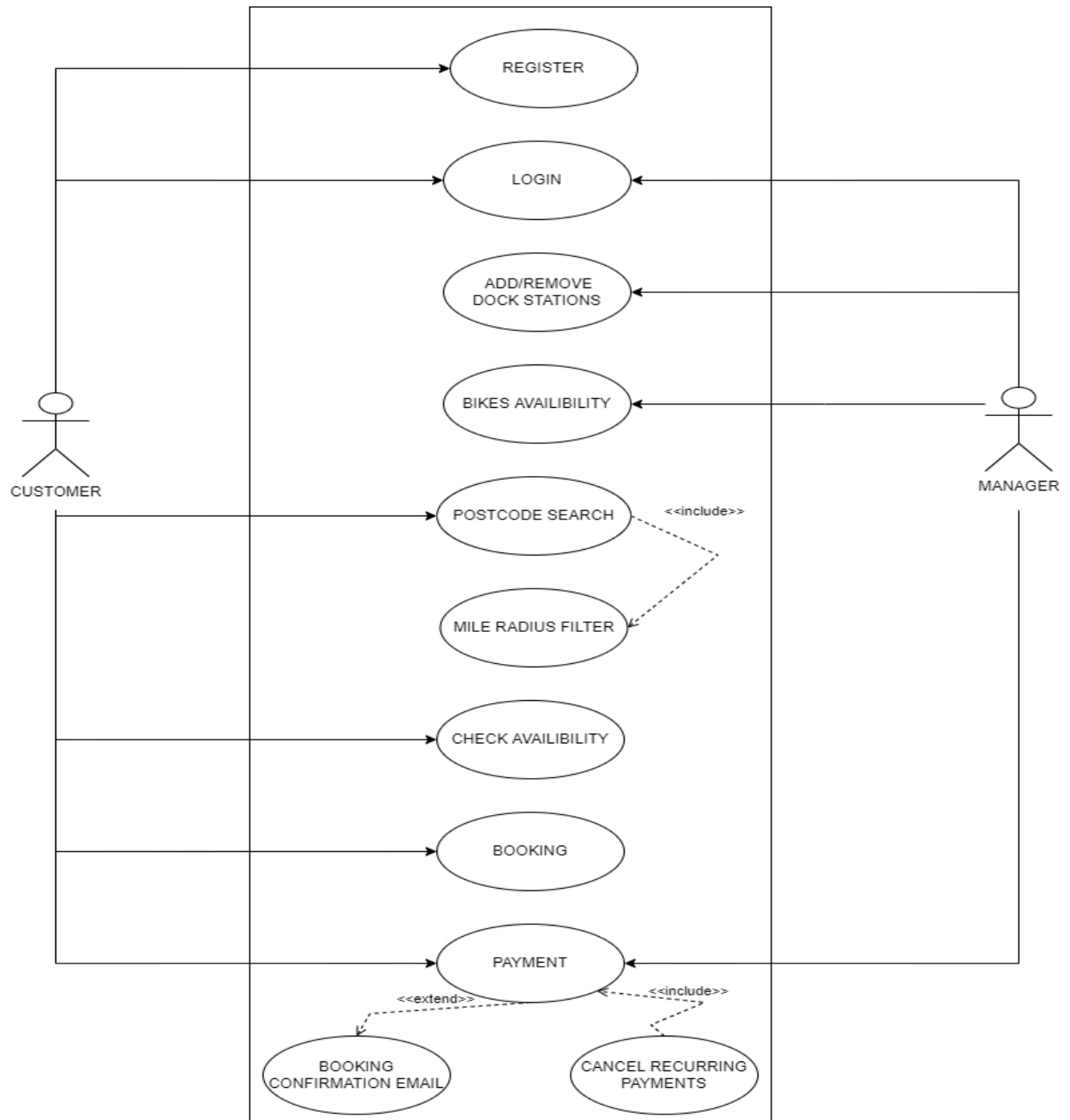
Flow charts are often used to describe the process flow of applications. Basically, flow chart is a pictorial representation of an algorithm. Figure -2 shows the flow chart diagram of the project. The process flow starts from the home page. If user exists, then they can proceed with the login. If the user details are not registered in the database, the customer should create an account using the register page. The user needs to confirm their email address to complete the registration process. After signing in, the users can find the nearby docking station by using the post code search facility. If the user entered credentials are valid, then users can see the information regarding the selected dock station. After the viewing the bikes availability, the user should redirect to the booking view, where the users can cross check the booking details before proceeding with the payment. If the payment was unsuccessful due to any issues, the process flow will again get executed from the booking session. If the payment succeeds, day pass will be issued to the customer.



**Figure 2- Flow Chart Diagram**

## 5.2 USE CASE DIAGRAM

In the Unified Modelling Language (UML), Use Case Diagram is used to show the possible interactions of the actors with the system.



**Figure 3- Use Case Diagram**



The two actors in this system are customer and manager. Figure-3 shows the list of all possible interactions of the actors with the system. The details are mentioned below:

- **Register:** Customers can create their account by entering their email address, username, password. The manager doesn't have the register functionality as the manager details are manually added.
- **Login:** To validate, the customers or the manager must input valid username and password which are used while registering for the webpage.
- **Add or remove dock stations:** After successful login, manager can add new dock stations by inputting the required fields.
- **Bikes Availability:** Manager can use this functionality to change the availability of bikes in a particular dock station.
- **Postcode Search:** Customers can use this functionality to check the nearest available dock station. The input fields are the postcode and a mile radius to carry the search.
- **Check Availability:** Customers can use this functionality to view the available bikes.
- **Booking:** Customers can use this functionality to verify the details like pickup station, email address, charges before continuing for the payment.
- **Payment:** Customers can complete the booking after successful payment.
- **Booking Confirmation Email:** After successful payment, booking confirmation email is automatically sent to the customer.
- **Cancel Recurring Payment:** Customers can use this functionality to cancel the next day recurring payments.

### 5.3 ENTITY RELATIONSHIP DIAGRAM (ER DIAGRAM)

Entity relationship diagrams are often used to describe the relationships between the entities in a database. The logical structure of the database can easily be explained by using the ER Diagram. Figure-4 shows the database tables and their relations which are used in this project.

- **auth\_user:** auth\_user table or User model is inherited from the Django framework (django.contrib.auth). This table stores the login information of all users within the application network. Django's built-in forms, 'NewUserForm' and 'AuthenticationForm' were used in this project to get the input from customers.

The auth\_user database table contains the following model objects

- user\_id – INTEGER FIELD (primary key): automatically generated by the framework.
- username – VARCHAR (unique key): validators make sure that the username is unique
- password – PASSWORD FIELD (text): password field in Django framework makes sure that the passwords are hashed before storing in the database. Users are requested to enter password twice for confirmation.

- email – EMAIL FIELD (text): This field stores the customer's email address.
- is\_active – BOOLEAN FIELD (text): The newly registered customers are requested to verify their email address by entering a one-time which is sent to their registered email ID. Initially the 'is\_active' field is set to FALSE. When the OTP entered by the customer matches with the OTP in generated, the 'is\_active' field will become TRUE and the user can login to the website without any errors.

There are several other objects in the User table which are automatically generated by the Django Framework, which includes last\_login, date\_joined. 'is\_staff' and 'is\_superuser' status will be set to FALSE for all the customers. 'is\_staff' status will only be TRUE for the manager, whose details are manually entered to the database by admin.

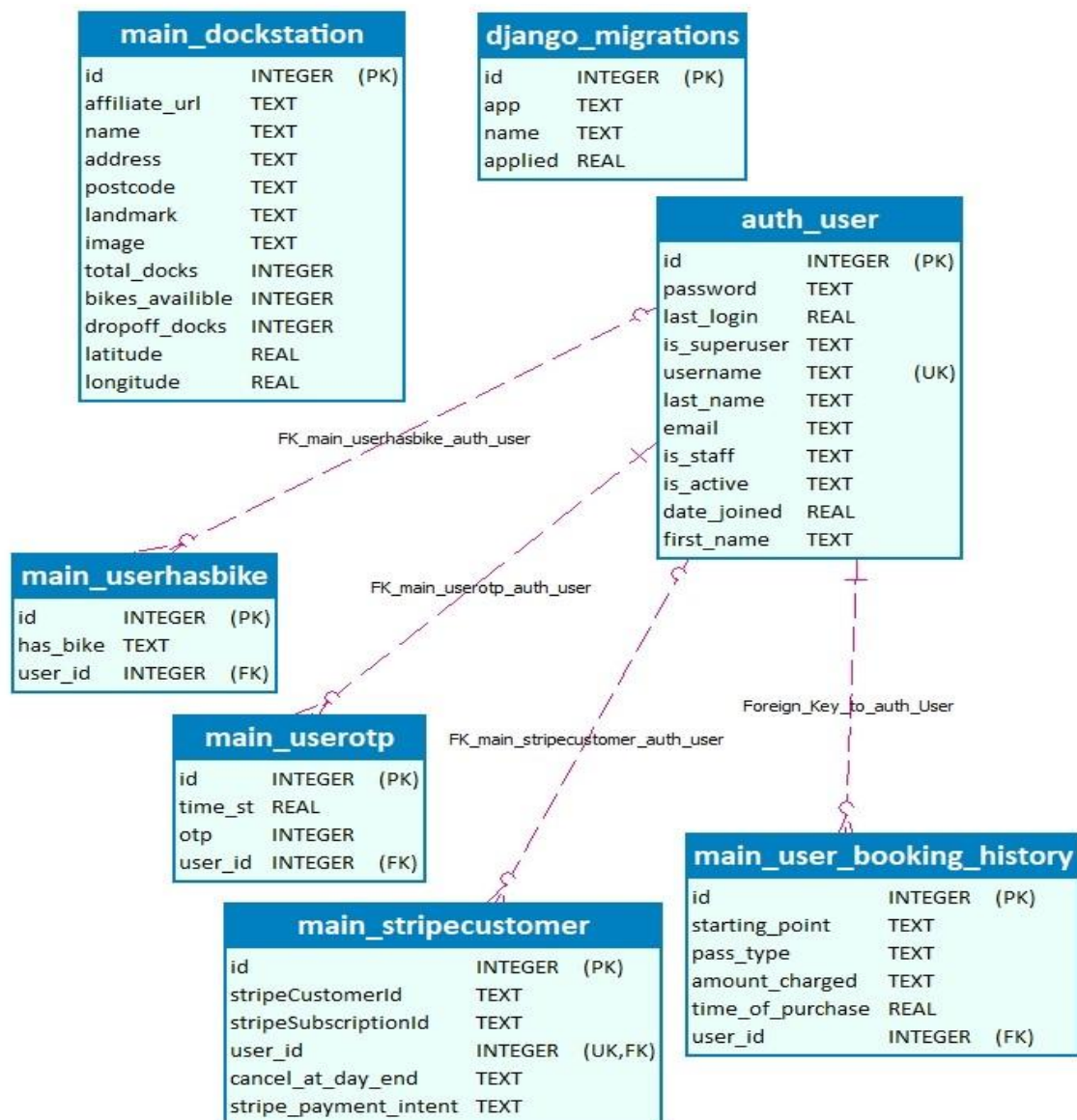
- **main\_userOTP:** The OTPs which are generated to the customers while registration process is stored in main\_userOTP database table.

The main\_userOTP database table contains the following model objects

- user\_id – INTEGER FIELD (FOREIGN KEY): The user\_id field in 'main\_userOTP' table is related to the primary key of 'AUTH\_USER' table.
- otp – INTERGER FIELD (MAX LENGTH = 6): otp field stores the passcodes which are sent to the customer by the backend application. When the customer enters their otp, the system will check for the user and otp value in main\_userOTP table and change the 'is\_active' status.

'Object ID (PRIMARY KEY)' and 'time\_stamp' which is set as auto\_now = TRUE, are the auto generated fields by Django framework.

- **django\_migrations:** Django includes schema for django\_migrations, which stores the record of all the database migrations happened in the backend.



**Figure 4- ER DIAGRAM**

- **main\_dockstation:** The dock station table in the database is managed by the manager. Manager is responsible for adding and removing the dock stations in the administrative panel.

The main\_dockstation database table contains the following model objects

- **name – CHARFIELD:** The name of Dock Station, entered by the manager will get stored in the name field.
- **address – CHARFIELD:** The address of Dock Station, entered by the manager will get stored in the address field.

- **postcode** – CHARFIELD: The Postcode of Dock Station, entered by the manager will get stored in the Postcode field.
  - **landmark** – TEXT FIELD: The landmark, which is nearest to the Dock Station, entered by the manager will get stored in the landmark field.
  - **image** – URL FIELD: The images which are used to display the dock station are stored in the form of an URL in URL Field.
  - **total\_docks** – INTEGER FIELD: Total\_docks store the value all the docks present in dock station. Total count includes the bikes available in the station and empty docks.
  - **bikes\_available** – INTEGER FIELD: bikes\_available field is used to store the value of number of bikes available in dock\_station table
  - **dropoff\_docks** – INTEGER FIELD: dropoff\_docks field is used to store the value of number of vacant docks available in dock\_station table
  - **latitude** – DECIMAL FIELD (max\_length = 6): Latitude field is used to store the location of dock station
  - **longitude** – DECIMAL FIELD (max\_length = 6): Latitude field is used to store the location of dock station
- **main\_stripecustomer:** Stripe customer table holds the payment information of customer. The data in stripe customer table is created by using the real-time notifications generated by stripe webhooks.
- The 'main\_stripecustomer' database table contains the following model objects
- **user\_id** (ONE TO ONE RELATION TO auth\_user): The user\_id in stripe customer table holds int to one relationship with username in auth\_user table. Which means that the auth\_user is associated with one and only record in stripe\_customer table. Update () method is used to update the value fields when the customer already exists in the database table.
  - **stripe\_customer\_id** – CHARFIELD: the stripe\_customer\_id is added to the database from the incoming webhook notifications
  - **stripe\_subscription\_id**– CHARFIELD: the stripe\_subscription\_id is added to the database from the incoming webhook notifications
  - **cancel\_at\_period\_end** – BOOLEAN FIELD: 'cancel\_at\_period\_end' field determines whether the recurring payments are turned on or not.
- **main\_user\_has\_bike:** The table user\_has\_bike, stores the information about the customers who are having the bikes.
- The 'main\_user\_has\_bike' database table contains the following model objects
- **user\_id** – INTEGER FIELD (FOREIGN KEY): The user\_id field in 'main\_user\_has\_bike' table is related to the primary key of 'AUTH\_USER' table.

- has\_bike – BOOLEAN FIELD: if has\_bike is TRUE, the customer will be prompted to select the dropoff station. Similarly, if has\_bike is set to FALSE, the customer will be prompted to select the pickup station.
- **main\_user\_booking\_history:** The table main\_user\_booking\_history, stores the record of all the customer's previous bookings  
The 'main\_user\_booking\_history' database table contains the following model objects
  - user\_id – INTEGER FIELD (FOREIGN KEY): The user\_id field in 'main\_user\_booking\_history' table is related to the primary key of 'AUTH\_USER' table.
  - starting\_point – Starting point field holds the record of the station address, where the customer pass is issued
  - pass\_type – Pass type holds the record of type of pass
  - amount\_charged- Amount Charged field holds the information about the amount charged from the customer.

## 6 PROJECT OUTCOME

The project is loaded with all the working functionalities. The webapp has customer module and manager module. Application server runs on localhost with port 8000. The URL endpoints for both the customer and manager modules are discussed in the sub sections 6.1 and 6.2.

### 6.1 URL ENDPOINTS FOR FUNCTIONALITIES AND WEBPAGES – CUSTOMER MODULE

**Table 3- Customer URL Endpoints**

URL ENDPOINT	DESCRIPTION	INPUT FIELDS
<a href="http://127.0.0.1:8000/">http://127.0.0.1:8000/</a>	This URL endpoint lands the customer in the home page of application (shown in Figure-3). After user signed in to the application the home page will display Tom Tom map with all dock stations and the customers are also provided with the postcode search function (Shown in Figure-9).	NONE

<a href="http://127.0.0.1:8000/register/">http://127.0.0.1:8000/register/</a>	This URL endpoint redirect the customer to registration page. After successful submission, the user will be prompted to verify the email address.	username email password confirm password
<a href="http://127.0.0.1:8000/signin/">http://127.0.0.1:8000/signin/</a>	This URL endpoint redirects the customer to sign in page.	username password
<a href="http://127.0.0.1:8000/contact/">http://127.0.0.1:8000/contact/</a>	This URL endpoint redirects the customer to contact us page. This facility is provided for unregistered customers as well	user_email subject message
<a href="http://127.0.0.1:8000/postcodesearch/">http://127.0.0.1:8000/postcodesearch/</a>	This URL endpoint redirects the registered customers from home page to postcode search page. The customers will enter their postcode and get the details of their nearby dock stations within the given mile radius. The search API shown in figure-12 will fetch the results and display it to the user.	postcode mile_radius
<a href="http://127.0.0.1:8000/password_reset/">http://127.0.0.1:8000/password reset/</a>	This URL endpoint redirects the customer to forgot password page.	user_email
<a href="http://127.0.0.1:8000/bikeavailability/&lt;int:id&gt;">http://127.0.0.1:8000/bikeavailability/&lt;int:id&gt;</a>	This endpoint lands the customers to bike availability view page. <int:id> in URL holds the dockstation_id value (Shown in Figure-15)	None
<a href="http://127.0.0.1:8000/bookings/&lt;int:id&gt;">http://127.0.0.1:8000/bookings/&lt;int:id&gt;</a>	This endpoint lands the customers to booking view page where the customers can view their selection before proceeding with the payment. <int:id> in URL holds the dockstation_id value (Shown in Figure-15)	NONE

path('stripe/create-checkout-session')  <a href="https://checkout.stripe.com/pay/cs_test_{{session}}">https://checkout.stripe.com/pay/cs_test_{{session}}</a>	The backend URL which is indicated by the variable path creates the checkout session for customer and redirect the customer to checkout.stripe URL mentioned. The input fields which are needed to complete the transaction is given in the next column	CARD_NUMBER NAME_ON_CARD EXIPRY_DATE SECURITY_CODE POSTCODE
<a href="http://127.0.0.1:8000/complete/?{{session_id}}">http://127.0.0.1:8000/complete/?{{session_id}}</a>	The customer will redirect to this end point after the transaction was successful.	NONE
<a href="http://127.0.0.1:8000/cancelled_transaction">http://127.0.0.1:8000/cancelled transaction</a>	The customer will redirect to this URL endpoint when the transaction was cancelled	NONE
<a href="http://127.0.0.1:8000/mypass">http://127.0.0.1:8000/mypass</a>	This URL endpoint lands the customer in My Pass page where the customers can view their pass and do the pickup and drop off operations and cancel next dat transactions.	dockstation_address cancel_at_period_end
<a href="http://127.0.0.1:8000/booking_history">http://127.0.0.1:8000/booking history</a>	Customers can view their booking history by navigating to this url	NONE

## 6.2 URL ENDPOINTS FOR FUNCTIONALITIES AND WEBPAGES – MANAGER MODULE

**Table 4- Manager URL Endpoints**

URL ENDPOINT	DESCRIPTION	INPUT FIELDS
<a href="http://127.0.0.1:8000/admin/">http://127.0.0.1:8000/admin/</a>	This URL endpoint redirects the manager to sign in page. The login credentials are manually entered to the database	username password

<a href="http://127.0.0.1:8000/admin/auth/user/">http://127.0.0.1:8000/admin/auth/user/</a>	Redirecting to this URL, the manager can find all the information of registered users.	NONE
<a href="http://127.0.0.1:8000/admin/main/dockstation/">http://127.0.0.1:8000/admin/main/dockstation/</a>	This URL redirects the manager to dock station page where the customer can add/edit/remove the dock station	NONE
<a href="http://127.0.0.1:8000/admin/main/dockstation/add/">http://127.0.0.1:8000/admin/main/dockstation/add/</a>	By accessing this URL, manager can add new dock stations.	station_name station_address station_postcode nearest_landmark station_description station_image total_docks available_docks dropoff_docks station_latitude station_longitude
<a href="http://127.0.0.1:8000/admin/main/stripecustomer/">http://127.0.0.1:8000/admin/main/stripecustomer/</a>	The manager will get the stripe payment details of the customer by accessing this URL	NONE
<a href="http://127.0.0.1:8000/admin/main/userhasbike/">http://127.0.0.1:8000/admin/main/userhasbike/</a>	By accessing this URL, manager will get the information regarding the customers who haven't dropped their bikes yet.	NONE
<a href="http://127.0.0.1:8000/admin/main/user_booking_history/">http://127.0.0.1:8000/admin/main/user_booking_history/</a>	By accessing this URL, manager can view all the bookings made by the customers	NONE



## 6.3 BUILDING AND RUNNING THE SERVER

To run the server in production environment, Open the project file in python source code editor. To install all dependencies, go to terminal and write the following command inside project directory

```
pip install -r requirements.txt
```

For migrating data to the database and running the server, type the following commands in terminal

```
python manage.py makemigrations  
python manage.py migrate  
python manage.py runserver
```

After completing these steps navigate to <http://127.0.0.1:8000/> , the server will be up and running.

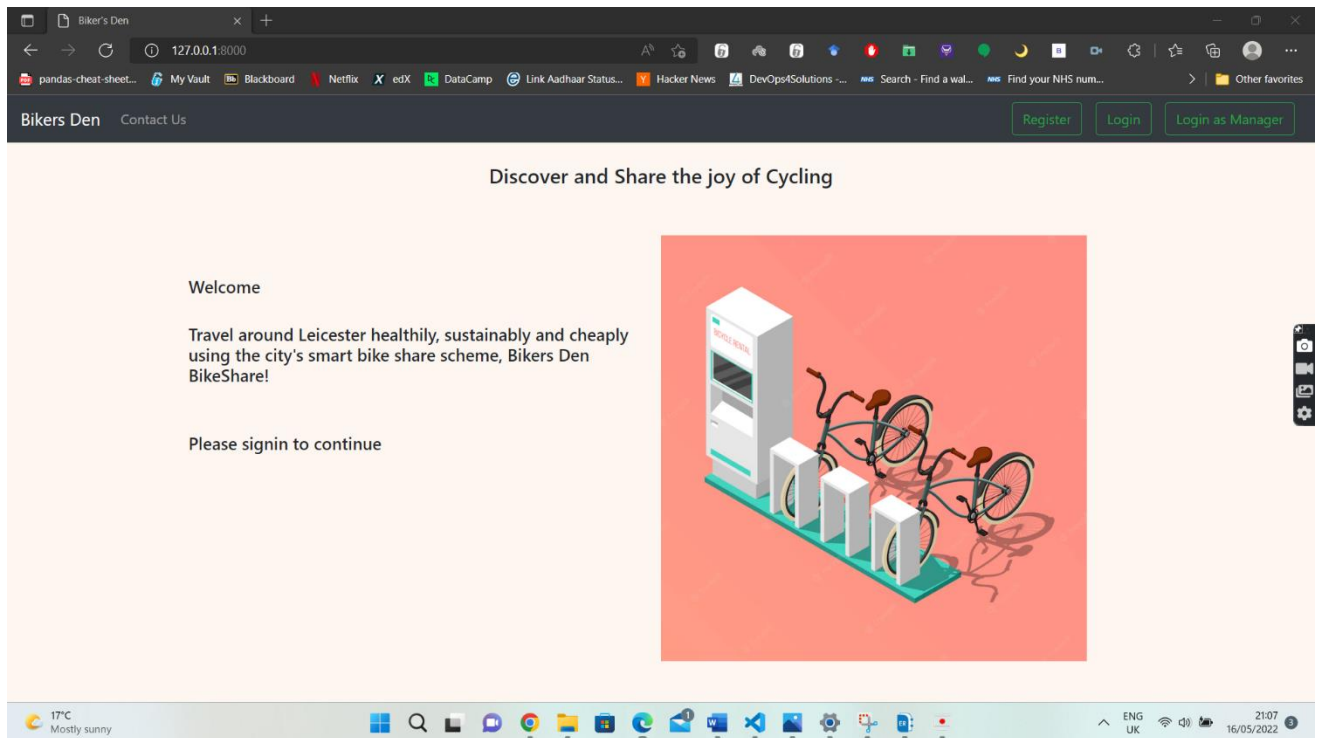
‘My Pass’ page takes the information generated from stripe webhooks. In test mode, before going for the payment, it is necessary to configure the webhooks in our local environment. The detailed instructions for setting up the stripe webhooks using stripe command line interface is explained in section- 6.4.8.

## 6.4 WEB PAGES AND CODE SNIPPETS

The logic behind the implementation and the outcome of results are clearly explained in the upcoming sub sections.

### 6.4.1 HOME PAGE

This is the landing page of the website with a navigation bar where the customers can register and login to the website is shown in the Figure-5. Manager/administrator can login into the panel by clicking login as manager button. This welcome page has Contact Us option in the navigation bar. Customers can get in touch with the team in case of any questions or concerns. This facility is also available for the non-registered customers.



**Figure 5 - Biker's Den Homepage**

## 6.4.2 CUSTOMER REGISTRATION PAGE WITH EMAIL VERIFICATION

To increase the security and prevention of dummy accounts, it's advised to verify the user email address before completion of registration process. After completing the registration form, an email with an OTP will be sent to the customer to complete the registration process by submitting the OTP. The logic involved in the customer registration process is shown in Figure- 6.

The One Time Password is generated by using the python random module. Random module is used to generate random numbers within the given range. After the submission of registration form, a random 6 digit number is generated by using `random.randint(100000,999999)` function. And the email to the customer is sent by using the Gmail SMTP settings. After the registration form submission, the input details will be stored in the database and the customer 'is\_active' status is set to FALSE. This means that the customer details are successfully stored in the database and the email address of the customer is not verified.

```

1 @csrf_protect
2 def register_request(request):
3     if request.method == 'POST':
4         get_otp = request.POST.get('otp') #213243 #None
5         if get_otp:
6             get_usr = request.POST.get('usr')
7             usr = User.objects.get(username=get_usr)
8             if int(get_otp) == UserOTP.objects.filter(user = usr).last().otp:
9                 usr.is_active = True
10                usr.save()
11                messages.success(request, f'Account is Created For {usr.username}')
12                return redirect('main:signin')
13            else:
14                messages.warning(request, f'You Entered a Wrong OTP')
15                return render(request, 'main/register.html', {'otp': True, 'usr': usr})
16        form = NewUserForm(request.POST)
17        if form.is_valid():
18            form.save()
19            username = form.cleaned_data.get('username')
20            email = form.cleaned_data.get('email')
21            usr = User.objects.get(username=username)
22            usr.email = email
23            usr.username = username
24            usr.is_active = False
25            usr.save()
26            usr_otp = random.randint(100000, 999999)
27            UserOTP.objects.create(user = usr, otp = usr_otp)
28            message = f"Hello {usr.username}\n\nTo authenticate, please enter the following one time password:\n {usr_otp} \n Thanks!"
29            send_mail("Welcome to Bikers Hub - Verify Your Email",message,settings.EMAIL_HOST_USER,[usr.email],fail_silently = False)
30            return render(request, 'main/register.html', {'otp': True, 'usr': usr})
31        else:
32            form = NewUserForm()
33        return render(request, 'main/register.html', {'register_form':form})
34

```

**Figure 6- Customer Registration page logic**

The image shows the Postman application interface for testing a REST API. A POST request is configured to the endpoint `http://localhost:8000/register/`. The request body is set to 'form-data' and contains the following parameters:

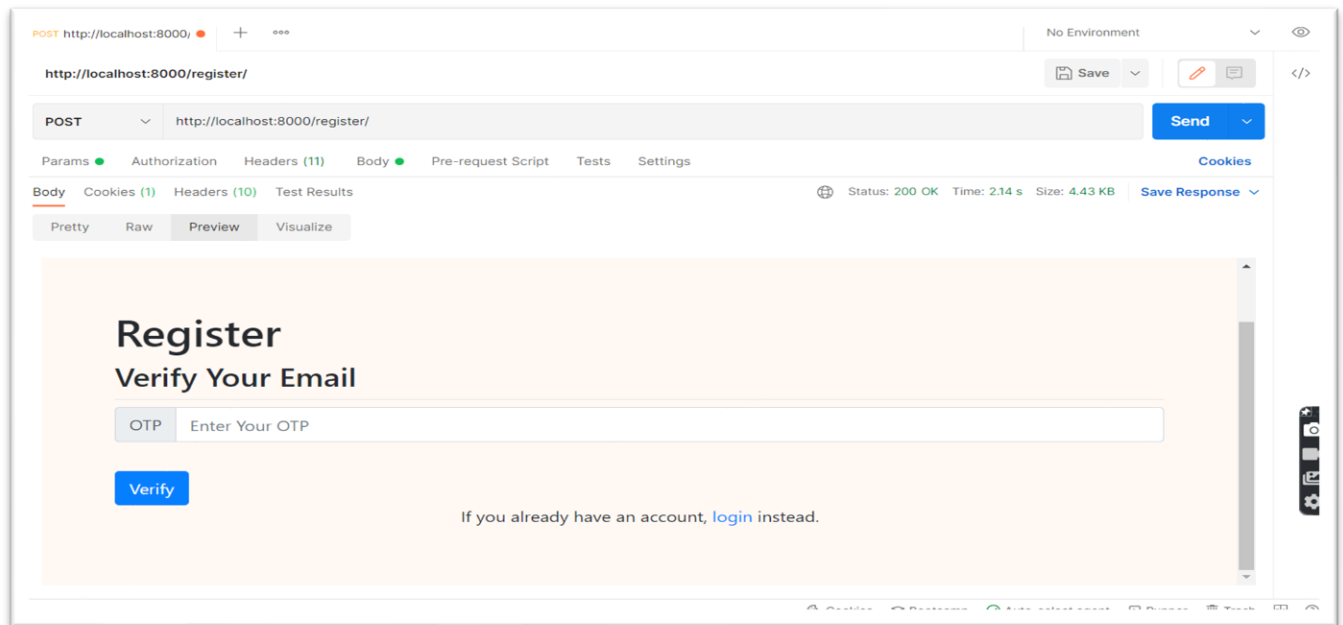
KEY	VALUE	DESCRIPTION
username	msr31	
email	msr31@student.le.ac.uk	
password1	Leicester1997!	
password2	Leicester1997!	
Key	Value	Description

The bottom status bar indicates the request was successful with a status of 200 OK, a response time of 78 ms, and a size of 5.18 KB. The response body is currently empty.

**Figure 7- Postman Test for Register endpoint**

Figure-7 shows the POST request for register functionality test in postman.

The input fields required for registration process are username and email which are Unique, two password input fields to verify the password. The output preview for the request is shown in Figure-8, the test was successfully returned 200 status response and the customer is redirected to enter the one-time password which is sent to their email ID.



**Figure 8- Postman Test Result Preview for Register endpoint**

After submitting one-time passcode, the validator will check whether the given OTP is correct or not. If OTP is valid, then the user active status will change to TRUE, which means that the registration process is completed successfully.

### 6.4.3 ADDING AND REMOVING DOCK STATIONS

When the manager clicks on 'login as manager button' from the homepage, they will be redirect to the administration panel, where the login form takes username and password as input fields for authentication. The login credentials of manager is manually entered to the database by admin.

After signing in, the manager can add a dock station by filling the required fields – Dock station name, address, postcode, image, Latitude and Longitude for the particular station.

Figure-9 shows the dock station at university of Leicester which is added to the database by the manager by inputting the required fields.

History

View on site

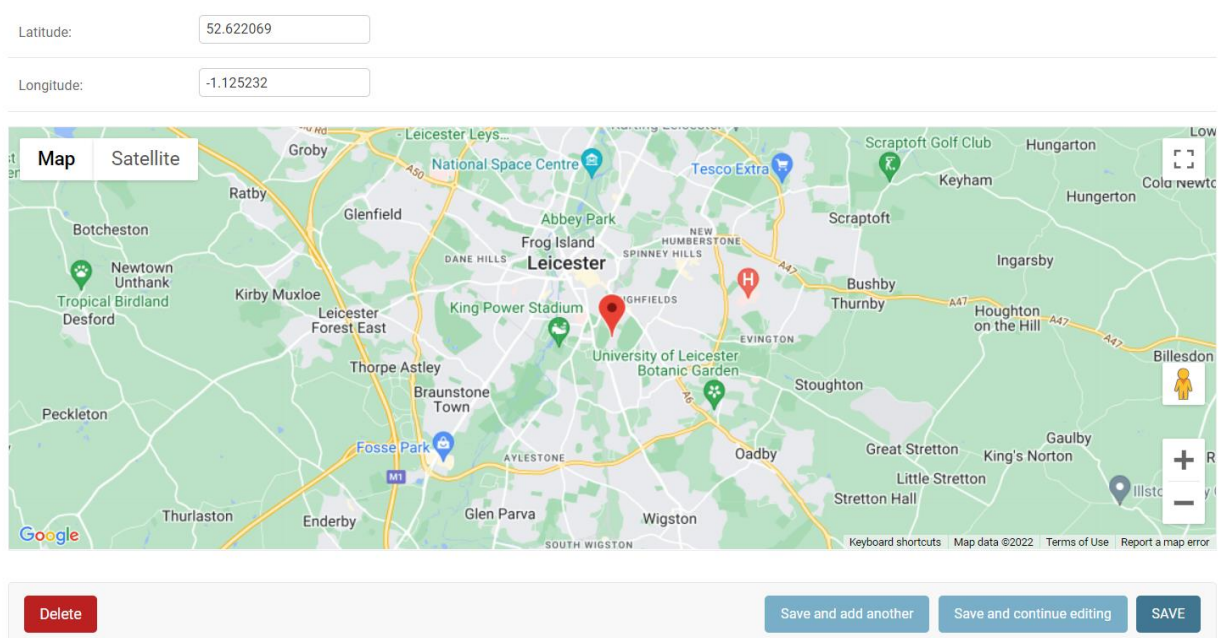
### University Road Station

Name:	<input type="text" value="University Road Station"/>
Address:	<input type="text" value="University Rd, Leicester"/>
Postcode:	<input type="text" value="LE1 7RH"/>
Landmark:	<div><div>Near University of Leicester</div><div></div></div>
Image:	<input type="text" value="https://images.unsplash.com/uploads/14122"/>
Total docks:	<input type="text" value="20"/>
Bikes available:	<input type="text" value="2"/>
Dropoff docks:	<input type="text" value="18"/>
Latitude:	<input type="text" value="52.622069"/>
Longitude:	<input type="text" value="-1.125232"/>

**Figure 9- Adding Dock Station from manager site**

By using google maps API, a simple google map widget was added to the manager panel, to mark the exact coordinates of the dock station. For displaying the google map based on latitudes and longitudes with a marker, Google Maps JavaScript API was used in this project. The steps involved in creating the GOOGLE\_MAP\_API\_KEY is shown below.

- ✓ Create Developer Account in Google Cloud Platform
- ✓ Navigate to APIs and Services
- ✓ In API Library Search for Maps JavaScript API and click ENABLE
- ✓ Navigate to credentials under APIs and Services to get the secret key



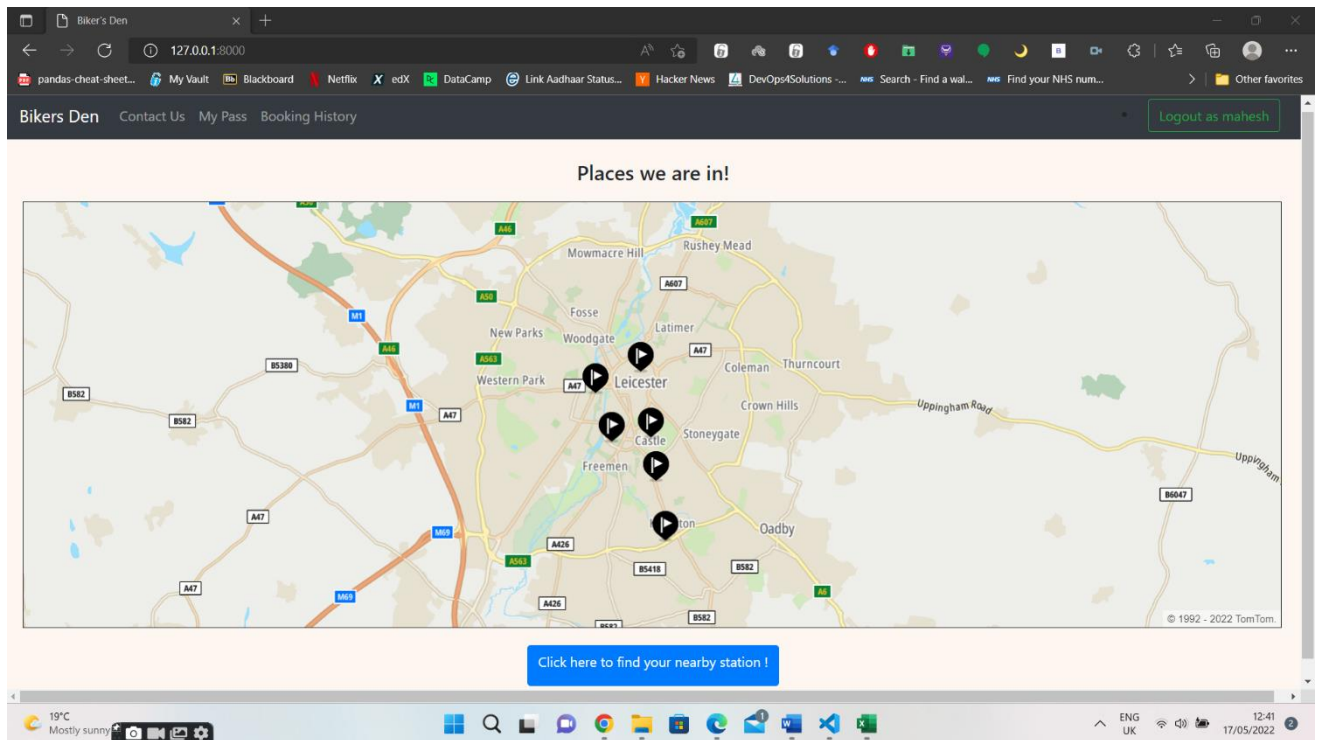
**Figure 10- Google Map marked with coordinated entered by manager**

Figure-10 shows the google map widget with a marker marked at the coordinates (52.622069 and -1.125232) which is at university of Leicester. Further the manager also has option to Delete and edit the existing dock stations.

#### 6.4.4 DOCKSTATIONS MAP VIEW

The dock stations which are added by the manager, will get stored in the database with all the fields including latitudes and longitudes. By using TomTom Maps API [8], a dynamic multiple marker map was created to view all the dock stations which are currently in service. Tom Tom maps secret key is needed to generate and display the map in the frontend of customer. Secret key is generated after registering for the developers account by following simple instructions given in the portal. The map is generated with responsive markers where the customer can further click on those markers to view the details like station name, station address postcode etc.

Figure-11 shows the customer's map view, with all the six dock stations which are added to the database.



**Figure 11- Tom Tom map with all the Dock Stations in database**

Initially, all the dock station's coordinates in the database are stored in JSON format and the JSON data is passed to the HTML file via context. The logic involved in this process is clearly shown in Figure-12.

```

1 location_list = list(DockStation.objects.order_by('name').values())
2 location_json = simplejson.dumps(location_list, use_decimal = True)
3 context = {'locations': location_json}
4 return render(request, 'main/home.html', context)

```

**Figure 12- Sending dock station coordinates as JSON to Tom Tom maps API**

The JavaScript code which is used to generate the map based on the context data is shown in Figure-13. Initially, an empty map without markers was created by passing the secret key and map product info.



```

1  <script>
2      // create the map
3      tt.setProductInfo('TomTom Maps Django Demo', '1.0');
4      let map = tt.map({
5          key: 'nnGFDAG5fbU1G2fImGXlvJ84EEfVPGuC',
6          container: 'map'
7      });
8
9      // add markers
10     let bounds = []
11     let stationLocations = JSON.parse("{ locations|escapejs }");
12
13     for (let stationLocation of stationLocations) {
14         let coordinates = [stationLocation.longitude, stationLocation.latitude];
15         bounds.push(coordinates);
16
17         // create popup to display store information when the marker is clicked
18         let popup = new tt.Popup().setHTML(`
19             <div class="locator-popup">
20                 <h6>Station Name</h6>
21                 <p>${stationLocation.name}</p>
22                 <h6>Address</h6>
23                 <p>${stationLocation.address}</p>
24                 <h6>Post Code</h6>
25                 <p>${stationLocation.postcode}</p>
26             </div>
27         `);
28
29         let marker = new tt.Marker()
30             .setLngLat(coordinates)
31             .setPopup(popup)
32             .addTo(map);
33     }

```

**Figure 13- Tom Tom Maps JavaScript code snippet**

To display the map with dynamic multiple markers, an empty list named 'bound' was created and JSON data is parsed inside that list. By using the for loop, the geographical data with latitudes and longitudes is iterated and stored inside the bound list. We have used the popup function to display the information whenever the marker is clicked. Now we have the latitude and longitude information of all the dock stations stored inside the list called bound. Popup variable stores the information which is needed to display when the customer clicks the map marker. As shown in Figure-11, the fully functional map is displayed by creating markers and passing the coordinates data. The customers can view the popup information simply by clicking the marker.



#### 6.4.5 FINDING NEARBY STATIONS USING POSTCODE SEARCH

To find the nearby docking stations, the customer will be asked to enter their postcode and the mile radius. Mile radius is nothing but the distance within their entered postcode range. When the customer clicks on submit button after filling in the required fields, an API fetch request will be sent to the search endpoint and the results are displayed using bootstrap cards. The code snippet to explain the logic involved in search API is shown in Figure-14.

```
1  @csrf_protect
2  def search(request):
3      DockStation_objects = DockStation.objects.all()
4      postcode = request.GET.get('postcode')
5      mile_radius = request.GET.get('mile_radius')
6      user_latitude = None
7      user_longitude = None
8      if postcode:
9          geolocator = Nominatim(user_agent='main')
10         location = geolocator.geocode(postcode)
11         user_latitude = location.latitude
12         user_longitude = location.longitude
13     output = []
14     for DockStation_object in DockStation_objects:
15         result = {}
16         result['name'] = DockStation_object.name
17         result['image'] = DockStation_object.image
18         result['landmark'] = DockStation_object.landmark
19         result['postcode'] = DockStation_object.postcode
20         result['address'] = DockStation_object.address
21         result['total_docks'] = DockStation_object.total_docks
22         result['bikes_available'] = DockStation_object.bikes_available
23         result['dropoff_docks'] = DockStation_object.dropoff_docks
24         result['id'] = DockStation_object.id
25         result['url'] = '127.0.0.1:8000/bikeavailability/' + str(DockStation_object.id)
26         if postcode:
27             initial_coords = (float(user_latitude), float(user_longitude))
28             station_coords = (float(DockStation_object.latitude), float(DockStation_object.longitude))
29             result['distance'] = round(great_circle(initial_coords, station_coords).miles, 3)
30         output.append(result)
31         output = sorted(output, key=lambda d: d['distance'])
32         if mile_radius:
33             if result['distance'] > float(mile_radius):
34                 output.pop()
35     return JsonResponse(output, safe=False)
```

**Figure 14- Postcode Search Backend Logic**

At first, the dock station objects in the database are triggered by using `Model.objects.all ()` method. The model objects information is then stored inside the 'DockStation\_objects' variable. By using the GET request, the user entered details about the postcode and the mile-radius are fetched and stored inside their respective variables. The initial values of the local variables, `user_latitude` and `user_longitude` are set to `None`. Validators are put in place and

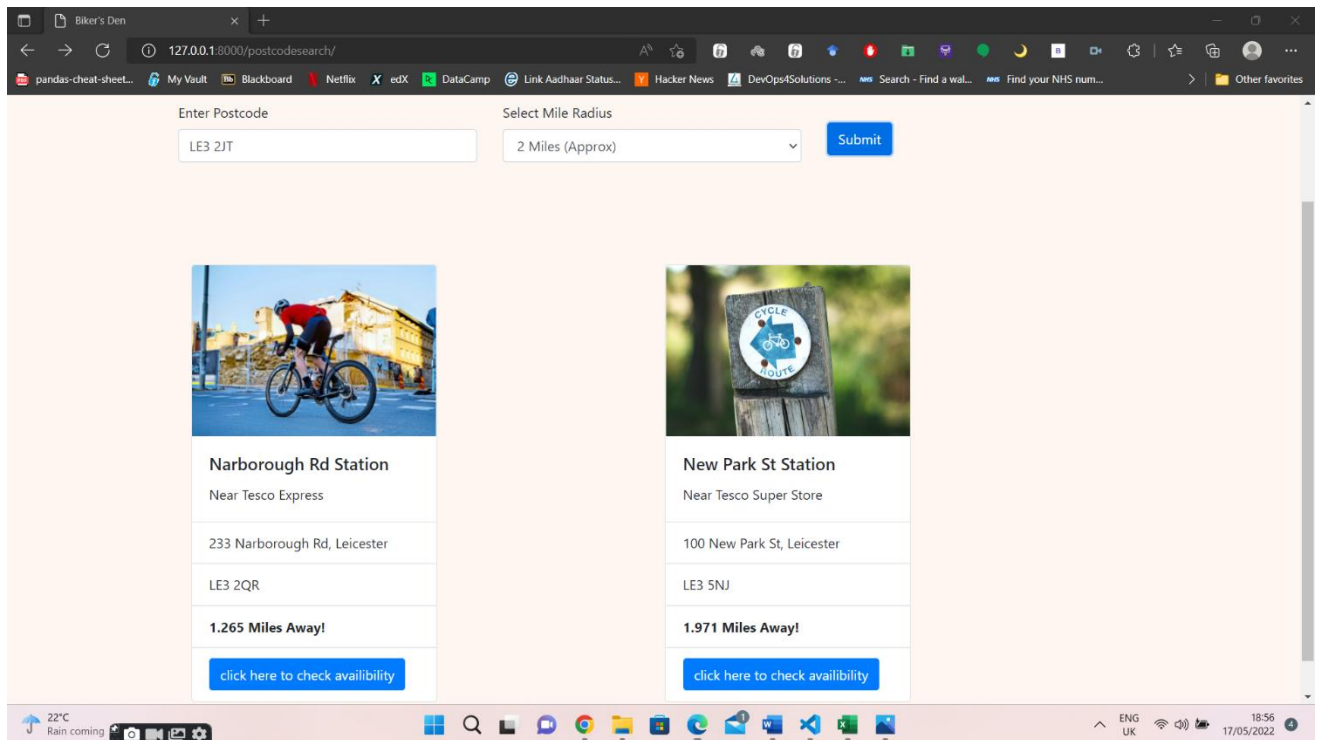
if the submitted information is invalid, the customer will get an error message and will be asked to re-enter the postcode and mile radius for their search. If the customer entered valid information, then the input postcode which is in the string format is passed into a geolocator to convert the postcode into a set of latitudes and longitudes. We have used the Geopy's Nominatim geolocator to convert the entered postcode into geospatial coordinates. The converted coordinates are taken as the user's location and the values stored in `user_latitude` and `user_longitude` variables will get updated. A key-value pair dictionary called `result` is created by looping over the data present in the dock station table. The information regarding each dock station is stored inside the output dictionary. When the user sends data to the server using POST request, the search API returns the response in JSON Format. The response now will have a extra field called 'Distance'. The response is created by drawing a circle with the mile radius given by the customer, taking the given postcode as centre. In JSON response, distance field stores the values obtained by calculating the distance between customer's postcode and the respective dock station. The output response is sorted in Ascending order by using the lambda function.

Figure 15 shows the Vue JS command to fetch the API request from backend.



*Figure 15- Command to fetch data from API request*

Vue JS [11] is used in this project to fetch and display the results from search API. The Vue component `v-for` is used in iterating the API response. The response is then displayed to the customer in the form of bootstrap cards. Figure-16 shows the output when the customer enters LE3 2JT as postcode and integer- 2 as `mile_radius`.



**Figure 16- Postcode search output**

Check Availability button redirects the customer to the absolute URL of the dock station with endpoint extending the ID of that dock station. In that page, the customer can view all the information including the bikes availability.

#### 6.4.6 BIKE AVAILABILITY VIEW

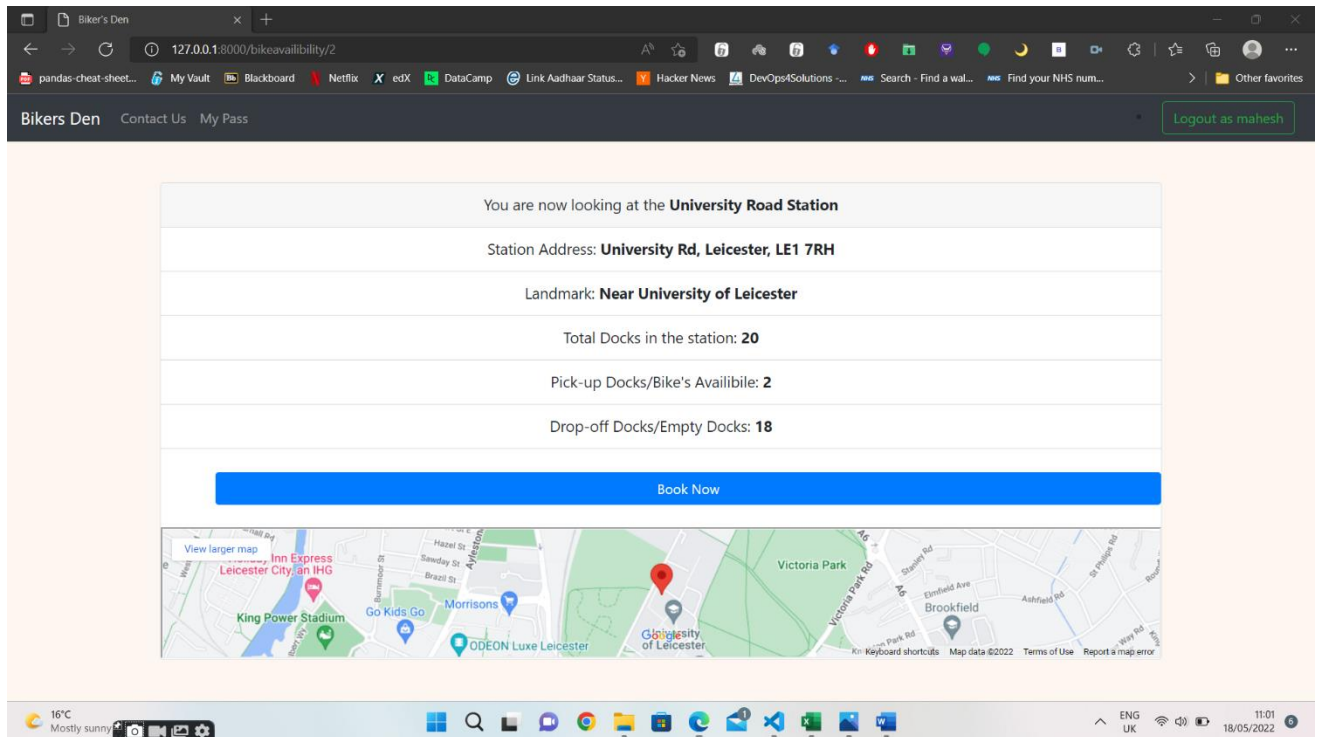
Figure-17 shows the bike availability view of the dock station which is at university of Leicester.

The url endpoint which holds the information regarding bike availability is shown below

<http://127.0.0.1:8000/bikeavailability/<int:id>>

Here <int:id> holds the dynamic id value of each dockstation

A google map iframe with the station location is embedded under the card component. The google map iframe helps the users navigating to their preferred dock station, a single click on the map will redirect to the google maps service and from there the customer can view the directions, traffic etc. After viewing the requested information, the customers can proceed with the booking by clicking on the book now button.

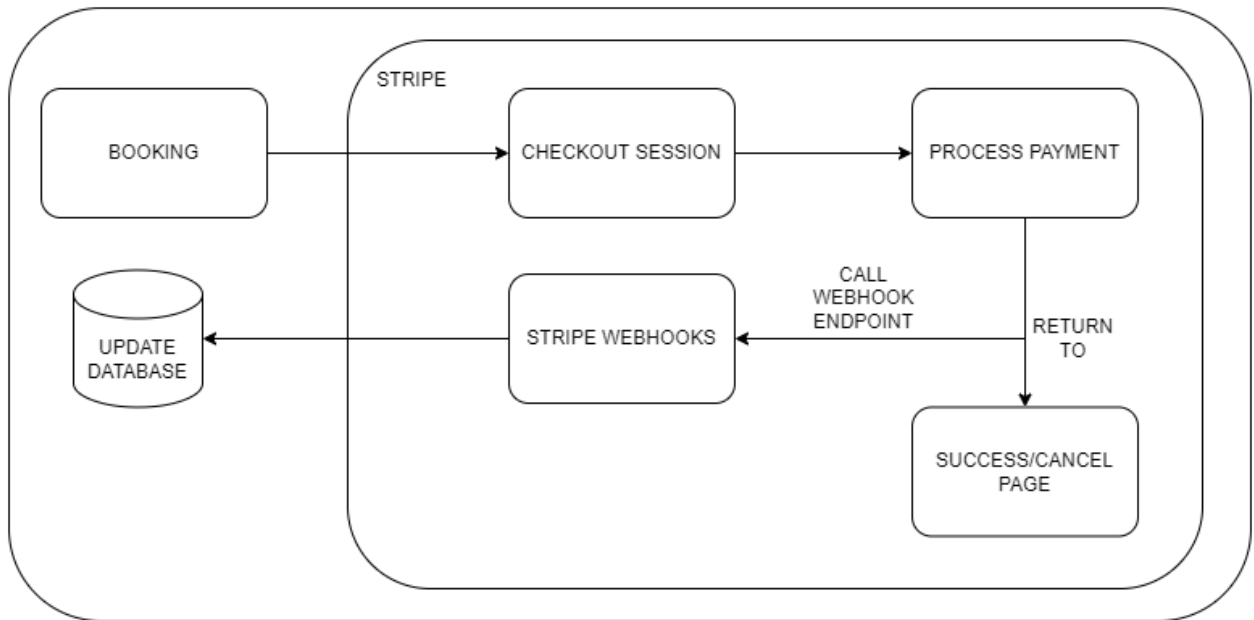


**Figure 17- Bike Availability View**

## 6.4.7 BIKE BOOKING AND PAYMENT

Biker's den platform offers their services on full day-pass basis. The payment gateway used in this project is Stripe. Stripe API allows the developers to access the stripe functionalities which are needed to make and manage payments online [10]. The two commonly used payment modes in stripe are one-time payments and stripe subscriptions. For issuing day pass on time, stripe subscriptions were used as a payment mode in this project. Stripe subscriptions allows the customers to access the product for a fixed time. Initially the recurring payment option will be set to True by default, meaning that the pass will auto renew after the expiration time. In this project, the customers are provided with an option to cancel those recurring payments any time.

After clicking 'Book now' in bike availability page shown in Figure-16, the customer will be redirected to the booking page where they can recheck the information like booking station, charges for the day pass, registered email etc. After confirming the details, the page will take the customer to stripe checkout session. After successful payment, the customer will be redirected to the payment success page and day pass will be issued to the customer. The customer can view his pass at 'My Pass' section on navigation bar. The payment flow diagram for this process is shown in Figure-18.



**Figure 18- Stripe Payment Flow**

To accept and map the payments, a product called Full Day Pass is created in stripe developer's account. The product holds all the information like the product name, product id, product price, billing plan (daily or weekly) etc.

The JSON response for the day pass(product) after its creation is shown below

```

{
  "id": "prod_LYux340aW0BVhd",
  "object": "product",
  "active": true,
  "attributes": [
  ],
  "created": 1650736291,
  "description": "Valid for one day between opening hours",
  "images": [
  ],
  "livemode": false,
  "metadata": {
  },
  "name": "Full Day Pass",
  "type": "service",
  "updated": 1650736291,
}

```

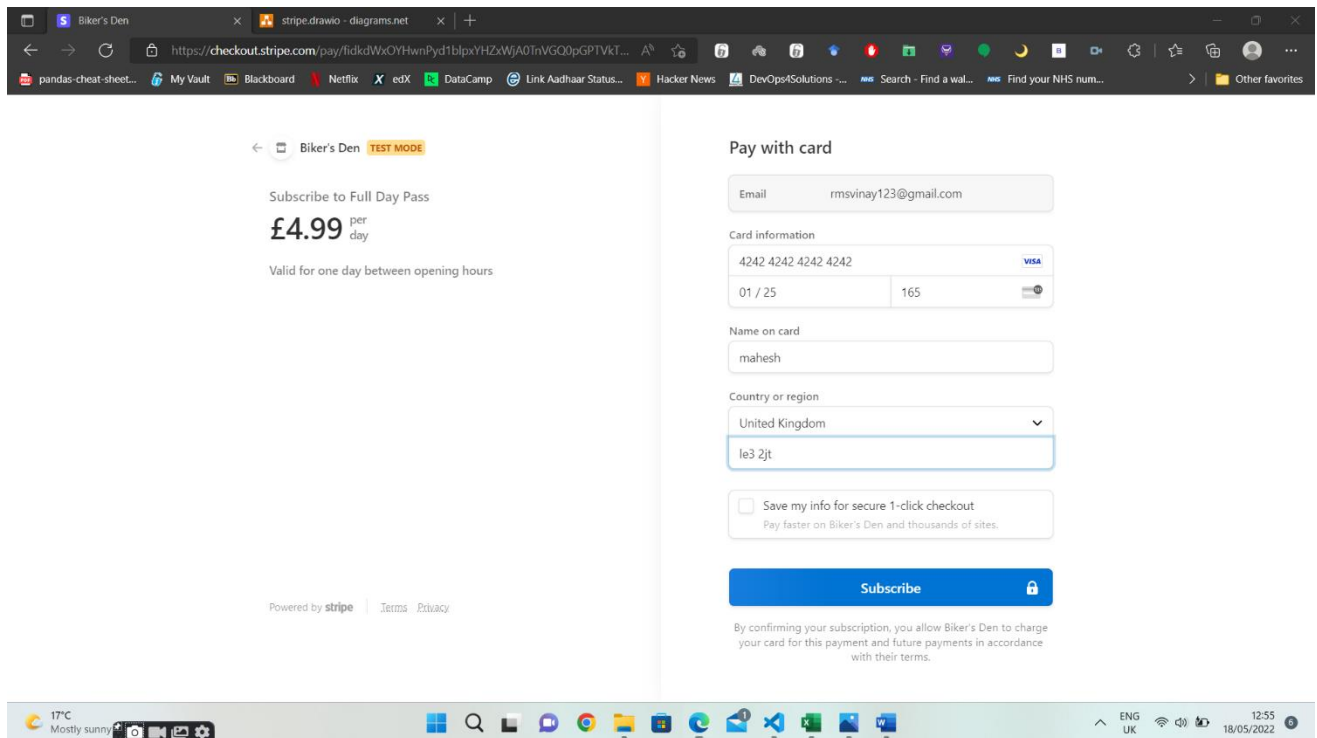
```

1  @csrf_protect
2  def create_checkout_session(request):
3      if request.method == 'POST':
4          email = request.POST.get('email')
5          booking_to = request.POST.get('booking_to')
6          station = DockStation.objects.get(address = booking_to)
7          user = User.objects.get(username=request.user)
8          if userhasbike.objects.filter(user = user).exists():
9              userhasbike.objects.filter(user=user).delete()
10             userhasbike.objects.create(user = request.user, has_bike = False)
11             stripe.api_key = settings.STRIPE_SECRET_KEY
12             checkout_session = stripe.checkout.Session.create(
13                 success_url=request.build_absolute_uri(reverse('main:complete') ) + "?session_id={CHECKOUT_SESSION_ID}",
14                 cancel_url=request.build_absolute_uri(reverse('main:cancelled_transaction')),
15                 client_reference_id=request.user.id if request.user.is_authenticated else None,
16                 customer_email = email,
17                 payment_method_types=['card'],
18                 line_items=[
19                     {
20                         'price': settings.STRIPE_PRICE_ID,
21                         'quantity': 1,
22                     }
23                 ],
24                 mode='subscription',
25             )
26             stripecustomer = userhasbike.objects.get(user = request.user)
27             stripecustomer.has_bike = True
28             station.bikes_available = station.bikes_available -1
29             station.dropoff_docks = station.dropoff_docks + 1
30             user_booking_history.objects.create(user = request.user, starting_point = station.address, pass_type = 'Full Day Pass', amount_charged = '4.99' )
31             station.save()
32             stripecustomer.save()
33             return redirect(checkout_session["url"])

```

**Figure 19- Stripe Checkout Session Logic**

Figure-19 shows the logic for creating a stripe checkout session, the customer's email and dock station address for the booking is fetched from the session. The fetched dock station address is used to make the GET request and the response is stored in 'station' variable. The user\_has\_bike table holds the information of the customers who haven't dropped their bikes yet. The Stripe Secret, Stripe Publishable key and the Product ID which are needed to create a payment checkout are saved in settings.py file. The API keys can be found in stripe developer's dashboard. Stripe is currently configured in Test mode. If the payment is successful, then the customer will be redirected to the payment success page. For the cancelled or incomplete transactions, the customer will be redirect to the cancelled\_transaction. Simulated payments can be made in order to test the system. Stripe provides a list of test card numbers based on card type and location. The interactive test for the checkout session form using test card details is shown in Figure-20.



**Figure 20- Stripe Checkout Session Output Snippet**

The card details used to make the interactive test is mentioned below

```
CARD NUMBER: 4242 4242 4242 4242 (stripe test card number - visa)
EXPIRY DATE: 01/25 (any date in the future)
SECURITY CODE: 165 (any three digits)
NAME ON CARD: MAHESH (any name)
POSTCODE: LE3 2JT (any postcode)
```

If the transaction is successful, the customer's subscription details will be uploaded to the database in Stripe Customer table. The stripe customer table object fields- user (One to One relation with auth user), stripe subscription id and stripe customer id. These details are fetched from the stripe webhooks. The details of configuring and using the webhooks with stripe transactions is clearly explained in 6.1.8 – Webhooks.

If a customer books their bike from University Road Station, the pickup bikes or available bikes present at the given station will be decreased by one and the drop off docks or vacant docks in that station will increase by 1.

The required data will be uploaded to the user\_booking\_history table in the database. The customers can view their booking under 'Booking History' section.



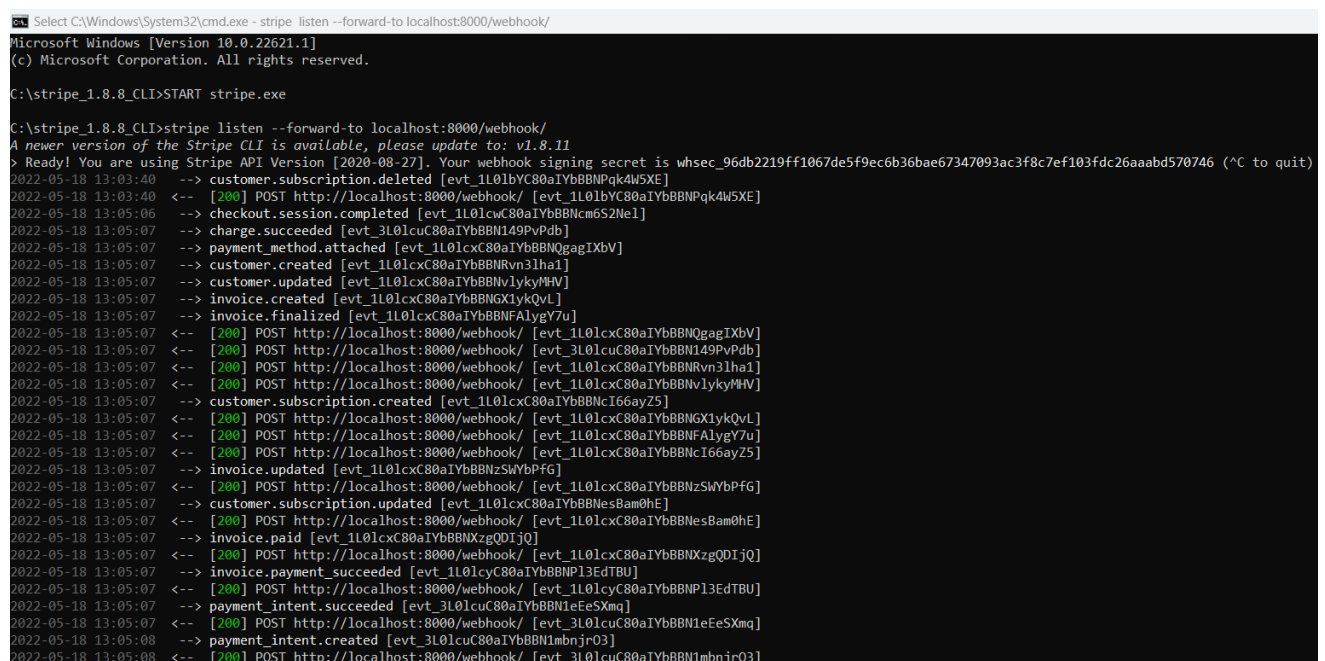
## 6.4.8 USING STRIPE WEBHOOKS TO CONFIRM THE USER PAYMENT

After the successful payment, the user is now redirected to the payment success page which was defined while creating the checkout session. The user will redirect to 'success\_url' as shown in Figure-19. We haven't confirmed this transaction programmatically yet. We can't just rely on the success page redirection as payment confirmation. So webhooks are used in the project to confirm the status of the payment. Stripe API uses incoming webhooks to notify the application when there is an event took place. Stripe Command Line Interface (CLI) is needed to configure the stripe webhooks in local environment [11]. The following commands are used to start stripe CLI in local environment.

```
<--- DOWNLOAD STRIPE CLI FROM https://stripe.com/docs/stripe-cli --->
<--- RUN COMMAND PROMPT--->
-----
<--- OPEN STRIPE CLI --->
START stripe.exe

<---STRIPE LOGIN--->
stripe login

<---ACTIVATE WEBHOOK--->
stripe listen --forward-to localhost:8000/webhook/
```



```

C:\> Select C:\Windows\System32\cmd.exe - stripe listen --forward-to localhost:8000/webhook/
Microsoft Windows [Version 10.0.22621.1]
(c) Microsoft Corporation. All rights reserved.

C:\stripe_1.8.8_CLI>START stripe.exe

C:\stripe_1.8.8_CLI>stripe listen --forward-to localhost:8000/webhook/
A newer version of the Stripe CLI is available, please update to: v1.8.11
> Ready! You are using Stripe API Version [2020-08-27]. Your webhook signing secret is whsec_96db2219ff1067de5f9ec6b36bae67347093ac3f8c7ef103fdc26aaabd570746 (^C to quit)
2022-05-18 13:03:40 --> customer.subscription.deleted [evt_1L0lbYC80aIYbBBNPqk4W5XE]
2022-05-18 13:03:40 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lbYC80aIYbBBNPqk4W5XE]
2022-05-18 13:05:06 --> checkout.session.completed [evt_1L0lcwC80aIYbBBNcm6S2Ne1]
2022-05-18 13:05:07 --> charge.succeeded [evt_3L0lcuC80aIYbBBN149PvPdb]
2022-05-18 13:05:07 --> payment_method.attached [evt_1L0lcx80aIYbBBNQagIXbV]
2022-05-18 13:05:07 --> customer.created [evt_1L0lcx80aIYbBBNRvn3lha1]
2022-05-18 13:05:07 --> customer.updated [evt_1L0lcx80aIYbBBNvlykyMHV]
2022-05-18 13:05:07 --> invoice.created [evt_1L0lcx80aIYbBBNGX1ykQvL]
2022-05-18 13:05:07 --> invoice.finalized [evt_1L0lcx80aIYbBBNFAlgyY7u]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNQagIXbV]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_3L0lcuC80aIYbBBN149PvPdb]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNRvn3lha1]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNvlykyMHV]
2022-05-18 13:05:07 --> customer.subscription.created [evt_1L0lcx80aIYbBBNcI66ayZ5]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNGX1ykQvL]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNFAlgyY7u]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNcI66ayZ5]
2022-05-18 13:05:07 --> invoice.updated [evt_1L0lcx80aIYbBBNzSWYbPfG]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNzSWYbPfG]
2022-05-18 13:05:07 --> customer.subscription.updated [evt_1L0lcx80aIYbBBNesBam0hE]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNesBam0hE]
2022-05-18 13:05:07 --> invoice.paid [evt_1L0lcx80aIYbBBNXzgQDIjQ]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNXzgQDIjQ]
2022-05-18 13:05:07 --> invoice.payment_succeeded [evt_1L0lcx80aIYbBBNP13EdTBU]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_1L0lcx80aIYbBBNP13EdTBU]
2022-05-18 13:05:07 --> payment_intent.succeeded [evt_3L0lcuC80aIYbBBN1eFeSXmq]
2022-05-18 13:05:07 <-- [200] POST http://localhost:8000/webhook/ [evt_3L0lcuC80aIYbBBN1eFeSXmq]
2022-05-18 13:05:08 --> payment_intent.created [evt_3L0lcuC80aIYbBBN1mbnjr03]
2022-05-18 13:05:08 <-- [200] POST http://localhost:8000/webhook/ [evt_3L0lcuC80aIYbBBN1mbnjr03]
```

Figure 21- Stripe CLI Webhook Response



Activating webhook generates webhook endpoint key. This key is used in our logic to get the real time updates from webhooks and trigger actions accordingly. Figure-21 shows the webhook response in local environment when the customer is redirected from the checkout session to payment confirmation page.

```
1  @csrf_exempt
2  def stripe_webhook(request):
3      stripe.api_key = settings.STRIPE_SECRET_KEY
4      endpoint_secret = settings.STRIPE_ENDPOINT_SECRET
5      payload = request.body
6      sig_header = request.META['HTTP_STRIPE_SIGNATURE']
7      try:
8          event = stripe.Webhook.construct_event(
9              payload, sig_header, endpoint_secret
10             )
11     except ValueError as e:
12         # Invalid payload
13         return HttpResponse(status=400)
14     except stripe.error.SignatureVerificationError as e:
15         # Invalid signature
16         return HttpResponse(status=400)
17     # Handle the checkout.session.completed event
18     if event['type'] == 'checkout.session.completed':
19         session = event['data']['object']
20         # Fetch all the required data from session
21         client_reference_id = session.get('client_reference_id')
22         stripe_customer_id = session.get('customer')
23         stripe_subscription_id = session.get('subscription')
24         payment_intnt_id = session.get('payment_intent')
25         # Get the user and create a new StripeCustomer
26         user = User.objects.get(id=client_reference_id)
27         send_mail(subject='You Bought a Day Pass!',
28                 message=email_message,
29                 from_email=settings.EMAIL_HOST_USER,
30                 recipient_list=[user.email],
31                 fail_silently = False
32             )
33         if StripeCustomer.objects.filter(user = user).exists():
34             StripeCustomer.objects.filter(user=user).update(
35                 user=user,
36                 stripeCustomerId=stripe_customer_id,
37                 stripeSubscriptionId=stripe_subscription_id,
38                 stripe_payment_intent = payment_intnt_id,
39                 cancel_at_day_end = False
40             )
41         StripeCustomer.objects.create(
42             user=user,
43             stripeCustomerId=stripe_customer_id,
44             stripeSubscriptionId=stripe_subscription_id,
45             stripe_payment_intent = payment_intnt_id,
46             cancel_at_day_end = False
47         )
48         print(user.username + ' just subscribed.')
49         return HttpResponse(status=200)
```

*Figure 22 - Webhook Logic for fetching event data and triggering actions*

Figure-22 shows the logic for webhook view. After proving the STRIPE SECRET KEY and WEBHOOK ENDPOINT KEY, the event data from stripe command line interface is fetched and stored in payload variable. The header signature will be of HTTP type. Since we configured our API in test mode, stripe allows us to communicate via HTTP protocol. HTTPS is required when the configuration is changed from test mode to live mode. After getting the webhook data, the event handler will check for the event 'checkout session completed'. The 'checkout session completed' event pops up only when the transaction is succeeded. So, after getting the completed event notification, the subscription details of the customer are uploaded to the database. These details can be used to issue the day pass and other services.

#### 6.4.9 ISSUING DAY PASS TO THE CUSTOMER

The final step after confirming the payment is to issue day pass to the customer. The logic for issuing day pass is shown in Figure- 23.

```

1  qr_data = None
2  @login_required
3  def mypass(request):
4      global qr_data
5      try:
6          stripe_customer = StripeCustomer.objects.get(user=request.user)
7          stripe.api_key = settings.STRIPE_SECRET_KEY
8          subscription = stripe.Subscription.retrieve(stripe_customer.stripeSubscriptionId)
9          product = stripe.Product.retrieve(subscription.plan.product)
10         address1= DockStation.objects.values_list('address',flat=True)
11         availability = DockStation.objects.values_list('bikes_available',flat=True)
12         stations = DockStation.objects.all()
13         qr_data = {
14
15             'user' : stripe_customer.user.username,
16             'pass':product.name,
17             'status' : subscription.status,
18         }
19         img = make(qr_data,box_size =5.5)
20         img.save("main/static/image/qrcode.png")
21         stripe_customer_status = stripe_customer.cancel_at_day_end
22         if userhasbike.objects.filter(user = request.user).exists():
23             status = userhasbike.objects.get(user = request.user)
24             has_bike = status.has_bike
25         return render(request, 'main/mypass.html', {
26             'subscription': subscription,
27             'product': product,
28             'stations':stations,
29             'address1':address1,
30             'availability':availability,
31             'has_bike':has_bike,
32             'stripe_customer_status' : stripe_customer_status,
33             'qr_data':qr_data,
34         })
35     except StripeCustomer.DoesNotExist:
36         return render(request, 'main/mypass.html')

```

*Figure 23- Logic for issuing day pass to the customer*

Initially, the auth user objects in stripe customer table is fetched and stored in 'stripe\_customer' variable. The stripe customer table holds the details of stripe subscription id and stripe customer id of auth user. STRIPE SECRET KEY is provided to verify the details of the customer from Stripe Account. The details regarding the customer subscription are retrieved from stripe account, by providing the subscription id which exists in the database. JSON response which is mentioned below shows the output data for the 'subscription' variable when the transaction is successful.

```
{
  "object": {
    "id": "sub\_1L1709C80aIYbBBNZuuKdGrG",
    "object": "subscription",
  },
  "billing_cycle_anchor": 1652957681,
  "cancel_at_period_end": false,
  "collection_method": "charge_automatically",
  "created": 1652957681,
  "customer": "cus\_LiY6TI4whJIPDC",
  "default_payment_method": "pm_1L1708C80aIYbBBNyqsfeVu",
  "items": {
    "object": "list",
    "data": [
      {
        "id": "si\_LiY6h9SLoledC7",
        "object": "subscription_item",
        "created": 1652957682,
        "metadata": {
        },
        "plan": {
          "id": "price\_1Krn7IC80aIYbBBNQ1AlMxLj",
          "object": "plan",
          "active": true,
          "amount_decimal": "499",
          "billing_scheme": "per_unit",
          "created": 1650736292,
          "currency": "gbp",
          "interval": "day",
          "interval_count": 1,
          "livemode": false,
          "metadata": {
          },
        },
      },
    ],
  },
}
```

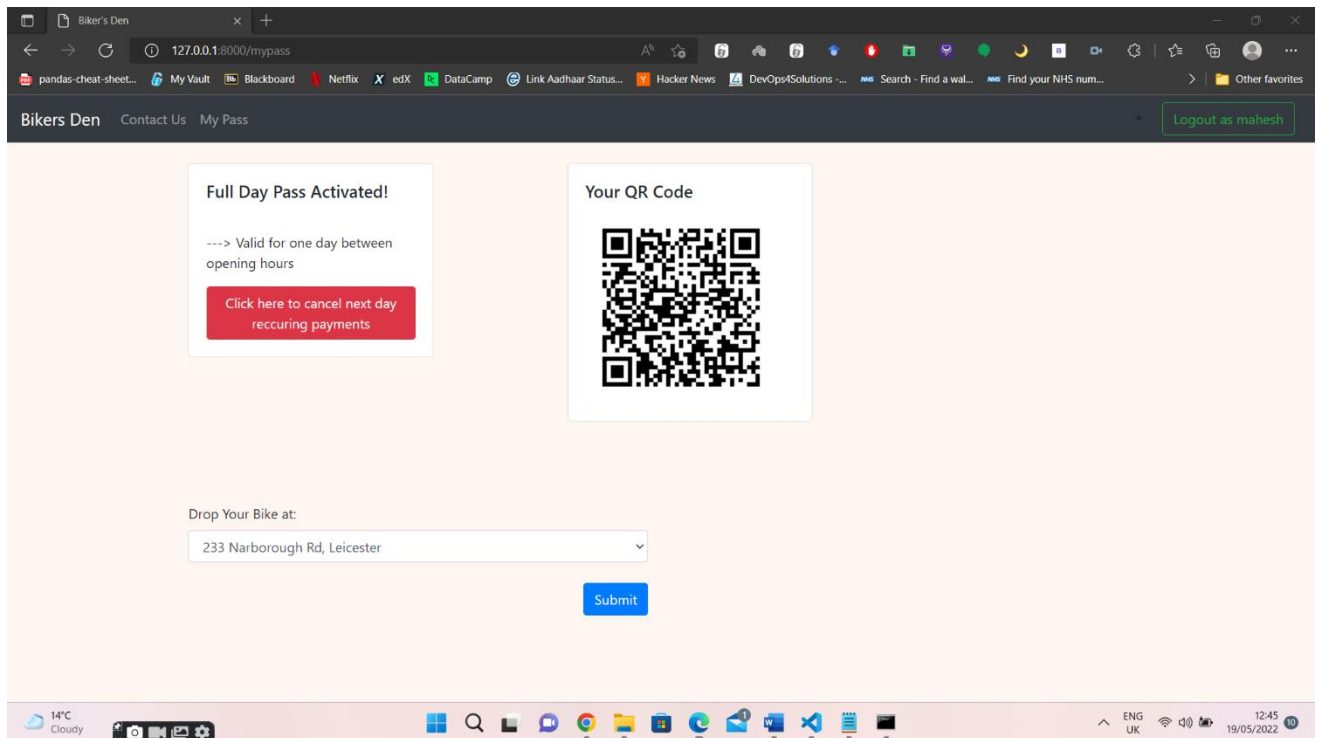
```

        "product": "prod_LYux340aW0BVhd",
        "usage_type": "licensed"
    },
    "price": {
        "id": "price_1Krn7IC80aIYbBBNQ1AlMxLj",
        "object": "price",
        "active": true,
        "billing_scheme": "per_unit",
        "created": 1650736292,
        "currency": "gbp",
        "livemode": false,
        "lookup_key": "daypass",
        "metadata": {
        },
        "product": "prod_LYux340aW0BVhd",
        "recurring": {
            "aggregate_usage": null,
            "interval": "day",
            "interval_count": 1,
            "trial_period_days": null,
            "usage_type": "licensed"
        },
    },
}

```

We can find all the data including product id, subscription type, recurring payments status etc. Under the subscription field, we can find the status of customer's subscription which is set to active. The subscription data is passed to the HTML file via context.

The condition in HTML file, `{% if subscription.status == "active" %}`, makes sure that the pass is only displayed to the customer whose payments are received and subscription status are not expired. A QR code is also attached with the pass. In this project, QR code is used as multi factor authentication for unlocking and locking the bikes at the customer preferred dock station. Figure-24 shows the My Pass page of customer, with pass details, QR code and options to select the dock station where the customer wants to drop the bike. QR code is generated by using python's QR-code package. The QR code holds the username, pass type and status data in it.



**Figure 24- Active Day Pass with QR code**

Customers are also provided with an option to cancel the next day recurring payments. The process involved in recurring payments cancellation is explained in the section 6.4.10.

#### 6.4.10 CANCELLING THE NEXT DAY RECCURING PAYMENTS

```

1  @login_required
2  def cancel(request):
3      try:
4          # Retrieve the subscription & product
5          stripe_customer = StripeCustomer.objects.get(user=request.user)
6          stripe.api_key = settings.STRIPE_SECRET_KEY
7          subscription = stripe.Subscription.retrieve(stripe_customer.stripeSubscriptionId)
8          subscription.cancel_at_period_end = True
9          stripe_customer.cancel_at_day_end = True
10         subscription.save()
11         stripe_customer.save()
12         messages.success(request, f"Reccuring payments cancelled successfully...")
13         return render(request, 'main/cancel.html')
14     except StripeCustomer.DoesNotExist:
15         return render(request, 'main/home.html')

```

**Figure 25- Cancelling Next Day Recurring Payments Logic**

Figure-25 shows the logic involved in cancelling the recurring payments. Initially the stripe customer is retrieved from the session and the auth user subscription details are fetched and stored under 'subscription' variable. When the customer clicks on 'cancel next day recurring payments' option which is shown in Figure- 22, the cancel at day end status of the stripe customer will be changed to True and the customer will not get be from the next day. The customers can still user their day pass until the period end. The JSON response log created in stripe after the cancellation of recurring payments is shown below

```
{
  "object": {
    "id": "sub\_1L1709C80aIYbBBNZuuKdGrG",
    "object": "subscription",
  },
  "billing_cycle_anchor": 1652957681,
  "cancel_at_period_end": true,
  "collection_method": "charge_automatically",
  "created": 1652957681,
  "customer": "cus\_LiY6TI4whJIPDC",
  "default_payment_method": "pm_1L1708C80aIYbBBNyqsfveVu",
  "items": {
    "object": "list",
    "data": [
      {
        "id": "si\_LiY6h9SLoledC7",
        "object": "subscription_item",
        "created": 1652957682,
        "metadata": {
        },
        "plan": {
          "id": "price\_1Krn7IC80aIYbBBNQ1AlMxLj",
          "object": "plan",
          "active": true,
          "amount_decimal": "499",
          "billing_scheme": "per_unit",
          "created": 1650736292,
          "currency": "gbp",
          "interval": "day",
          "interval_count": 1,
          "livemode": false,

```



### 6.4.11 PICK BIKE AND DROP BIKE LOGIC

Figure-26 shows the logic behind pick bike and drop bike functionalities. The logic revolves around 'bike availability' object and 'empty docks' object in the database. In this project, each dock station is created with a total number of 20 docks. Out of those 20 docks, 15 are for bike picking operations and the remaining 5 docks are vacant and will be used for bike dropping operations. If the customer books their day pass from university of Leicester station, the available bikes count in that station will be reduced by one and empty dock count will be increment by one. After picking the

bike at university road dock station, the 'user\_has\_bike' object for the customer is set to True, meaning a bike has been allotted to the customer. If the user wants to drop the bike, they can choose any dock station from the list which is closer to them. After dropping bike at the station, the 'user\_has\_bike' status is set to FALSE where, the user will again get an option to pick a bike from any of the stations located in the city. The customers can choose their bike picking and bike dropping stations at any time from 'my pass' section until the pass validity.

```
1 def pickbike(request):
2     if request.method == 'POST':
3         booking = Booking.objects.get(user = request.user)
4         booking_to = request.POST.get('booking_to')
5         station = DockStation.objects.get(address = booking_to)
6         station.bikes_availible = station.bikes_availible - 1
7         station.dropoff_docks = station.dropoff_docks + 1
8         stripecustomer = userhasbike.objects.get(user = request.user)
9         stripecustomer.has_bike = False
10        stripecustomer.save()
11        station.save()
12        messages.success(request, f"Your Request was successfull. You can pick your bike at {booking_to}")
13        return redirect("main:homepage")
14
15 def dropbike(request):
16     if request.method == 'POST':
17         booking = Booking.objects.get(user = request.user)
18         booking_to = request.POST.get('booking_to')
19         station = DockStation.objects.get(address = booking_to)
20         station.bikes_availible = station.bikes_availible + 1
21         station.dropoff_docks = station.dropoff_docks -1
22         stripecustomer = userhasbike.objects.get(user = request.user)
23         stripecustomer.has_bike = False
24         stripecustomer.save()
25         station.save()
26         messages.success(request, f"Bike Succesfully dropped at {booking_to}")
27         return redirect("main:homepage")
```

**Figure 26- Pick Bike and Drop Bike Logic**

## 7 CONCLUSION

From the beginning of the thesis, the purpose and scope of the project are clearly exposed. The concepts for the future improvements are also presented in section-7.2. The main aim of the project is to develop a web application which allows customers to hire bicycles for their everyday commute. According to a recent survey, most people living in cities are now using bike sharing schemes as a medium of transport. Biker's Den customers will get benefited from a new service which will allow them to book bikes online based on day passes.

The purpose and objectives of Biker's Den project is achieved and the web application is loaded with all the essential and recommended features (mentioned in 2.2.1 and 2.2.2). All the other optional features have been successfully implemented. The implemented optional features include QR code generation after successful payment, cancelling the next day recurring payments and inclusion of highway code rules. The real time session management can't be achieved in this project, as it requires physical hardware equipment like trackers and sensors to communicate with the server.

During the thesis and development process, I learned a lot about the general structure of many software systems and the overall notion of the system as a whole.

This project is loaded with all the fully working functionalities which can be accessed by the customer and manager. Customer functionalities include postcode search option, buying day pass, picking and dropping of bikes from the dock stations, cancelling the next day recurring payments. Manager functionalities include adding and maintaining dock stations, managing users etc., The third party APIs which are used in this project are Google Maps API, Tom Tom Maps API and Stripe Payments API.

The thesis thus, demonstrates the best use of technologies and tools available to develop the project with fully working functionalities and APIs.

### 7.1 TESTING

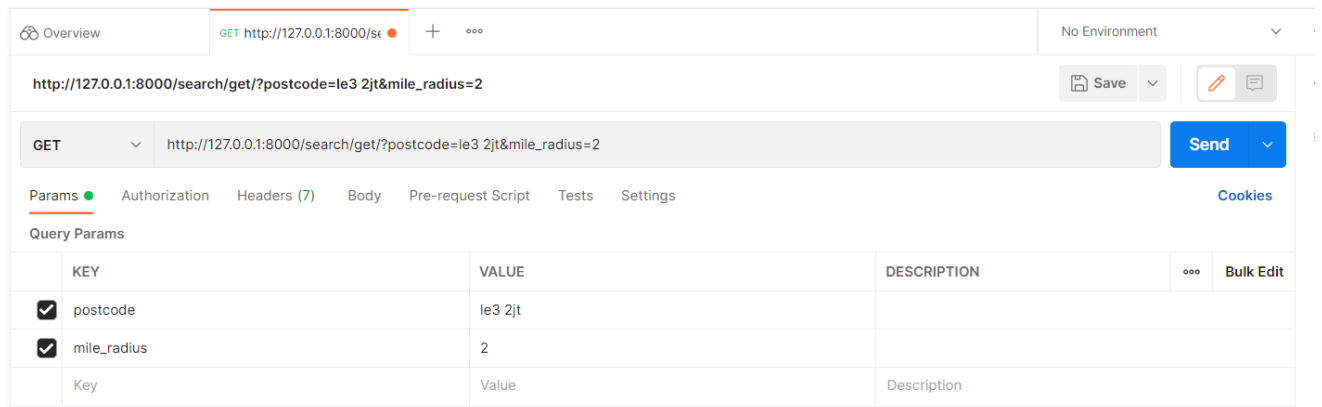
Website testing is basically checking the webapp to identify any potential bugs before it is publicly available to the end-user. Since this author followed waterfall approach to develop this project, testing is done in almost every phase. The testing methods followed while developing the project is discussed in further sections.

#### 7.1.1 FUNCTIONALITY TESTING

Functionality testing is a method in which the application requirements will be mapped or compared with the application features to assess whether the output reaches the

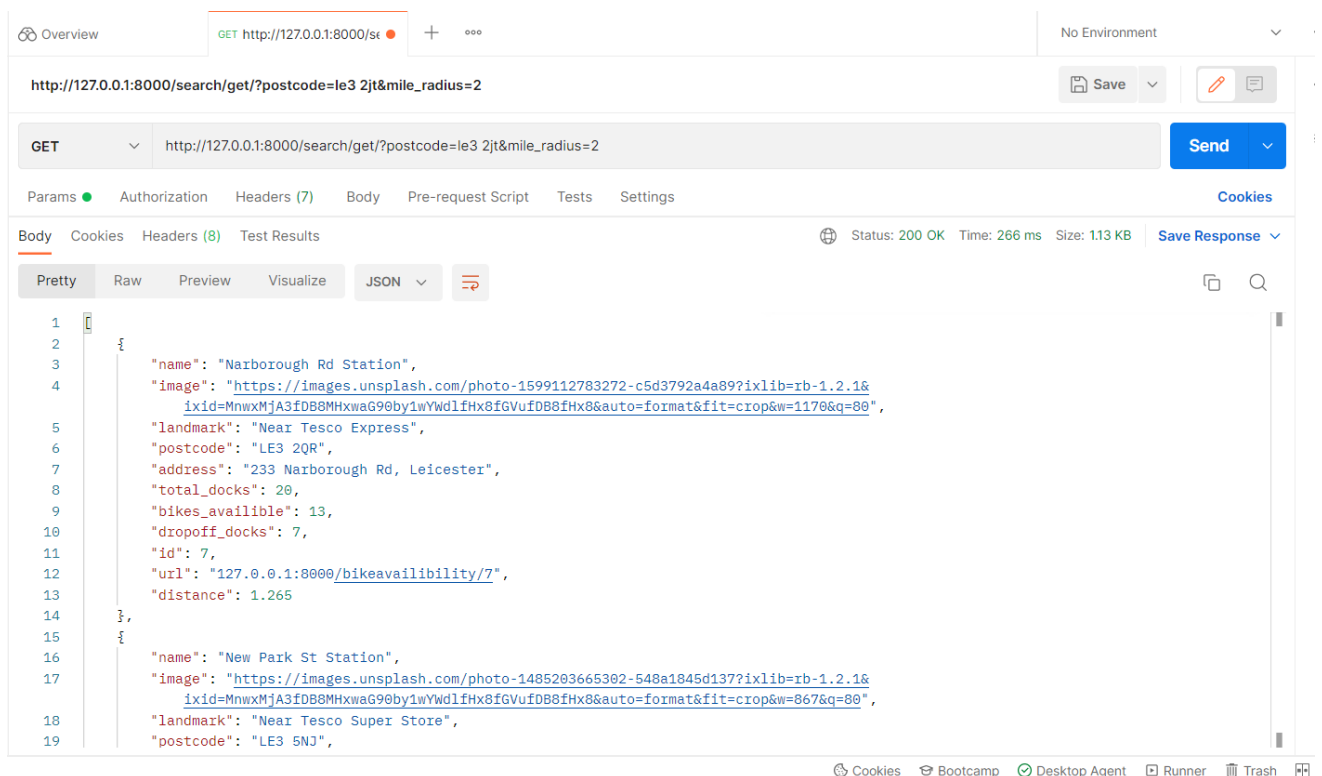


expectations of end-user. It basically tests each application feature with its software requirement.



**Figure 27- Functionality Test in Postman**

One of the challenging tasks in this project is implementing the postcode search functionality. The function should return the nearby dock stations to the customer when the given keyword arguments are valid. Postman tests the HTTP requests (GET, POST, PUT, PATCH, DELETE) and returns different responses based on the request. Figure-27 shows the testing of postcode search functionality in Postman API Platform. The test was conducted by inputting the postcode as 'LE3 2JT' and the mile radius as 2.



**Figure 28- Test Result for Postcode Search Functionality**

The test returned a '200 OK' status response and the output JSON format is shown in Figure-28. The nearest dock station from LE3 2JT is Narborough Road Station and it is 1.265 miles away. Similarly, the functionality is tested by inputting postcode as 'B15 1DA' and mile\_radius as 10. The test returned an empty JSON response as there are no dock stations available in the given postcode within the given mile radius. The results of all the other tests conducted in postman are given below

**Table 5- Postman Testing - Result**

Functionality	Input Fields	Status Code	Summary
Customer Registration function	Username: Mahesh Email: <a href="mailto:msr31@student.le.ac.uk">msr31@student.le.ac.uk</a> Password1: London1997! Password2: London1997!	200 OK	User registered successfully. 200 response denotes that the request was successful
Login function	Username: Mahesh Password1:Leicester1997!	400 ERROR	400 error is displayed because the password given Leicester1997! doesn't match with the password stored in database
Login Function	Username: Mahesh Password: London1997!	200 OK	User logged in successfully
Post code search API	Postcode: B15 1DA Mile_radius: 10	204	204 status code denotes that the response is empty. There are no dock stations available in B15 1DA postcode within 10 mile radius

Post code search API	Postcode: B15 1DA Mile_radius: 50	200 OK	The response returns 200 status code. A dock station is present within the search criteria. And the distance from postcode to dock station is 42.843 m.
Post code search API	Postcode: LE3 2JT Mile_radius: 1	204	204 status code denotes that the response is empty. There are no dock stations available near LE3 2JT postcode within 1 mile radius. The user will be asked to expand their search by increasing the mile radius

### 7.1.2 CROSS BROWSER TESTING

Cross browser testing comes under non-functionality testing where the application is manually launched in several popular web browsers like Google Chrome, Microsoft Edge, Opera, Mozilla Firefox etc. and tested to find out if there are any bugs or issues present. The application worked well in all browsers except Google Chrome. The local host kept throwing '404 Page Not Found' error. After doing further research, the author found out that all the other browsers are appending slashes automatically after the URL and Google Chrome is looking for the exact URL which is defined in the backend application. To fix this issue a simple command `APPEND_SLASH = True` is added to the settings file of the project. This command will ensure that the slashes are going to append while calling for URL even when there are no slashes present in the backend URLs.

## 7.2 FUTURE IMPROVEMENTS

Real time projects often have larger scope for improvements. For postcode search, this project uses Geopy's Nominatim geolocator to convert the postcode into latitude and longitude pair. Nominatim comes free with the Geopy package but it offers limited functionalities and low request limits. We can make more queries efficiently by using a paid service like Google Geolocator, Open Quest Map or Pick Point. For calculating the distance between dock station and customer's location, the author had used Geopy's 'great\_circle' function. The result distance field is not 100 percent accurate, as the distance is calculated from point to point in the form of a straight line. By using the Directions API from google maps and other service providers, we can determine the exact distance between the two points by considering the traffic and one way routes. Due to lack of time, the author had only issued day passes. Weekly passes and monthly passes can also be included on the booking view page where the user will have an option to select their pass type. The QR code generated at 'my pass' page is not encrypted. Anyone with a smart phone can view the information stored in that QR code. For adding another layer of security, the QR codes should be encrypted, so that it requires a proper decoder to decrypt the information stored in QR code. Python's OpenCV offers great support in encrypting and decrypting the data in image files. Further, a directions API can also be integrated to show the information regarding the routes. The stripe payments API used in this project accepts only credit card or debit card payments. Additional payment options like Google Pay and Apple Pay can also be integrated for quicker checkouts.

## 8 REFERENCES

- [1] Zhou, J., Guo, Y., Sun, J., Yu, E. and Wang, R., 2022. Review of bike-sharing system studies using bibliometrics method. Journal of Traffic and Transportation Engineering (English Edition).
- [2] Pojani, D., Chen, J., Mateo-Babiano, I., Bean, R. and Corcoran, J., 2020. Docked and dockless public bike-sharing schemes: research, practice, and discourse. In Handbook of Sustainable Transport. Edward Elgar Publishing.
- [3] Ricci, M., 2015. Bike sharing: A review of evidence on impacts and processes of implementation and operation. Research in Transportation Business & Management, 15, pp.28-38.
- [4] Guo, Y., Yang, L. and Chen, Y., 2022. Bike share usage and the built environment: a review. Frontiers in public health, 10.
- [5] GOV.UK. 2022. Transport and environment statistics: Autumn 2021. [online]  
Available at: [https://www.gov.uk/government/statistics/transport-and-environment-statisticsautumn-2021/transport-and-environment-statistics-autumn-2021#:~:text=Data%20sources&text=Transport%20produced%207%25%20of%20the,transport%20vehicles%20\(111%20MtCO2e\)](https://www.gov.uk/government/statistics/transport-and-environment-statisticsautumn-2021/transport-and-environment-statistics-autumn-2021#:~:text=Data%20sources&text=Transport%20produced%207%25%20of%20the,transport%20vehicles%20(111%20MtCO2e))
- [6] "3.7.3 Documentation." Python.org, 2019, Available: <https://docs.python.org/3/>
- [7] "Django documentation | Django documentation | Django", Docs.djangoproject.com, 2022. [Online]. Available at: <https://docs.djangoproject.com/en/4.0/>
- [8] "Django Web Framework (Python) - Learn web development | MDN", developer.mozilla.org 2022. [Online]. Available at: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django>
- [9] Tom Tom Maps API Developers Documentation  
Developer.tomtom.com. 2022. Introduction | Map Display API. [online]  
Available at: <https://developer.tomtom.com/map-display-api/documentation/product-information/introduction>
- [10] "Django", Fullstackpython.com, 2022. [Online].  
Available at: <https://www.fullstackpython.com/django.html>
- [11] Stripe.com. 2022. Documentation. [online] Available at: <https://stripe.com/docs>
- [12] Stripe.com. 2022. Stripe CLI Reference. [online]  
Available at: <https://stripe.com/docs/cli/trigger#trigger-event>
- [13] Vuejs.org. 2022. Vue.js - The Progressive JavaScript Framework | Vue.js. [online]  
Available at: <https://vuejs.org/>
- [14] Docs.microsoft.com. 2022. Learn Django tutorial in Visual Studio, step 1, Django basics. [online]  
Available at: <https://docs.microsoft.com/en-us/visualstudio/python/learn-django-in-visual-studio-step-01-project-and-solution?view=vs-2022>