BIKE SHARE APP PROJECT

Authors	Student ID	Coding section	Project management & report
Ze Li	24203409	60%	10%
Anju Mariyam Joseph	24219517	20%	45%
Sabrina Alzbeta Ragulova	24231301	20%	45%
URL to the video: https://github.com/zeli8888/BikeShareApp/blob/main/report/video3060177501.mp4			
URL of the GitHub project: https://github.com/zeli8888/BikeShareApp.git			

1. Project Overview

1.1. Project Objective

Cycling is a widely used form of transportation in Ireland. The demand is growing due to promotion of sustainable transportation, which supports health and benefits of economics. The demand is growing due to promotion of sustainable transportation, which supports health and benefits of economics. In recent years, cycling rates have been increasing due to the multiple schemes as a government initiative. The most common challenge faced by the users is locating these bikes or identifying the bike docking stations. This causes frustrations leading to the inefficiency of the journey for the user.

Furthermore, another factor affecting the user journey efficiency is the weather conditions. Dublin is experiencing changing weather, due to this, riders are most likely to choose biking on sunny days rather than rainy days. As a result of these observations, real-time updates on bike availability and weather are necessary for users. To accommodate these challenges faced by the user, the BikeShareApp web application was created. It is primarily designed to help the users get the bike's information in Dublin and receive real-time updates of the current weather condition.

1.2. Project goals

The goal of this project was to develop a mobile app which acts as an all-inclusive solution for users. It covers the following aspects:

- Real-time updates about the bike and stand availability to avoid the ineffectual journey planning.
- Real-time weather information integrated in the app to assist users to take informative decisions in terms of trip planning.
- Data visualization for bike stations to identify the busy docking stations and daily trends to plan for optimal travel.
- Route planning effectively for a seamless user experience.
- Promote sustainability by providing users with a web app which is dependable and approachable.

1.3. Target audience

The app was designed with the following user's demand:

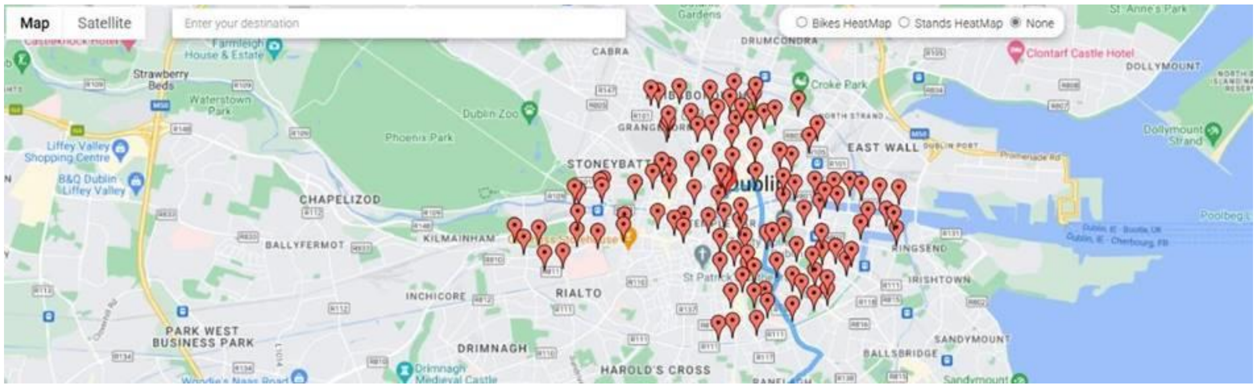
- Daily travellers: These are people who travel regularly, possibly on habitual time.
- Visitors / Tourists: People who would like to explore the city on bikes for various reasons such as flexibility, cost-effectiveness and eco-friendly travel experience.

- Recreational / casual bikers: People who just enjoy the occasional bike trips either for leisure or other casual reasons.

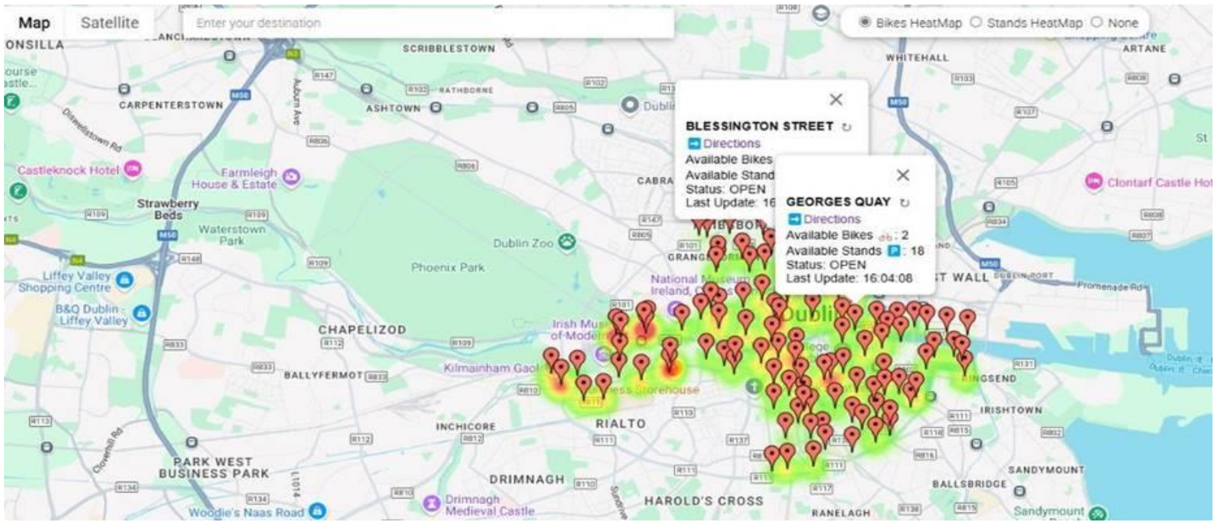
1.4. App Functionality or features:

These are the distinctive features that were integrated in the application:

- **Interactive Google Map Interface:** Google map interface was integrated by using the Google maps JavaScript API. To provide the users with an easy-to-user or familiar mapping interface. The basic functionality of the maps can be used, such as the zoom in and out to view the different bike station data, hover on to explore the different bike stations across different areas of Dublin.

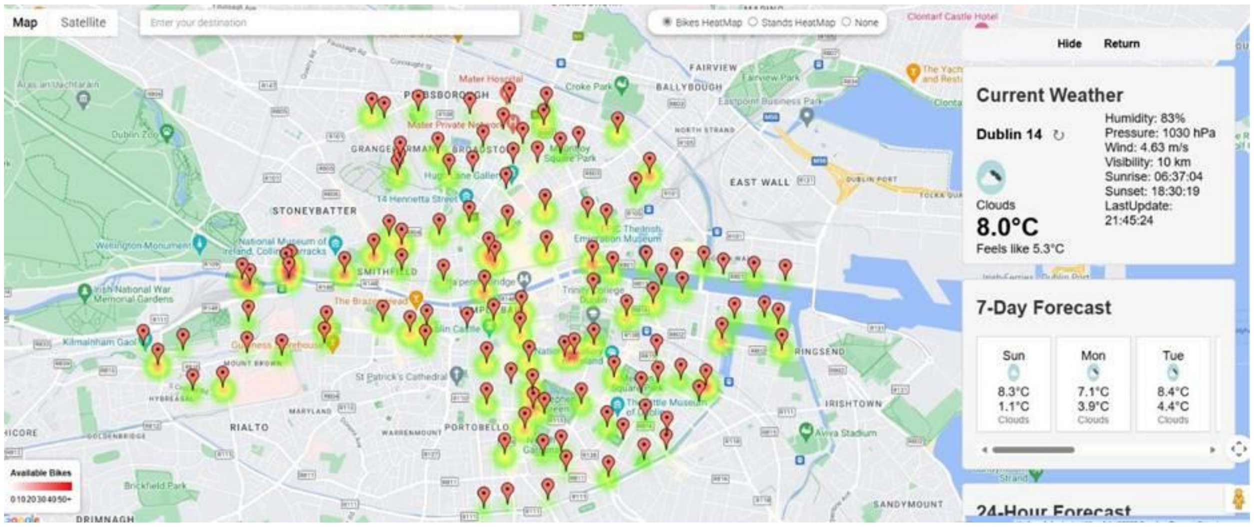


- **Real-time Bikes Information:** One of the objectives of this app is for it to provide live data on the bike and stand availability throughout all the stations in Dublin. Achieving this, the JCDecaux API and SQL database are used to fetch and store data. Queries are sent to the JCDecaux API to acquire the live feedback on the availability. The last updated time, the number of available bikes, the number of available stands, and status are all updated per station in the web application for the convenience of the user.


BLESSINGTON STREET
Directions
Available Bikes
Available Stand
Status: OPEN
Last Update: 16

GEORGES QUAY
Directions
Available Bikes: 2
Available Stands: 18
Status: OPEN
Last Update: 16:04:08

- **Real-time Weather Information Based on Your Location:** To make informed decisions about planning of the trip, it is essential for the users to be updated about the weather information. In this app, the weather information is acquired from the Open Weather API to provide the potential users with real-time updates about the weather. The key data on weather provided are temperature, humidity, pressure, wind, visibility, sunrise, sunset and the updated time. The 7-day forecast of weather is displayed as well. This is convenient for any trip planning of users.



Current Weather

Dublin 14

Clouds

8.0°C

Feels like 5.3°C

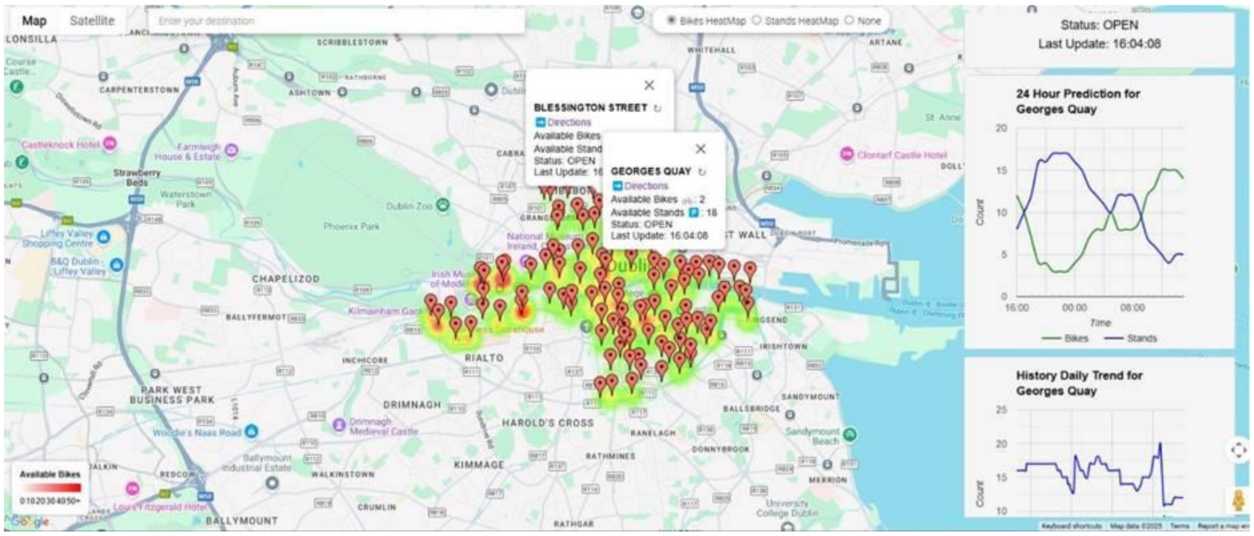
Humidity: 83%
Pressure: 1030 hPa
Wind: 4.63 m/s
Visibility: 10 km
Sunrise: 06:37:04
Sunset: 18:30:19
LastUpdate: 21:45:24

7-Day Forecast

Sun	Mon	Tue
8.3°C	7.1°C	8.4°C
1.1°C	3.9°C	4.4°C
Clouds	Clouds	Clouds

24-Hour Forecast

- **Visualized History and 24-Hour Availability Trends for Each Bike Station:** Bike trends are available to assist users with proper trip planning. In terms of less congested times of the bike stations. These are available daily for 24-hours. This helps the users to predict whether a station is more likely to be full or empty.

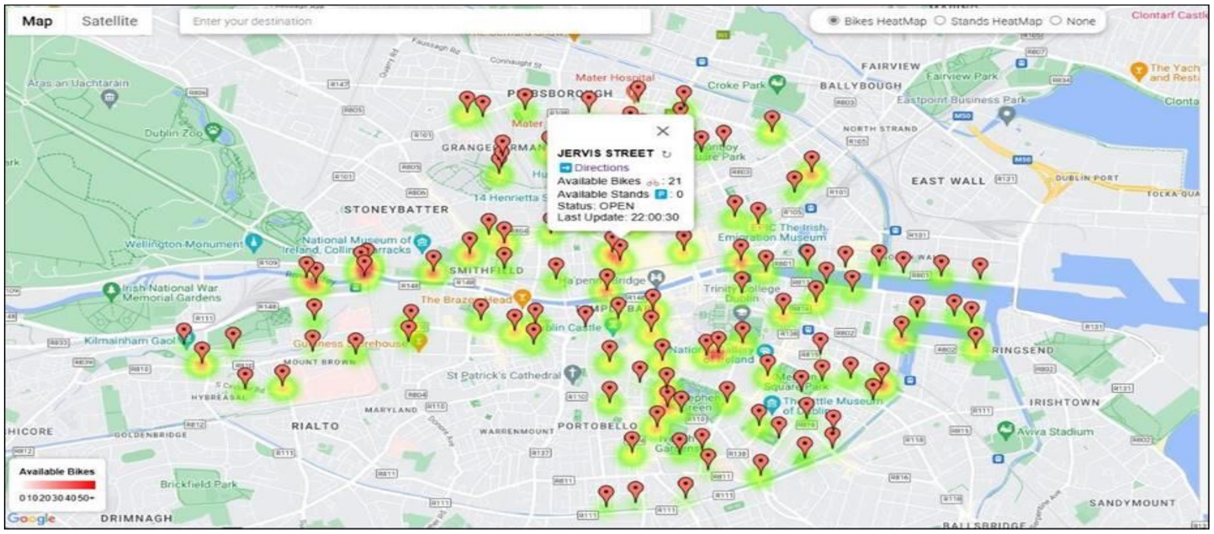


BLESSINGTON STREET
Directions
Available Bikes
Available Stand
Status: OPEN
Last Update: 16

GEORGES QUAY
Directions
Available Bikes : 2
Available Stands : 18
Status: OPEN
Last Update: 16:04:08

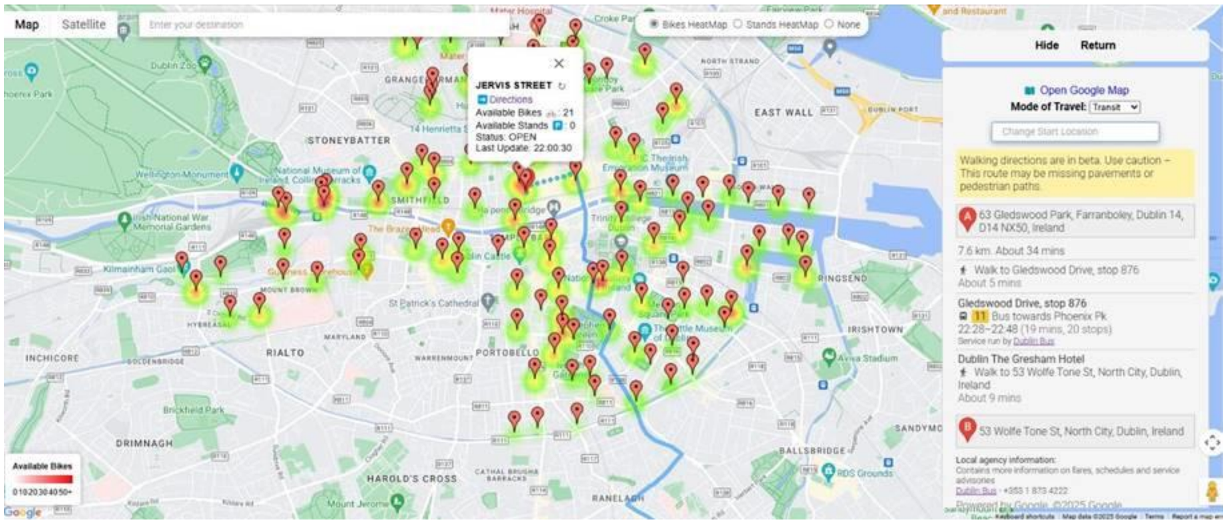
Status: OPEN
Last Update: 16:04:08

- **Visualized Availability Heat Map for Each Bike Station:** To visually capture the attention of the users, heatmaps were implemented. They are color-coded to display the high-demand and the comparatively low-demand stations among the stations.



JERVIS STREET
Directions
Available Bikes: 21
Available Stands: 0
Status: OPEN
Last Update: 22:00:30

- **Route Suggestions with Google Map Interface:** The application also helps to integrate with public transportation to plan any transit journeys as well as provide optimum routes for biking between stations and destinations.



JERVIS STREET
Directions
Available Bikes: 21
Available Stands: 0
Status: OPEN
Last Update: 22:00:30

Mode of Travel: Transit

Change Start Location

Walking directions are in beta. Use caution – This route may be missing pavements or pedestrian paths.

A 63 Gledswood Park, Farranboley, Dublin 14, D14 NX50, Ireland

7.6 km. About 34 mins

Walk to Gledswood Drive, stop 876
About 5 mins

Gledswood Drive, stop 876
11 Bus towards Phoenix Pk
22:28–22:48 (19 mins, 20 stops)
Service run by Dublin Bus

Dublin The Gresham Hotel
Walk to 53 Wolfe Tone St, North City, Dublin, Ireland
About 9 mins

B 53 Wolfe Tone St, North City, Dublin, Ireland

Local agency information:
Contains more information on fares, schedules and service advisories
Dublin Bus · +353 1 873 4222

Powered by Google, ©2025 Google

- **Every Fetched Bikes and Weather Data Stored in Database, Minimal External API Calls:** Every data fetched from external API calls is used to update the database, making sure data is reusable and minimizing external API calls. This is to further optimize the app with regards to cost and performance.

- **Automatically Update and Remove Old Database Data with Customizable Time Interval and Asynchronous Service:** This feature ensures the database to be updated and well-organized. If the data from the database is older than a customizable time interval, it will be updated from external API calls automatically, and the database will update and remove old data on background asynchronously.

- **Responsive Website suitable for different size of mobile phones, tablets and desktop computers:** This web application has been tested on different sizes of screens. It makes sure every functionality is still usable on all the common sizes of screens, from mobile phones to desktop computers.

2. Requirements

2.1. Mock-ups

Rough mock-ups of the user interface were designed to provide a visual representation of the final design. It was done using the Figma tool.

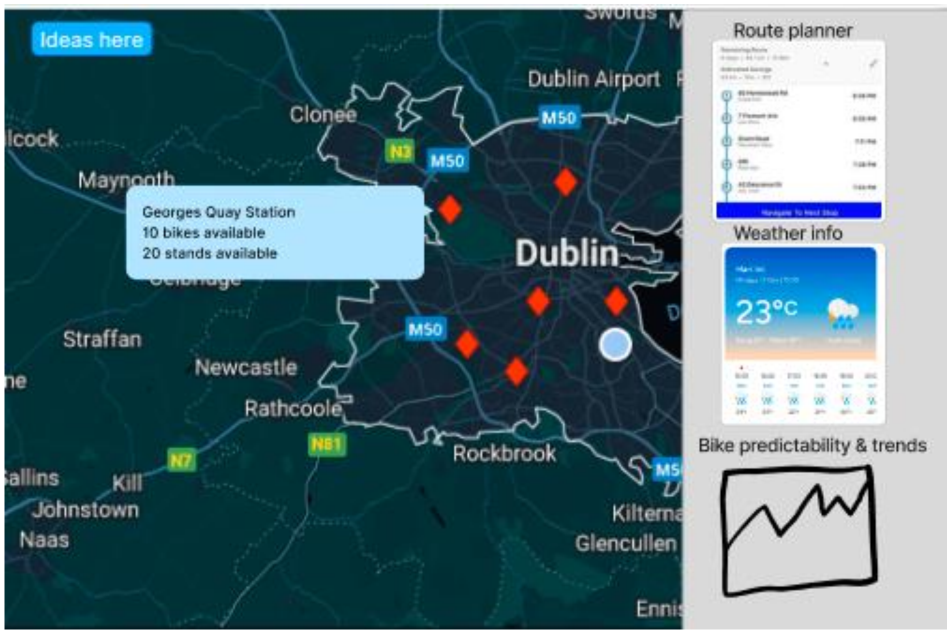


Fig 2.1: Rough mock-up of the user interface created on Figma

2.2. List of user stories:

The user stories defined what are the critical features that a user could use for when using the app. The user stories emphasizing on this product are as below:

- As a user, I would like to have access to map in which I can find the bike stations easily and can easily navigate around the area.
- As a user, I want access to the up-to-date availability of each bike station so I can plan my journeys effectively.
- As a user, I would like to see the most recent weather information for the place that I currently am to plan my ride.
- As a user, I would appreciate seeing the past trends and prediction of bike availability so that I can have more information to plan my ride.
- As a user, I would like to see the weather predictions for a few days.
- As a user, it would be great if there is a way to notice the high bike availability stations.
- As a user, I would like to get route suggestions on how to reach the bike station and plan my journey from there.
- As a user, I would like to use this app on my various devices like PCs or mobile phones.

2.2. Acceptance Criteria of the user stories

2.2.1 Interactive Google map interface

- The map should load properly on all devices
- Bike stations should be clearly indicated in the map
- Map should be able to zoom in and out

2.2.2. Real-time Bike information

- Real time information about the bikes and stand availability should be displayed
- Data should be fetched and updated automatically
- Current bikes data should be fetched when users press the refresh button.

2.2.3. Real-time weather information and prediction

- Accurate weather information depending on the user’s location is to be provided.
- Overview of weather information should be available.
- Current weather data should be fetched when users press the refresh button.
- Users should be able to see the weather prediction for the next 24 hours and for 7-days

2.2.4. Bike availability trends

- Historical data on the availability trends of the bikes
- Graph should be clear on all devices
- Users should be able to select a bike station and see the bike availability trend.

2.2.5. 24-hour Bike availability prediction

- The prediction chart for 24 hours should be available for each station.

2.2.6. Bike availability/ Stands availability Heat Map

- Heat map based on the bike availability or stand availability which is colour coded for each station for easy identification.
- The heat map needs to be real time updated to align with the current bike information.
- If the user doesn't require the visual aid, then they should be able to turn it off.

2.2.7. Route suggestions through google map

- User should be able to find their current location.
- Users should be able to feed in the current location and the end location.

2.2.8. Database storage of data and auto cleanup

- Fetched data from the API should be stored inside the database.
- API calls are to be made to update the database automatically when the data is outdated or if the user clicks the refresh button.
- Any data which is updated is replaced with the current data which was fetched from the API.
- Update and cleanup are performed asynchronously.

2.2.9. Responsive Website

- App layout should be responsive as per the screen used such as mobile phone, tablet or desktop.
- Performance should be constant in all the devices.

3. Architecture and Design

3.1. Overview of the architecture

The software architecture (Fig 2.1) of this web application is devised such as to provide the real time information of the bike/stand availability and the weather information with the help of a user-friendly portal. The user primarily interacts with the web application user interface and from that the requests are made to the backend. The web application was built using HTML, CSS and JavaScript and it comprises a Google Map interface which is provided by the Google Maps JavaScript API. For the bike data, stand data availability and weather information, the data is fetched from the Amazon RDS (Relational Database Service). If the fetched data from the RDS is old, then the API calls are made to the JCDecaux API and Open Weather API to fetch the Bike/station data and weather data. Asynchronously, this data replaces the old data available in the Amazon RDS and are sent to the web application. The users are provided with a physical refresh button, which fetch directly from the APIs. This is done instead of the RDS and updates the data in the RDS asynchronously. To offer predictive features like predicting bike availability. Machine learning models are also included in the EC2 instance.

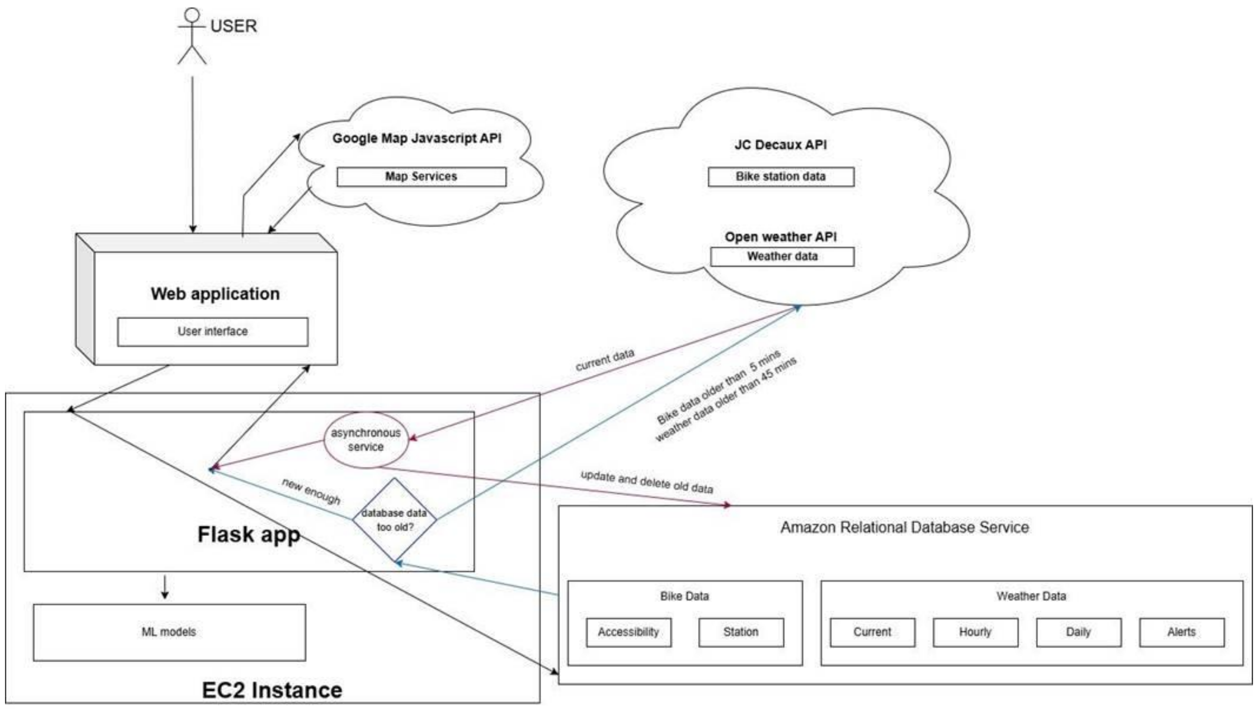


Fig 2.1: Overall Architecture of the application

3.2. Class diagram of the web application and its elements

Fig 2.2 illustrates the structured class diagram for the web application's backend architecture which was designed using the Controller-Service-Repository (CSR) pattern. The entry point for the backend of the application is the bike_share_application.py file, which initiates the system through a main(database) function. The backend has 3-minute layers which are the controllers, services, models and repositories. The controller is mainly used to handle incoming requests and contains separate modules such as the bikes_controller.py, prediction_controller.py and weather_controller.py. These are responsible for handling specific types of requests associated with the bikes, prediction and weather data. The controller then delegates the requests to the services layer, which performs the complex operations for business logic, and it contains BikesService, PredictionService, WeatherService. The repository layer is responsible for manipulating data models which is composed of various classes such as the StationRepository, AvailabilityRepository, CurrentRepository, DailyRepository, HourlyRepository, and AlertsRepository. The data model classes represent the data structure in the database. It consists of stored information such as Station, Availability, Current, Daily, Hourly and Alerts. These models contain methods like as_dict() for serialization and incorporate attributes like station location, bike availability, weather, and alarms. The backend is simpler to administer and grow, because of the layered design's support for its modularity, scaling, and concern segregation.

Fig 2.3 depicts the structured class diagram for the web application's frontend architecture which is designed to manage bike stations, routes, weather data, and trends. The system is designed modular such that each file is handling specific functionalities. The main entry point is the index.js file. It sets up the basic structure of the page, including header, main container for the map and sidebar. The map.js file initializes the map on the page, using the user's location as the centre. The route.js file’s main function is that it calculates the route from start coordinates to the end coordinates and displays it on the map. The route is displayed using the Google Maps DirectionsRenderer. A link to Google Maps is also displayed allowing the user to view the route in Google Maps. The sidebar.js is designed which shows all the necessary information such as the route planning, weather information, bike and weather predictions and so on. It also offers the users with a hide button, which is used to hide away the sidebar. Weather-related data is handled by weather.js, which retrieves current weather conditions and the biketrend.js file processes and visualizes bike trend data. The user location is managed by the userlocation.js file and predictive data is managed by the prediction.js file. All things considered, this modular design guarantees a clear division of responsibilities, which makes the program scalable and simpler to maintain. Notice that index.js also import some other modules in the codes, but only to export some functions to be global. To simplify the graph, these dependencies are not shown.

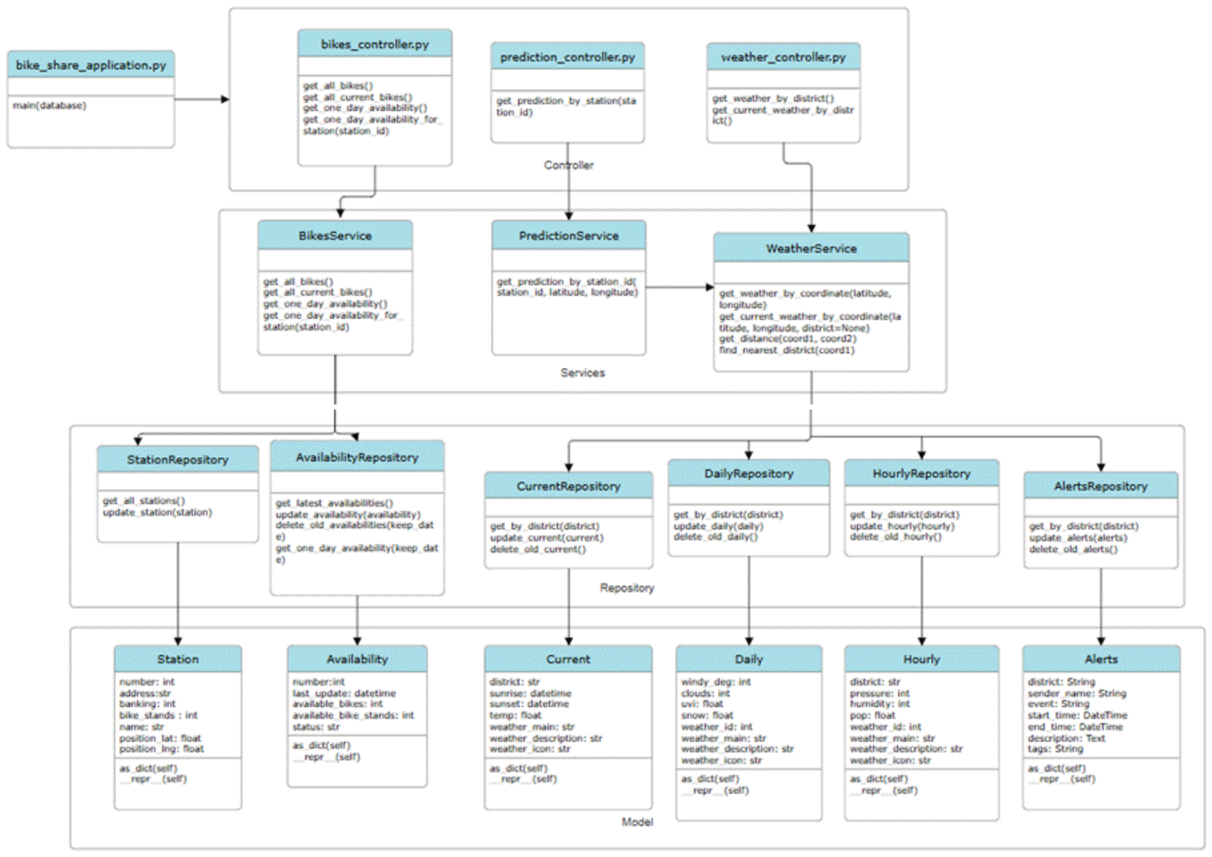


Fig 2.2: Class Diagram of the back end of the web-application and its elements

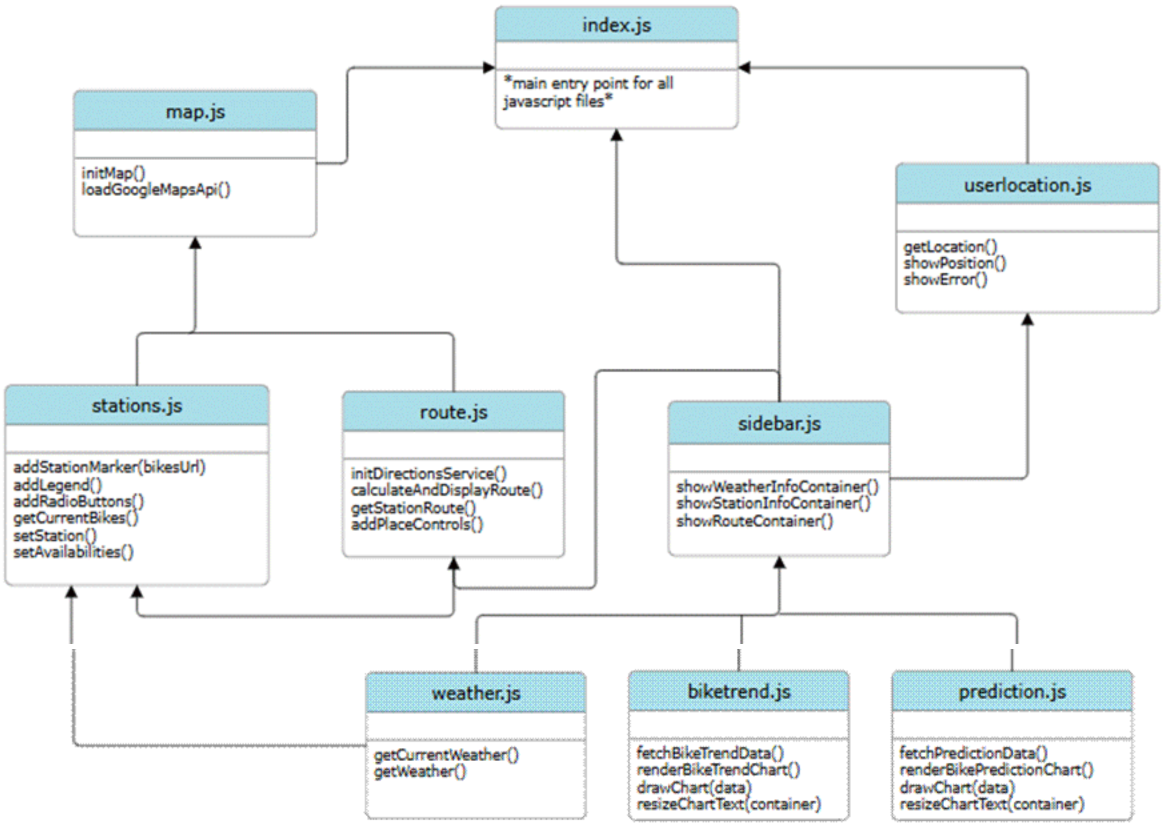


Fig 2.3: Class Diagram of the frontend of the web-application and its elements

3.3. Sequence diagram of the interactions between web application elements

Fig 2.4 depicts the sequence diagram of the interactions between web application elements which were formulated such that the application provides the bike and weather data to the user along with the predictive capabilities. The process starts with the user who requests for the services. This prompts the web app to get the map services from the Google Map JavaScript API which returns the user with the displayed map view. Simultaneously, the bike and weather data are also requested by the web app to the EC2 instance which hosts Flask app with ML (Machine learning) capacity. The Flask App retrieves the data from the Amazon RDS (Relational Database Server). If the data stored in the RDS is outdated, then the API call is initiated to the JCDecaux and the Open Weather API to get the most recent data. If the user force refreshes the data, the same process is followed. Asynchronously with this task, the RDS is updated with the new data fetched from the APIs. The new data is then displayed to the user, completing the cycle. This graphic demonstrates how user actions, backend processing, external API calls, and database updates are seamlessly integrated to provide insights that are both dynamic and predictive.

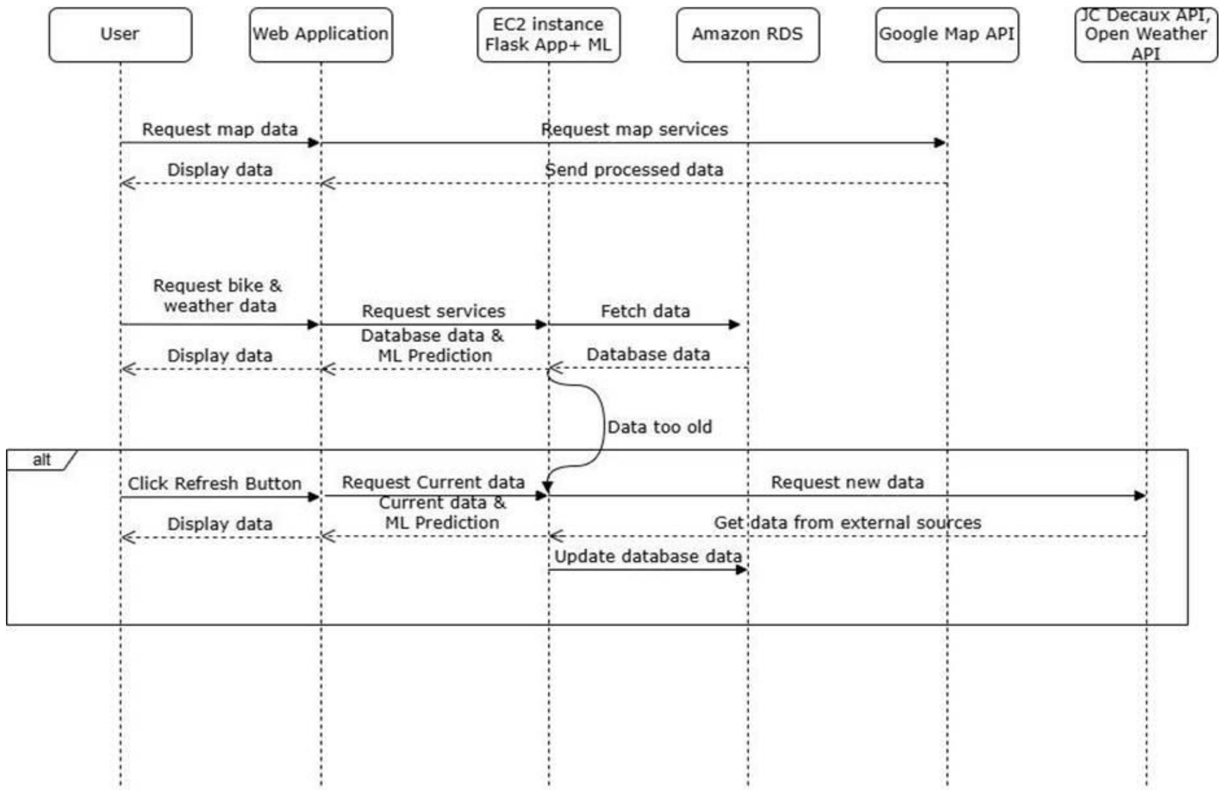


Fig 2.4: Sequence diagram of the interactions between web application elements

3.4. Technologies used

3.4.1. Frontend Technology

The frontend of this app is built on the basic web frontend-based technologies such as HTML, CSS and JavaScript. The structural foundation is built on HTML for most of the content and interface elements and CSS was used to style the elements. The dynamic and interactive aspects of this app were built using JavaScript technology. Together, all these technologies allow us to create a user-centered, end to end experience which doesn't rely on bulky frontend frameworks.

3.4.2. Backend Technology

The application's backend is coded in Python Flask, a quick and adaptable web framework that is perfect for creating web services and scalable APIs. Communication between the frontend and the backend is made through Flask, which manages the server-side logic, routes, and API endpoints. To efficiently store bike and weather data, MySQL is employed as the local relational database, guaranteeing fast access and requiring few external API calls. Utilizing an AWS EC2 instance for deployment and scalability, the backend offers a dependable cloud environment with adaptable resource management. AWS RDS (Relational Database Service) is also used for production-level managed MySQL database hosting; it provides scalability, improved security, and automated backups, making the system reliable and prepared for practical use.

4. Machine Learning Model

The machine learning model was developed for the bike sharing website serves to predict the number of available bikes and docks at each station on an hourly basis. This predictive functionality enhances user experience by enabling forward planning based on anticipated availability and supports the operational efficiency of the system by offering insight into usage trends.

4.1. Feature Selection and Preprocessing

The selection of features was guided by a combination of domain knowledge and data availability. The final input features chosen were:

- Hour of the day (Cyclical encoding)
- Temperature (°C)
- Relative humidity (%)
- Barometric pressure (hPa)
- Is weekend (Boolean flag)
- Is holiday (Boolean flag)

Hourly granularity was selected in accordance with the weather prediction data provided by OpenWeatherMap. To better capture temporal patterns, the hour feature was encoded cyclically (using sine and cosine functions), allowing the model to learn that 23:00 and 00:00 are temporally adjacent, a relationship that linear encodings cannot express. This transformation significantly improved temporal generalization.

Environmental features were estimated using statistical averages. For each hour, temperature, humidity, and pressure were calculated as the mean of maximum and minimum values, adjusted using the standard deviation. This method reduced noise and helped generate consistent hourly inputs. It allowed to contribute to more stable predictions.

Each station was treated as a separate modelling task, and separate models were trained for each one. Before training, the data for each station was pre-processed. Incomplete or invalid station datasets were excluded. For instance, stations with no available data or stations where all records indicated zero bikes and docks were not modelled. The dataset was cleaned by removing station ID and timestamp columns, and by applying cyclical encoding to the hour feature.

After pre-processing, data was organized into individual CSV files—one for each station. The dataset for each station included only the selected features and the two target variables: available_bikes and available_docks.

4.2. Model Training

The dataset was then split into training, validation, and testing sets using an 80-10-10 ratio through stratified random sampling to ensure representativeness. Numerical features were standardized using StandardScaler from the scikit-learn library to bring all inputs into the same scale, which is crucial for neural network training stability.

The machine learning model architecture was a feedforward neural network implemented using the TensorFlow Keras API. Each model was trained independently for each station to allow for station-specific trends and usage patterns. The model concluded with a linear output layer to produce two continuous outputs corresponding to available bikes and available docks. The structure of the network consisted of the following components:

- Input layer with dimensionality matching the number of features (after encoding)
- Three hidden layers with 64, 32, and 16 neurons respectively, each followed by:
    - ReLU activation
    - Batch normalization
    - Dropout (0.2 rate) for regularization
- Output layer with two neurons (corresponding to the two targets), using linear activation

The optimizer used was Adam with a learning rate of 0.001, and the loss function was Mean Squared Error (MSE), suitable for continuous regression problems. Training was performed over a maximum of 800 epochs, although an early stopping mechanism was applied to monitor validation loss and halt training when improvements plateaued. This approach improved training efficiency and prevented overfitting.

To further stabilize training, all numerical features were standardized using scikit-learn's StandardScaler, and this scaler was saved alongside each model for consistent inference during deployment. The dataset was split into three partitions:

- 80% for training
- 10% for validation
- 10% for testing

This stratified random split helped ensure that the model generalized well, and that performance metrics were not biased by particular time intervals.

4.3. Model Testing and Evaluation

After training, models were evaluated using the test subset of the data. The performance was measured using two primary metrics: Mean Absolute Error (MAE) and R^2 Score (coefficient of determination), calculated separately for each target variable. These metrics provided a clear understanding of how well the model predicted real-world availability conditions.

- MAE was used to assess the average absolute difference between predicted and actual values, offering an intuitive sense of prediction error in terms of bike or dock counts.
- R^2 Score measured the proportion of variance explained by the model, indicating how well the model captured the underlying data distribution.

Evaluation was carried out independently for each station model. It was observed that stations with higher traffic and more diverse data (e.g., central city locations) yielded better predictive performance, while outlier stations with limited or noisy data displayed lower accuracy. Stations with entirely zero availability data or corrupted records were excluded from evaluation. For instance, station 46 and station 81 had no usable records. While station 51 and station 70's data contained 2173 and 59 entries, they are all filled with zero values, making them unsuitable for learning.

Visualizations were generated to compare actual vs. predicted availability for randomly selected test periods. These plots offered an interpretable way to assess performance and revealed that the model was generally able to follow real-time trends and respond appropriately to time-of-day and environmental changes.

To evaluate the model's predictive performance, forecasts for both available bikes and available docks were generated for each station. Here, we only show the evaluation graph for Station 1 over a randomly selected 250-hour period. As illustrated in Figures 1 and 2, the model's predictions closely follow the actual values in both cases. For available bikes, the model achieved a Mean Absolute Error (MAE) of 2.12 and an R^2 score of 0.88, indicating strong alignment between predicted and real-time availability. Similarly, the forecast for available docks yielded an MAE of 2.11 and an R^2 score of 0.88, reflecting consistent performance across both target variables. These results demonstrate that the trained model is effective at capturing temporal patterns in bike station usage and can reliably generalize to unseen data, making it a practical tool for supporting real-time decision-making and resource allocation within the system. Evaluation results for other stations can be accessed in the model_results.txt on GitHub. The average MAE of bikes and docks for all models are 1.64 and 1.62. The average R^2 score of bikes and docks for all models are 0.89 and 0.89.

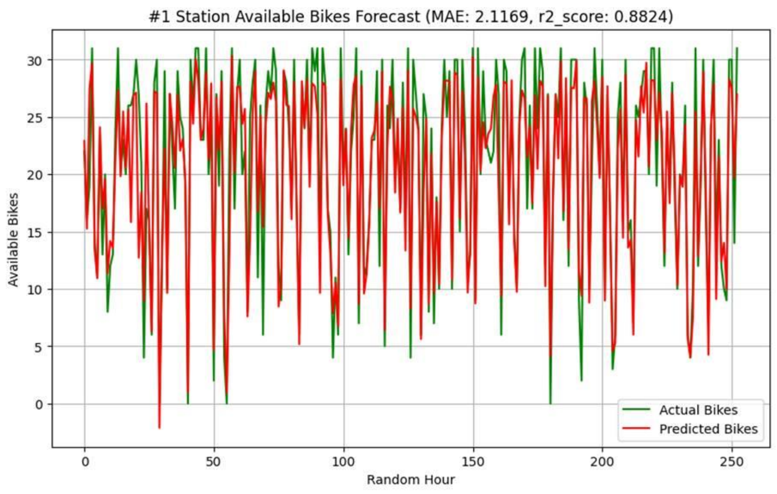


Figure 4.1: Predicted vs. Actual Available Bikes graph with caption

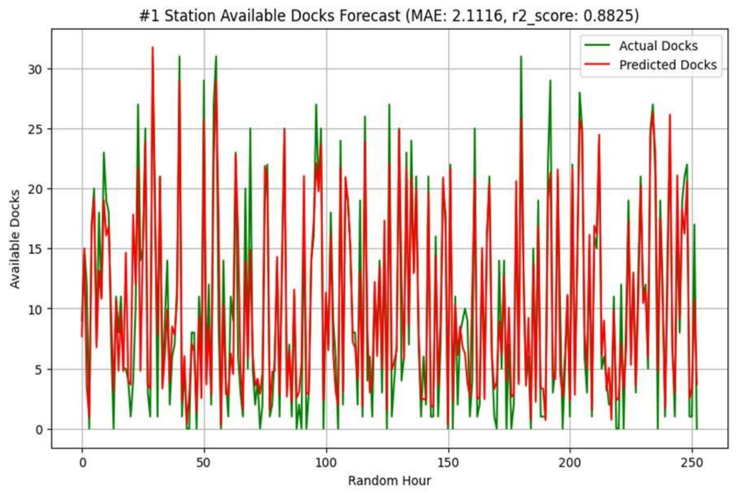


Figure 4.2: Predicted vs. Actual Available Docks graph with caption

4.4. Machine Learning Conclusion

The machine learning component of the bike sharing website played a central role in forecasting hourly availability. Through careful feature selection, preprocessing, and model tuning, reliable predictive models were built for individual stations. The integration of environmental and temporal data allowed for context-aware forecasting, and rigorous evaluation ensured that predictions were both accurate and interpretable. Overall, the approach successfully enhanced the intelligence and responsiveness of the bike sharing system.

5. Testing

5.1. Overview

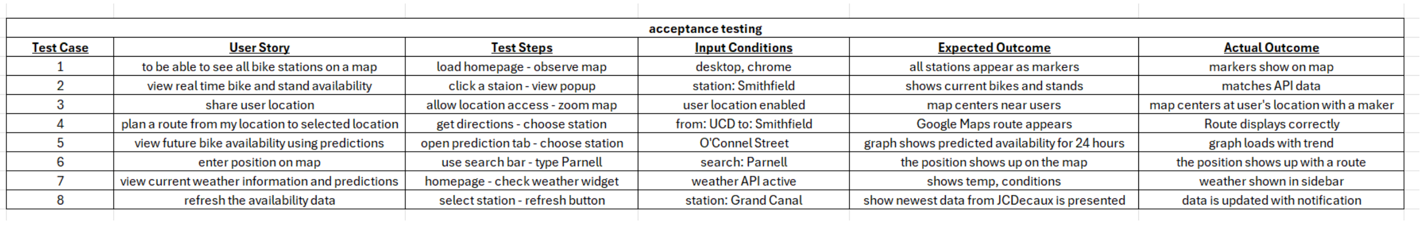
To ensure the quality, reliability, and usability of the webpage, a comprehensive testing process was implemented. The testing strategy encompassed both:

- Acceptance testing: used to validate that functional requirements were successfully implemented.
- Usability testing, which evaluated how real users interact with the system and how intuitive the interface is.

5.2. Acceptance testing

Acceptance testing was conducted to validate whether the features of the Dublin Bikes webpage met the functional requirements and user expectations, as defined by project's user stories. A total of eight core user stories were selected for this phase, each representing an essential part of the user journey—from loading the map to viewing live data, predictions, and navigation.

Each user story was translated into a test case with defined steps, input conditions, and expected outcomes. The actual application behaviour was compared to the expected outcomes. A pass was recorded if all acceptance criteria were met without issues. All eight test cases passed successfully. The following table summarized the test cases, conditions and outcomes:



acceptance testing					
Test Case	**User Story**	**Test Steps**	**Input Conditions**	**Expected Outcome**	**Actual Outcome**
1	to be able to see all bike stations on a map	load homepage - observe map	desktop, chrome	all stations appear as markers	markers show on map
2	view real time bike and stand availability	click a staion - view popup	station: Smithfield	shows current bikes and stands	matches API data
3	share user location	allow location access - zoom map	user location enabled	map centers near users	map centers at user's location with a maker
4	plan a route from my location to selected location	get directions - choose station	from: UCD to: Smithfield	Google Maps route appears	Route displays correctly
5	view future bike availability using predictions	open prediction tab - choose station	O'Connel Street	graph shows predicted availability for 24 hours	graph loads with trend
6	enter position on map	use search bar - type Parnell	search: Parnell	the position shows up on the map	the position shows up with a route
7	view current weather information and predictions	homepage - check weather widget	weather API active	shows temp, conditions	weather shown in sidebar
8	refresh the availability data	select station - refresh button	station: Grand Canal	show newest data from JCDecaux is presented	data is updated with notification

Fig 5.1: Acceptance Testing Results

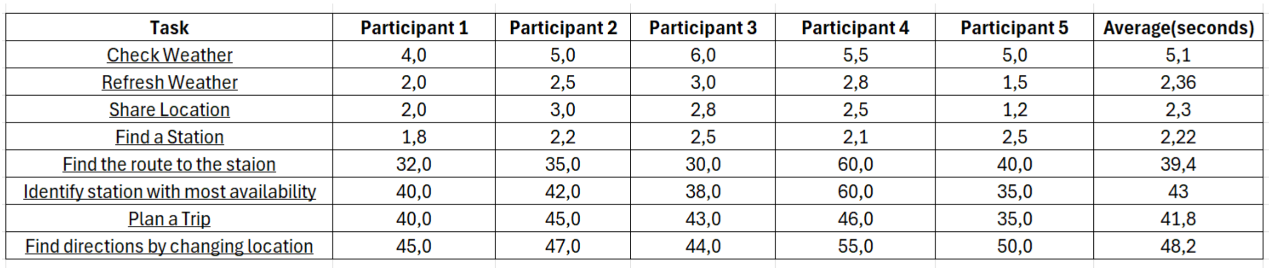
The following summarizes key findings for each area:

- Map and Station Loading: users were able to view all bike stations clearly on the map upon loading the homepage. The station markers appeared consistently across different devices. This confirmed that the initial map loading functionality was working as expected and integrating correctly with the station data.
- Real-Time Availability: clicking on a station brought up an info window with live availability of bikes and stands. Updates occurred automatically at the defined refresh interval, confirming both accuracy and reliability of the real-time data system.
- Location Detection: when users allowed location access, the map automatically centred around their current position. This feature worked smoothly and significantly enhanced the user experience by helping users identify nearby stations.
- Route Planning: using the get directions feature, users could select a starting location and a bike station to generate a route they will have to take. This functionality helped users easily navigate to their desired station and was essential for first-time users unfamiliar with the area.
- Prediction Feature: the prediction tab allowed users to select a station and view anticipated availability at any time. During testing, predicted trends aligned with expected usage behaviours – for example, higher demand during morning commute hours and lower demand in the afternoon.
- Position Search: this feature allowed users to type a location name and quickly locate it on the map. Upon selection, the map cantered on the location with a route to it, and the info window appeared automatically.
- Weather Widget: the real-time weather widget was successfully integrated and displayed relevant data such as temperature, and conditions. This provided context to users, especially those checking availability in poor weather conditions.
- Refresh Button Behaviour: testing confirmed that when a user selected a station and clicked the refresh button, the current data is updated and shown appropriately with a notification.

The acceptance testing phase confirmed that the core features of the webpage aligned with the initial user stories and project requirements. The systematic approach ensured that each function was tested in isolation and in context, providing confidence in the stability of the system for end users. No critical bugs were found during this phase, and all cases passed.

5.3. Usability testing

To evaluate the user-friendliness of the webpage, the usability testing was conducted with five participants simulating first-time users. This type of testing provided insight into the user experience, identifying any areas of confusion, or inefficiency during normal site interaction. Each participant was given a set of eight tasks. After completing these tasks, a survey was filled out by each participant. This combination of task performance metrics and qualitative user feedback provided insights into both the interface design and the user experience. The table below shows the average time in seconds that participants took to complete each of the eight tested tasks:

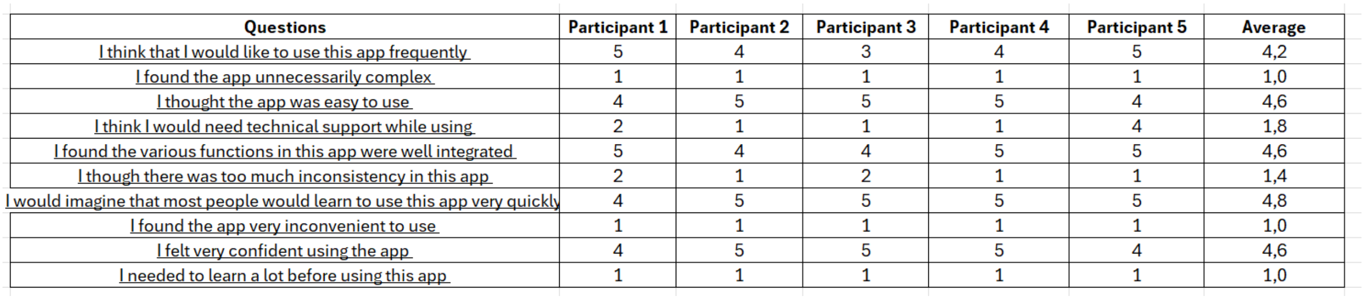


Task	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Average(seconds)
Check Weather	4,0	5,0	6,0	5,5	5,0	5,1
Refresh Weather	2,0	2,5	3,0	2,8	1,5	2,36
Share Location	2,0	3,0	2,8	2,5	1,2	2,3
Find a Station	1,8	2,2	2,5	2,1	2,5	2,22
Find the route to the staion	32,0	35,0	30,0	60,0	40,0	39,4
Identify station with most availability	40,0	42,0	38,0	60,0	35,0	43
Plan a Trip	40,0	45,0	43,0	46,0	35,0	41,8
Find directions by changing location	45,0	47,0	44,0	55,0	50,0	48,2

Figure 5.2: Usability Testing Results in Seconds

- Fast Completion Tasks (< 5 seconds): tasks such as checking weather, refreshing weather data, sharing location, and finding a station had very short completion times. This indicates that the interface for these features is intuitive, well-placed, and quickly understood even by first-time users. These tasks were likely completed with just one or two clicks, this could suggest effective visual flow.
- Slower Completion Tasks (> 40 seconds): more complex tasks, such as finding routes, identifying availability, and planning trips took considerably longer to complete. These tasks may require multi-step interactions, which could require more guidance.

Some gaps presented could suggest that new users might complete simple tasks easily but may initially struggle with advanced features until they become more familiar. The following results summarize average participant responses to 10 usability-focused survey statements, rated from 1 (Strongly Disagree) to 5 (Strongly Agree):



Questions	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Average
I think that I would like to use this app frequently	5	4	3	4	5	4,2
I found the app unnecessarily complex	1	1	1	1	1	1,0
I thought the app was easy to use	4	5	5	5	4	4,6
I think I would need technical support while using	2	1	1	1	4	1,8
I found the various functions in this app were well integrated	5	4	4	5	5	4,6
I though there was too much inconsistency in this app	2	1	2	1	1	1,4
I would imagine that most people would learn to use this app very quickly	4	5	5	5	5	4,8
I found the app very inconvenient to use	1	1	1	1	1	1,0
I felt very confident using the app	4	5	5	5	4	4,6
I needed to learn a lot before using this app	1	1	1	1	1	1,0

Figure 5.3: Usability Testing Survey Results

The participants' contact information have been uploaded to GitHub

- Ease of Use and Learnability: participants gave high scores to ease of use and learnability, strongly indicating that the site is accessible even for first-time users. The low scores for needing support and needing to learn a lot further support this.
- Simplicity and Integration: responses to questions about complexity, inconsistency, and inconvenience were very low, which supports the conclusion that the interface is both cohesive and simple to navigate. Participants recognized that different features were well integrated into a consistent experience.
- User Confidence and Frequency of Use: users felt confident using the app and were likely to use it regularly. These are strong indicators of user satisfaction, meaning the app could serve real users in everyday scenarios without the need for training or further technical support.

Using the standardized System Usability Scale (SUS) scoring method based on these responses, the overall usability score would land between 85–90/100, which is considered "Excellent" by usability standards. Overall, users completed the tasks with minimal guidance and demonstrated a good understanding of the interface, suggesting the design was intuitive and accessible.

5.4. Future testing

Testing played a crucial role in shaping the final version of the webpage. Acceptance testing confirmed that functional requirements were implemented correctly, while usability testing helped polish the user experience. In future iterations, the following improvements could be recommended for testing process:

- Broader usability testing: Involving more participants with varied technical backgrounds to gain diverse feedback.
- Mobile testing: Ensuring full responsiveness and functionality on different mobile devices and screen sizes.
- Automated testing: Implementing automated unit and integration tests for critical functions such as API calls and map rendering.

6. Process

6.1. Overview

The development of the Dublin Bikes webpage followed an iterative process throughout the semester. The approach was centred around Agile methodologies, allowing for flexibility and continuous improvements. Initial planning focused on defining core features, including bike availability tracking, weather information, and route mapping using Google Maps API. The project evolved through regular check-ins, feedback, and the implementation of new functionalities, ensuring the webpage met the outlined objectives. These objectives were proposed in the first meeting of the team.

As the semester progressed, the project evolved through a series of sprints, with each sprint focusing on implementing specific features, refining user experience, and testing functionality. Feedback, which was given between each other, was gathered after each iteration, allowing the development team to adapt and make necessary adjustments to enhance the overall product. This cyclical process of design, feedback, implementation, and testing ensured that the project aligned with the initial goals and responded to user needs effectively.

By using Agile principles, the project was kept flexible and responsive to both technical challenges and evolving requirements, ultimately enabling the team to deliver a high-quality and functional bike-sharing webpage.

6.2. Communication methods

Communication played a crucial role in ensuring the project remained on track and meeting deadlines. There were 2 primary methods of communication utilized throughout the project.

- WhatsApp, this was employed for immediate, real-time communication with facilitation for quick problem resolution and updates. This instant communication tool helped keep the team connected, especially when dealing with smaller tasks or clarifying technical skills.
- Weekly in-person meetings were held every Tuesday to review progress, address challenges and synchronize work. Every 2 weeks, the sprint would have finished, and new steps were created. These meetings ensured that the entire team was aligned and allowed for more detailed discussions.

6.3. Scrum

This project was managed by a small team of 3 who shared different responsibilities. The core responsibilities were collaboratively handled by all members.

- Shared Scrum Master duties: the main responsibilities were to facilitate the sprint planning, updates of the sprints and product backlogs, and update the user stories accordingly.
- Shared Product ownership: this was conducted to make everyone participate in discussion. Members agreed on the priorities of the tasks, defined the tasks in the product backlog and refined the features.
- Shared development team: the main task was to always split tasks according to personal experiences and knowledge of software necessities.
- Sprint retrospective: at the end of each sprint, the team gathered to provide feedback to the Scrum Master on what went well and what could’ve been improved. This feedback was necessary action plans for the next sprint.

6.4. Sprint Reviews

The project was structured into a series of two-week sprints, following Agile principles to ensure steady progress. At the start of each sprint, a sprint planning session was held to define the tasks and features to be accomplished within the given timeframe. These tasks were prioritized based on their importance to the project's goals, ensuring that critical functionalities were addressed first.

During the sprint, team members worked on their assigned tasks, which included implementing new features, debugging existing code, and refining the user interface. Daily standups were held to provide quick updates on individual progress, identify potential issues, and maintain focus on the sprint objectives. This helped ensure that the project stayed on track and that everyone was aware of what was being worked on at any given time. At the end of each sprint, a sprint review was conducted, where completed tasks were demonstrated and evaluated. Feedback was collected from team members, and any necessary adjustments were made before moving on to the next sprint. Retrospectives were also held after each sprint to reflect on what worked well and what could be improved in the next iteration. This continuous feedback loop contributed to the iterative improvement of the project and helped the team stay adaptable in the face of challenges.

6.4.1. Feb 4 – Feb 18

The first sprint focused on establishing the foundation for the project. The team began by identifying all relevant user stories to guide feature development. Required software tools and dependencies were installed, and initial integration with third-party APIs was carried out. Specifically:

- Defined and listed all user stories
- Downloaded and installed necessary software tools
- Implemented API requests for weather data and JCDecaux bike data
- Established a database and configured data storage for incoming API data

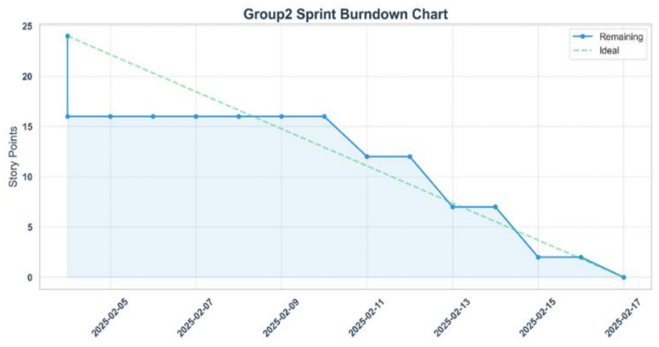


Fig 6.1: Burndown chart Sprint 1

6.4.2. Feb 18 – Mar 4

The second sprint emphasized building the first functional version of the webpage and integrating key APIs. Backend and frontend components were connected to display bike and weather data in real-time. The website began to take shape, and multiple layers of testing were introduced to ensure basic stability:

- Built the initial version of the website
- Linked Google Maps API to the frontend
- Implemented and tested API requests for bike and weather data
- Stored API responses in the database
- Created backend APIs to retrieve bike and weather data from the database
- Connected frontend requests to the backend APIs and displayed the data
- Performed functional testing of the website

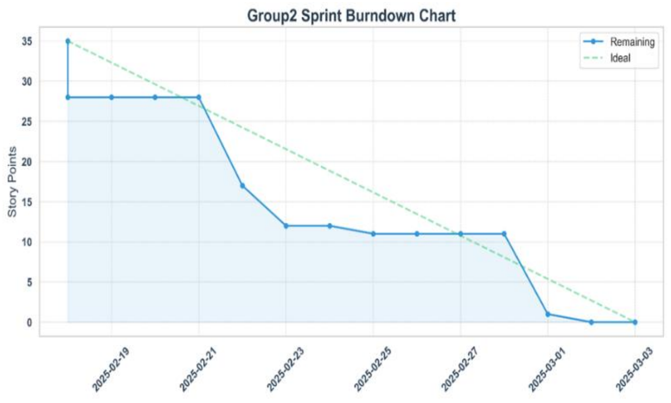


Fig 6.2: Burndown chart Sprint 2

6.4.3. Mar 4 – Apr 1

Third sprint introduced advanced functionality and significantly improved the user experience. Dynamic components were added to the website, and the project was deployed to an Amazon EC2 server for real-time access and testing. Map features were enhanced to support user location and routing:

- Added a refresh button for real-time data updates
- Created backend endpoints and frontend links for fetching and displaying live data
- Deployed the full application on an EC2 server
- Prompted users to share location and displayed a marker accordingly
- Implemented weather data retrieval based on shared location
- Added route calculation for the most efficient path
- Modified the Google Maps interface to better accommodate route display and UI (User Interface) elements.

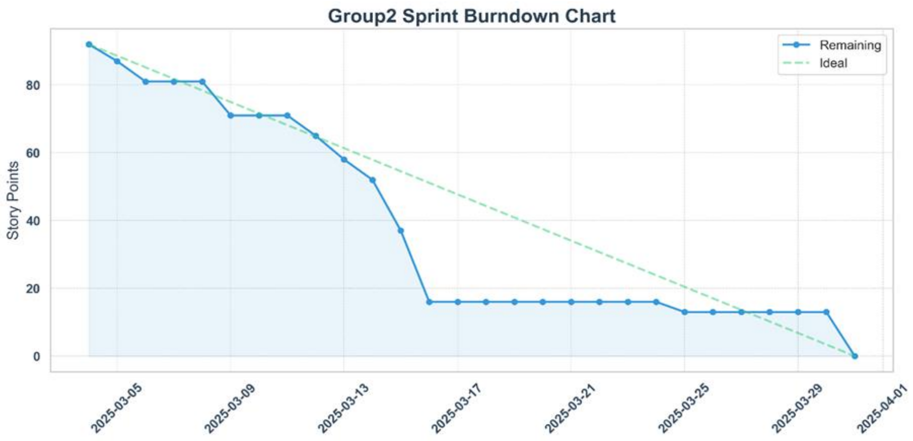


Fig 6.3: Burndown chart Sprint 3

6.4.5. Apr 1 – Apr 15

The final sprint was dedicated to integrating the machine learning model, refining the website, and preparing deliverables such as the project report and additional pages. The model was trained to predict bike availability, and UI responsiveness was improved for a more user-friendly experience:

- Developed, trained, and tested a machine learning model for bike availability prediction
- Integrated the prediction model with the main application
- Added detailed annotations to functions for improved code readability
- Made the website responsive across devices
- Finalized and compiled the project report

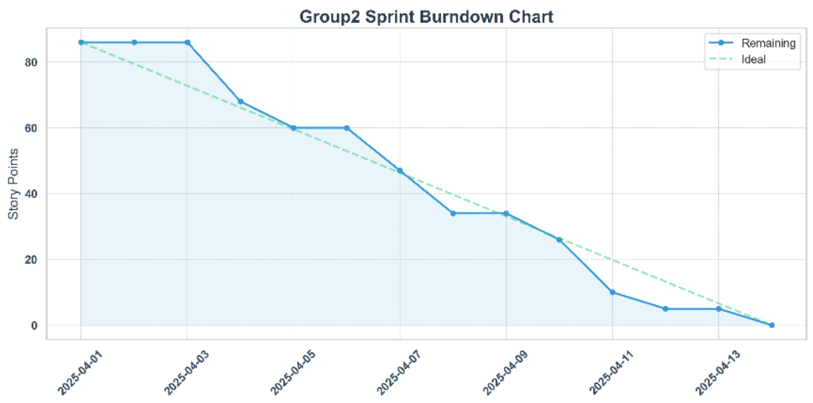


Fig 6.4: Burndown chart Sprint 4

6.5. Sprint Retrospective

6.5.1. Sprint 1

- What Went Well: The team successfully set up all necessary tools, including the database, GitHub, AWS EC2, and RDS. User stories were effectively discussed and aligned with project goals.
- What Didn't Go Well: There was some confusion about sprint requirements and unexpected RDS costs.
- Action Plan: Improve sprint documentation, organize the product backlog, and enhance software proficiency.

6.5.2. Sprint 2

- What Went Well: Backend was completed with minimal bugs, and Flask-SQL Alchemy helped streamline work. Git usage improved communication. The front end with Google Maps and new features was implemented successfully.
- What Didn't Go Well: Time constraints due to academic workload limited progress.
- Action Plan: Maintain strong team communication, set realistic sprint goals, and align workload with other commitments.

6.5.3. Sprint 3

- What Went Well: Additional time during holidays allowed more user stories to be added, early report progress, and efficient task completion.
- What Didn't Go Well: Some tasks were difficult due to limited experience, but overall progress was steady.
- Action Plan: Finish remaining stories, refine the report, and enhance the website design.

6.5.4. Sprint 4

- What Went Well: All user stories were completed, and the full project report was finalized.
- What Didn't Go Well: Limited time in the final sprint caused rushed task completion and restricted testing due to late instruction.
- Action Plan: Finalize the report and create the application video.

6.6. Tools used

A variety of tools and technologies were utilized throughout the development of the Dublin Bikes webpage to streamline development, enhance collaboration, and ensure the successful of the project.

- GitHub: it was used as a primary version control system for sharing and reviewing the code. Each team member worked on a separate feature branch, ensuring that their individual work did not interfere with other's contributions, this process helped maintain code quality. GitHub provided the platform for hosting the repository, managing pull requests, and conducting code reviews.
- Visual Studio code: the primary integrated development environment (IDE) used for writing the webpage code was Visual Studio Code. Its extensive feature set, including syntax highlighting, IntelliSense, and Git integration, made it an ideal tool for coding in JavaScript, HTML, CSS, and Python (for the Flask backend).

- Google Maps API: one of the key features of the Dublin Bikes webpage was the integration of real-time bike availability and route mapping. The Google Maps API provided the necessary functionality for displaying bike stations and routes, as well as mapping out the most efficient paths for users.
- Amazon EC2: it was used to deploy the backend of the Dublin Bikes webpage. As a scalable cloud service, EC2 provided the necessary computing power to host the Flask application and handle incoming traffic. By using EC2, the team ensured that the server could be easily scaled up or down based on traffic demands, improving reliability and performance throughout the project.

- Amazon RDS: The team utilized Amazon RDS to create, manage, and scale a MySQL database in the cloud, providing a secure and reliable storage solution for bikes and weather data. By storing historical data in database, the application can quickly access and reuse the data, reducing the need for external API calls and improving overall performance.

7. Conclusion

The development of the web application provided valuable insights into both technical implementation and collaborative project management. Working in sprints enabled the team to effectively divide tasks, respond to challenges, and incrementally build a user-oriented solution. Despite occasional difficulties such as limited time and unfamiliar technologies, steady progress was achieved through structured planning and continuous improvement. The project enhanced skills in backend and frontend development, as well as in communication, coordination, and workflow organization. Overall, the process resulted in a functional and refined application that aligns with the original goals.