Table of Contents

[Section 1: Planning 2](#_Toc193380231)

[Proposal: Business Case & Scope 2](#_Toc193380232)

[Context Diagram: 3](#_Toc193380233)

[Section 2: Requirements 4](#_Toc193380234)

[UML Use Case Diagram: 4](#_Toc193380235)

[Software Requirements Specification (SRS): 4](#_Toc193380236)

[1. Introduction: 4](#_Toc193380237)

[2. Functional Requirements: 5](#_Toc193380238)

[3. Non-Functional Requirements: 5](#_Toc193380239)

[4. System Constraints: 6](#_Toc193380240)

[5. Assumptions & Dependencies: 6](#_Toc193380241)

[6. User Stories: 6](#_Toc193380242)

[7. Requirements Traceability Matrix: 6](#_Toc193380243)

[Section 3: Design 7](#_Toc193380244)

[1. User Interface (UI) Design Overview 7](#_Toc193380245)

[2. Wireframe and Wireflow Diagrams 7](#_Toc193380246)

[3. High-Fidelity Mockup 8](#_Toc193380247)

[4. Accessibility Considerations 9](#_Toc193380248)

[5. User Interaction Flow 9](#_Toc193380249)

[6. Design Justification 10](#_Toc193380250)

[Section 4: Implementation 10](#_Toc193380251)

[UML Component Diagram 10](#_Toc193380252)

[Screenshots and User Guide 12](#_Toc193380253)

[Section 5: Testing 16](#_Toc193380254)

[Test Plan 16](#_Toc193380255)

[Test Cases 16](#_Toc193380256)

[Section 6: Individual Contribution 18](#_Toc193380257)

[Requirements 19](#_Toc193380258)

[Implementation 20](#_Toc193380259)

[UML Class Diagram Representing JSON Query Results 20](#_Toc193380260)

[Code Quality and Comments 21](#_Toc193380261)

[Manual Testing 21](#_Toc193380262)

# Tables and Screenshots

[Table 1: Functional Requirements 6](#_Toc193380296)

[Table 2: Non-Functional Requirements 6](#_Toc193380297)

[Table 3: User Stories 7](#_Toc193380298)

[Table 4: Requirements Traceability Matrix 8](#_Toc193380299)

[Table 5: Test Cases 18](#_Toc193380300)

[Table 6: Requirements Traceability Matrix 19](#_Toc193380301)

[Table 7: Use-Case Model 20](#_Toc193380302)

[Table 8: Requirements Traceability Matrix 23](#_Toc193380303)

[Screenshot 1: Homepage 13](#_Toc193380347)

[Screenshot 2: Homepage on Scroll 14](#_Toc193380348)

[Screenshot 3: Search Functionality in Action 15](#_Toc193380349)

[ScreenShot 4: main.js code overview 22](#_Toc193380350)

# Section 1: Planning

## Proposal: Business Case & Scope

**Business Case:**

Parking management has become a significant issue in the modern urban environment. Such cities as Bristol face daily traffic jams due to the increased number of vehicles and scarcity of parking lots. It is not uncommon for drivers to spend a considerable amount of time searching for parking spaces, which results in traffic congestion, pollution, and customer dissatisfaction. In order to meet these challenges, we recommend the development of a Bristol Car Park Finder Web Application.

This web-based system pulls data from Bristol's open datasets in real time. It presents it in a map and table format where users can search for car parks by location, availability, operators, occupancy, and other features. It enables users to search for car parks by name, identify their location on the map, and view the information in a tabular form.

The business benefits include:

* Less traffic congestion - Identification of available car parks reduces the time spent in circulation.
* Better user experience - Search and visualization make it easier for users to find parking spaces.
* Environmental benefits - Lowered emissions through reduced search times.
* Public data utilization - Efficient usage of existing city data assets.
* Scalability - The ability to expand to other cities or link with payment services in the future.

**Scope:**

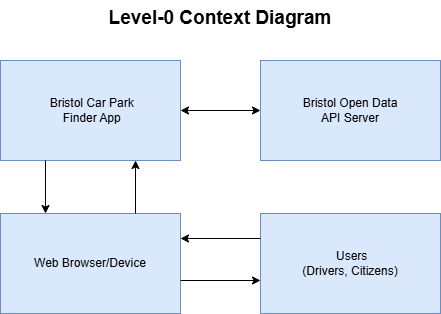
The project scope includes:

* Integration with Bristol's public car park API.
* Interactive map using the Leaflet.js library, showing all the car parks available.
* A search bar that allows users to filter car parks by name and automatically updates the map markers and table rows.
* Tabular data representation of the detailed features, including operator, number of spaces, availability of CCTV, etc.
* The interface design is friendly to both the desktop and mobile users.
* Business case, system diagrams, design, implementation, and testing documentation.

**Out of Scope:**

* Payment or reservation features.
* Mobile app development (web app only).
* Private parking data integration.

## Context Diagram:



Explanation:

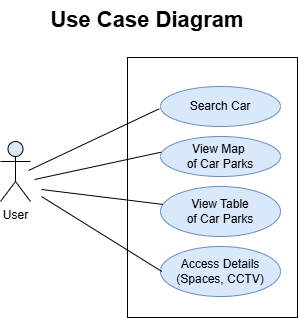
The Car Park Finder App is a web application that users (drivers) access through their browsers.

The Bristol Open Data API Server uses the app to obtain the most current parking information.

The data is presented to the users in two formats: as markers on the map and as a data table that can be searched.

# ****Section 2: Requirements****

## ****UML Use Case Diagram:****



Actors:

* User (Primary Actor) - Any person using the web application to search for information about car parks.

Use Cases:

* Search Car Parks - This feature allows users to search for a specific car park by entering the name of the car park.
* Map of Car Parks - This feature allows users to view all the car parks on a map.
* Table of Car Parks - This feature allows users to view more specific information, such as the operator, the number of floors, and the current occupancy.
* Get More Information - When a user clicks on a car park, they can see more information, such as whether it has CCTV, the time it opens and closes, and occupancy rates.

## ****Software Requirements Specification (SRS):****

### ****1. Introduction:****

**1.1 Purpose:**

The goal of the Car Park Finder Web Application is to offer up-to-date information on car parks in Bristol for both residents and tourists. The system pulls data from the Bristol Open Data API and provides a map-based interface with search functionality, allowing users to locate available parking spaces quickly.

**1.2 Intended Audience:**

* The general public (drivers, residents, visitors)
* City council (as a proof of public data utilization)
* Developers and maintainers (for future enhancements)

### ****2. Functional Requirements:****

Table 1: Functional Requirements

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| FR1 | The system shall retrieve car park data from the Bristol Open Data API. |
| FR2 | The system shall display car park locations on an interactive map. |
| FR3 | The system shall present car park details in a tabular format. |
| FR4 | The system shall allow users to search car parks by name |
| FR5 | The system shall dynamically update both map and table based on search input. |
| FR6 | The system shall allow users to click on map markers for detailed info pop-ups. |
| FR7 | The system shall highlight corresponding car parks in the map and table after search. |

### ****3. Non-Functional Requirements:****

Table 2: Non-Functional Requirements

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| NFR1 | The application shall be responsive, ensuring usability across desktops and mobiles. |
| NFR2 | The application shall use valid HTML5, CSS, and JavaScript standards. |
| NFR3 | Data fetching and rendering operations shall be optimized for minimal load times. |
| NFR4 | The system shall be designed to support scalability, allowing future integration with additional datasets. |
| NFR5 | The application shall maintain accessibility standards for all user demographics. |

### ****4. System Constraints:****

* Data will only be collected from the Bristol Open Data API endpoint.
* No private data integration will be allowed; only public datasets will be considered.
* Payment, reservation, or booking features are not allowed.

### ****5. Assumptions & Dependencies:****

* It is assumed that the Bristol Open Data API is accessible and is working as expected.
* An Internet connection is needed to use the app.
* Users can interact with the app through web browsers that meet HTML5, CSS3, and JavaScript ES6+ standards.

### ****6. User Stories:****

Table 3: User Stories

|  |  |
| --- | --- |
| **ID** | **User Story** |
| US1 | As a driver, I want to search for a car park by name so I can quickly locate it. |
| US2 | As a user, I want to view car parks on a map so that I can understand their locations. |
| US3 | As a user, I want to see detailed information about each car park (operator, spaces, CCTV) to make informed decisions. |

### ****7. Requirements Traceability Matrix:****

Table 4: Requirements Traceability Matrix

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Use Case** | **User Story** |
| FR1 | View Map, View Table | US2 |
| FR2 | View Map | US2 |
| FR3 | View Table | US3 |
| FR4 | Search Car Parks | US1 |
| FR5 | Search Car Parks, View Map, View Table | US1, US2 |
| FR6 | Access Details | US3 |
| FR7 | Search Car Parks, View Map, View Table | US1, US2 |

# Section 3: Design

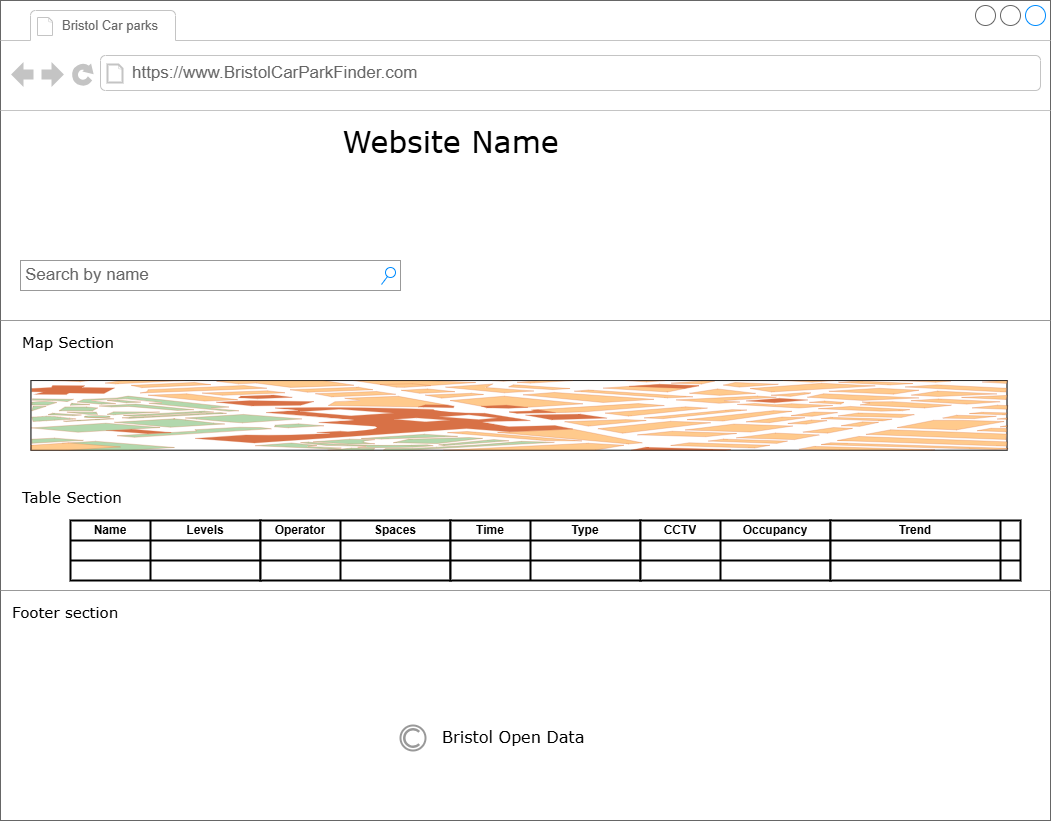
## 1. User Interface (UI) Design Overview

The design philosophy of the Bristol Car Park Finder Web Application is clean, intuitive, and mobile-first. The aim is to allow users to search, navigate, and view car park information easily on both the map and table views. The design process was in line with the UI/UX design process, from low-fidelity wireframes to high-fidelity mockups.

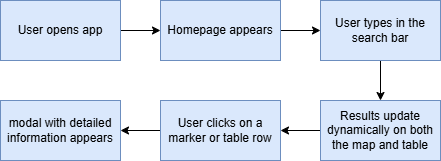
## 2. Wireframe and Wireflow Diagrams

**Low-Fidelity Wireframe:**

**Home Page Layout:**

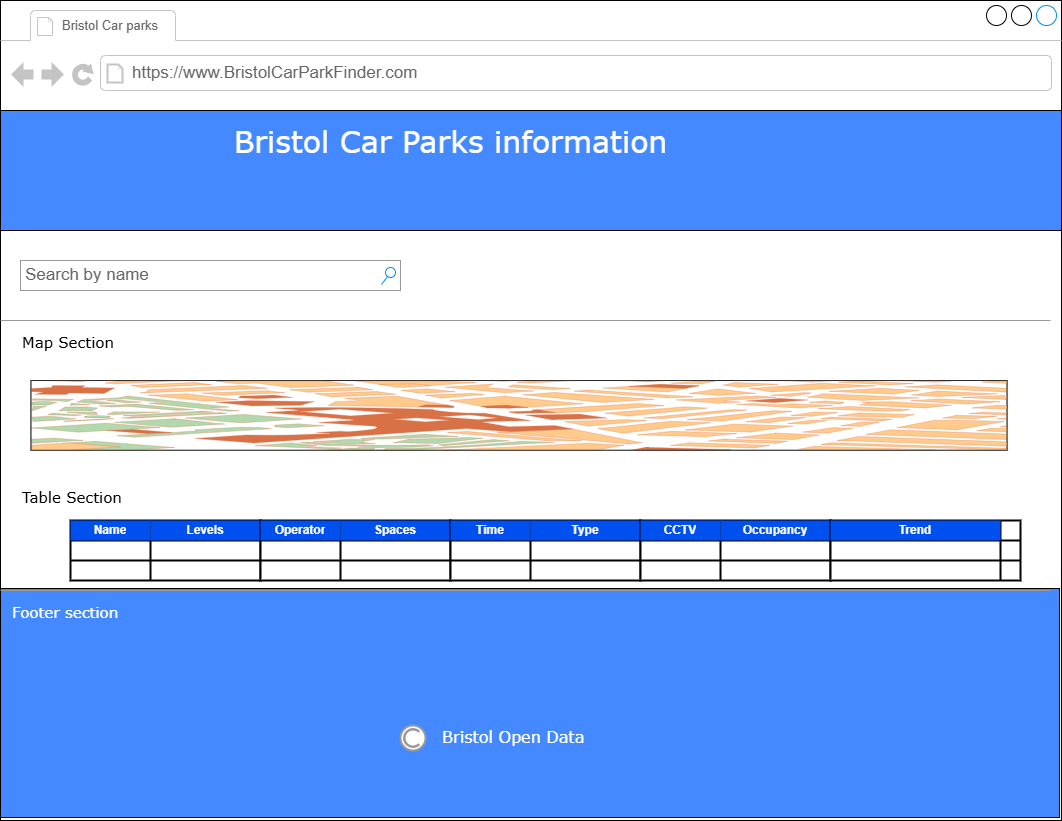
****

**Wireflow Diagram**



## 3. High-Fidelity Mockup

The high-fidelity mockup includes the graphic design, which gives a close-to-real look and feel of the application:



## 4. Accessibility Considerations

* The search bar and the table rows are also properly labelled according to the ARIA (Accessible Rich Internet Applications) standards.
* The colour contrast is high enough to provide visibility for users with vision impairment.
* The application is fully keyboard accessible, where the user can navigate through the search, map markers, and table entries using the tab key.
* The map markers have alt text to make the screen readers read out the right car park details.

## 5. User Interaction Flow

* Landing Page - The user first sees the map and the table.
* Search Interaction - When the user starts typing in the search bar, suggestions appear, and the map and table update their results in real time.
* Map/Table Integration - When a row in the table is selected, or a marker on the map is clicked, the other is highlighted.
* Additional Information - Hovering over a marker or a table cell opens a pop-up or modal window with more information, such as the number of spaces, CCTV, and operator.
* Scroll Responsiveness - The table is scrollable, displaying large data sets without affecting the page load time.

## 6. Design Justification

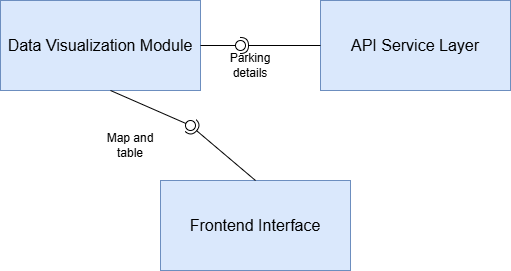
The design is practical and user-oriented, which is the primary focus of the concept. This is because the map and table components are placed side by side, which makes it easy for users to switch between the two and compare spatial and tabular data. The use of colours is consistent, the fonts are clear, and the grid layout makes the app look professional. Also, the integration of real-time filtering improves the application's functionality, making it easier for users to find a particular car park without being overwhelmed.

Responsiveness ensures that the interface is usable and readable no matter the device used, be it a computer, a tablet, or a mobile phone. In addition, the accessibility principles were followed to ensure that the application could be used by all users, including those with disabilities and those who use assistive devices.

# Section 4: Implementation

## UML Component Diagram

The UML Component diagram of the web application shows the modularity of the application and how the various components of the application interact with each other. The system is divided into three major parts: the Frontend Interface, the API Service Layer, and the Data Visualization Module. Each component is designed to be modular and independent, which makes it easier to maintain and scale.



**Frontend Interface:**

* index.html: This is the main HTML file containing the page's basic structure and links to CSS and JavaScript files.
* styles.css: This file is used to style the interface and ensure that the interface's look and feel are consistent.
* main.js: Manages event listeners, data loading, and DOM manipulation based on the user's actions.

**API Service Layer:**

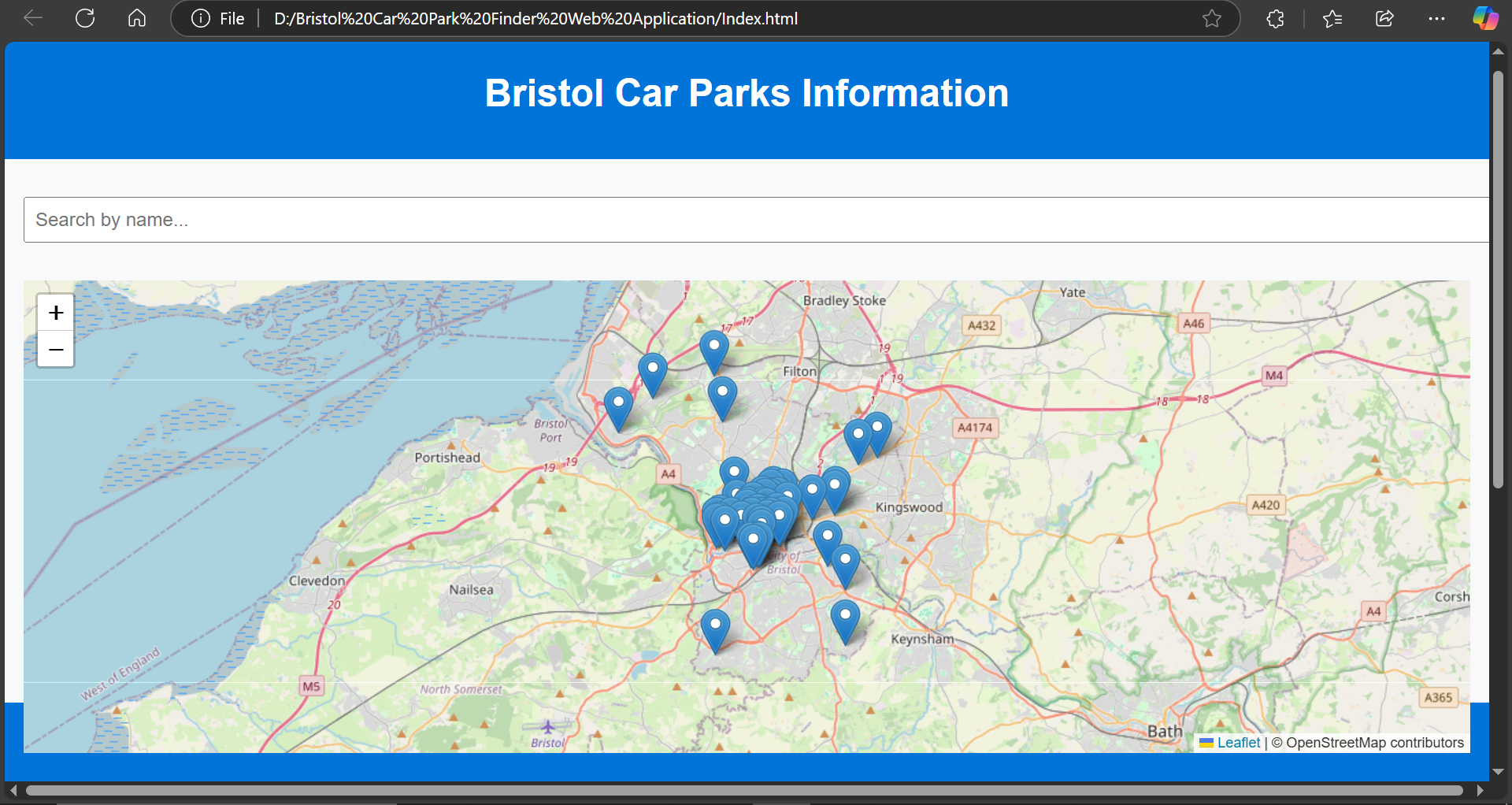
* Uses JavaScript's Fetch API to make HTTP GET requests to the Bristol Open Data API endpoint.
* Responsible for parsing and preprocessing JSON data.
* Serves as a data provider for the Table and Map components while filtering the data.

**Data Visualization Module:**

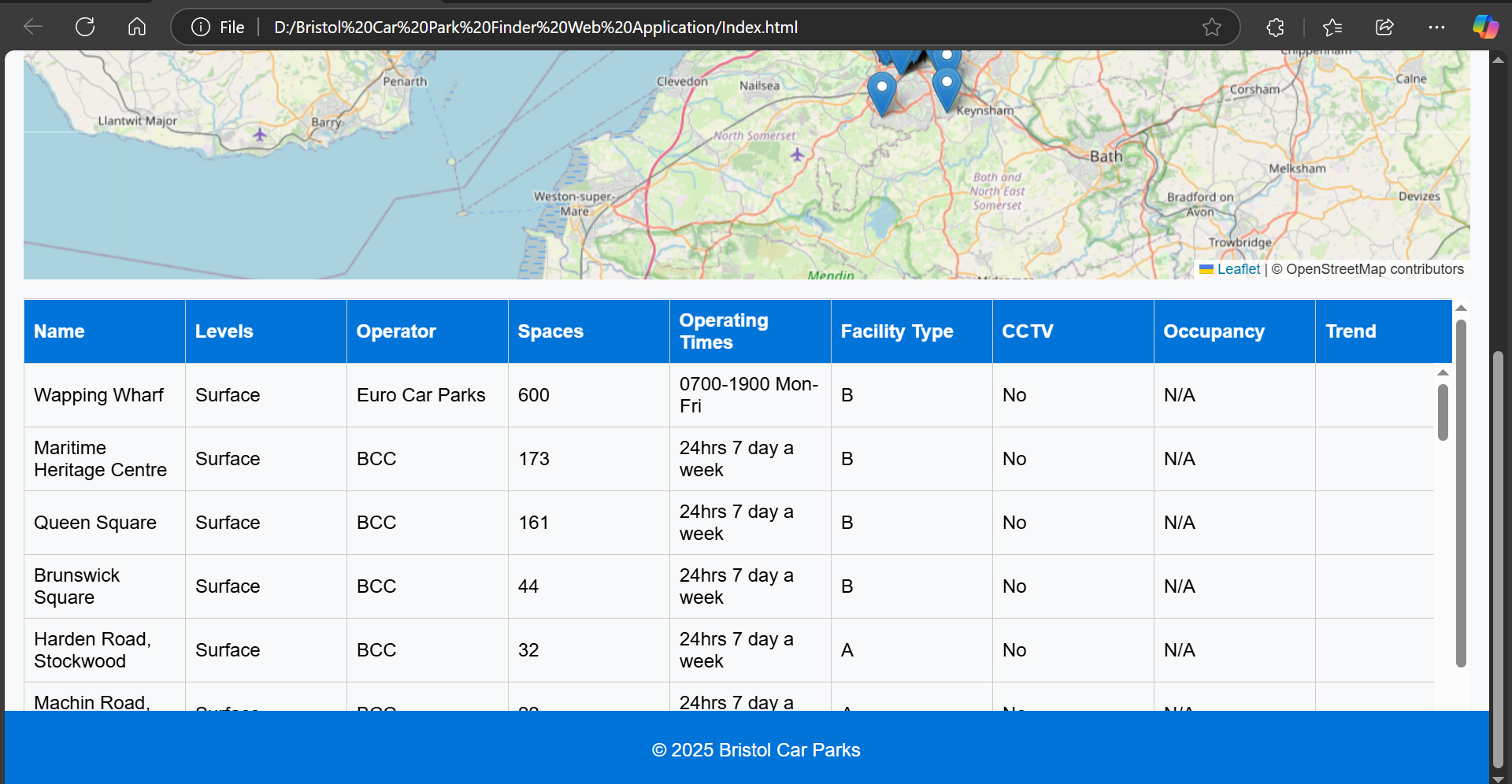
* Integrates Leaflet.js to display data points on the map.
* Updates map markers according to the user's search query.
* Updates the table dynamically based on the current data and the filters set by the user.

The interactions between these components are well-coordinated to reduce dependencies and facilitate data exchange. The API Service Layer is the intermediary that retrieves data and sends it to the visualization module and the frontend interface so that the user can interact with the most current data.

## Screenshots and User Guide

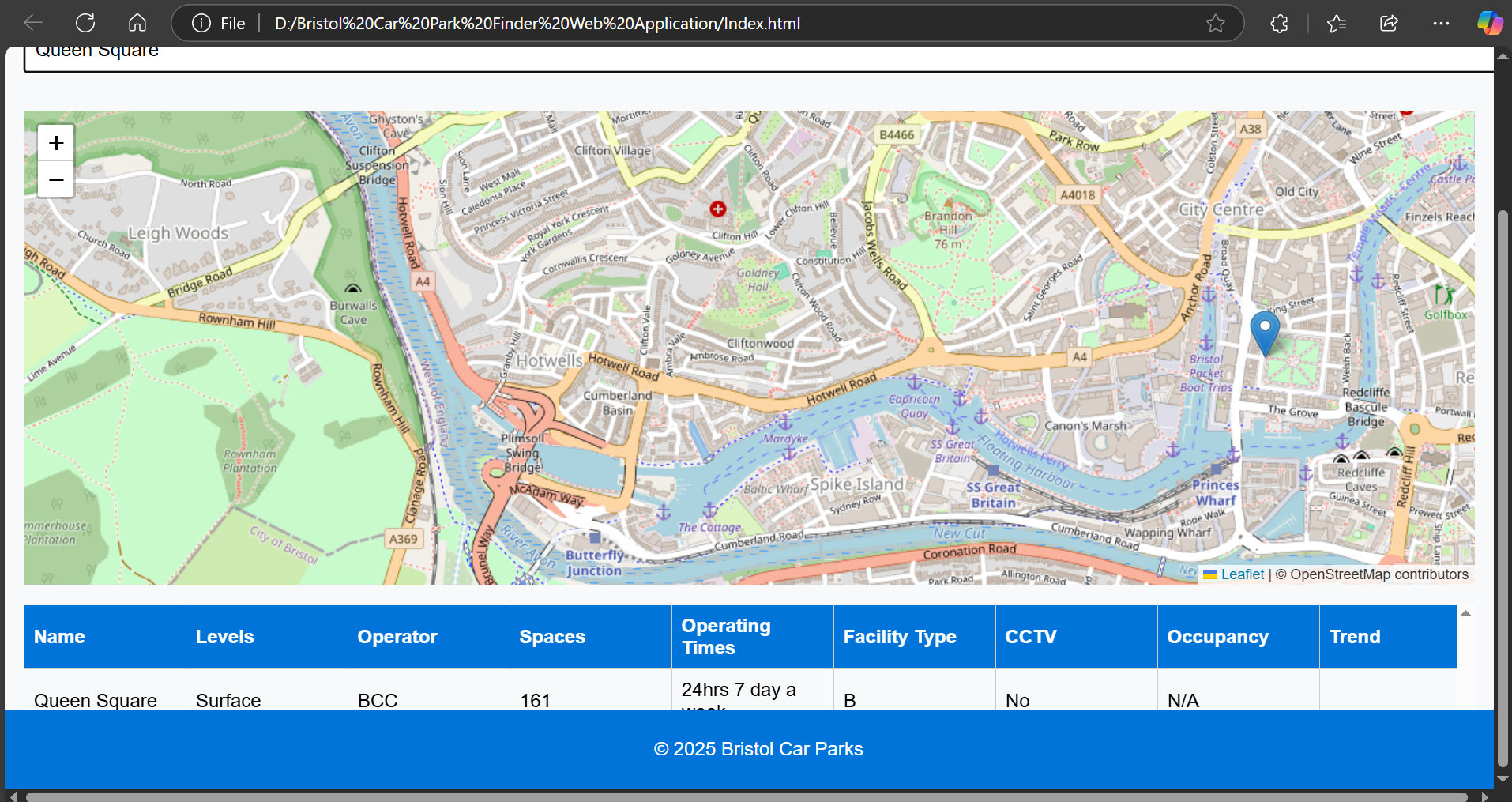
****

Screenshot 1: Homepage



Screenshot 2: Homepage on Scroll

These screenshots shows the application after data has been fetched from the database and displayed on the screen. It focuses on the map with markers indicating transport facilities. At the bottom of the map is a table that provides additional information about the facility, such as Name, Number of Levels, Operator, Spaces, etc.



Screenshot 3: Search Functionality in Action

A snapshot of the search feature where a user has entered the name of a particular facility. The map zooms in on the relevant marker, and the table is updated to show only the record that matches the search criteria.

**User Guide:**

**Launching the Application:**

* To do this, open the index.html file in any modern web browser, such as Google Chrome or Mozilla Firefox.
* Ensure that there is an active internet connection to retrieve data from the Bristol Open Data API.

**Navigating the Interface:**

* At the top of the page is a search bar where users can enter keywords related to transport facilities.
* Below the search bar is an interactive map created with the Leaflet library's help, showing markers for each facility.

**Using Search Functionality:**

* Type the name or any keyword related to a facility in the search bar.
* The map zooms in, and the marker corresponding to the selected location is highlighted.
* The table refreshes to display only the details of the searched facility.

**Viewing Facility Details:**

* Below the map is a table that provides information about each facility: Name, Number of Levels, Operator, Spaces, Operating Times, Facility Type, CCTV availability, Occupancy, and Trend.

**Responsive Design:**

* The application is cross-platform, which means it can be used on desktop and mobile platforms.

**Key Implementation Highlights**

**HTML5 Compliance:**

* All the HTML elements are checked with an HTML5 validator to check the structure and compatibility.

**CSS Styling:**

* The layout is clean and modern and built using the grid and flexbox.
* Consistent colour scheme and accessible font choices as per the design specifications.

**JavaScript Integration:**

* Effective utilization of event listeners to capture user inputs.
* Use asynchronous Fetch API for efficient API data acquisition without needing to refresh the page.
* Dynamic DOM manipulation to update both the map and table at the same time.

**Accessibility Considerations:**

* Semantic HTML tags for improved screen reader accessibility.
* High contrast color scheme and readable font sizes.

**Version Control:**

* All code and documentation are hosted on GitHub and committed to following standard commit conventions for ease of review and collaboration.

# Section 5: Testing

Testing is an important software development phase to check the web application's functionality, performance, and reliability. It ensures the application fits its intended use and functions properly in different scenarios. In developing our HTML web application based on the Bristol Open Data, we performed various testing activities, including manual and automated testing. The testing section includes the test plan, test cases, results, and the requirements traceability matrix.

## Test Plan

The testing strategy for the web application includes the following categories:

* Functional Testing - Ensures that all features of the application function correctly. This includes pulling data from APIs, displaying maps, populating tables, and performing searches.
* Usability testing - Ensures the application is easy to use and has a user-friendly interface.
* Compatibility Testing - Checks the application's compatibility across browsers (Chrome, Firefox, Edge) and devices (desktop, tablet, mobile).
* Performance Testing - This tests the application's ability to handle load and the time it takes to render the map and load data.
* Security Testing - Checks that external API calls are secure, and that user input in the search bar does not pose a threat.
* Regression Testing - It checks that the recent changes or bug fixes have not impacted the functionality incorrectly.

## Test Cases

The following test cases were developed and run:

Table 5: Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Description | Input | Expected Output | Actual Result | Status |
| TC01 | Verify API data is fetched successfully | API URL | JSON data displayed on map and table | JSON data retrieved and displayed | Pass |
| TC02 | Verify map loads and pins locations correctly | API data | Markers shown on map | Markers appear accurately | Pass |
| TC03 | Verify search functionality filters table correctly | Location name | Table shows matching entry | Correct entry displayed | Pass |
| TC04 | Verify search functionality highlights map pin | Location name | Corresponding map pin highlighted | Pin highlighted correctly | Pass |
| TC05 | Test UI responsiveness on mobile | Mobile viewport | UI adjusts to screen size | UI responsive | Pass |
| TC06 | Check cross-browser compatibility | Chrome, Firefox, Edge | Application loads correctly | Works fine on all browsers | Pass |
| TC07 | Verify table and map refresh when new data is loaded | Updated API data | Map and table reflect new data | Correctly updated | Pass |
| TC08 | Ensure input fields prevent XSS | Malicious script input | Input sanitized, no execution | No security issues | Pass |

**Requirements Traceability Matrix**

Table 6: Requirements Traceability Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement ID | Requirement Description | Test Case ID | Test Result |
| RQ01 | Retrieve and display data from API | TC01 | Pass |
| RQ02 | Display map with correct markers | TC02 | Pass |
| RQ03 | Search functionality filters table and highlights map pin | TC03, TC04 | Pass |
| RQ04 | User interface is responsive | TC05 | Pass |
| RQ05 | Application works across browsers | TC06 | Pass |
| RQ06 | Reflect updates to API data | TC07 | Pass |
| RQ07 | Prevent security vulnerabilities | TC08 | Pass |

**Bug Fixes**

Some of the problems that were noted and addressed during testing include the following:

* Search Highlight Issue -The first problem I encountered was that the search did not highlight the map marker. I made some changes in the JavaScript code to make the corresponding marker change color during a search.
* Table Problem - Sometimes, the table did not automatically scroll down to the matching entry. This was done by incorporating JavaScript code that scrolls the page to the row the user is searching for.
* Browser Compatibility - Some minor CSS differences were observed in Firefox, and the stylesheet was modified to add browser prefixes.

# Section 6: Individual Contribution

## Requirements

In the individual use case, my particular focus was on the search functionality for Bristol Car Parks, where users can search for car parks based on certain parameters such as name, operator, or type of facility. This feature improves the web application's functionality by allowing users to search for data on the map and in a table format.

Individual Use-Case Model Table:

Table 7: Use-Case Model

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Use Case ID | Use Case Name | Actors | Description | Preconditions | Postconditions | Basic Flow | Alternate Flow |
| UC1 | Search Car Park Data | User | Allows the user to input a search term to filter car park data displayed on the map and table. | User accesses the web app | Filtered results displayed | 1. User accesses the web page.  2. User inputs a search term in the search box.  3. Application filters and highlights matching car parks on the map and in the table.  4. Display relevant information. | No matches found: Notify the user and reset search field. |

This use case was developed based on the user requirements and the necessity of an efficient search tool for big data. It also provides the option for additional filter criteria (e.g., by availability or number of spaces) to be added.

## Implementation

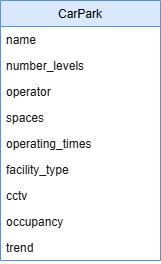
Querying Bristol Open Data

The data was collected from the following Bristol Open Data REST API:

<https://maps2.bristol.gov.uk/server2/rest/services/ext/ll_transport/MapServer/5/query?where=1%3D1&outFields=NAME,NUMBER_LEVELS,OPERATOR,SPACES,OPERATING_TIMES,FACILITY_TYPE,CCTV,OCCUPANCY,TREND&outSR=4326&f=json>

This API was accessed using the JavaScript fetch() function, and the JSON response was managed asynchronously. The query parameters were chosen to include all the fields necessary for map display and table data population. It also considers the possibility of API failures and provides for error messages in case data is not retrieved.

## UML Class Diagram Representing JSON Query Results

****

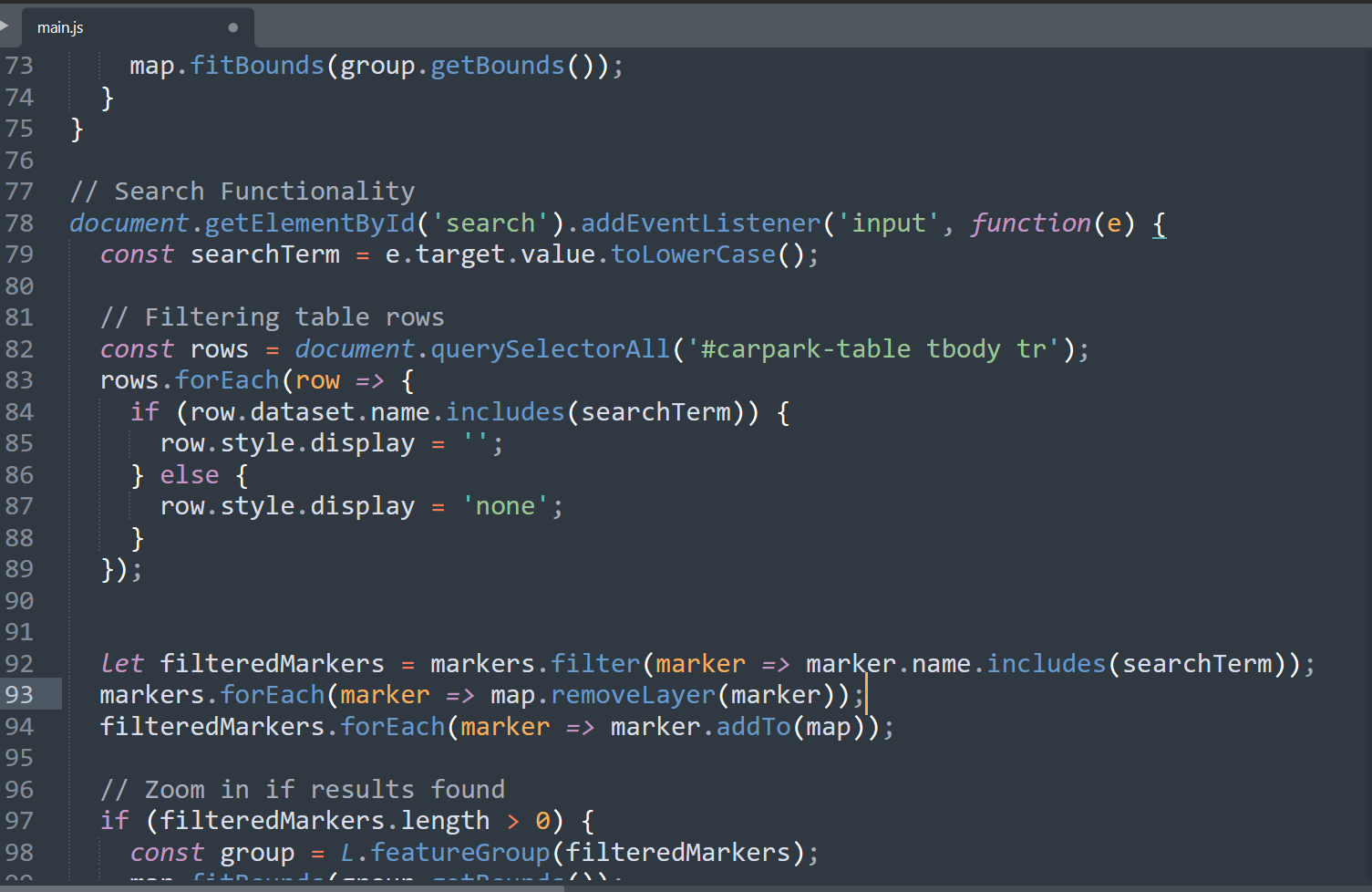
Each attribute is mapped to a field in the JSON response from the API, which is a direct mapping.

## Code Quality and Comments

JavaScript, HTML, and CSS code are commented on thoroughly to explain the code's purpose and logic. Specific attention was paid to:

* Discussing the fetch API calls and the error handling strategies.
* Adding comments to the DOM manipulation steps when populating the table and map with data.
* Ensure that variable names are clear and easy to understand.
* The division of work is into main.js, index.html, and styles.css.

Example snippet from main.js:



Screenshot 4: main.js code overview

## Manual Testing

Manual testing was performed by:

* Different search queries were performed to check the effectiveness of the filter.
* Ensuring that the corresponding markers are highlighted on the map.
* Testing for special cases (searching for an empty string or a string that does not exist).
* Cross-browser testing on Chrome, Firefox, and Edge.
* Ensuring all fields are displayed correctly (operator, spaces, CCTV, etc.).

**Requirements Traceability Matrix**

Table 8: Requirements Traceability Matrix

|  |  |  |
| --- | --- | --- |
| Requirement | Implemented In | Verified By |
| Search functionality filters table & map | main.js (filterData, plotMarkers) | Manual testing |
| Display relevant car park details | index.html, main.js | Manual UI verification |
| Error handling for failed API requests | main.js (fetch error handling) | manual testing |