Maintainability Document for TRESA Reserve Nature of Totterdown:

Introduction:

Welcome to the TRESA (Totterdown Urban Nature Reserve) Maintainability Document. This document outlines the necessary information and guidelines for maintaining and integrating the Totterdown Urban Nature Reserve project into existing systems or platforms. The project was undertaken with the objective of mapping and enhancing green spaces within the Totterdown area, with the overarching goal of covering 30% of the city into green spaces, as per TRESA's initiative.

Brief Description of the Project

The Totterdown Urban Nature Reserve project aimed to design a map highlighting green spaces within the Totterdown area. The project was initiated in collaboration with TRESA, an organization dedicated to expanding green spaces within urban environments. The project involved the development of a web-based mapping application that displays public and user-submitted green spaces on an interactive map interface.

Purpose of the Document

The purpose of this document is to provide comprehensive guidance on maintaining and integrating the Totterdown Urban Nature Reserve project into existing systems or platforms. It outlines the various components of the project, including the database structure, codebase, and external dependencies, and provides recommendations for ensuring the project's continued functionality and scalability.

Audience

This document is intended for developers, system administrators, and other stakeholders involved in the maintenance and integration of the Totterdown Urban Nature Reserve project. It assumes a basic understanding of web development technologies, databases, and system integration principles.

Database Schema:

The database schema provides a structured way to store information related to posts and categories. Here's a detailed breakdown of each table:

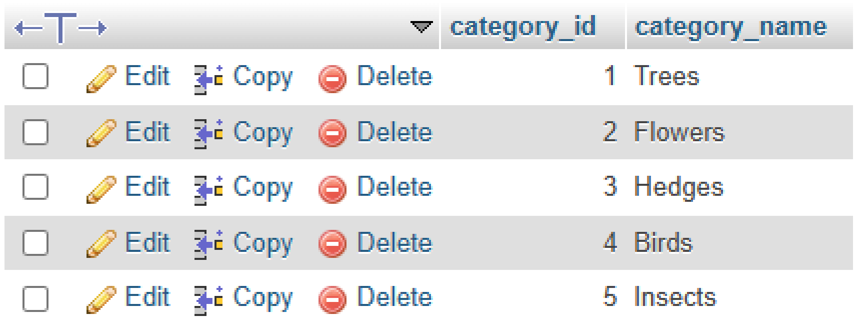
Category Table:

category_id (int): Unique identifier for each category.

category_name (varchar): Name of the category.

Primary Key: category_id

This table stores various categories such as Trees, Flowers, Hedges, Birds, and Insects.



←T→				category_id	category_name
☐	Edit	Copy	Delete	1	Trees
☐	Edit	Copy	Delete	2	Flowers
☐	Edit	Copy	Delete	3	Hedges
☐	Edit	Copy	Delete	4	Birds
☐	Edit	Copy	Delete	5	Insects

Category_has_post Table:

FK_post_id (int): Foreign key referencing the post_id in the privatespace_post table.

FK_category_id (int): Foreign key referencing the category_id in the category table.

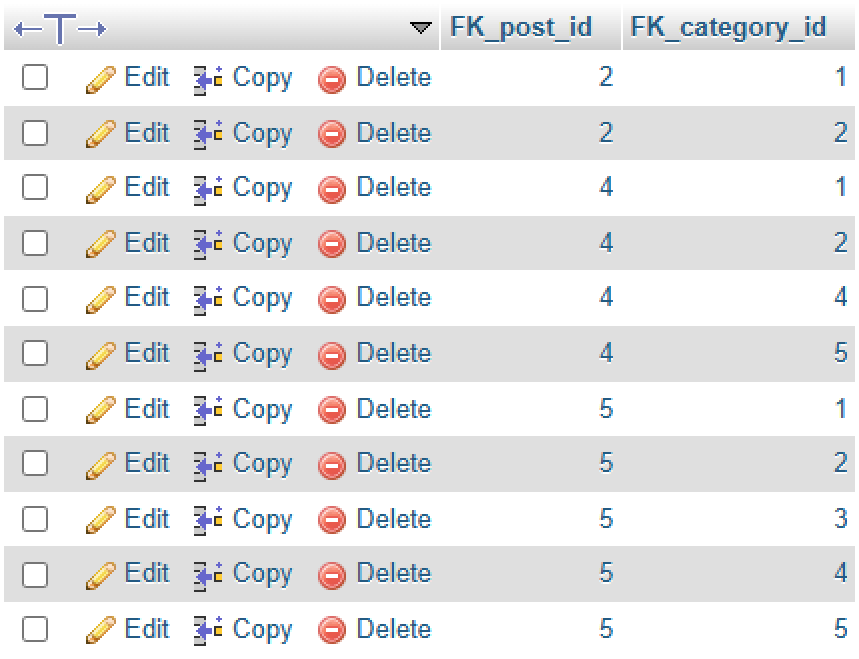
Primary Key: (FK_post_id, FK_category_id)

Foreign Key Constraints:

FK_category_has_post_category: References category_id in the category table.

FK_category_has_post_post1: References post_id in the privatespace_post table.

This table establishes a many-to-many relationship between posts and categories, allowing a post to be associated with multiple categories.



←T→				FK_post_id	FK_category_id
☐	Edit	Copy	Delete	2	1
☐	Edit	Copy	Delete	2	2
☐	Edit	Copy	Delete	4	1
☐	Edit	Copy	Delete	4	2
☐	Edit	Copy	Delete	4	4
☐	Edit	Copy	Delete	4	5
☐	Edit	Copy	Delete	5	1
☐	Edit	Copy	Delete	5	2
☐	Edit	Copy	Delete	5	3
☐	Edit	Copy	Delete	5	4
☐	Edit	Copy	Delete	5	5

Privatespace_post Table:

post_id (int): Unique identifier for each post.

post_resident_name (varchar): Name of the resident who made the post.

post_resident_email (varchar): Email of the resident who made the post.

post_lat (float): Latitude coordinate of the post location.

post_long (float): Longitude coordinate of the post location.

post_desc (varchar): Description of the post.

post_dimens (varchar): Dimensions of the post area.

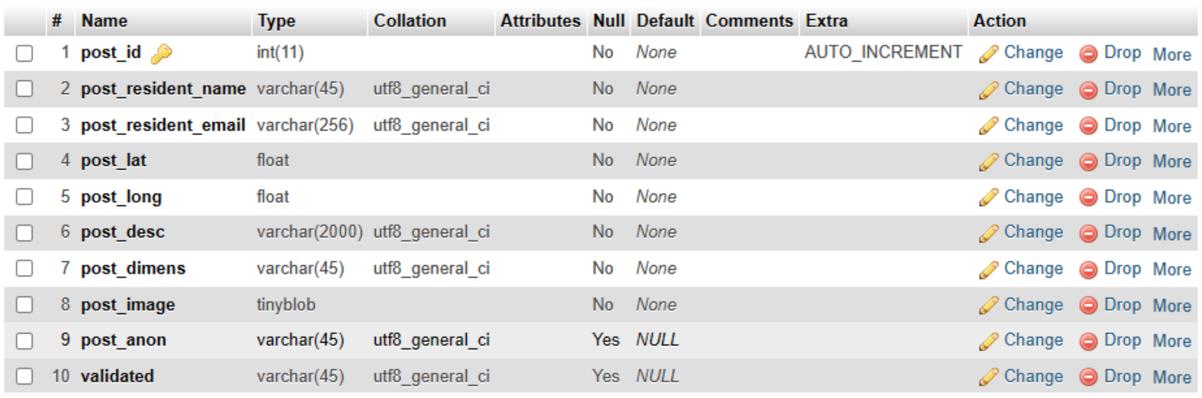
post_image (tinyblob): Image associated with the post.

post_anon (varchar): Indicates if the post is anonymous.

validated (varchar): Indicates if the post has been validated.

Primary Key: post_id

This table stores detailed information about private space posts, including resident details, location coordinates, description, dimensions, image, anonymity status, and validation status:



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	**post_id**	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
☐	2	**post_resident_name**	varchar(45)	utf8_general_ci		No	*None*			Change Drop More
☐	3	**post_resident_email**	varchar(256)	utf8_general_ci		No	*None*			Change Drop More
☐	4	**post_lat**	float			No	*None*			Change Drop More
☐	5	**post_long**	float			No	*None*			Change Drop More
☐	6	**post_desc**	varchar(2000)	utf8_general_ci		No	*None*			Change Drop More
☐	7	**post_dimens**	varchar(45)	utf8_general_ci		No	*None*			Change Drop More
☐	8	**post_image**	tinyblob			No	*None*			Change Drop More
☐	9	**post_anon**	varchar(45)	utf8_general_ci		Yes	*NULL*			Change Drop More
☐	10	**validated**	varchar(45)	utf8_general_ci		Yes	*NULL*			Change Drop More

Publicspace_post Table:

post_id (int): Unique identifier for each post.

post_area_name (varchar): Name of the public space area.

post_lat (float): Latitude coordinate of the post location.

post_long (float): Longitude coordinate of the post location.

post_desc (varchar): Description of the post.

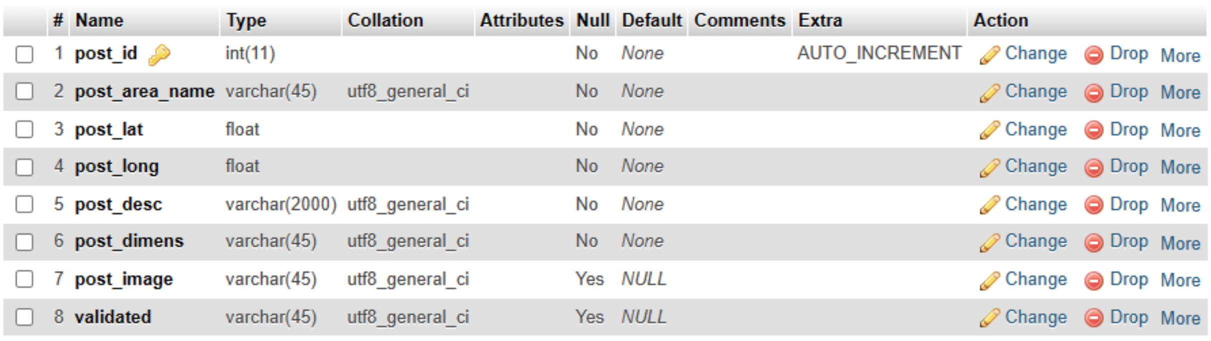
post_dimens (varchar): Dimensions of the post area.

post_image (varchar): Image associated with the post.

validated (varchar): Indicates if the post has been validated.

Primary Key: post_id

This table stores information about public space posts, including area name, location coordinates, description, dimensions, image, and validation status:



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	**post_id** 🔑	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
☐	2	**post_area_name**	varchar(45)	utf8_general_ci		No	*None*			Change Drop More
☐	3	**post_lat**	float			No	*None*			Change Drop More
☐	4	**post_long**	float			No	*None*			Change Drop More
☐	5	**post_desc**	varchar(2000)	utf8_general_ci		No	*None*			Change Drop More
☐	6	**post_dimens**	varchar(45)	utf8_general_ci		No	*None*			Change Drop More
☐	7	**post_image**	varchar(45)	utf8_general_ci		Yes	*NULL*			Change Drop More
☐	8	**validated**	varchar(45)	utf8_general_ci		Yes	*NULL*			Change Drop More

Feedback Table:

feedback_id (int): Unique identifier for each feedback entry.

post_id (int): Foreign key referencing the post_id in the privatespace_post or publicspace_post table, depending on the type of post associated with the feedback.

feedback_content (varchar): Content of the feedback provided by users.

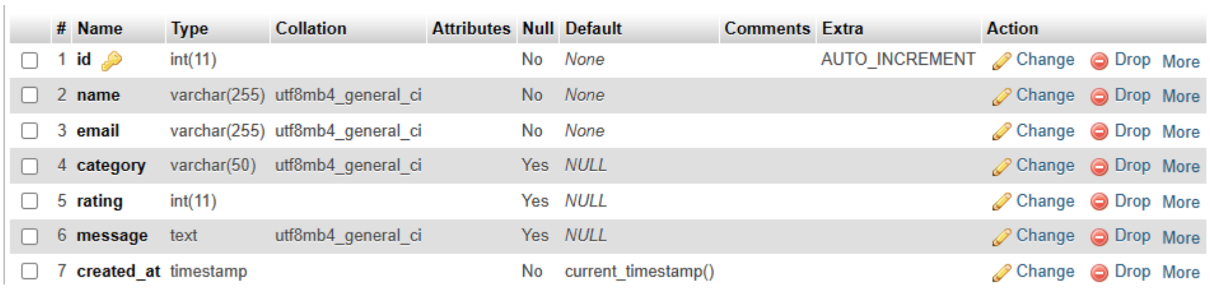
feedback_date (datetime): Date and time when the feedback was submitted.

Primary Key: feedback_id

Foreign Key Constraint:

FK_feedback_post_id: References post_id in either the privatespace_post or publicspace_post table, depending on the associated post type.

This table stores feedback submitted by users regarding specific posts. Each feedback entry is associated with a particular post, allowing administrators to review user comments and suggestions for improvement:



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	**id** 🔑	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
☐	2	**name**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	3	**email**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	4	**category**	varchar(50)	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
☐	5	**rating**	int(11)			Yes	*NULL*			Change Drop More
☐	6	**message**	text	utf8mb4_general_ci		Yes	*NULL*			Change Drop More
☐	7	**created_at**	timestamp			No	current_timestamp()			Change Drop More

Including the feedback table in the database schema enhances the platform's functionality by enabling user engagement and providing valuable insights for post moderation and enhancement.

Code Structure:

Admin Dashboard:

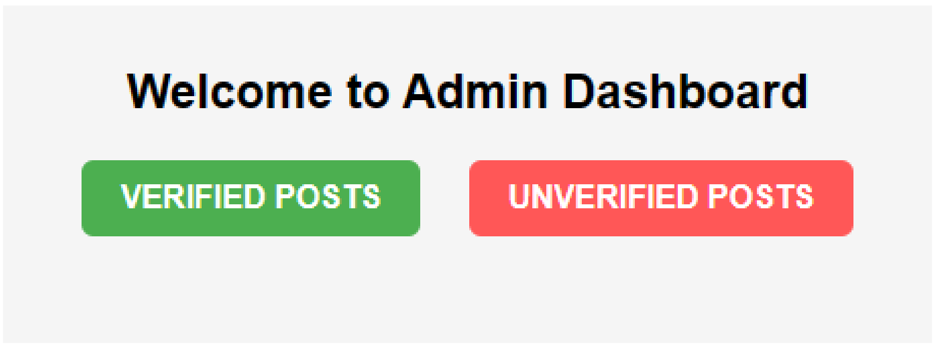
The admin dashboard application follows a structured approach to ensure maintainability and scalability. The codebase is organized into several files, each serving a specific purpose within the application. Below is an overview of the main components and their functionalities:

1. HTML Structure (admin.php)

This file defines the user interface of the admin dashboard, including layout, buttons, and other elements.

Purpose: Present a user-friendly interface for the admin to navigate and manage posts and feedback.

Location: admin.php



Welcome to Admin Dashboard

2. Action Handling (admin.php)

This PHP script handles actions initiated by the admin, such as validating or deleting posts and feedback.

Purpose: Process user requests and interact with the database accordingly to perform CRUD (Create, Read, Update, Delete) operations.

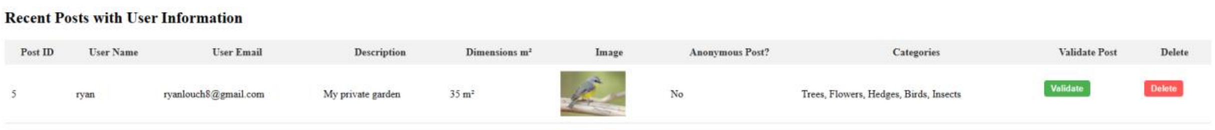
Location: admin.php

3. Display Recent Posts (unverified.php)

This PHP script retrieves recent posts from the database and displays them in the admin dashboard.

Purpose: Provide the admin with an overview of recent posts submitted by users, enabling validation or deletion.

Location: unverified.php



Recent Posts with User Information

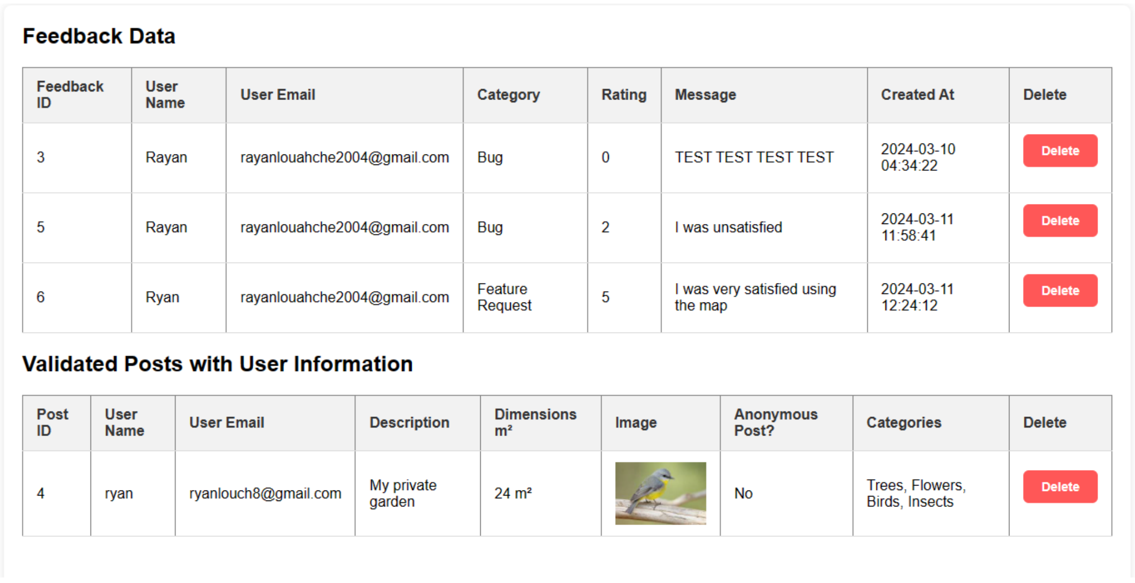
Post ID	User Name	User Email	Description	Dimensions m²	Image	Anonymous Post?	Categories	Validate Post	Delete
5	ryan	ryanlouch8@gmail.com	My private garden	35 m²		No	Trees, Flowers, Hedges, Birds, Insects	Validate	Delete

4. Display Feedback Data (verified.php)

This PHP script fetches feedback data submitted by users and presents it to the admin for review.

Purpose: Allow the admin to monitor user feedback, facilitating improvements and addressing concerns.

Location: verified.php



Feedback Data

Feedback ID	User Name	User Email	Category	Rating	Message	Created At	Delete
3	Rayan	rayanlouahche2004@gmail.com	Bug	0	TEST TEST TEST TEST	2024-03-10 04:34:22	Delete
5	Rayan	rayanlouahche2004@gmail.com	Bug	2	I was unsatisfied	2024-03-11 11:58:41	Delete
6	Ryan	rayanlouahche2004@gmail.com	Feature Request	5	I was very satisfied using the map	2024-03-11 12:24:12	Delete

Validated Posts with User Information

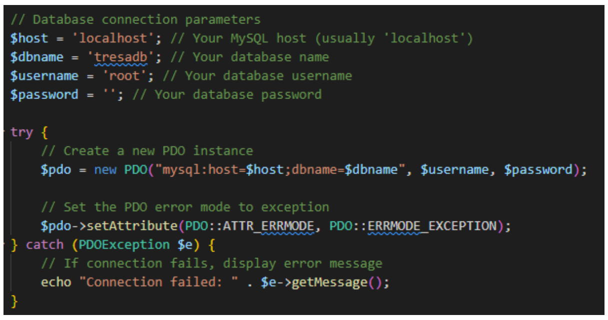
Post ID	User Name	User Email	Description	Dimensions m²	Image	Anonymous Post?	Categories	Delete
4	ryan	ryanlouch8@gmail.com	My private garden	24 m²		No	Trees, Flowers, Birds, Insects	Delete

5. Database Connection (config.php)

This PHP script establishes a connection to the MySQL database used by the admin dashboard application.

Purpose: Provides a mechanism for connecting to the database, ensuring consistency and reusability across multiple components.

Location: config.php



```php
// Database connection parameters
$host = 'localhost'; // Your MySQL host (usually 'localhost')
$dbname = 'tresadb'; // Your database name
$username = 'root'; // Your database username
$password = ''; // Your database password

try {
    // Create a new PDO instance
    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);

    // Set the PDO error mode to exception
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    // If connection fails, display error message
    echo "Connection failed: " . $e->getMessage();
}
```

6. JavaScript Functions

Several JavaScript functions are embedded within the HTML files to handle user interactions, confirmations, and other dynamic behaviors.

Purpose: Enhance the functionality of the admin dashboard interface by providing interactive features and feedback mechanisms.

Location: Embedded within HTML files.Styles (CSS) and JavaScript files: These files are used for styling the dashboard interface and adding client-side functionality, enhancing user experience.

Validate and Delete Post Logic: PHP code for handling validation and deletion of posts. This logic ensures that posts can be managed efficiently and securely.

HTML and PHP Mixed Code: Used for generating dynamic content and displaying posts in a tabular format. Proper indentation and comments are added to improve code readability.

This organized structure facilitates easy navigation, debugging, and future enhancements of the codebase.

7. Overall Functionality

The admin dashboard provides several key functionalities, including:

Viewing recent posts with user information, allowing administrators to monitor new submissions easily.

Validating posts to mark them as validated, ensuring that only approved posts are displayed to users.

Deleting posts from the system, enabling administrators to manage content effectively and remove inappropriate or outdated posts.

These functionalities empower administrators to maintain the platform efficiently and ensure the quality and integrity of posted content.

Main Component: ''Map'':

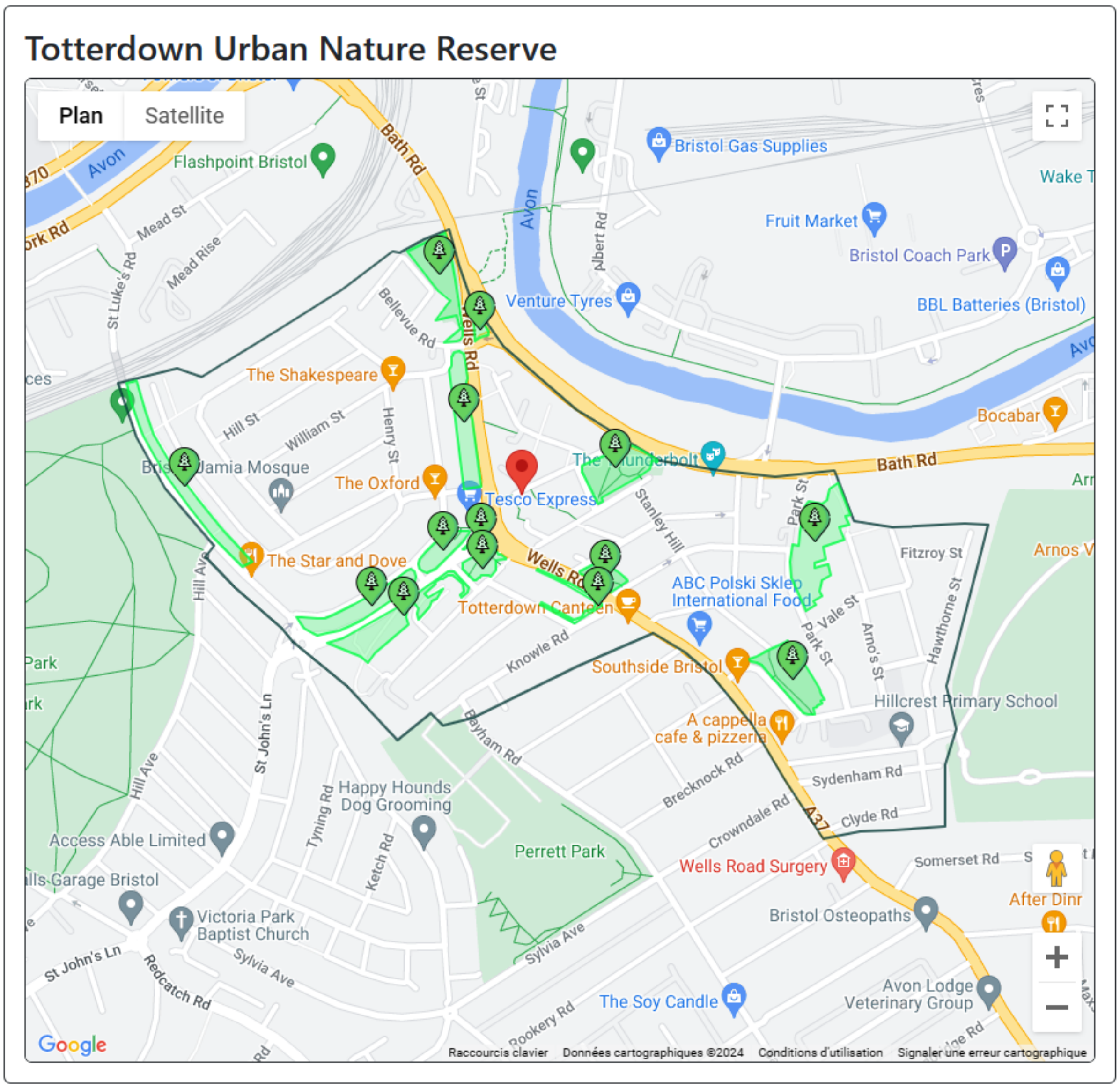
1. Main PHP Script (Map.php)

This file serves as the main entry point for the map application and contains PHP scripts to fetch data from the database and initialize the map.

Location: Map.php

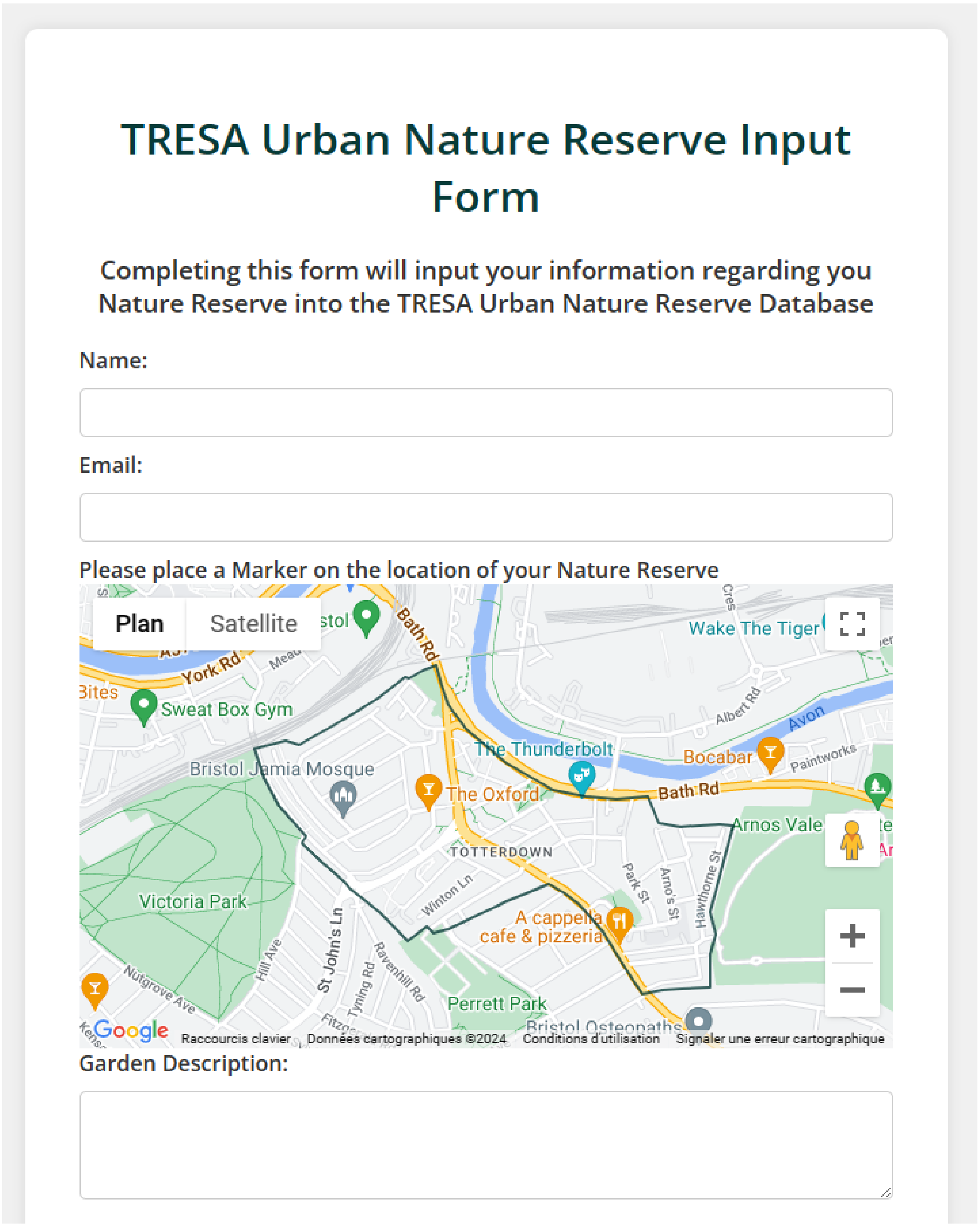
Functionalities:

-Fetches data from the database regarding public and private green spaces and Initializes the Google Maps API and displays the map:



Totterdown Urban Nature Reserve

-Adds markers for public and private green spaces on the map using a Form :



TRESA Urban Nature Reserve Input Form

Completing this form will input your information regarding you Nature Reserve into the TRESA Urban Nature Reserve Database

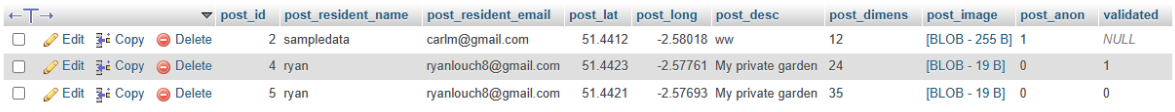
Name:

Email:

Please place a Marker on the location of your Nature Reserve

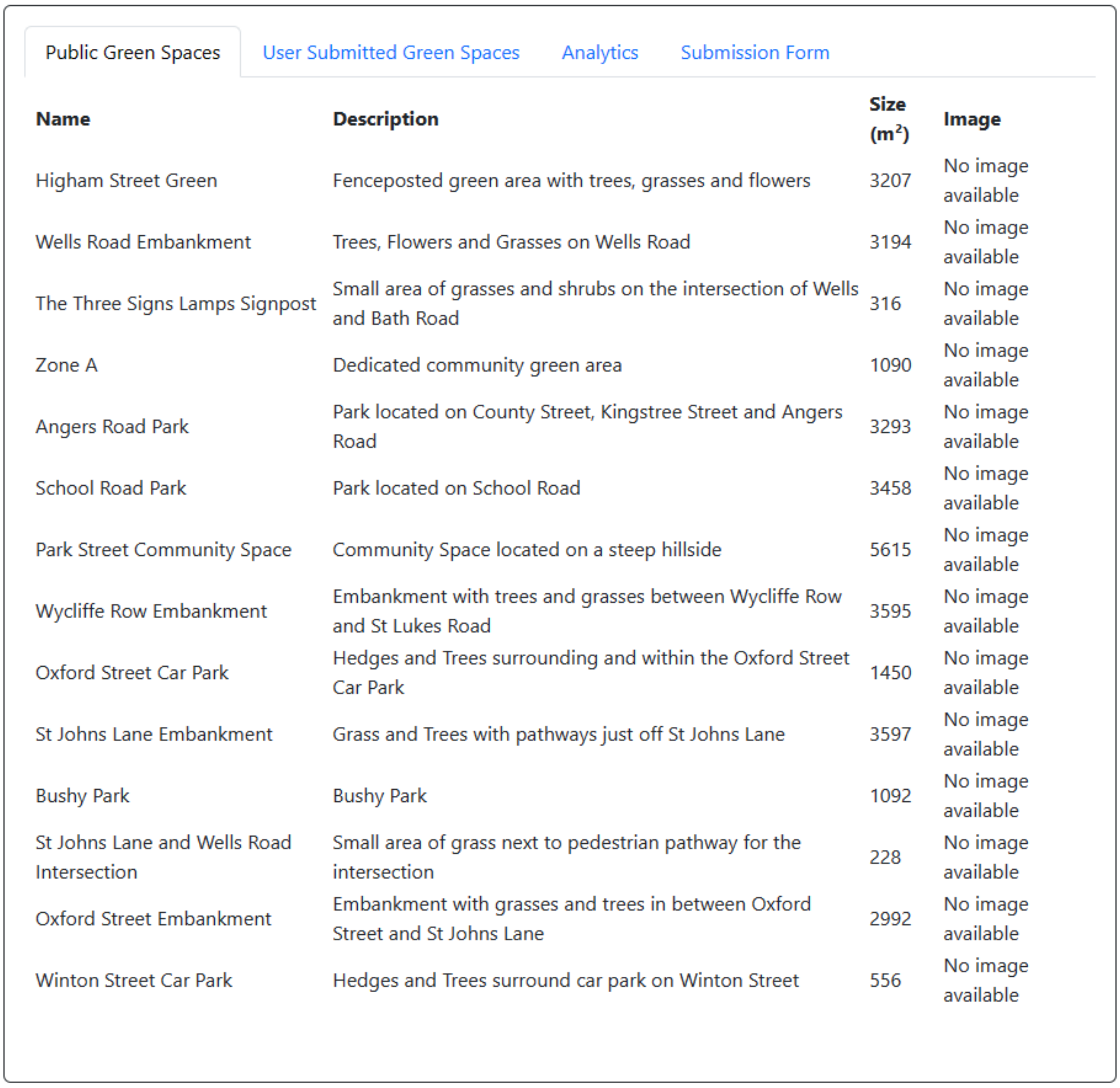
Garden Description:

-Populates tables with data retrieved from the database:



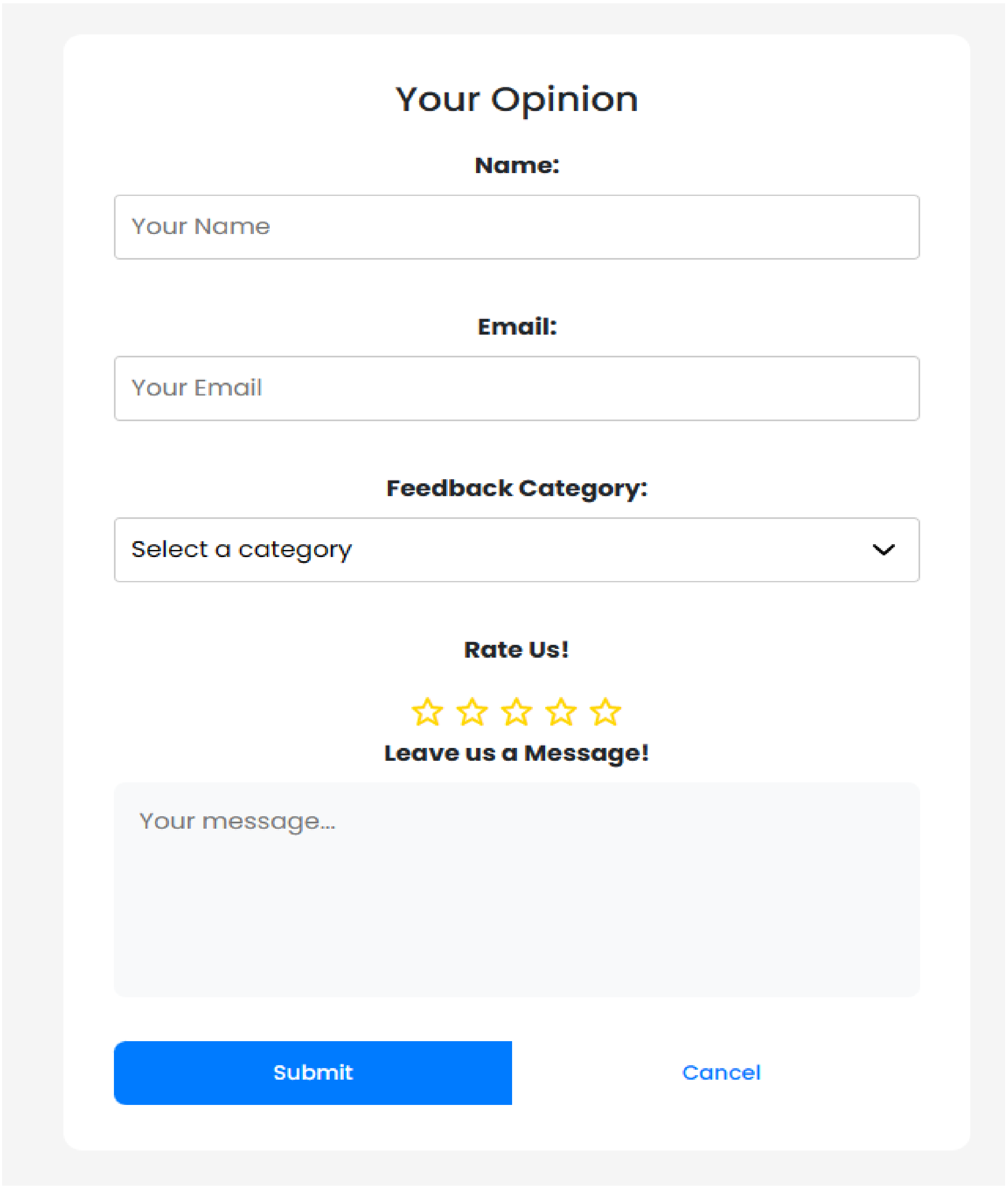
	post_id	post_resident_name	post_resident_email	post_lat	post_long	post_desc	post_dimens	post_image	post_anon	validated
Edit Copy Delete	2	sampledata	carlm@gmail.com	51.4412	-2.58018	ww	12	[BLOB - 255 B]	1	NULL
Edit Copy Delete	4	ryan	ryanlouch8@gmail.com	51.4423	-2.57761	My private garden	24	[BLOB - 19 B]	0	1
Edit Copy Delete	5	ryan	ryanlouch8@gmail.com	51.4421	-2.57693	My private garden	35	[BLOB - 19 B]	0	0

-Implements tabbed navigation for user interaction:



Name	Description	Size (m^2)	Image
Higham Street Green	Fenceposted green area with trees, grasses and flowers	3207	No image available
Wells Road Embankment	Trees, Flowers and Grasses on Wells Road	3194	No image available
The Three Signs Lamps Signpost	Small area of grasses and shrubs on the intersection of Wells and Bath Road	316	No image available
Zone A	Dedicated community green area	1090	No image available
Angers Road Park	Park located on County Street, Kingstree Street and Angers Road	3293	No image available
School Road Park	Park located on School Road	3458	No image available
Park Street Community Space	Community Space located on a steep hillside	5615	No image available
Wycliffe Row Embankment	Embankment with trees and grasses between Wycliffe Row and St Lukes Road	3595	No image available
Oxford Street Car Park	Hedges and Trees surrounding and within the Oxford Street Car Park	1450	No image available
St Johns Lane Embankment	Grass and Trees with pathways just off St Johns Lane	3597	No image available
Bushy Park	Bushy Park	1092	No image available
St Johns Lane and Wells Road Intersection	Small area of grass next to pedestrian pathway for the intersection	228	No image available
Oxford Street Embankment	Embankment with grasses and trees in between Oxford Street and St Johns Lane	2992	No image available
Winton Street Car Park	Hedges and Trees surround car park on Winton Street	556	No image available

-Navigation to a feedback page:



2. Configuration File (config.php)

This file contains configuration variables used by the main PHP script, such as database connection details and coordinates for TRESA area and public green spaces.

Location: config.php

Functionality:

Provides configuration variables for the main PHP script.

Defines constants for database connection details and other settings.

4. JavaScript Functionality

Several JavaScript functions are embedded within the HTML section of Map.php to enhance interactivity and map functionality.

Location: Embedded within Map.php

Functionality:

Initializes the Google Maps API and sets up the map.

Adds markers for public and private green spaces.

Handles user interactions such as clicking on markers and switching tabs.

5. Stylesheet (styles.css)

This CSS file contains styles used to enhance the visual appearance and layout of the map application.

Location: styles.css

Functionality:

Defines styles for various elements within the map application, such as tables, buttons, and map components.

6. External Libraries

The map application includes external libraries such as Bootstrap and Chart.js for additional functionality and styling.

Libraries Used:

Bootstrap (CSS and JavaScript)

Chart.js (JavaScript)

7. HTML Structure

The HTML section of Map.php defines the structure of the map application interface, including the map container, tabbed navigation, tables for displaying data, and other elements.

Functionality:

Renders the user interface elements of the map application.

Integrates PHP scripts, JavaScript functions, and external libraries.

Implements tabbed navigation for organizing content.

Code Quality:

Readability:

PHP Code Readability: The PHP code is generally well-structured and easy to follow. Variable names are descriptive, and comments are provided to explain complex logic or database queries.

JavaScript Code Readability: JavaScript functions are adequately commented and follow a consistent style. Variable names are meaningful, enhancing code readability.

HTML Structure: HTML markup is organized and follows best practices. Elements are appropriately nested, and IDs/classes are used effectively for styling and JavaScript interactions.

Maintainability:

Modular Structure: The code is divided into separate files for better organization and maintenance. Components like database connection, configuration, and JavaScript functionality are isolated, making it easier to update or modify specific parts.

Code Comments: Comments are used throughout the codebase to explain functionality, SQL queries, and important logic. This aids in understanding and maintaining the code in the future.

Consistent Coding Style: The code follows a consistent coding style, both in PHP and JavaScript sections, promoting maintainability and readability.

Performance:

Database Queries: Database queries are necessary but could be optimized further for performance, especially if the dataset grows large. Consideration should be given to indexing, query optimization, and caching mechanisms.

Client-Side Processing: JavaScript functions, especially those related to map rendering and marker handling, should be optimized to minimize client-side processing, and improve user experience.

Security:

SQL Injection Prevention: PDO is used for database interactions, which helps prevent SQL injection attacks. However, prepared statements should be utilized consistently throughout the application to mitigate security risks.

Data Validation: Input data should be properly validated, especially when interacting with user-submitted content, to prevent XSS (Cross-Site Scripting) and other security vulnerabilities.

Adherence to Coding Standards:

Consistency: The code generally adheres to coding standards, but minor inconsistencies exist (e.g., variable naming conventions, indentation). Ensuring strict adherence to a consistent coding style can improve maintainability and collaboration.

Overall Assessment:

The Map.php file and its associated components demonstrate good code quality overall. However, there are areas for improvement, such as optimizing database queries for performance, enhancing client-side processing efficiency, and ensuring consistent adherence to coding standards. Regular code reviews and refactoring efforts can further improve the quality, maintainability, and security of the application.