Project „Broker checker“

A web application with a specific purpose, designed to improve the flow of information among employees in a US based carrier’s logistics centre.

Aleksandar Petrović

v0725.alpha1

Table of Contents

1. Introduction

1.1. Project Overview

Project Broker Checker is an early WIP web application designed to improve communications in a developing logistics coordination team for the United States of America market and operations. This software was not meant to include many functionalities, but to make Customer Relations a simple endeavor with its 'storage warehouse'-type database, and easy to learn UI, designed to be appealing to carrier's logistics coordination team members and other employees. With its smart use of **SaferWeb API**, this web app provides all the necessary data from **USDOT** along with smartly packed local database entries, providing minimal storage requirements.

1.2. Purpose and scope

Main purpose of this software is to be the carrier's **Customer Relations Internal Management System**. With the provided database solution, this software can store all the information one carrier requires about every other carrier noted in the **USA Department of Transportation**. It was made to prevent loss of income, or loss of another kind, on a potentially booked load based on previous experiences. However, this software's greatest advantage is its ability to provide **negotiation leverage** based on previous experiences, potentially improving profits over both the short and long term.

1.3. Target users

This software is designed as an essential tool for a developing US carrier's logistics coordination team – improving general relations knowledge for new team members, while reminding senior members of any positive or negative feedback about a certain customer.

Its usefulness comes mainly from its users. A team with internal rules about making entries will develop the app's database to its full potential.

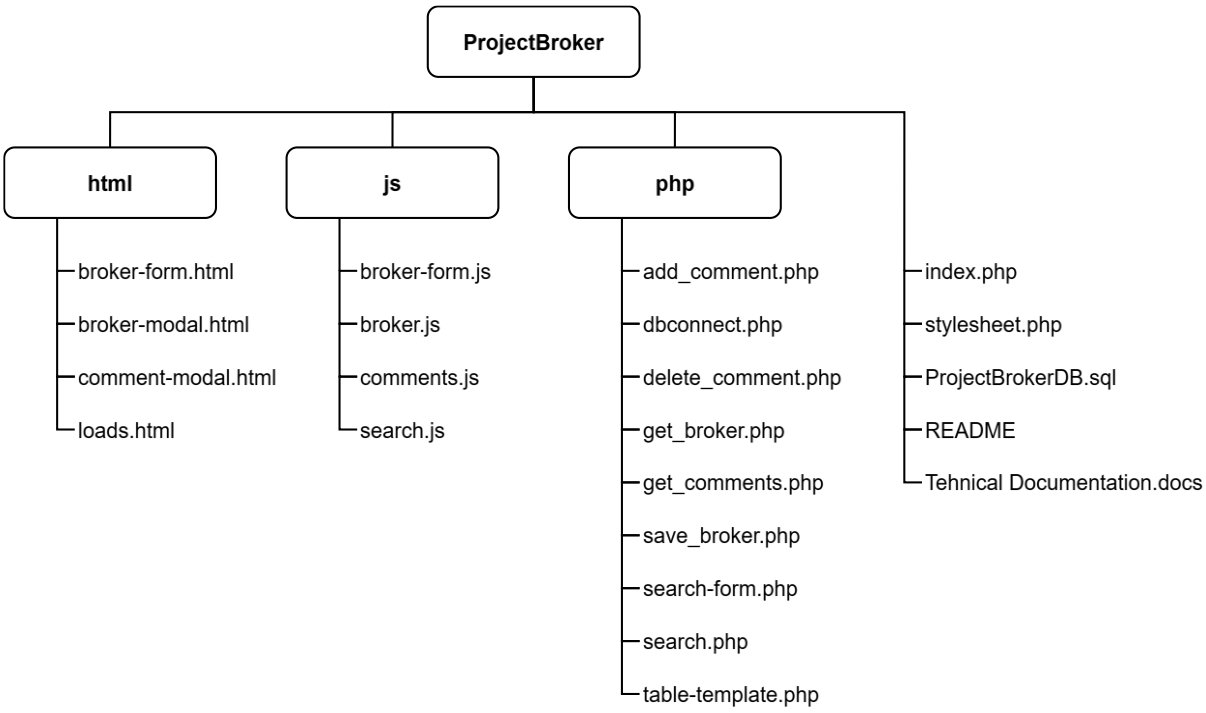
2. Tech Stack & Architecture

2.1. Technologies Used

The project uses **PHP 8** as the server-side language for API search and database manipulation, running on an **Apache 2.4 HTTP Server** used for local development. Its frontend was developed using **HTML** and **JavaScript** for static and dynamic elements, as well as **Bootstrap 5.3.7** for building a user-friendly interface. **MariaDB** is used for the database design and is connected to via **phpMyAdmin** for testing and development.

These languages and development tools were chosen due to their recognition and reliability in this area of software engineering, allowing for improved modularity and easier enhancements based on the provided source code.

2.2. File & Folder Structure



Picture 1: Folder Tree

3. Setup Instructions

3.1. Prerequisites

For the project to be set up, it requires a host PC with either **Windows** or **Linux OS** with installed web server of choice and a DMBS.

The project was developed on **Apache 2.4** web server with **MariaDB,** connected via **phpMyAdmin** for database management. These tools are recommended to minimize the need for adjustments.

3.2. Local Installation

Granted the host PC has the required web server and DBMS, installation is straightforward.

1. Place the `ProjectBroker` folder in the web server's source directory.
2. Run `ProjectBrokerDB.sql` with a DBMS of your choice.
3. In the file `ProjectBroker\php\dbconnect.php`, update the $host, $user and $pass variables with your server's info.
4. Make sure the web server and DBMS are up and running.
5. Open the browser and access the application via `http://localhost/ProjectBroker`.

4. Main Components

4.1. Backend

This project's backend was developed using the **PHP** server-side language. Backend scripts are mainly used for database manipulation, external API communication and returning the results to the frontend JS scripts.

Regarding database manipulation, these scripts were handling queries such as `SELECT`, `INSERT`, `UPDATE` and `DELETE`.

Notable scripts include:

- `search.php` — searches the database and fetches basic carrier info from the SaferWeb API, returning JSON data based on the search input and selected mode (search/edit).
- `save_broker.php` — inserts or updates carrier entries in the database.
- `comment.php` — manages comment entries linked to brokers.

4.2. Frontend

The development of dynamic elements and UI was made with **HTML**, **CSS**, **JavaScript** and Bootstrap.

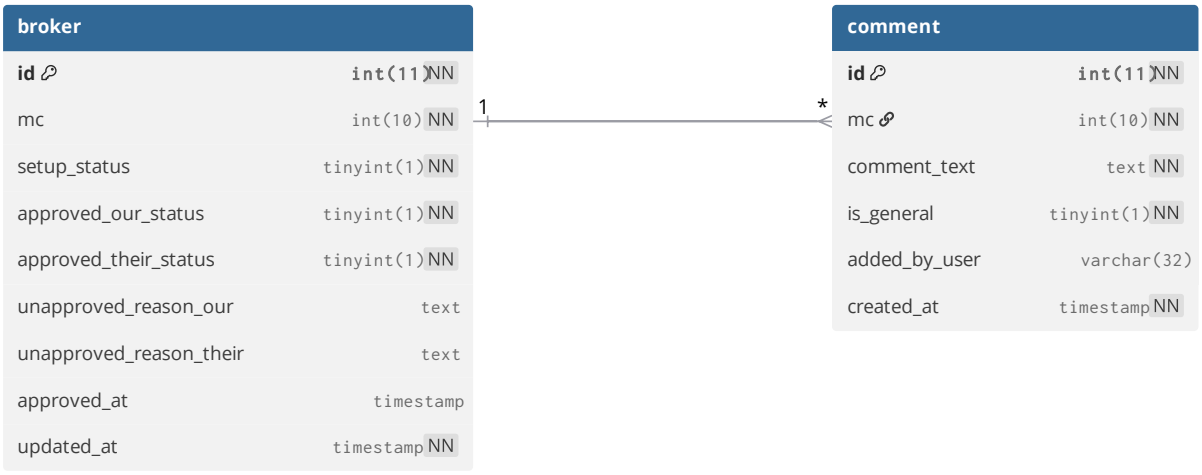
HTML was used for creating static elements of the web app, along with some dynamic elements which are called with JS when needed.

JS was used for calling dynamic elements when needed and populating them with information provided in JSON format by the PHP scripts.

Bootstrap and **CSS** were used for styling and layout, resulting in a clean, user-friendly interface

4.3. Database

The SQL database was made using the **phpMyAdmin** tool. It consists of two tables with one-to-many relationship between them, based on the `mc (INT)` column.



Picture 2: Database design

5. Functionality

5.1. Search Brokers

This is the main function of the project. A user enters info in one of the provided textboxes (**MC#**, **DOT#** or **carrier name**) and clicks the **Search** button. The app first searches through the **SaferWeb API**, then checks whether the carrier already exists in the local database.

If no data is entered, an error message is displayed requiring the user to enter data in at least one of the search boxes.

The user may then be presented with one of five possible outcomes:

1. **No broker found** – If the MC#, DOT#, or name do not exist, an error message is displayed.
2. **Yellow row (API only)** – If a matching entry exists in the USDOT API but not in the database, the MC#, DOT#, and name are shown. The rest of the data is not provided. The Actions dropdown only displays the **"Add Broker to System"** button.
3. **Green row (Approved)** – If the carrier exists in both the API and the database, and the 'Approved' status is positive, the row is green. Additional information (setup status, approval info, general comment, and last edited date) is shown. The Actions dropdown now includes options to **edit broker**, **view/add comments**, and **book load** (WIP).
4. **Red row (Not Approved)** – If the carrier exists in both the API and the database, but the 'Approved' status is negative, the row is red. All information is displayed, and the Actions dropdown offers the same options as above.
5. **Multiple matches** – If the entered carrier name is a 100% match for more than one carrier in USDOT, a list of entries is displayed, each behaving according to its respective status (as in 2–4).

5.2. Add/Edit Brokers

Both functions open a **modal** that allows full customization of database entry information.

- When **adding** a carrier, the user can set **Setup Status**, **Approved (Our Side)**, and **Approved (Their Side)** using Yes/No options.
  A **General Comment** field is also available but not required.
  If either approval status is set to **No**, an additional **Unapproved Reason** field appears, which is also optional.
- When **editing** a carrier, all current database values are pre-filled into the modal, allowing the user to make changes accordingly.

5.3. View/Add Comments

When the user clicks the **View Comments** button, a modal opens showing all previously added comments. Each comment includes the author and its full text.

When the user clicks the **Add Comment** option (from inside the comment modal), a secondary modal appears with a textbox and a checkbox to indicate whether the comment is **general** or not.

If a general comment is added and submitted, the page is refreshed, and the **General Comment** field in the broker row is updated.

6. Known Issues & Future Plans

6.1. Known Issues

- No User Integration

The current version of the project doesn't include any user accounts or authentication method. The User Integration is only mimicked by the navbar selection option which, for now, hosts two pseudo users.

6.2. Future Plans

- User Management System

User roles and login authentication will be introduced in future versions. This will allow restricted access to sensitive data and tracking of actions, improving security.

- Rating System

A star or number based rating system may be introduced to allow users to rate carriers, which will provide a quick visual reference, further improving the project's purpose

- Load booking

A load booking system may be introduced, expanding the data warehouse, further increasing the data available for analytics and detailed info on the Customer Relations.