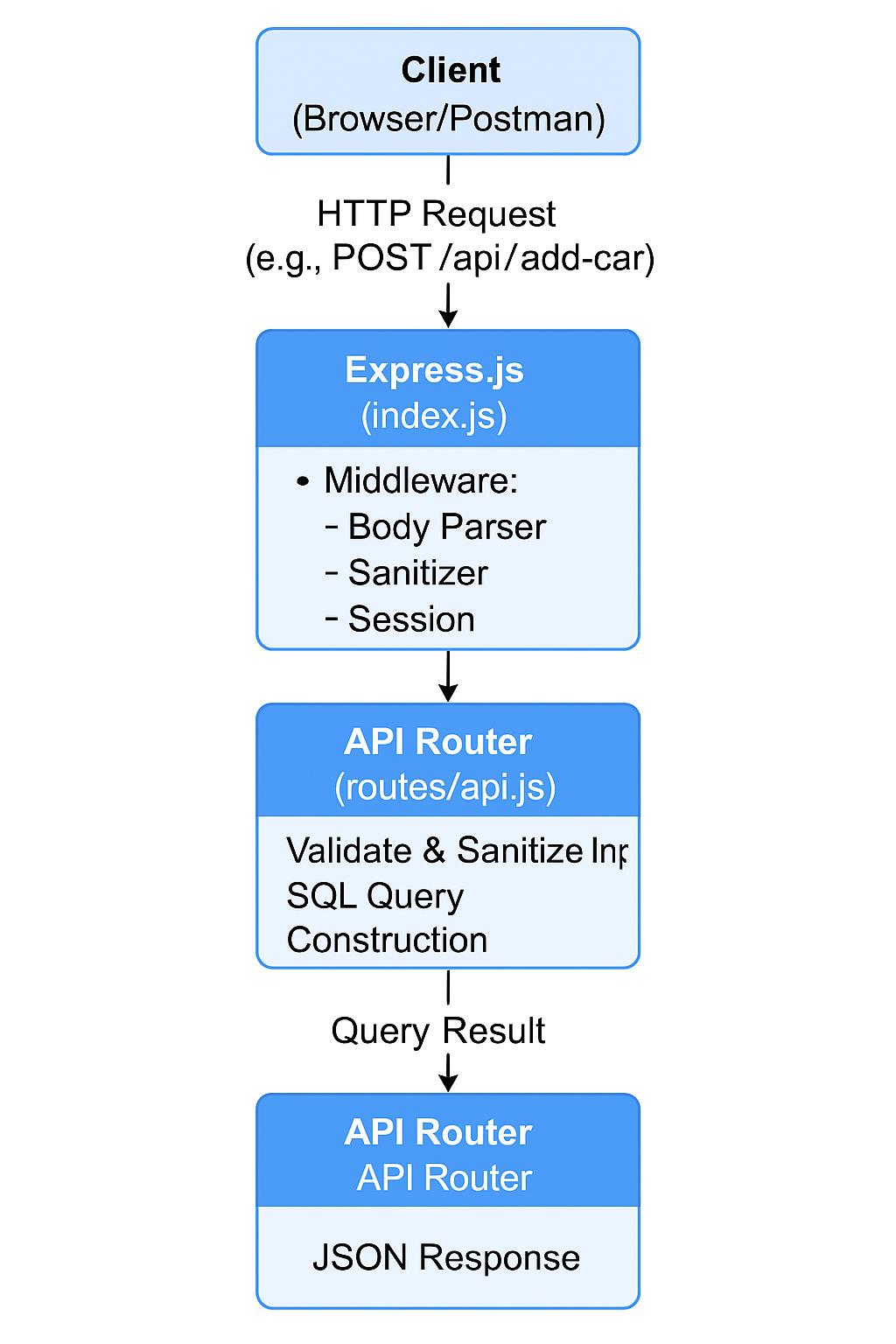
**Portfolio Project Report**

**Outline**

The Car Inventory is a comprehensive web application designed for managing automotive inventory and related business operations. The system includes car management capabilities, allowing users to add, view, search, and manage vehicle inventory with detailed information including make, model, colour, price, condition, and dealer associations. The application features a sophisticated dealer management system for tracking car dealers, including their contact information and locations, while also providing comprehensive maintenance record tracking that logs service history, dates, details, and costs for each vehicle. There is also a bargain finder feature that quickly identifies cars priced under £20,000. For system control, the application includes a complete user authentication system with secure login and registration capabilities and provides a comprehensive RESTful API with an interactive API Explorer for programmatic access to all system functions.

Additional features include user administration tools, weather integration for multiple locations, and a modern interactive interface enhanced with spinning car animations. The system is built with security in mind, incorporating input sanitization, SQL injection protection, comprehensive validation, and secure error handling to ensure data integrity and system reliability.



In the application tier, the templating engine EJS is used to render dynamic HTML views, while static assets like CSS and images are served from the public directory. Middleware such as express-session manages user sessions, express-validator ensures input validation, and express-sanitizer protects against malicious input. The application is modularized with separate route handlers for main site navigation, user management, car management, and API endpoints, promoting maintainability and scalability.

In the data tier, the application uses MySQL as its relational database management system. The mysql2 Node.js package facilitates secure and efficient communication between the Express server and the MySQL database. The database stores all persistent data, including car details, user accounts, dealers, and maintenance records. SQL scripts (create\_db.sql and insert\_test\_data.sql) are used to define the schema and populate initial data.

**Car Inventory Data Model**

| **Table Name** | **Columns (Field Name)** | **Data Type** | **Description** |
| --- | --- | --- | --- |
| **users** | id (PK) | INT, AUTO\_INCREMENT | Unique user ID |
|  | First Name | VARCHAR(50) | User’s First Name |
|  | Last Name | VARCHAR(50) | User’s Last Name |
|  | username | VARCHAR(50) | User’s login name |
|  | Password | VARCHAR(20) | User’s Password |
|  | Hashed password | VARCHAR(255) | Hashed password |
|  | email | VARCHAR(100) | User’s email address |
| **dealers** | id (PK) | INT, AUTO\_INCREMENT | Unique dealer ID |
|  | name | VARCHAR(100) | Dealer name |
|  | location | VARCHAR(255) | Dealer address |
|  | contact | VARCHAR(50) | Dealer phone number |
| **cars** | id (PK) | INT, AUTO\_INCREMENT | Unique car ID |
|  | Colour | VARCHAR(20) | Colour of the car |
|  | make | VARCHAR(50) | Car make (e.g., Toyota) |
|  | model | VARCHAR(50) | Car model (e.g., Corolla) |
|  | year | INT | Year of manufacture |
|  | dealer\_id (FK) | INT | Linked dealer |
|  | price | DECIMAL(10,2) | Car price |
|  | Condition | ENUM('used', 'new', ‘certified’) | Car status |
| **maintenance** | id (PK) | INT, AUTO\_INCREMENT | Unique maintenance record ID |
|  | car\_id (FK) | INT | Linked car |
|  | description | VARCHAR(255) | Maintenance description |
|  | date | DATE | Maintenance date |
|  | cost | DECIMAL(10,2) | Maintenance cost |

The Car Inventory data model is designed to efficiently manage information about vehicles, users, dealers, owners, and maintenance activities.  At its core is the cars table, which stores essential details for each vehicle, such as make, model, year, VIN, price, and status (available or sold). Each car record is linked to a dealer and an owner through foreign keys.

The users table manages authentication and authorization, storing user credentials and contact information to ensure secure access within the application. The dealers table contains information about car dealerships, including their names, addresses, and contact details. Maintenance history is tracked in the maintenance table, which records each service event for a car, including the date, description, and cost. This table references the cars table, enabling a comprehensive maintenance log for every vehicle in the inventory.

By normalizing data and using foreign keys, the model ensures data integrity, reduces redundancy, and supports complex queries, such as retrieving all cars for a specific dealer or viewing a car’s complete maintenance history.

A screenshot of a computer

AI-generated content may be incorrect.

The login page uses a simple and sleek design but having forms where the use fills in their username and password to ensure only verified accounts can access the site. There is also the option to register if the user does not have a pre-existing password and username. There is also a footer with a spinning car gif to add flair to the page, with a clear identification of the Car Inventory System.

A screenshot of a car inventory

AI-generated content may be incorrect.

The homepage includes links to various functions, such as car inventory, user management, dealer interaction, and additional tools. The user functions available are:

**Inventory Management**

* **Add a new car**: Allows the user to add a new vehicle entry to the system.
* **List of cars**: Displays a full list of all vehicles currently in the inventory.
* **Bargain cars**: Filters and shows cars available at discounted or bargain prices.

**Maintenance Management**

* **Add new maintenance**: Enables users to record new maintenance activity for cars.
* **Maintenance history**: Allows viewing the history of recorded maintenance for a specific car.

**Car Dealer Management**

* **Add a new car dealer**: Allows users add a new car dealer to the system.
* **List of car dealers**: Displays all dealers currently in the system.

**User Management**

* **List users**: Shows a list of registered users.
* **Register with us**: Allows new users to register for an account.
* **Login to your account**: For existing users to log in.
* **Logout**: Logs the current user out of the system.

**Search & External Tools**

* **Search for cars**: Provides a way to search the car database using filters or keywords.
* **Check the current weather forecast**: Pulls weather info using an API.

**Information & Development Tools**

* **About Car Inventory**: Provides information about the system and its purpose.
* **API Explorer**: Allows users to interact with the system's API, useful for integrations or testing.

**Visual**:

The background features a stylized image of a car on a road, reinforcing the theme.

A screenshot of a search box

AI-generated content may be incorrect.

The screenshot above shows the car searching function of the Car Inventory and allows users to filter and find cars based on certain criteria. The page includes the:

* **Make**: Filters cars by manufacturer or brand.
* **Model**: Filters by specific model name under the chosen model.
* **Colour**: Narrows results by the car colour.
* **Max Price**: Limits the search to cars at or below the specified price.

**Buttons**

* **Search Cars (orange button)**
  + Submits the form and displays search results matching the input criteria.
* **Home (dark button)**
  + Redirects the user back to the main dashboard or home page.

**Security Measures in the Application**

**1. Input Validation**

* **Express Validator**: The application uses the express-validator library to validate incoming data for API endpoints. For example, when adding or updating a car, fields like Make, Model, Price, DealerID, and Condition are checked for presence, type, and allowed values.

**2. Input Sanitization**

* **Sanitization**: The code uses req.sanitize() on all user -supplied input (from req.body, req.params, and req.query) before using it in SQL queries.

**3. Parameterized SQL Queries**

* **Prepared Statements**: All SQL queries use parameterized statements (i.e., ? placeholder values passed as an array).
* This ensures that user input is never directly concatenated into SQL queries, which is a primary defence against SQL injection attacks.

**4. Error Handling**

* **Consistent Error Responses**: The application returns generic error messages for database errors and validation failures, without exposing sensitive details about the database or server internals.

**How to use the API**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Select an Endpoint:**

The API Explorer displays a list of available API endpoints (such as /api/cars, /api/dealers, etc.). Click on the endpoint you want to test.

1. **Choose the HTTP Method:**

For each endpoint, you can select the HTTP method (GET, POST, PUT, DELETE) you want to use.

1. **Enter Parameters or Request Body:**

* For GET requests, you can enter query parameters (e.g., make, model, condition).
* For POST or PUT requests, you can fill in the request body fields (such as car details or dealer information).

1. **Send the Request:**

Click the button, and the API Explorer will send the request to the server.

1. **View the Response:**

The response from the server (including data, success messages, or error details) will be displayed directly in the API Explorer interface. This helps you see exactly what the API returns for your request.

**Username:** AB

**Password:** b