Abdelmalek Essaadi University
Faculty of Sciences and Techniques of Tangier
Department of Computer Engineering

Technical Control and Insurance Management System for Automobiles
A Comprehensive Web Application Using React JS and Spring Boot

Supervised by:

- Pr. MAHBOUB AZIZ

Developed by:

- BOUASSAB Chaimae

Program:

- Master's in IT Security & Big Data

Academic year:

- 2024-2025

General Description:

This project is a comprehensive web application designed to streamline the management of vehicle-related information, including technical inspections, insurance policies, and regular maintenance tasks such as oil changes. The goal is to provide an innovative and user-friendly tool for vehicle owners, technical inspection centers, insurance providers, and administrators. By leveraging modern technologies like React JS and Spring Boot, the application centralizes vehicle data management, automates reminders, and offers personalized recommendations.

Project Objectives:

The primary objectives of this project are as follows:

1. **Simplify and Centralize Vehicle Management:** Provide an intuitive platform for users to manage vehicle-related data efficiently.
2. **Automate Reminders:** Notify users of upcoming deadlines for insurance renewals, technical inspections, and maintenance tasks.
3. **Offer Personalized Recommendations:** Suggest maintenance actions based on vehicle data and usage patterns.
4. **Innovative Features:** Incorporate document-scanning capabilities to simplify administrative processes.

Key Features:

The application includes the following core functionalities:

1. **Vehicle Management**
   - Register vehicles with details such as make, model, chassis number, etc.
   - Update and track vehicle information seamlessly.
2. **Insurance Verification and Payment**
   - Verify insurance document validity through scanned uploads.
   - Facilitate quarterly or annual premium payments.
   - Send notifications for renewal deadlines and generate payment receipts.
3. **Technical Control**
   - Track technical inspection results.
   - Automate reminders for upcoming inspections.
   - Generate detailed reports on safety, emissions, and overall vehicle condition.
4. **History and Tracking**
   - Access a full history of maintenance and technical visits.
   - Archive documents and reports for easy reference.
5. **"Scan Tonobil" Feature**
   - Use Optical Character Recognition (OCR) to extract data from scanned documents.
   - Automatically retrieve detailed vehicle information from external databases.
6. **Notifications and Dashboard**
   - Send reminders for insurance, maintenance, and inspection deadlines.

- Provide an interactive dashboard displaying vehicle status and required actions.

7. **Analysis and Recommendations**
    - Deliver tailored maintenance suggestions based on vehicle data.
    - Utilize Machine Learning (optional implementation) to predict future maintenance needs.

Minimum Viable Product (MVP)Features:

For the initial release, the following functionalities were prioritized:

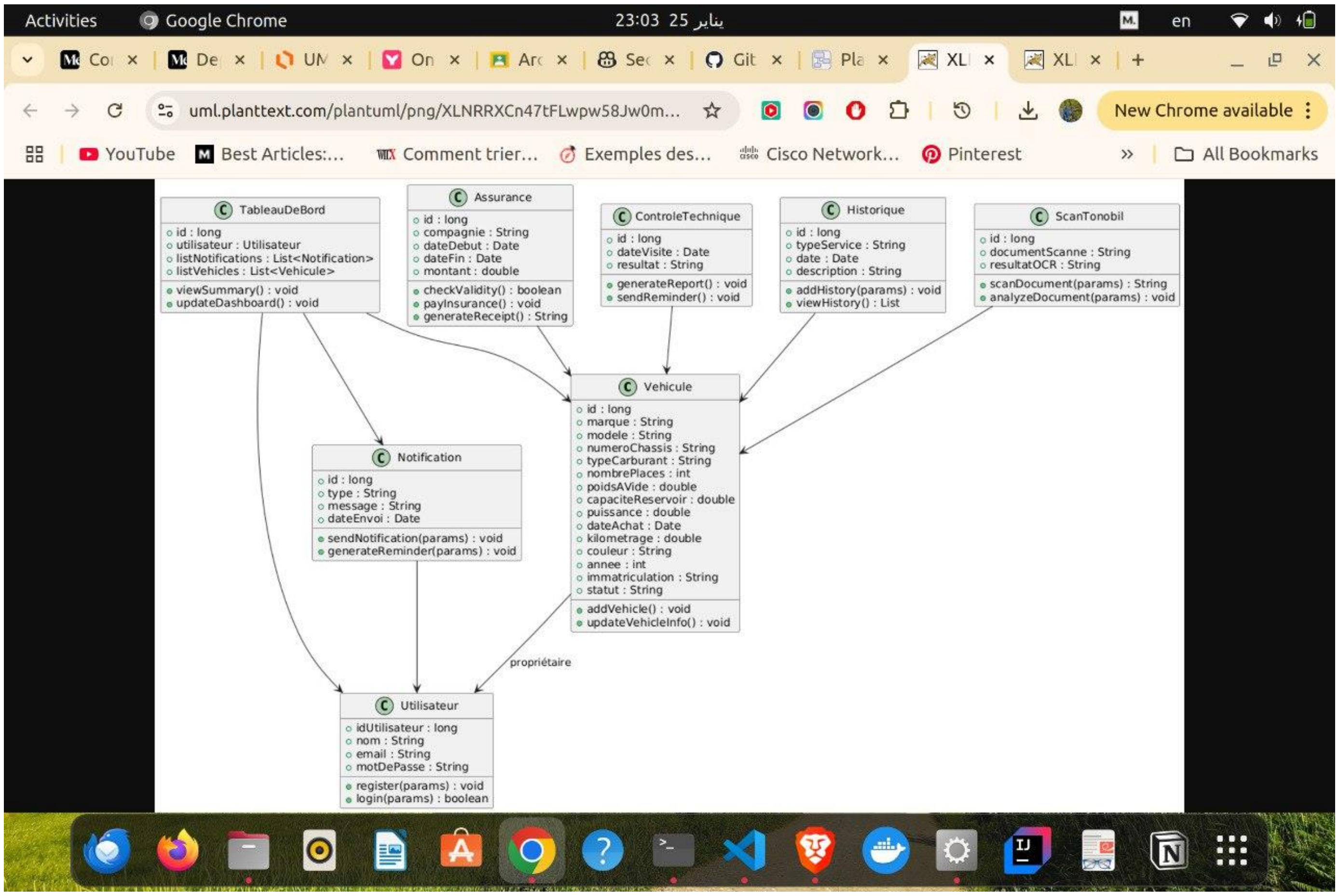
→ Vehicle and client management.

→ Service history tracking (e.g., oil changes, technical inspections).

→ Automated notifications for reminders.

→ Insurance payment and verification system

System Architecture &&Conception:



The class diagram depicts a web application designed to manage vehicle-related data, including technical inspections, insurance, notifications, and user interactions. It includes seven main classes, each representing a critical component of the system. The classes are interconnected to show relationships such as associations and dependencies, reflecting how data flows and interacts within the application. The diagram is structured hierarchically, with the TableauDeBord (Dashboard) class at the top, branching out to other classes like Vehicule, Utilisateur, Assurance, ControleTechnique, Historique, and ScanTonobil.

Relationships Between Classes

TableauDeBord (Dashboard):

- ✓ Associated with Utilisateur (one-to-one, as each dashboard belongs to a user).
- ✓ Contains lists of Vehicule and Notification (one-to-many relationships), allowing users to manage multiple vehicles and receive multiple notifications.

Utilisateur (User):

- ✓ Linked to Vehicule (one-to-many, as a user can own multiple vehicles).
- ✓ Connected to TableauDeBord (one-to-one, as each user has a dashboard).

Vehicule (Vehicle):

- ✓ Associated with Assurance and ControleTechnique (one-to-many, as a vehicle can have multiple insurance policies and inspections over time).
- ✓ Linked to Utilisateur (many-to-one, as multiple vehicles can belong to one user).
- ✓ Related to Historique (one-to-many, as a vehicle has a history of services).
- ✓ Assurance, ControleTechnique, Historique, and ScanTonobil:
- ✓ These classes are associated with Vehicule (one-to-many or many-to-one relationships), tracking specific aspects of vehicle management (insurance, inspections, history, and document scanning).

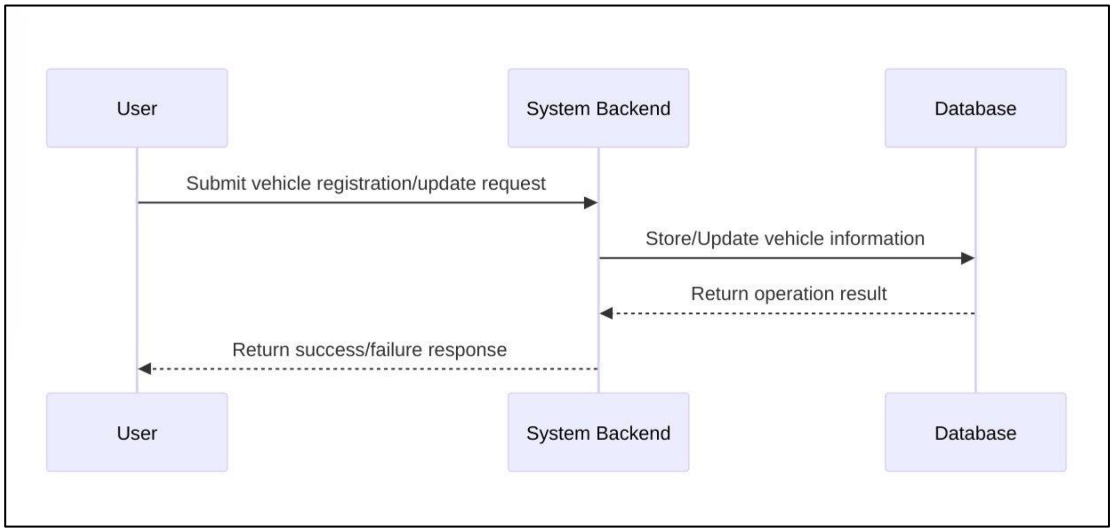
Notification:

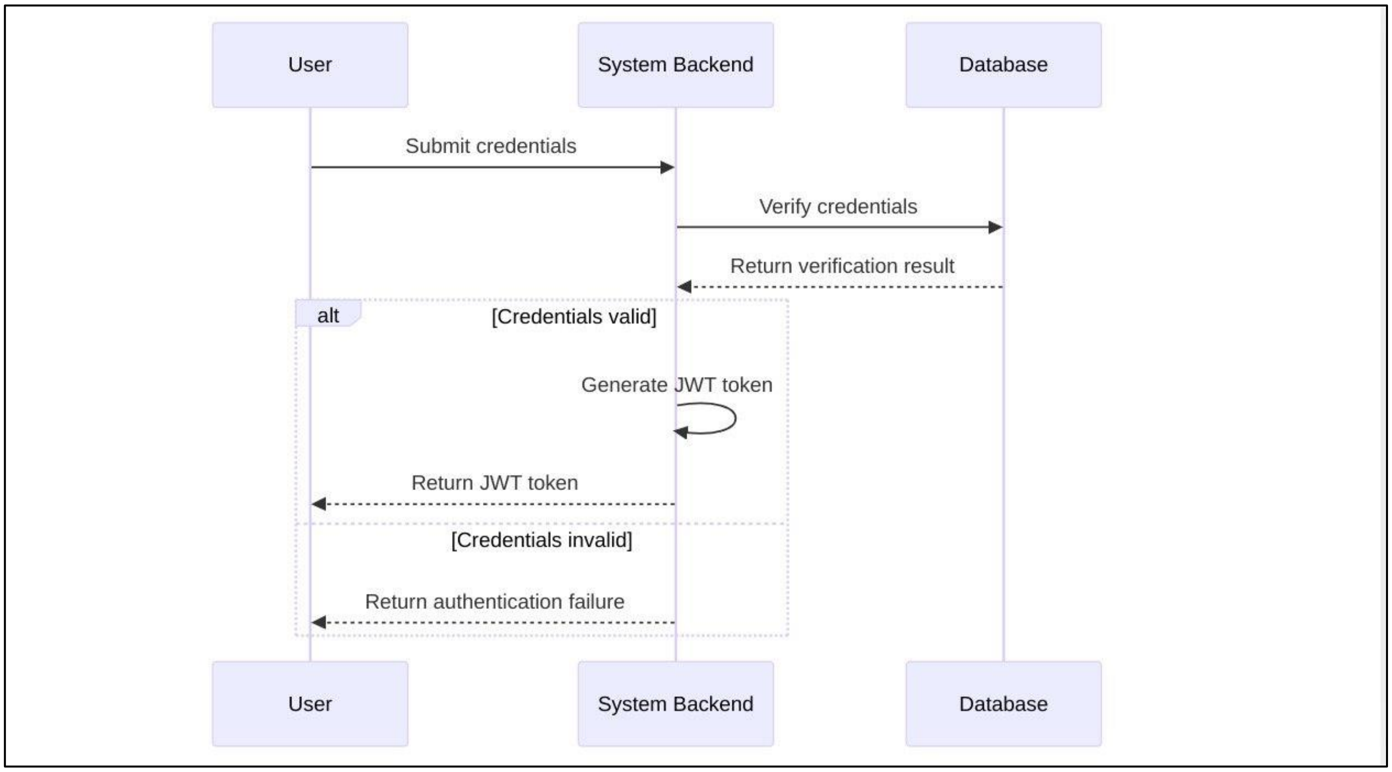
- ✓ Connected to TableauDeBord (many-to-one, as notifications are displayed on the user's dashboard) and potentially triggered by Assurance, ControleTechnique, or other classes for reminders.

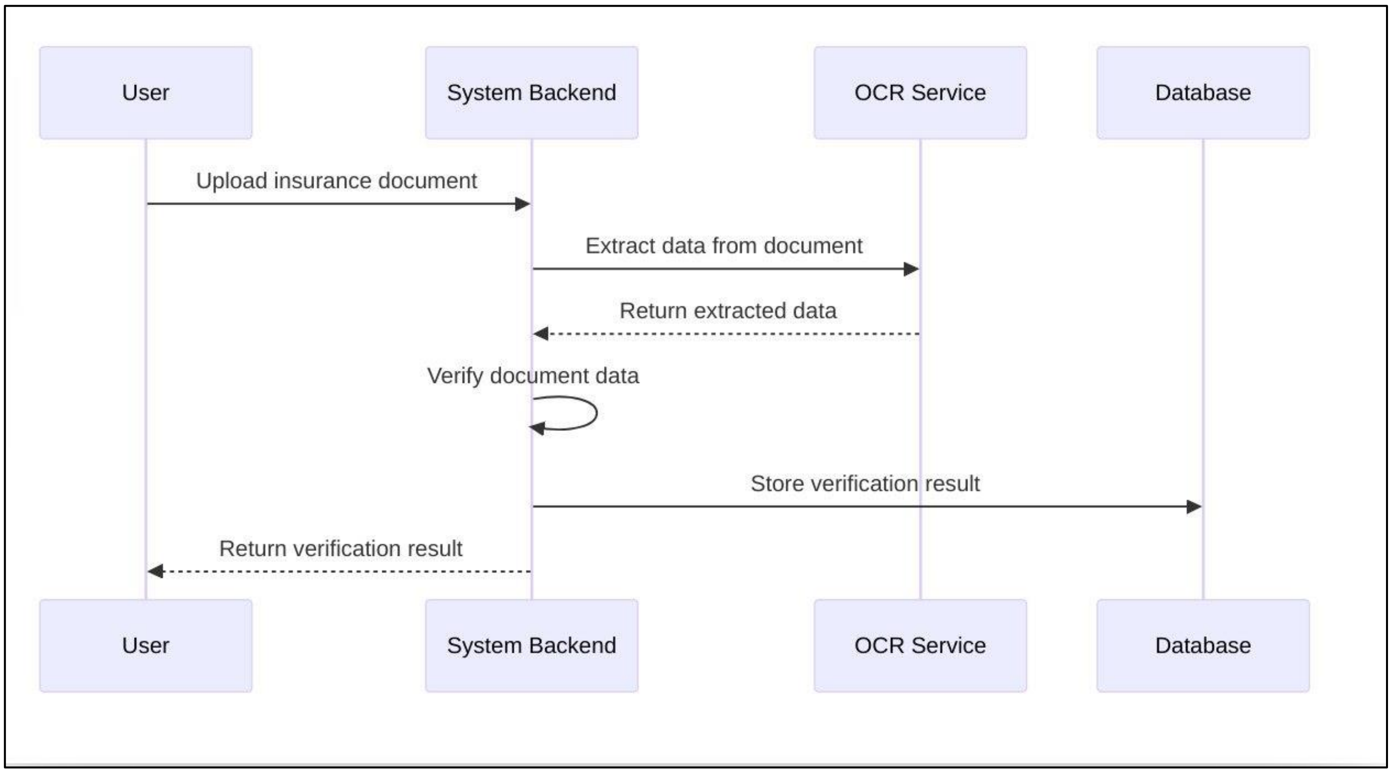
Purpose and Functionality

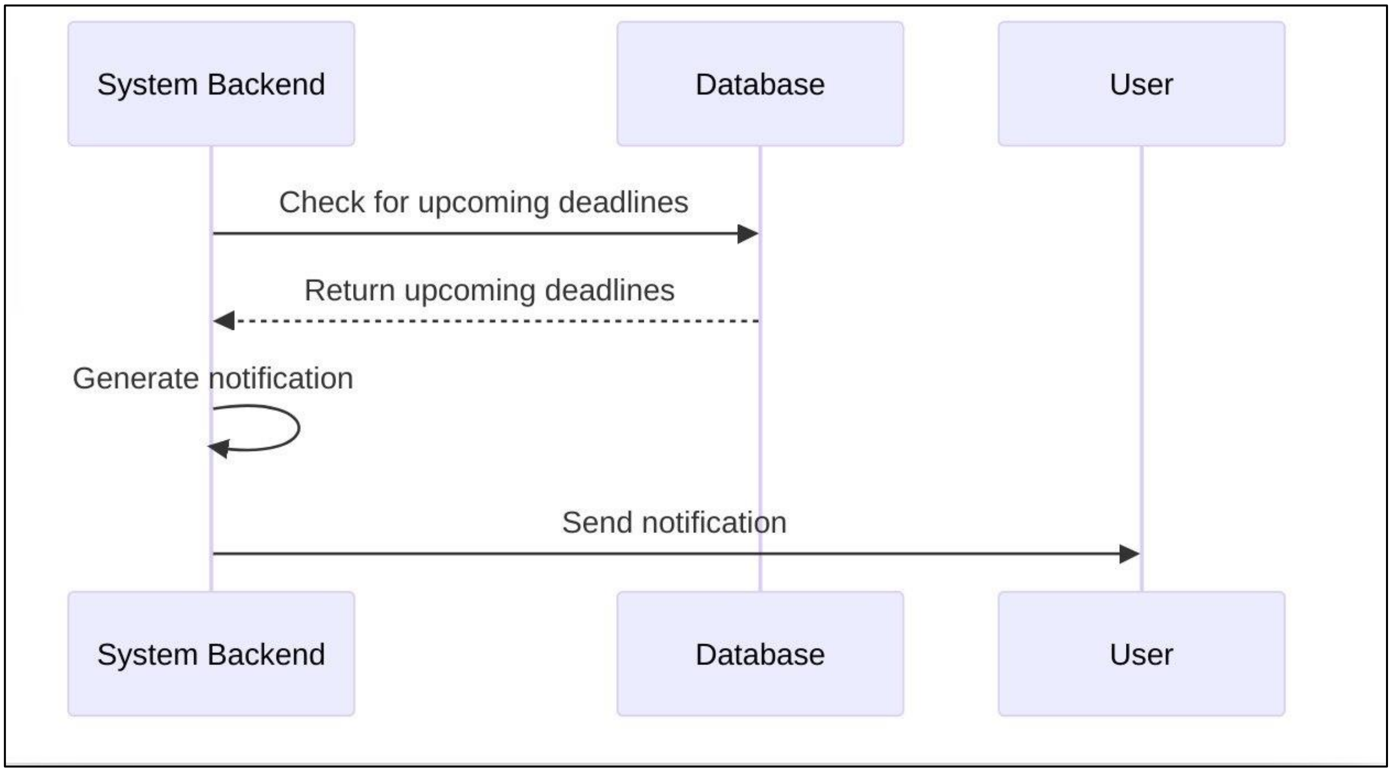
- ✓ This class diagram models the data structure and behavior of a web application built with React JS (frontend) and Spring Boot (backend), as described in your earlier document. It supports the system's key features, such as vehicle registration, insurance management, technical inspections, document scanning (via ScanTonobil), and automated notifications. The diagram ensures that the system can handle complex relationships between users, vehicles, and services while maintaining data integrity and providing a user-friendly experience through the dashboard.

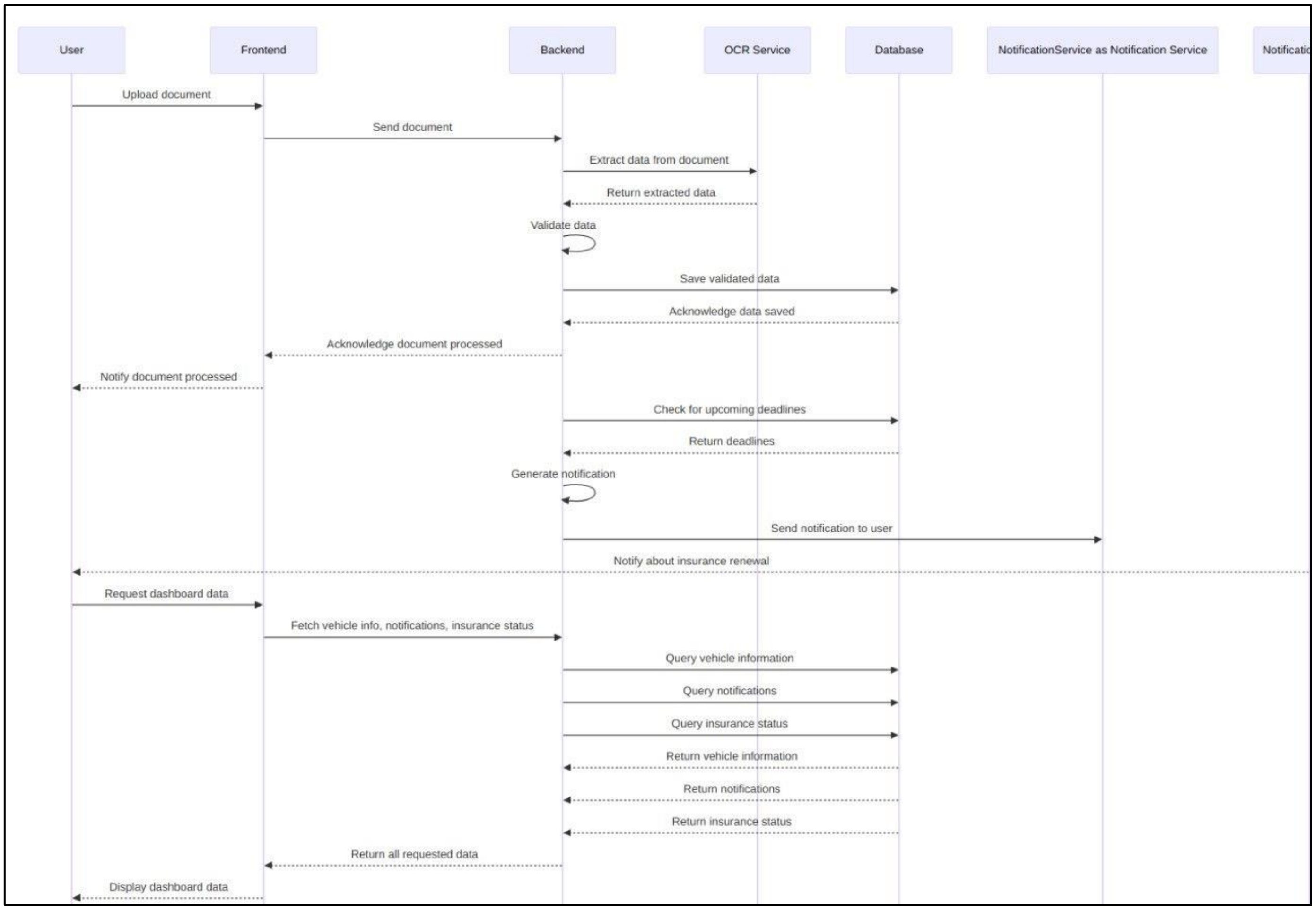
- ✓ This structure aligns with the project's objectives of centralizing vehicle management, automating reminders, and integrating innovative features like OCR-based document processing. The classes and their methods facilitate CRUD (Create, Read, Update, Delete) operations, data validation, and user interaction, making it a robust foundation for the application.


User
System Backend
Database
Submit vehicle registration/update request
Store/Update vehicle information
Return operation result
Return success/failure response
User
System Backend
Database


User
System Backend
Database
Submit credentials
Verify credentials
Return verification result
alt
[Credentials valid]
Generate JWT token
Return JWT token
[Credentials invalid]
Return authentication failure
User
System Backend
Database


User
System Backend
OCR Service
Database
Upload insurance document
Extract data from document
Return extracted data
Verify document data
Store verification result
Return verification result
User
System Backend
OCR Service
Database


System Backend
Database
User
Check for upcoming deadlines
Return upcoming deadlines
Generate notification
Send notification
System Backend
Database
User


User
Frontend
Backend
OCR Service
Database
NotificationService as Notification Service
Notificatio
Upload document
Send document
Extract data from document
Return extracted data
Validate data
Save validated data
Acknowledge data saved
Acknowledge document processed
Notify document processed
Check for upcoming deadlines
Return deadlines
Generate notification
Send notification to user
Notify about insurance renewal
Request dashboard data
Fetch vehicle info, notifications, insurance status
Query vehicle information
Query notifications
Query insurance status
Return vehicle information
Return notifications
Return insurance status
Return all requested data
Display dashboard data

This diagram illustrates the **document processing and dashboard data retrieval workflow** in the system. It represents interactions between different components, including the **User, Frontend, Backend, OCR Service, Database, and Notification Service**.

Process Breakdown:

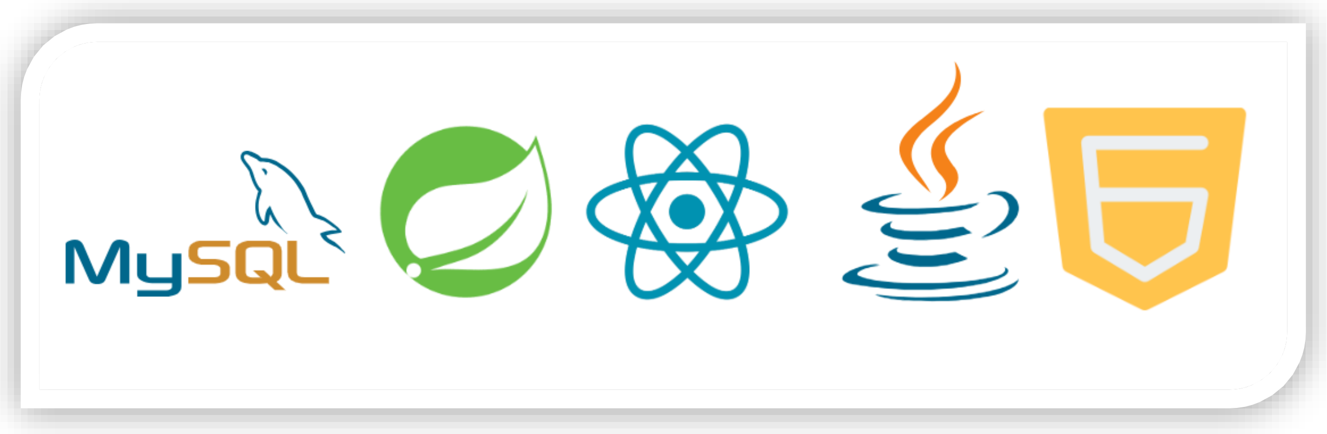
1. **Document Upload & Processing**:
   - The user uploads a document via the frontend.
   - The frontend sends it to the backend, which forwards it to the **OCR Service** for data extraction.
   - Extracted data is validated and saved in the **Database**.
   - Once processed, a notification is sent to inform the user.
2. **Notification Handling**:
   - The system checks for upcoming deadlines (e.g., insurance renewal).
   - If a deadline is found, a notification is generated and sent to the user.

3. **Dashboard Data Retrieval**:
    - The user requests dashboard data.
    - The backend fetches vehicle-related details, notifications, and insurance status from the **Database**.
    - The retrieved data is sent back and displayed on the user's dashboard.

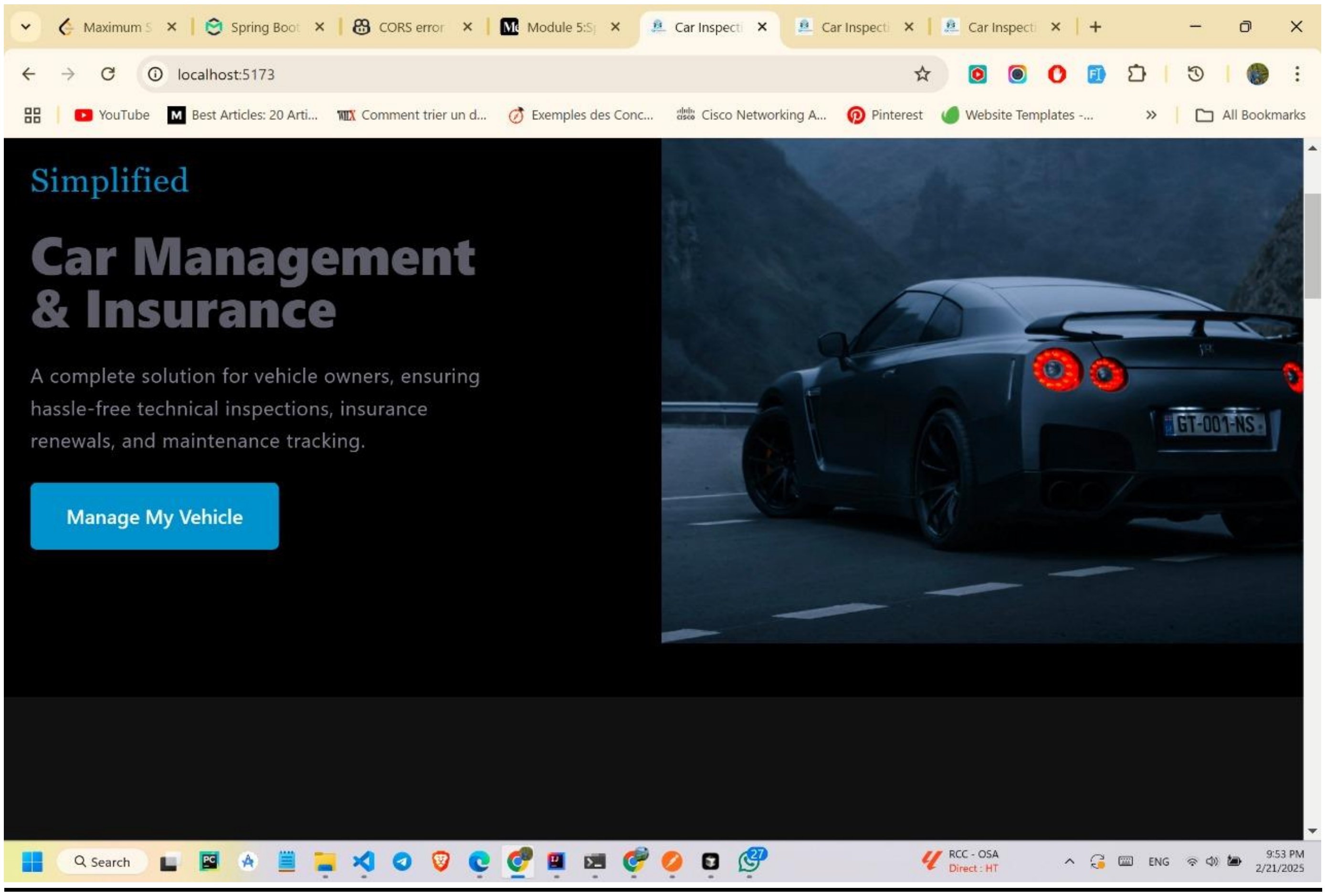
This workflow ensures efficient document processing, timely notifications, and an organized dashboard view for the user.

Technologies used:

- **Frontend**: React.js, Tailwind CSS, Framer Motion, AOS, React Router
- **Backend**: Spring Boot
- **Database**: MYSQL
- **Authentication**: JWT
- **Other Tools**: Postman (for API testing), GitHub (version control)



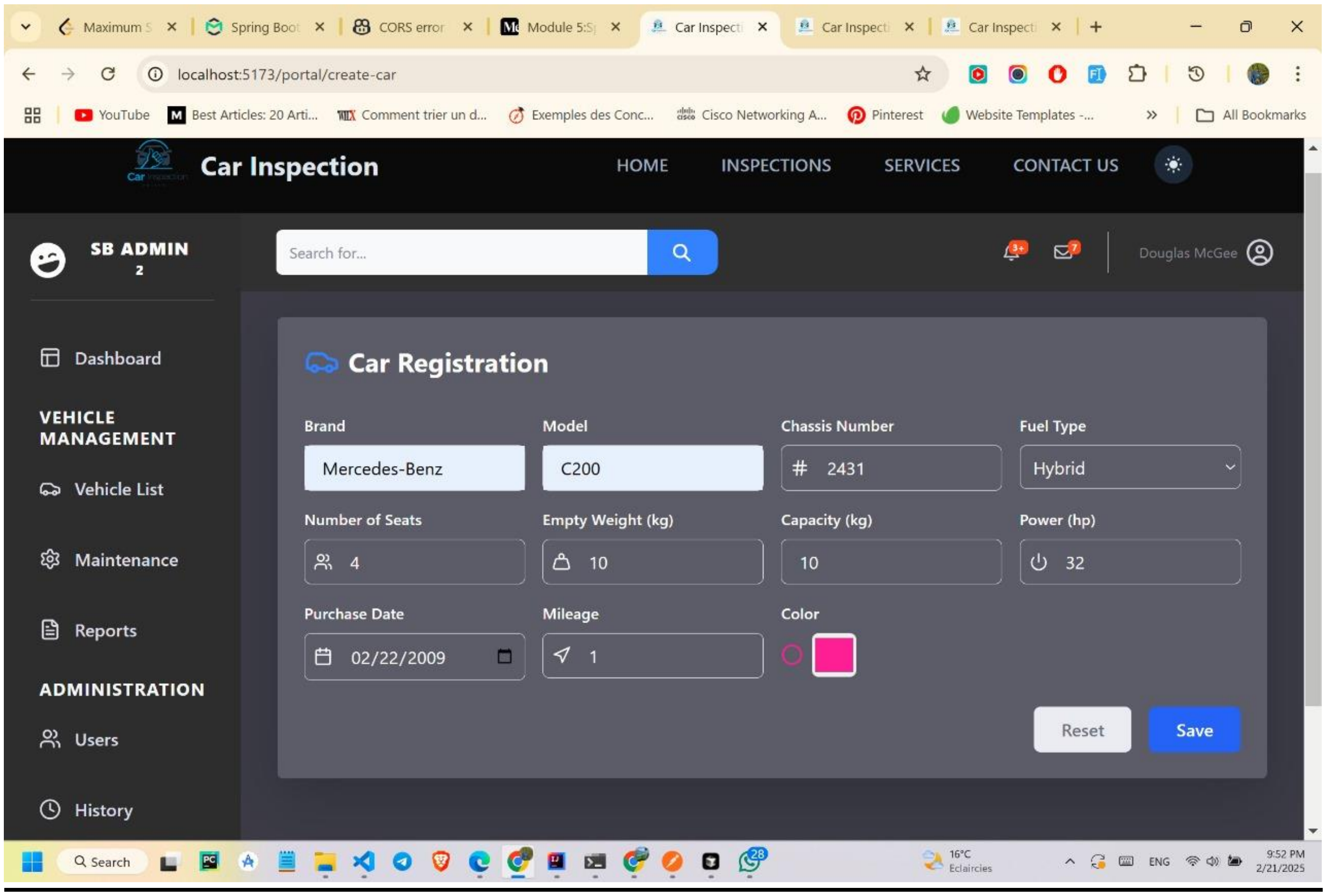
Interfaces:



Simplified

Car Management & Insurance

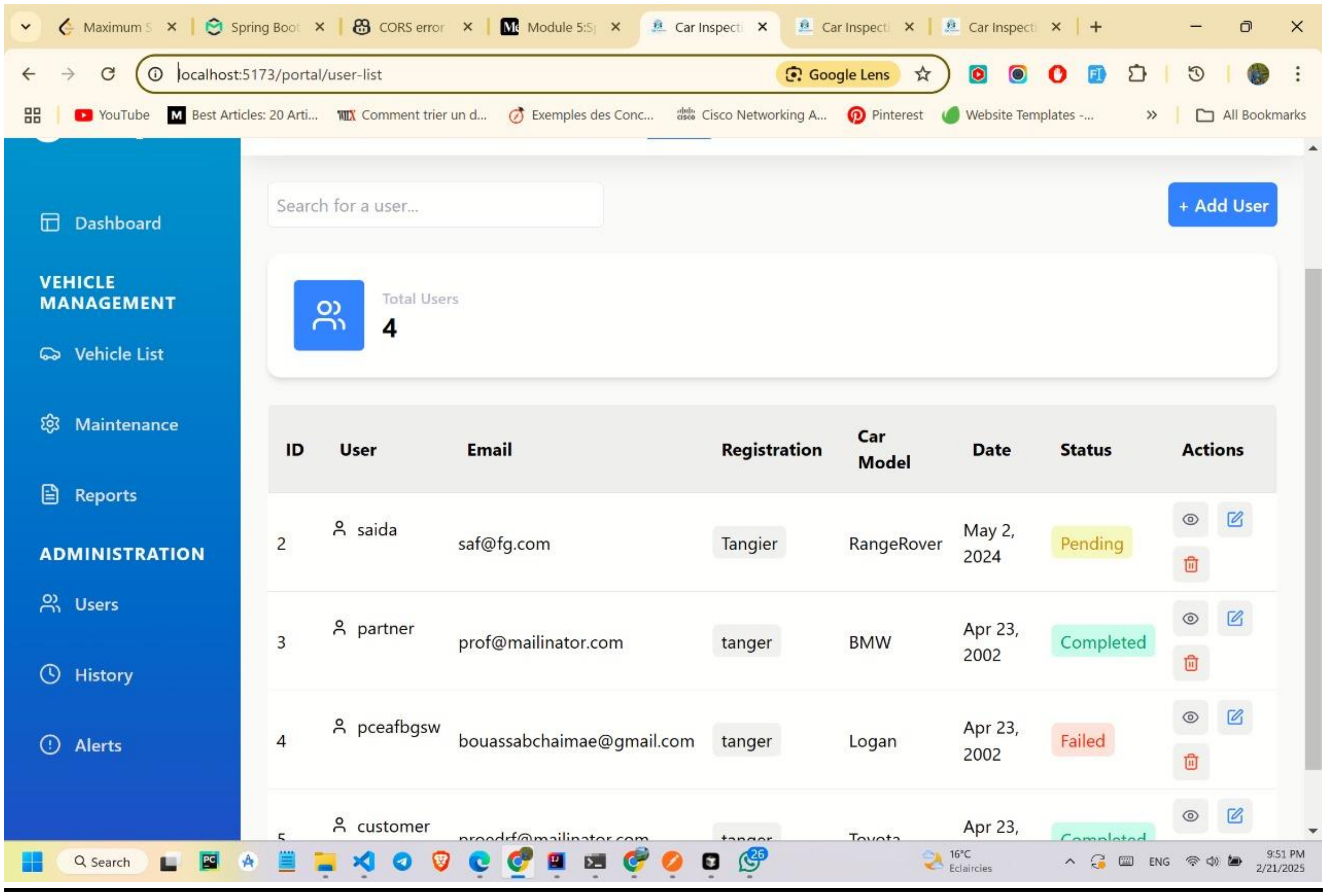
A complete solution for vehicle owners, ensuring hassle-free technical inspections, insurance renewals, and maintenance tracking.



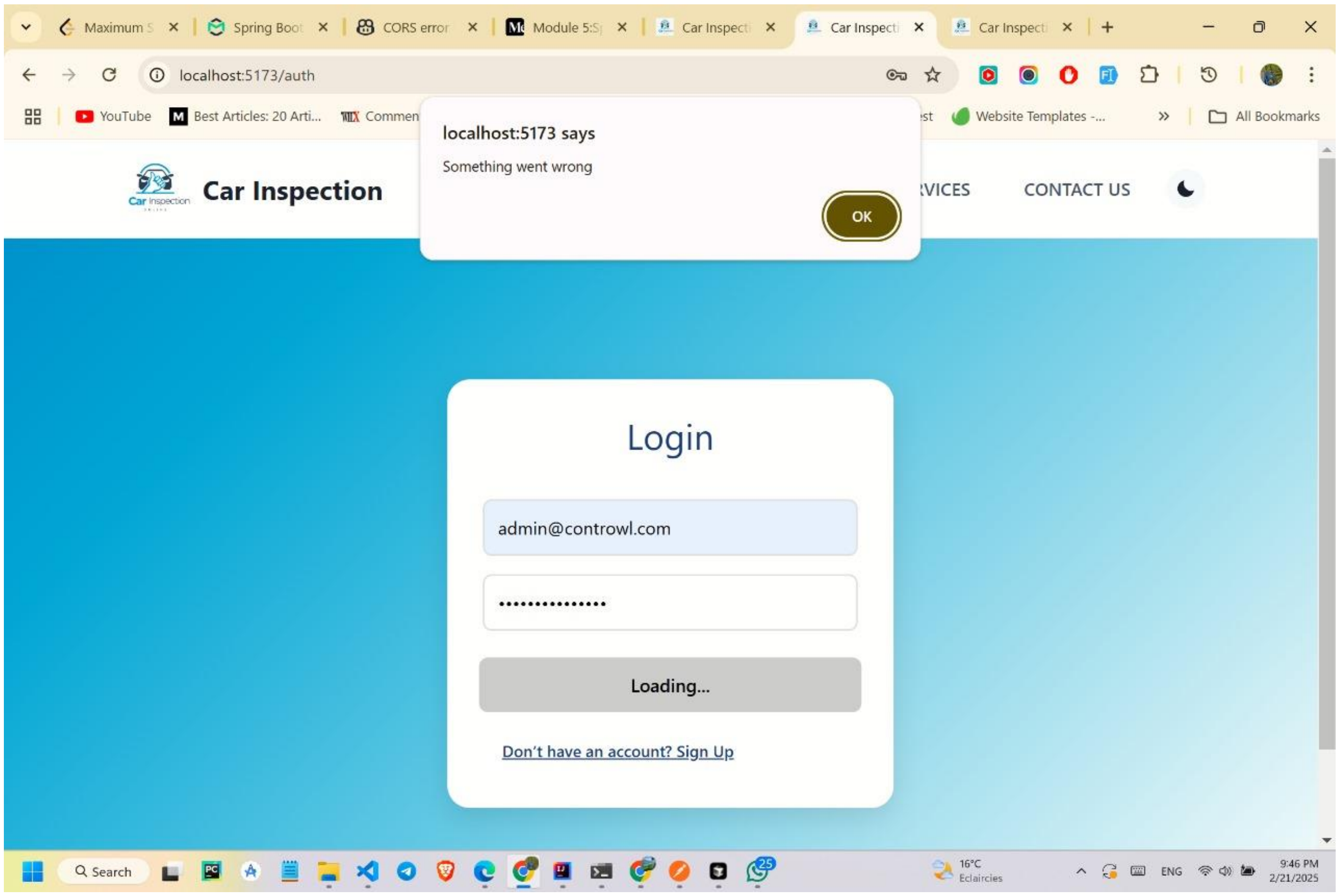
Car Registration

Brand	Model	Chassis Number	Fuel Type
Mercedes-Benz	C200	2431	Hybrid
Number of Seats	Empty Weight (kg)	Capacity (kg)	Power (hp)
4	10	10	32
Purchase Date	Mileage	Color	
02/22/2009	1	🟥	

Reset Save



ID	User	Email	Registration	Car Model	Date	Status	Actions
2	saida	saf@fg.com	Tangier	RangeRover	May 2, 2024	Pending	
3	partner	prof@mailinator.com	tanger	BMW	Apr 23, 2002	Completed	
4	pceafbgsw	bouassabchaimae@gmail.com	tanger	Logan	Apr 23, 2002	Failed	
5	customer				Apr 23,		

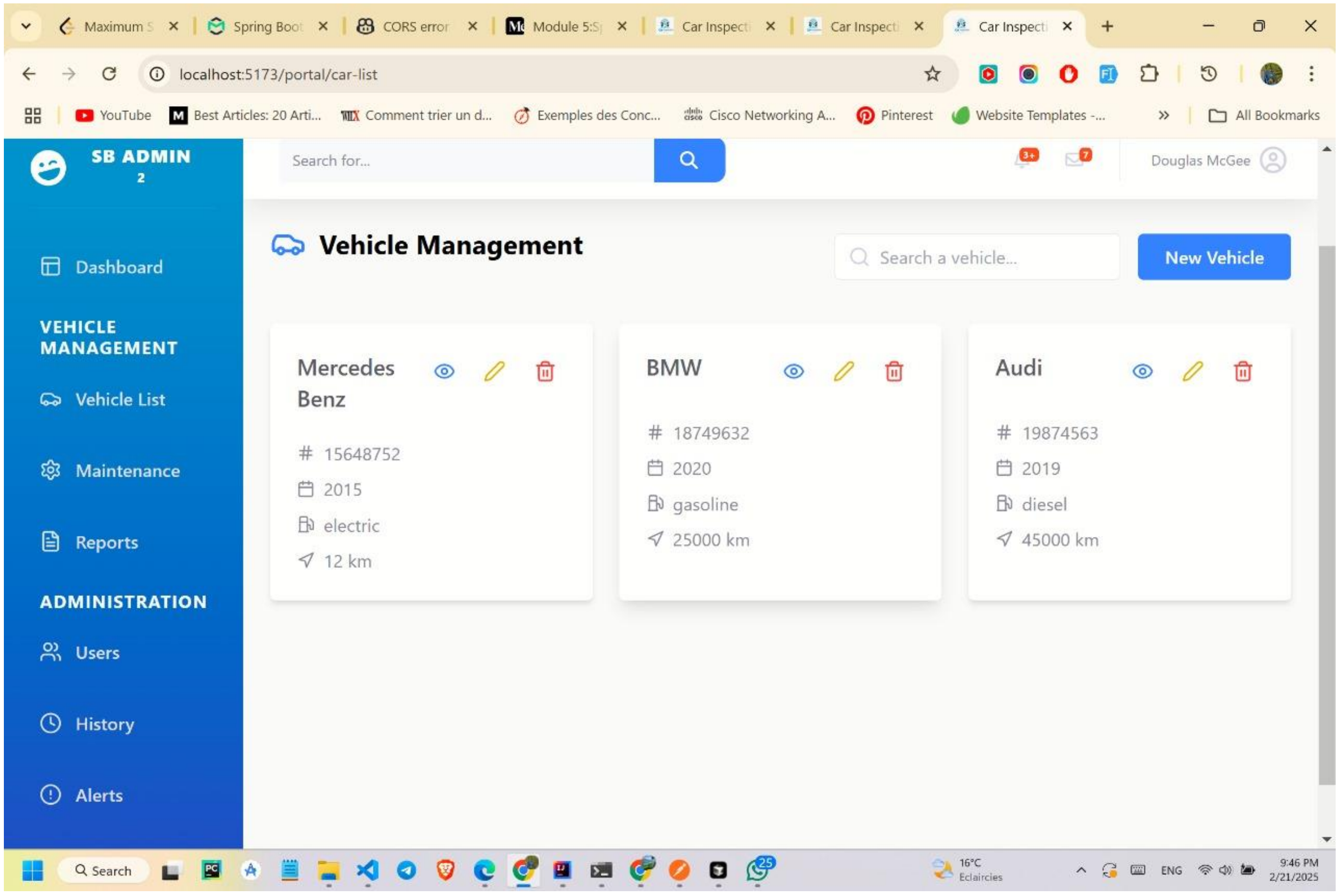


localhost:5173 says

Something went wrong

OK

admin@controwl.com



Vehicle Management

Mercedes Benz	# 15648752	2015	electric	12 km
BMW	# 18749632	2020	gasoline	25000 km
Audi	# 19874563	2019	diesel	45000 km

Authentication Endpoints (UtilisateurController):

POST /api/utilisateurs/register

- Body: UserRegistrationDTO (nom, email, motDePasse)

POST /api/utilisateurs/login

- Body: UserLoginDTO (email, motDePasse)

POST /api/utilisateurs/logout

Vehicle Endpoints (VehiculeController)

GET /api/vehicules

- Requires: Authentication

GET /api/vehicules/{id}

- Requires: Authentication
- Path Variable: id (Long)

POST /api/vehicules

- Requires: Authentication + USER or ADMIN role
- Body: Vehicule object
- Query Param: utilisateurId (Long)

DELETE /api/vehicules/{id}

- Requires: Authentication
- Path Variable: id (Long)

User Management Endpoints (UtilisateurController)

GET /api/utilisateurs/users

- Requires: Authentication

GET /api/utilisateurs/{id}

- Requires: Authentication
- Path Variable: id (Long)

POST /api/utilisateurs/add

- Requires: Authentication
- Body: Utilisateur object

DELETE /api/utilisateurs/{id}

- Requires: Authentication + Admin token
- Path Variable: id (Long)
- Header: Authorization (Bearer token)

Admin Endpoints (AdminController)

GET /api/admin/users

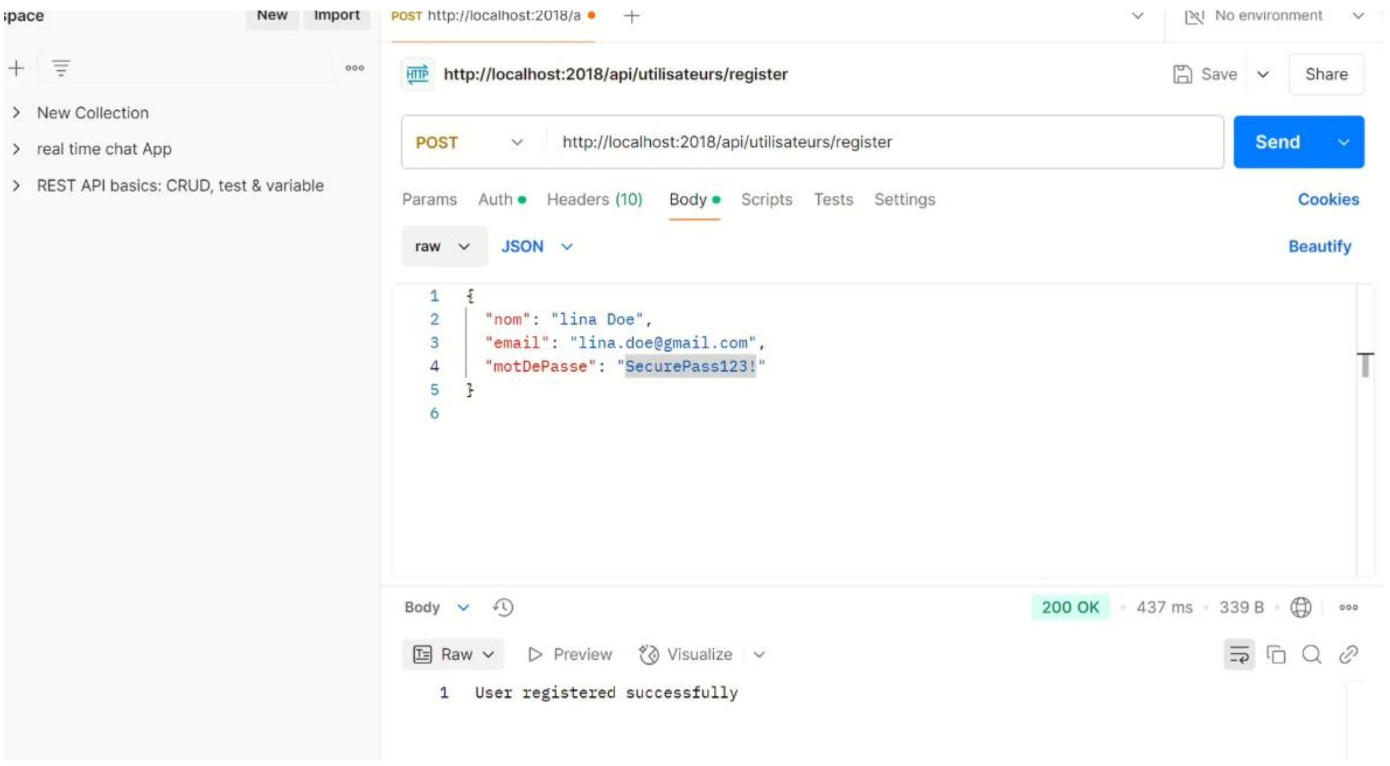
- Requires: ADMIN role

PUT /api/admin/users/{id}

- Requires: ADMIN role
- Path Variable: id (Long)
- Body: UserManagementDTO

POST /api/admin/users/admin

- Requires: ADMIN role
- Body: UserRegistrationDTO

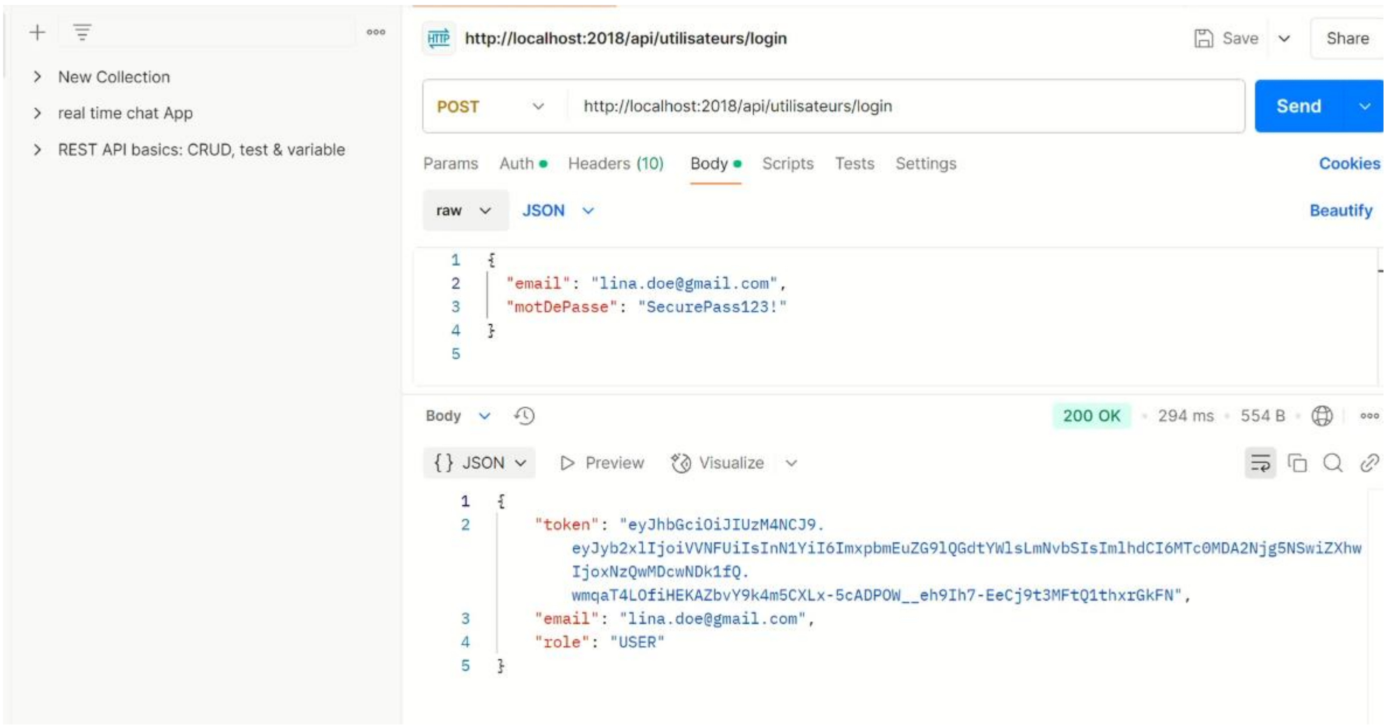


POST http://localhost:2018/api/utilisateurs/register

```
{
  "nom": "lina Doe",
  "email": "lina.doe@gmail.com",
  "motDePasse": "SecurePass123!"
}
```

200 OK · 437 ms · 339 B

```
User registered successfully
```



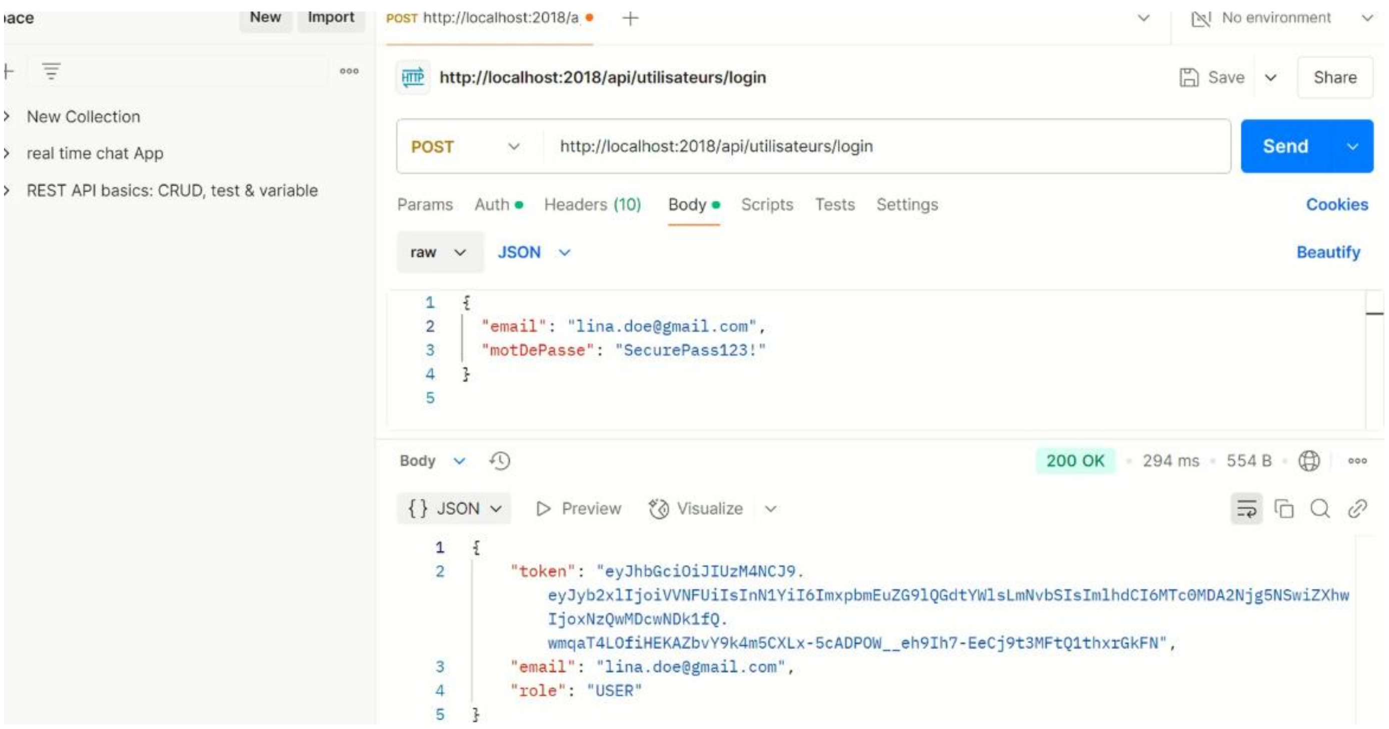
http://localhost:2018/api/utilisateurs/login

POST http://localhost:2018/api/utilisateurs/login

```
{
  "email": "lina.doe@gmail.com",
  "motDePasse": "SecurePass123!"
}
```

200 OK · 294 ms · 554 B

```
{
    "token": "eyJhbGciOiJIUzM4NCJ9.
        eyJyb2xlIjoiVVNFUiIsInN1YiI6ImxpbmEuZG9lQGdtYWlsLmNvbSIsImlhdCI6MTc0MDA2Njg5NSwiZXhw
        IjoxNzQwMDcwNDk1fQ.
        wmqaT4LOfiHEKAZbvY9k4m5CXLx-5cADPOW__eh9Ih7-EeCj9t3MFtQ1thxrGkFN",
    "email": "lina.doe@gmail.com",
    "role": "USER"
}
```



POST http://localhost:2018/a

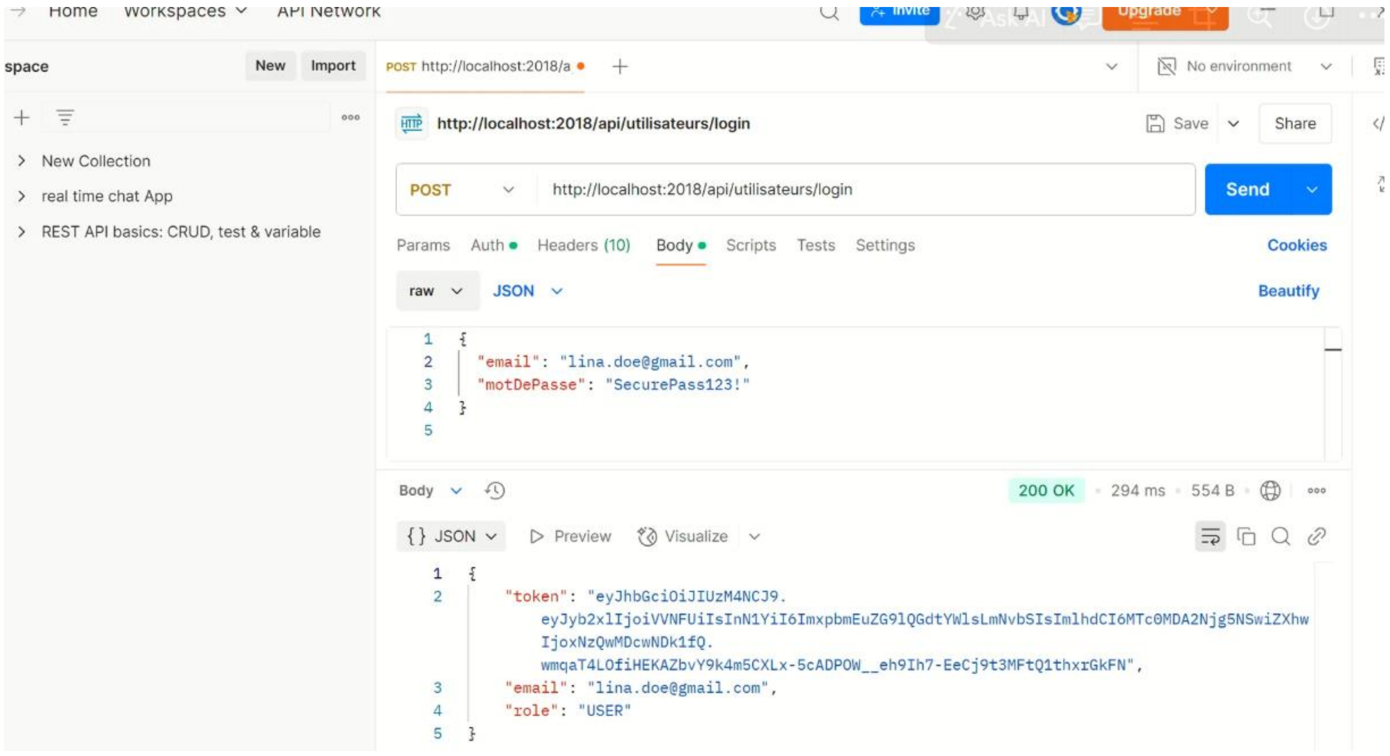
http://localhost:2018/api/utilisateurs/login

POST http://localhost:2018/api/utilisateurs/login

```
{
  "email": "lina.doe@gmail.com",
  "motDePasse": "SecurePass123!"
}
```

200 OK · 294 ms · 554 B

```
{
    "token": "eyJhbGciOiJIUzM4NCJ9.
        eyJyb2xlIjoiVVNFUiIsInN1YiI6ImxpbmEuZG9lQGdtYWlsLmNvbSIsImlhdCI6MTc0MDA2Njg5NSwiZXhw
        IjoxNzQwMDcwNDk1fQ.
        wmqaT4LOfiHEKAZbvY9k4m5CXLx-5cADPOW__eh9Ih7-EeCj9t3MFtQ1thxrGkFN",
    "email": "lina.doe@gmail.com",
    "role": "USER"
}
```



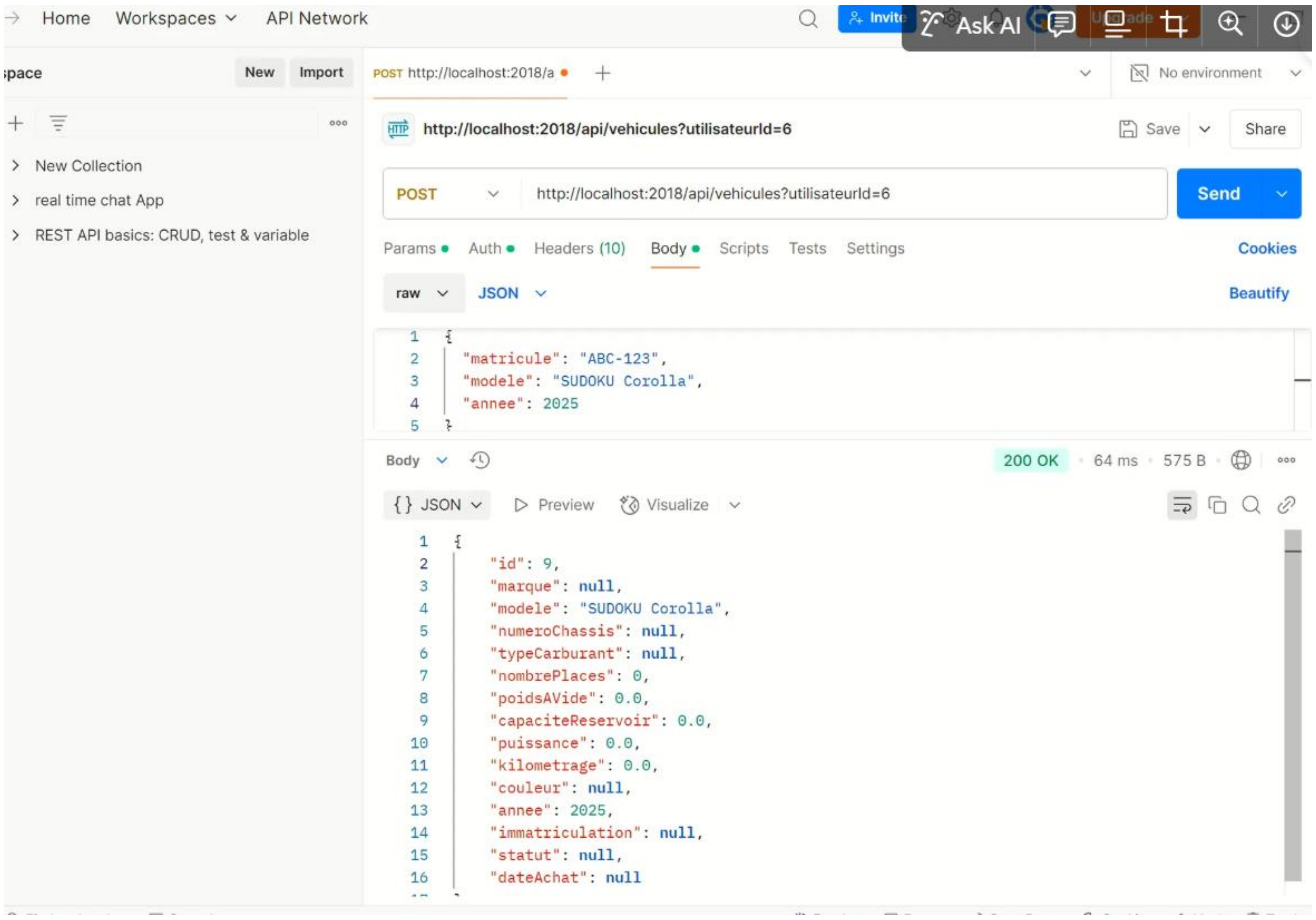
http://localhost:2018/api/utilisateurs/login

POST http://localhost:2018/api/utilisateurs/login

```
{
  "email": "lina.doe@gmail.com",
  "motDePasse": "SecurePass123!"
}
```

200 OK · 294 ms · 554 B

```
{
    "token": "eyJhbGciOiJIUzM4NCJ9.
        eyJyb2xlIjoiVVNFUiIsInN1YiI6ImxpbmEuZG9lQGdtYWlsLmNvbSIsImlhdCI6MTc0MDA2Njg5NSwiZXhw
        IjoxNzQwMDcwNDk1fQ.
        wmqaT4LOfiHEKAZbvY9k4m5CXLx-5cADPOW__eh9Ih7-EeCj9t3MFtQ1thxrGkFN",
    "email": "lina.doe@gmail.com",
    "role": "USER"
}
```



POST http://localhost:2018/api/vehicules?utilisateurId=6

```
{
  "matricule": "ABC-123",
  "modele": "SUDOKU Corolla",
  "annee": 2025
}
```

```
{
    "id": 9,
    "marque": null,
    "modele": "SUDOKU Corolla",
    "numeroChassis": null,
    "typeCarburant": null,
    "nombrePlaces": 0,
    "poidsAVide": 0.0,
    "capaciteReservoir": 0.0,
    "puissance": 0.0,
    "kilometrage": 0.0,
    "couleur": null,
    "annee": 2025,
    "immatriculation": null,
    "statut": null,
    "dateAchat": null
```

Project Structure

The project follows a standard Spring Boot architecture with the following main packages:

- **config**: Configuration classes for CORS, security, and database settings
- **controller**: HTTP request handlers and route definitions
- **dto**: Data Transfer Objects for request/response data encapsulation
- **entities**: Database model classes mapped to tables
- **exception**: Custom exception handling
- **mappers**: Entity-DTO conversion utilities
- **repositories**: JPA interfaces for database operations
- **security**: Authentication and authorization components
- **service**: Business logic implementation

Conclusion :

The **Car Inspection Management System** was developed to streamline the process of vehicle inspections, ensuring better efficiency, transparency, and ease of access for both administrators and vehicle owners. The system integrates a **Spring Boot** backend with a **React.js** frontend, providing a seamless user experience while maintaining robust security and data management through JWT authentication and a PostgreSQL database.

Through the implementation of key functionalities such as **user authentication, vehicle registration, reservation management, and document handling**, the platform enables administrators to monitor vehicle inspections effectively while allowing users to schedule their own inspections and track their vehicle's status. The integration of **role-based access control (RBAC)** ensures that only authorized users can perform specific actions, enhancing system security.

During the development process, challenges such as **binding the frontend with the backend, authentication errors, and database schema adjustments** were encountered and resolved, improving the overall system's reliability and functionality. Additionally, features like **file upload for document storage, notification handling, and search filtering** were added to enhance usability.

In the future, further enhancements such as **real-time updates, AI-based inspection recommendations, and advanced reporting dashboards** could be integrated to improve decision-making and user experience. This project demonstrates the potential of leveraging modern web technologies to optimize administrative processes in the automotive industry while ensuring compliance with inspection regulations.