Fleet Management System - Technical documentation

1. Introduction

Fleet Management System is a web application designed for efficient management of PONG's fleet. The aim of the system is to eliminate manual vehicle and trip order records and provide authorized users with easy access to all relevant information.

The application allows:

- Vehicle administration (adding, updating, deleting).
- Records of trip orders with details about the driver, route and trip status.
- Generating reports on vehicle usage within a defined time period.
- Different levels of user access, with administrators having full control over data, while regular users can enter trip orders and view data.

The system is built as a modern web application, divided into frontend and backend, which communicate with each other via a REST API.

Key benefits of the system include:

- Digitization of the fleet management process.
- Increased accuracy and availability of data.
- Efficient monitoring of vehicle status and trip orders.
- Reduction of administrative errors and manual work.

The following documentation describes in detail the technologies used, functionalities, system architecture, API endpoints, and the installation process.

2. Technologies

The Fleet Management System was developed using modern JavaScript ecosystem, based on React.js for frontend and Node.js with Express.js for the backend. The data is stored in PostgreSQL database, hosted on Neon.tech The application is distributed via the service Vercel(frontend) and Railway(backend).

2.1 Frontend

The frontend is developed as Single Page Application (SPA) using:

- React.js–JavaScript library for building interactive user interfaces.
- Tailwind CSS–Utility-first CSS framework for fast and modern styling.
- React Router–Managing routing within the application.
- Axios–HTTP client for sending requests to the backend API.

The frontend application is hosted on Vercel, which allows automatic deployment from the GitHub repository.

2.2 Backend

The backend application is based on Node.js and Express.js, with the following technologies:

- Express.js–Minimalist web framework for API development.
- PostgreSQL–Relational database for data storage.
- JSON Web Token (JWT)–It is used for user authentication and authorization.
- CORS middleware–It allows secure access to the API from different domains.
- Railway–Cloud hosting for backend service.

2.3 Database

The system uses PostgreSQL, a powerful relational database, hosted at Neon.tech, which enables: automatic backup and scalability, high database availability, and secure and fast query processing.

The database contains three key tables:

- Vehicles–Contains vehicle data.
- Trips–Records of trip orders.
- User–User data (admin).

2.4 Hosting and Deployment

- Frontend (React.js) → Vercel–Automatic deployment directly from GitHub. Link
- Backend (Node.js, Express.js) → Railway–Cloud hosting of backend applications.
- Database (PostgreSQL) → Neon.tech–Cloud database with automatic scaling.

This technological architecture enables high performance, flexibility and easy system maintenance.

3. Functionalities

Fleet Management System enables fleet administration, trip order recording and report generation. The system uses data validation to ensure the accuracy of entries and prevent irregularities.

3.1 Authentication and authorization

The system uses JWT authentication to ensure secure access to the API.
Users have different levels of access:

- The administrator can manage vehicles and trip orders.
- A regular user can create and view trip orders, but cannot edit vehicles.

User login is done through the login form, and the system checks the correctness of the credentials before granting access.

3.2 Vehicle management

Administrators have full control over vehicle data and can:

- Add new vehicles by entering information such as make, model, chassis number, engine power and fuel type.
- Update existing vehicles.
- Delete vehicles if they do not have active trip orders.
- View a list of all vehicles with all relevant information.

Ordinary users can only view vehicles, without the possibility of modifications.

Validation during vehicle entry:

- Check that the chassis number and engine number are not already entered into the system.
- The year of manufacture must be between 1900 and the current year.
- Engine power (kW and HP) must be greater than 0.

3.3 Management of trip orders

Trip orders represent the trip records of each vehicle.

Regular users can:

- Add new trip orders.
- View a list of your trip orders.

Administrators can additionally:

- Update trip order status.
- Delete trip orders if they are not in "Completed" status.

Trip order statuses:

- Recorded – The account has been created, but not yet confirmed.
- Confirmed – The vehicle has been collected and the order is active.
- Rejected – The order was not approved.

- Completed – The trip is complete.

Validation when entering a trip order:

- All fields are required.
- The start of the period must be before the end of the period.
- The number of passengers must be greater than 0.
- The driver's name must contain only letters and spaces.
- The origin and destination location cannot be the same.
- A vehicle must be selected from the list of available vehicles.
- It is not possible to create a new trip order for a vehicle that has already been reserved (if there is an active order in the "Recorded" or "Confirmed" status).

3.4 Generating vehicle usage reports

The system allows for an overview of vehicle utilization over a certain period of time.

The report is generated in the form of a table showing relevant vehicle usage data. Users can download the report in PDF format for archiving or further analysis.

4. System architecture

Fleet Management System is based on a modern architecture that includes frontend, backend and database. The system uses REST API for communication between the frontend and backend layers, which allows for modularity and easy maintenance.

4.1 Architecture description

The system architecture is divided into three basic layers:

1. **Frontend (React.js, Tailwind CSS)**

- Implemented using React.js for dynamic UI rendering. Use Tailwind CSS for quick and easy styling of elements.
- It communicates with the backend via REST API calls.
- Deployed on the Vercel platform, enabling fast and secure delivery to users.

2. **Backend (Node.js, Express.js)**

The backend is developed in the Node.js framework with Express.js for handling HTTP requests

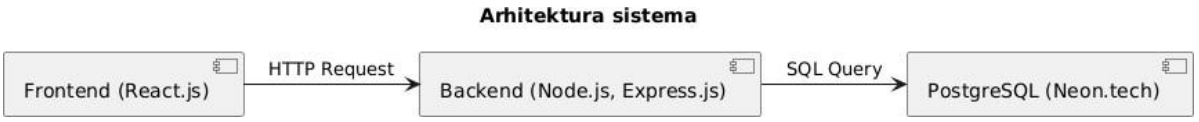
- Implements authentication and authorization using JWT (JSON Web Token).
- Maintains a REST API that enables CRUD operations on vehicles and trip orders.
- Deployed on the Railway platform for scalability and ease of maintenance.

3. **Database (PostgreSQL – Neon.tech)**

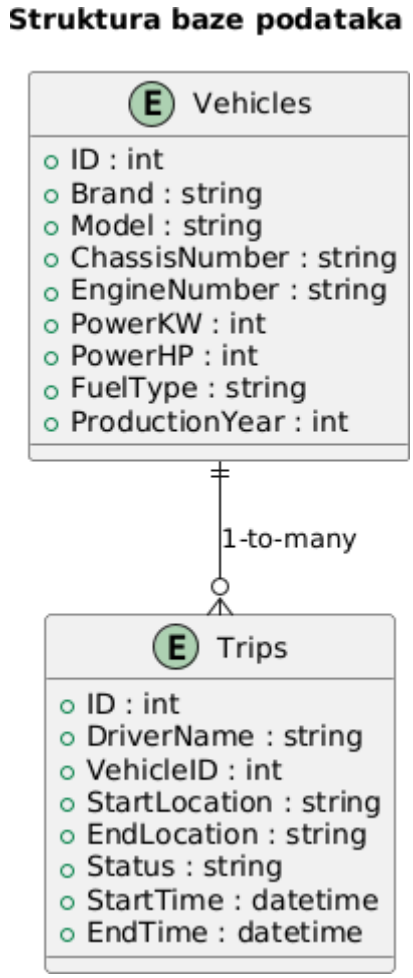
- PostgreSQL database hosted on Neon.tech.
- It contains two main tables: vehicles (for vehicle records) and trips (for trip order).

- It uses relational relationships to enable linking of vehicle and trip order data.

4.2 Architecture diagram



4.3 ERD diagram



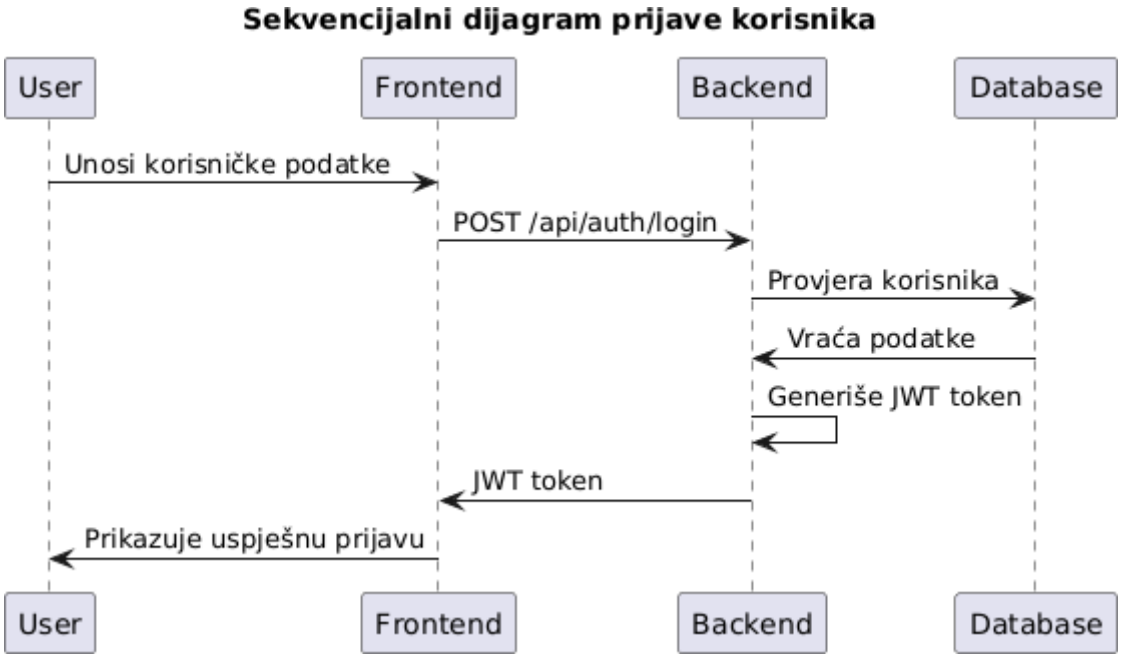
4.4 Description of communication between layers

- Frontend - sends HTTP requests to the backend using the REST API.
- Backend - processes requests, authenticates users via JWT tokens, and communicates with the database to perform CRUD operations.
- Database - It stores all information about vehicles and trip orders and enables data retrieval and manipulation.

4.5 Safety measures

- JWT authentication ensures access only to authorized users.

- Role-based Access Control (RBAC) differentiates privileges between administrators and regular users.
- PostgreSQL uses data encryption and security measures to protect data.



This architecture enables an efficient and scalable fleet management solution, with easy maintenance and the possibility of future functionality expansion.

5. Installation and startup

5.1 Cloning a project

To download and run the project, use the following commands:

git clone https://github.com/sejooo1/fleetmanagement.git
cd fleet-management

5.2 Starting the backend

cd backend
npm install
node index.js

5.3 Starting the frontend

cd frontend
npm install
npm start

6. API Endpoints

6.1 Authentication

Method	Route	Description
POST	/api/auth/login	User login

6.2 Vehicles

Method	Route	Description
GET	/api/vehicles	Retrieving all vehicles
POST	/api/vehicles	Adding a vehicle (admin)
ROAD	/api/vehicles/:id	Vehicle update (admin)
DELETE	/api/vehicles/:id	Delete vehicle (admin)

6.3 Trip orders

Method	Route	Description
GET	/api/trips	Retrieving all trip orders
POST	/api/trips	Adding trip orders
ROAD	/api/trips/:id/status	Update account status (admin)
DELETE	/api/trips/:id	Delete trip order (admin)

6.4 Creating reports

Method	Route	Description
GET	/api/reports/usage	Creating a report

7. Conclusion

Fleet Management System enables efficient fleet management and offers a simple user experience through a modern frontend design and reliable backend logic. Further improvements may include adding notifications and better report analytics.