# **Car Listing Application**

## **About**

This project is part of the **Prüfungsvorleistung | Winter 2024/25** for the **Middleware Technology** course at Hochschule für Technik Stuttgart. It is designed for educational purposes and demonstrates the development of a modern web application for managing cars. The application supports the insertion, updating, deletion, and listing of cars.

## **Team Overview**

* **Atharva Chaudhari**
* **Hardik Kathiriya**

## **Project Overview**

### **Frontend**

* **Framework**: React.js
* **Features**:
  + List cars
  + Add cars
  + Update cars
  + Delete cars
* **Data Management**: Fetches car data from the backend

### **Backend**

* **Framework**: Java Spring Boot
* **Features**:
  + Simplified development with auto-configuration and embedded servers
  + RESTful API for managing cars (GET, POST, DELETE)
  + Seamless integration with various databases and tools
* **Database**: MySQL (used with Java Spring Boot)

### **Technologies Used**

* **Frontend**: React.js
* **Backend**: Java Spring Boot
* **Database**: MySQL
* **Containerization**: Docker, Docker Compose

## **Documentation: 12-Factor Principles**

### **Codebase**

The project utilizes a single codebase managed in a version control system (GitHub Codespaces). Both the frontend and backend components reside in the same repository, ensuring streamlined version control, improved traceability, and simplified management.

### **Dependencies**

* **Frontend**: package.json for React.js
* **Backend**: pom.xml for Spring Boot

### **Config**

All configurations are externalized through environment variables in the docker-compose.yml file, eliminating hardcoded values such as ports, URLs, and database connection details.

### **Backing Services**

The project treats the MySQL database as an attached resource, running within the backend container. Its configuration is defined in the docker-compose.yml file, ensuring smooth integration and efficient management within the infrastructure.

### **Build, Release, Run**

* **Dockerfiles**: Separate Dockerfiles are created for both the frontend and backend services.
* **Docker Compose**: A combined docker-compose.yml file manages both the build and run stages by creating container images and deploying them as separate, isolated services, ensuring efficient management and seamless integration of application components.

### **Processes**

The Spring Boot backend is designed to be stateless. A MySQL database is used for data storage, emphasizing the separation of persistent data from the stateless nature of the application logic. This design ensures scalability and a clear distinction between application state and data persistence.

### **Port Binding**

* **Frontend**: Port 3000
* **Backend**: Port 8085

### **Concurrency**

The application supports horizontal scaling using the process model provided by Docker Compose. Multiple instances of both frontend and backend services can be deployed concurrently to handle increased demand when needed.

### **Disposability**

The project prioritizes rapid startup and smooth shutdown:

* **Spring Boot**: Optimized for efficient resource utilization and quick startup, ensuring consistent and reliable performance.
* **Docker Compose**: Supports seamless application management, allowing containers to be stopped and restarted with minimal downtime using the docker-compose down and docker-compose up commands.

### **Dev/Prod Parity**

Development and production environments are kept consistent:

* **Containerized Workflow**: The same Docker Compose configuration is used for both development and production.
* **Externalized Configuration**: Environment variables manage configuration, ensuring consistency across all stages.

### **Logs**

Logs are treated as event streams and are accessible via container logs (docker-compose logs).

### **Admin Processes**

Administrative tasks, such as database migrations, can be executed as one-off processes using Spring Boot Actuator or by running manual commands within the containers, ensuring efficient management of such operations.

## **Summary**

The Car Listing Application is a full-stack web application that provides a seamless interface for managing cars. It allows users to add, update, delete, and list cars. The frontend, built with React.js, ensures a smooth user experience, while the backend, powered by Java Spring Boot, provides robust API functionality. The application leverages MySQL for data persistence and uses Docker and Docker Compose for containerization and environment consistency. By following the 12-Factor App methodology, the project ensures scalability, maintainability, and adherence to modern software development best practices.