# Car selling application

## Exercise Guidelines

- Allowed resources: written materials, personal computers, laptops, and internet resources.
- Prohibited: mobile phones, communication with anyone other than the examiner, censor, and proctor.
- Do not store solutions on external networks or drives/hosts like GitHub, Facebook, Google Drive, etc.
- Start by skim reading the entire exercise. Move to the next task if stuck. Focus on demonstrating your approach.
- Submit your project in a zip file on wiseflow, including solutions and a README.md file with answers to theoretical questions.
- Duration: 4 hours. Restroom breaks only. No smoking.

---

## Introduction

Build a backend system for an e-commerce platform selling used cars. Tasks include managing cars and sellers. Theoretical questions are part of the exercise.

---

## Domain Description

The application facilitates selling used cars. Entities:

1. Car: brand, model, make, year, first registration date, price, id.
2. Seller: first name, last name, email, phone, city. One seller can have multiple cars.

---

## Task 1: Building a REST Service Provider with Javalin

1.1 Create a Java project using the Javalin framework.

1.2 Document your work in a README.md file.

1.3 Implement a CarDTO class.

1.4 Develop a REST API with Javalin for cars.

```
1.4.1 Implement a CarController.

1.4.2 Set routes in CarRoutes.

1.4.3 Use CarDAOMock to mock the database.
```

1.5 Test the endpoints using dev.http file.

---

## Task 2: REST Error Handling

2.1 Document error handling for each endpoint.

2.2 Return exceptions as JSON.

2.3 Implement a logger to log exceptions.

---

## Task 3: Streams and Generics

3.1 Create a method in CarDAOMock to filter cars by year.

3.2 Group cars by brand and get total price.

---

## Task 4: JPA

4.1 Establish a HibernateConfig class with a method that returns an EntityManagerFactory.

4.2 Implement a Car entity class with the following properties: brand, model, make, year, first registration date, price, id.

4.3 Implement a Seller entity class with properties: first name, last name, email, phone, city, and a OneToMany relationship to cars.

4.4 Create a DAO class CarDAO using JPA and Hibernate. The new DAO should implement iDAO and include additional methods:

```
- `void addCarToSeller(int sellerId, int carId)`
- `Set<Car> getCarsBySeller(int sellerId)`
```

4.5 Create a Populator class and populate the database with cars and sellers.

4.6 Modify the CarController to persist data in the database.

4.7 Test the endpoints using the dev.http file. Document the output in your README.md file to verify the functionality.

---

## Task 5: Testing REST Endpoints

5.1 Create a test class for the REST endpoints in your CarRoutes file.

5.2 Set up @BeforeAll to create the Javalin server, the CarController, CarRoutes, and the EntityManagerFactory for testing.

5.3 Configure the @BeforeEach and @AfterEach methods to create the test objects (Cars and Sellers).

5.4 Create a test method for each of the endpoints.

5.5 Explain the differences between testing REST endpoints and the tests conducted in Task 5 in your README.md file.

---

## Task 6: Security

6.1 Implement a authentication mechanism for the REST API using JWT (with login and protected endpoints)

6.2 Add allowed roles for each endpoint (make sure everyone can use the login endpoint)