Service Design and Engineering - Car Rental

Mateusz Chudek
Faculty of Mathematics and Information Science,
Warsaw University of Technology, Poland

1 Introduction

The SDE 2024 Car Rental System is a project focused on simplifying the management of users, cars, and reservations. It is built with a modular design, separating responsibilities into distinct services for user management and car rental operations. All services within the system require JWT authorization to ensure secure access and data integrity.

The project source code and additional resources can be found on the GitHub organization: https://github.com/car-rental-sde.

2 Technology

The SDE 2024 Car Rental System is developed using Spring Boot 3.4.1, enabling quick setup of REST APIs and application configuration. Database interactions are handled through JPA (Java Persistence API) with Hibernate as the ORM implementation.

MapStruct is used for performant object-to-object mapping, reducing manual implementation of DTO conversions. Security is implemented with Spring Security, leveraging JWT-based authentication and role-based access control for securing endpoints.

The application uses an H2 database which is an in-memory database, providing a lightweight and fast solution for temporary data storage and debugging. Controller and clients are generated automatically using the OpenAPI Generator Maven Plugin, which converts OpenAPI specifications into usable code artifacts for consistent API interactions.

3 Architecture

The system consists of two services: the car rental service and the user service. Each service has its own dedicated database, namely UsersDB and CarsDB. Additionally, the car rental service integrates with two external APIs to enhance its functionality.

4 Services

4.1 Car rental service

The car rental service is the core of the application, responsible for coordinating all key processes. Its API is divided into four groups: cars, reservations, users, and internal. The first three handle

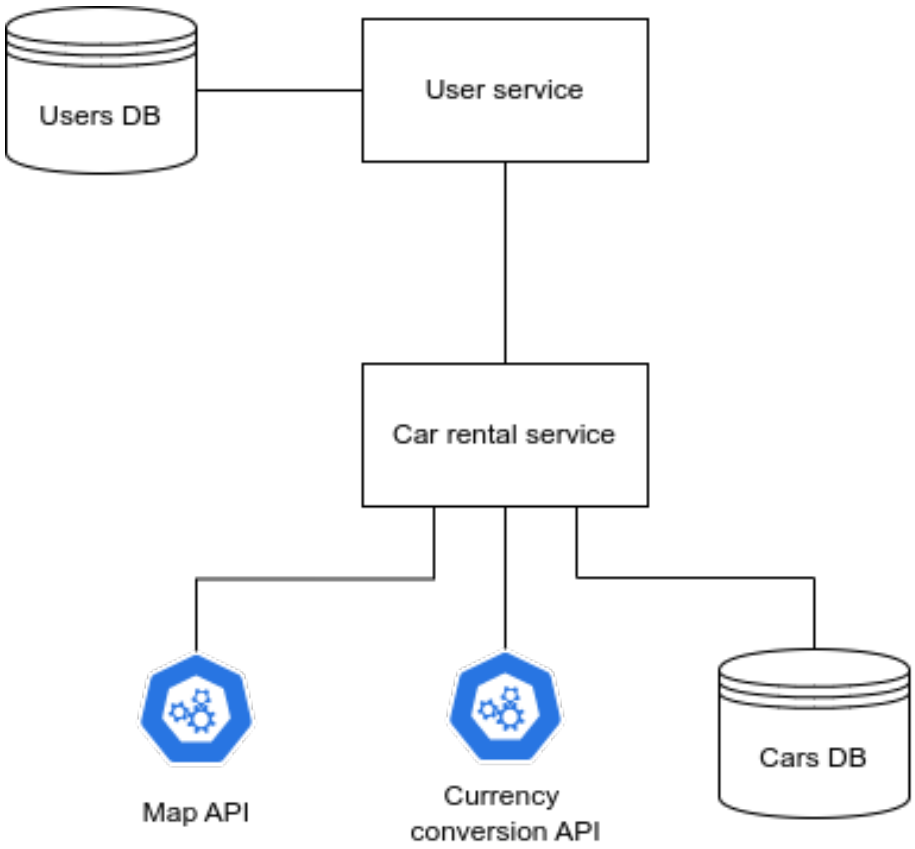

Figure 1: System Architecture of the Car Rental System

CRUD operations for their respective resources, while the last provides access to dictionaries of resources, such as car brands and models.

4.2 User service

The user service is designed for customers who want to use the car rental service. It maintains a database of registered users but relies on communication with the car rental service to handle reservations.

5 External APIs

5.1 Google Maps API

The Google Maps Distance Matrix API is used to calculate the distance between the car's pickup and drop-off locations and the base. This ensures accurate distance measurement for operational and cost calculations.

5.2 Exchangerates

The Exchange Rate API is utilized to calculate the daily price of a car and the total rental cost by converting amounts between euros and other supported currencies.

6 Possible improvements

6.1 Project for model

During development, it became evident that both services, being internal and one consuming the other, share many model objects. Currently, each service maintains its own independent set of generated classes, resulting in significant duplication and unnecessary object mapping. A better solution would be to create a standalone model project dedicated to generating and providing model classes for both services.

6.2 Refresh token

The current security implementation uses a JWT token as an access token, which is effective but missing one crucial component to be fully functional. Specifically, a second token, called a refresh token, should be generated. Its purpose is to allow users to renew their access without requiring them to log in again once the access token expires. This could be easily achieved by adding a new table to store the refresh tokens.

6.3 Payload objects for search operations

Certain search operations, such as searching for a car, involve numerous query parameters, which can make the implementation complex and harder to manage. An effective and cleaner alternative would be to use a payload object containing the search parameters. This approach simplifies the implementation, improves readability, and makes it easier to extend or maintain in the future.

6.4 Exception handler

Layered architecture mandates that all communication with the outside world be handled within controllers. However, there are cases where it becomes apparent earlier in the process, such as in the service layer, that a specific HTTP status code and message need to be returned. To address this, the project includes an example solution that utilizes a custom exception handler to manage such scenarios effectively.

7 Attachments

The documentation repository contains the following resources: Swagger documentation for the APIs of both services, providing detailed API specifications, and Postman collections for both services, enabling easier and more efficient testing.