

SWE 212 PROJECT STUDY 2 (2026)

Due date: 17 March 2026, in class.

The goal of this project is to develop a REST-API web application using Java Spring-Boot framework. MySQL or PostgreSQL database server can be used for data storage.

- All exceptions and errors must be handled carefully creating proper logs.
- Repository and service layers must be designed and implemented.
- A clear and concise way has to be chosen for controller end-points. Complex tasks has to be done in the service layer.
- Data returns must be handled with a proper Data Transfer Object (DTO). Do not use the entity classes!
- New data insertions and manipulations must create a log line at standard output.
- Tests will be conducted Postman (sometime browser can be used as well).
- Your application must include Swagger.

All your application endpoints must be shown in the browser Swagger output (see Figure 1). Practical examples and how your application is going to be tested with Postman and browser will be shown in the class.

Figure 2. shows some example views which will be shown in the class for car-rental example. You are expected to work on your problem and create similar Swagger and Postman outputs based on your problem.

The screenshot shows the Swagger UI interface for a REST API. At the top, there's a header bar with tabs for 'Explore' and 'v3/api-docs'. Below the header, the title 'OpenAPI definition' is displayed, along with a 'v0 OAS 3.0' badge. A 'Servers' dropdown is set to 'http://localhost:8080 - Generated server url'. The main content area is organized into sections for different controllers:

- rental-controller**: Contains methods for rental management:
 - PUT /rental/update/{id}
 - POST /rental/add
 - GET /rental/getfull/{id}
 - GET /rental/get/{id}
 - GET /rental/all
 - DELETE /rental/delete/{id}
- customer-controller**: Contains methods for customer management:
 - PUT /customer/update/{id}
 - POST /customer/add
 - GET /customer/get/{id}
 - GET /customer/all
 - DELETE /customer/delete/{id}
- car-controller**: Contains methods for car management:
 - PUT /car/update/{id}

Figure 1. Example Swagger output.

The figure consists of five separate Postman request windows arranged in a grid-like layout.

- Top Left:** A GET request to `http://localhost:8080/car/all`. The response status is 200 OK. The response body is a JSON array containing two car objects. Each car has an id, brand, model, plate, and a rents array. The first car's rents array contains one object with id 1, rentDate null, returnDate null, and returned false. The second car's rents array contains one object with id 2, rentDate null, returnDate null, and returned false.
- Top Right:** A GET request to `http://localhost:8080/customer/`. The response status is 200 OK. The response body is a JSON array containing two customer objects. Each customer has an id, name, address, and a rents array. The first customer's rents array contains one object with id 1, rentDate null, returnDate null, and returned false. The second customer's rents array contains one object with id 2, rentDate null, returnDate null, and returned false.
- Middle Left:** A GET request to `http://localhost:8080/car/get/1`. The response status is 200 OK. The response body is a JSON object representing the first car from the all-cars list. It includes id, brand, model, plate, and a rents array with one object (id 1, null dates, false returned).
- Middle Right:** A PUT request to `http://localhost:8080/car/update/1`. The response status is 200 OK. The response body is a JSON object representing a car with id 1, brand Fiat, model 2018, plate 54-RB-567, and an empty rents array.
- Bottom Left:** A DELETE request to `http://localhost:8080/rental/delete/1`. The response status is 200 OK. The response body is a JSON object representing the same car as the PUT request, but with an updated rents array containing one object (id 1, null dates, false returned).

Figure 2. Some Postman views for car-rental project.

PS: Your group subject was distributed in the beginning of the semester.

Grading:

No	Task	Grade
1	The database and tables are created in MySQL or PostgreSQL server by Hibernate.	10
2	The model entities and required DTOs are OK.	20
3	Exceptions, errors are handled properly. Logging is OK	20
4	Repository and service layers that implement all required DB operations are OK.	20
5	Swagger and all controller endpoints are OK (CRUD methods are implemented).	20
6	Everything works as described with no error, exceptions handled properly.	10

PS: Group presentation is required, group members must show up to in presentation to collect points.