Spring-MicroService-RestAPI-Project

**Summary:**

This project involves two microservices: a **Cab Microservice** and a **Booking Microservice**. The **Cab Microservice** manages details about different cabs, including fare per kilometer. The **Booking Microservice** handles bookings, calculates total fare based on the distance and cab fare, and stores booking details. These microservices interact via REST API calls.

**Description:**

1. **Cab Microservice (Port: 9090)**:
   * **Purpose**: Manages cab data, including fare per kilometer, seat capacity, type of cab, and company information.
   * **Key Endpoints**:
     + POST /travel/store\_cab: Used to add a new cab, including its fare per kilometer and other details.
     + GET /travel/find\_fare\_by\_cab\_id/{cid}: Retrieves the fare per kilometer for a specific cab using its ID.
2. **Booking Microservice (Port: 9191)**:
   * **Purpose**: Manages bookings and calculates the total fare for trips based on the distance traveled and the fare per kilometer fetched from the **Cab Microservice**.
   * **Key Endpoints**:
     + POST /booking/add\_booking: Adds a booking with origin, destination, distance, and cab ID, and calculates the total fare using the cab's fare rate.
     + GET /booking/find\_booking\_by\_id/{bid}: Retrieves booking details, including the calculated total fare.

**Workflow Overview:**

* The **Cab Microservice** stores fare details for different cabs.
* The **Booking Microservice** interacts with the Cab Microservice to fetch the fare for a specific cab and calculates the total fare based on the distance traveled.
* The total fare is stored and retrieved in the Booking Microservice, ensuring seamless integration between both services.

**Technologies Used:**

1. **Java**:
   * The primary programming language used to develop the core functionalities of the microservices. It provides object-oriented principles, robust libraries, and scalability to handle complex backend services.
2. **Spring Framework**:
   * A powerful Java framework that simplifies enterprise-level application development. It provides dependency injection, aspect-oriented programming, and many other features essential for large-scale application development.
3. **Spring Boot**:
   * A sub-framework of Spring, Spring Boot is designed to simplify the setup and development of new Spring applications. It eliminates the need for extensive XML configurations, making development faster and easier.
   * **Key Features**: Embedded web servers (like Tomcat or Jetty), auto-configuration, production-ready metrics, and more. It was used to create RESTful microservices with annotations like @RestController, @Service, @Repository, etc.
4. **Spring Data JPA**:
   * Part of the Spring ecosystem, Spring Data JPA simplifies database operations by handling basic CRUD operations and complex queries. It provides an abstraction over **JPA** (Java Persistence API) and interacts with the **Hibernate ORM** (Object-Relational Mapping) to persist data in the MySQL database.
5. **Spring Cloud & Eureka**:
   * **Spring Cloud** is used for building microservices and ensuring smooth communication between them. **Eureka** is a service discovery tool used to register and locate services.
   * In this project, the **Cab Microservice** and **Booking Microservice** both registered with **Eureka** for service discovery, allowing them to find and communicate with each other dynamically.
6. **Hibernate**:
   * Hibernate is used to handle ORM (Object Relational Mapping) for the project. It helps in mapping Java objects to database tables and vice versa. Hibernate manages data persistence by converting Java objects into relational database tables.
7. **MySQL**:
   * A relational database management system used to store information related to cabs and bookings. **Spring Data JPA** interacts with MySQL to handle CRUD operations for the cab and booking data.
8. **RestTemplate**:
   * A Spring tool for making HTTP requests from one microservice to another. It was used to call endpoints from the **Cab Microservice** to fetch data such as cab fare and calculate the booking fare.
9. **JUnit**:
   * A unit testing framework for Java, JUnit was used to write and run test cases. JUnit allows automated testing of the individual components in the application (such as services, controllers, and repositories) to ensure they function correctly.
10. **Web Services**:
    * RESTful Web Services were built using Spring Boot. These web services enable the application to expose endpoints via HTTP, allowing microservices to interact with each other and external systems. **GET**, **POST**, **PUT**, and **DELETE** requests were handled through annotated controller methods.
11. **Microservices**:
    * The architecture employed in this project. **Microservices** are small, loosely coupled services that interact via APIs. The **Cab Microservice** and **Booking Microservice** are independent services that handle specific functionalities, such as managing cab information and booking cabs, respectively. Each microservice can be developed, deployed, and scaled independently, ensuring flexibility and resilience.
12. **Lombok**:
    * A Java library used to minimize boilerplate code by auto-generating code such as getters, setters, constructors, and toString methods using annotations like @Data and @Getter.
13. **Maven**:
    * A build automation tool used for managing the project's dependencies, packaging, and building processes. It also helps in organizing the project structure, ensuring smooth integration with external libraries and tools.
14. **Postman**:
    * A widely used API client tool to test RESTful web services. Postman was used for manual testing of the endpoints in both the **Cab** and **Booking** microservices. It allowed sending HTTP requests to the microservices and verifying the responses for the APIs, ensuring they worked as expected.

Github Link: https://github.com/saitarun23/Spring-MicroService-RestAPI-Project