PROJECT SPECIFICATION

Build the Backend System for a Car Website

Convert the Pricing Service

CRITERIA	MEETS SPECIFICATIONS
The Pricing Service is converted to a microservice.	The Pricing Service API is converted to a microservice with Spring Data REST, without the need to explicitly include code for the Controller or Service.
A Eureka server is used and the Pricing Service is registered with that server.	A Eureka server is implemented and running on port 8761. The Pricing Service is registered on that server and is named `pricing-service`. The Vehicles API should be able to use the Eureka server to call the pricing service.
At least one additional test is added for the Pricing Service.	Within the `test` folder, at least one additional test has been implemented outside of `contextLoads()` for the overall Pricing Service Application.

Implement the Vehicles API

CRITERIA	MEETS SPECIFICATIONS
The Vehicles API can perform CREATE operations.	The Vehicles API is able to create a new vehicle based on input from the user with a POST request.
The Vehicles API can perform READ operations.	The Vehicles API can receive GET requests from a user, and read back either a list of all existing vehicles, or the data for a single vehicle.
The Vehicles API can perform UPDATE operations.	The Vehicles API can update an existing vehicle through input from the user.
The Vehicles API can perform DELETE operations.	The Vehicles API can delete an existing vehicle when requested by the user.

The Vehicles API can request location data from Boogle Maps, and pricing data from the Pricing Service.	The Vehicles API is able to consume information from the separate Boogle Maps and Pricing Service APIs, and return that information as part of the vehicle information for a single vehicle. *Note:* Boogle Maps will assign a new random address each time a query is called, so don't fret if it changes between queries.

Testing the Vehicles API

CRITERIA	MEETS SPECIFICATIONS
Tests are implemented for the Vehicles API `CarController` that cover the CRUD (Create, Read, Update, Delete) operations.	Within the `CarControllerTest.java` file, the `TODO`s for tests of CRUD operations have been implemented. You are welcome to add additional tests beyond these as desired!
All necessary dependencies are included in each Application's POM files, and the code runs successfully.	All necessary dependencies have been added to the relevant POM files, and the code is able to run both from tests and in launching the actual applications.

API Documentation

CRITERIA	MEETS SPECIFICATIONS
API documentation for the Vehicles API is implemented using Swagger.	API documentation for the Vehicles API is implemented using Swagger, and all related API queries are able to be run from there. The documentation is available at http://localhost:8080/swagger-ui.html when the application is running. *Note:* You are welcome to add Swagger API documentation for the other APIs, but it is not required.

Suggestions to Make Your Project Stand Out!

- The Boogle Maps application does not actually store the address assigned to a given vehicle based on latitude and longitude, and instead randomly assigns a new one each time it is called. How could you update this to track which address is assigned to which vehicle? What happens if the vehicle latitude and longitude is updated in the Vehicles API?
- The Pricing Service stores prices based on ID, but that ID is not truly assigned to a specific vehicle - if the vehicle is deleted and a new vehicle uses the old ID, the same price is used. How can you update the Pricing Service (and perhaps the Vehicles API) to assign a new (random) price when a vehicle is removed from the Vehicles API?
- An additional helpful service after the Vehicles API would be to have an Orders/Sales service when a customer wants to order a vehicle. How would you implement this Orders/Sales API and integrate it with the Vehicles API?
