DA

Distributed Algorithms
April 2021

TD | **REST**

Objective of the session :
Building a REST server using Spring.

1 Vehicle REST Server

Consider the following example of a company that sells vehicles. This company would like to have information about the vehicles it sells and make this information available to all sellers and potential buyers (in this example we will consider only some cars and some of their characteristics). For this purpose, a REST API is implemented to provide this information to the different clients.

The following table shows a subset of the available vehicles and some of their characteristics.

ID	**Vehicle**	**ModelYear**	**VehicleType**	**Gearbox**	**FuelType**
1	Renault - Capture	2019	SUV	Automatic	Hybrid
2	Peugeot - 308	2020	Car	Manual	Gasoline
3	Ford - Focus	2015	Car	Manual	Gasoline
4	Reanult - Clio	2021	Car	Manual	Gasoline
5	Mercedes Benz - EQC	2019	SUV	Automatic	Hybrid

Exercises

The objective of this exercise is to create a service using a REST server so that the company can have an inventory of available vehicles that can be accessed by different clients.

1. **Take a moment to familiarize yourself with the project files.**
2. **Add setters and getters for the different information of a vehicle.**
3. **The server must respect the following endpoints.**
   - **GET, `/cars` : returns the list of vehicles.**
   - **GET, `/cars/{id}` : returns the information of a vehicle.**
   - **POST, `/cars` : adds a vehicle.**
   - **PUT, `/cars/{id}` : updates the information about a vehicle.**
   - **DELETE, `/cars/{id}` : remove a vehicle.**
4. `VehicleNotFoundException` **is an exception used to indicate when a vehicle is looked up but not found. Change the** `VehicleNotFoundAdvice` **class to issue the right HTTP code status.**
5. **Let's test the server. You can use POSTMAN, RESTClient or** `curl`[1]**. You need to specify the HTTP method (GET, POST, PUT or DELETE)**

1. Client URL is a command-line tool for getting or sending data using various network protocols. cURL supports HTTPS.

followed by the parameters needed to call the various methods of the REST API[2]. Take some time to analyze the response to each request.
— Get all the vehicles.
— Get one particular vehicle (e.g. 3). What happens if vehicle 10 is searched ?
— Create some new vehicle records.
— Update a vehicle. Let's change his ModelYear.
— Delete a vehicle.

6. Consider the following design problem : our system is running and is used by many clients. Suddenly, it is necessary to separate the `vehicle` into `manufacturer` and `model`. For example, the `Vehicle: Renault - Clio` should be separated into `Manufacturer: Renault` and `Model: Clio`.
   How would you upgrade the server without requiring the clients to upgrade as well ?
7. What is missing to make this web-based service RESTful ? A critical ingredient to any RESTful service is adding links to relevant operations. To make your controller more RESTful, add links. Re-do some of the tests from item 6.
8. Think about testing your program with server and client on different machines.

---

2. for those who use curl, you can use the `| json_pp` command to make the JSON pretty.

Documentation about annotations that can be useful

— @Entity
This is a JPA annotation to make an object ready for storage in a JPA-based data store.

— @SprinbBootApplication
This is a very useful annotation for creating Java web application with Spring. By using Spring Boot, then you can run your application without deploying it into a web server, as it comes with an embedded Tomcat server.

— @RestController
This is a convenience annotation for developing a RESTful web service with the Spring MVC framework. When you annotate a controller class with @RestController it does two purposes, first, it says that the controller class is handling a request for REST APIs and second you don't need to annotate each method with the @ResposneBody annotation to signal that the response will be converted into a Resource using various HttpMessageConverers.
There are routes for each operation (@GetMapping, @PostMapping, @PutMapping and @DeleteMapping, corresponding to HTTP GET, POST, PUT, and DELETE methods).

— @ResponseBody
The @ResponseBody annotation is one of the most useful annotations for developing RESTful web service using Spring MVC. This annotation is used to transform a Java object returned from a controller to a resource representation requested by a REST client. The @ResponseBody is used to indicate that the response returned by this method will be converted into a resource that the client can consume. If you have created any RESTful web service with Spring MVC then you know that we need to annotate each method which generates REST response with the @ResponseBody annotation but with the introduction of @RestController, we can avoid this.

— @RequestBody
This annotation can convert inbound HTTP data into Java objects passed into the controller's handler method. Just as @ResponseBody tells the Spring MVC to use a message converter when sending a response to the client, the @RequestBody annotations tell the Spring to find a suitable message converter to convert a resource representation coming from a client into an object.

— @ExceptionHandler(e)
This annotation allows to configure the advice to only respond if a particular Exception "e" is thrown.

— @ResponseStatus
This annotation allows to issue HttpStatus, *e.g.* an HTTP 404.