Lab - CRUD operations with Spring Boot

Estimated Time Needed: 45 mins

In this lab, you will learn how to create a **Products's list** using a Spring Boot. Your application should allow you to add a product, retrieve the products, retrieve a specific product with its id, update a specific product with its id, and delete a product with its id. All these operations will be achieved through the REST API endpoints in your Spring Boot Application.

You will create an application with API endpoints to perform Create, Retrieve, Update, and Delete operations on the above data using a Spring Boot.

You will use cURL and POSTMAN to test the implemented endpoints.

Learning objectives

After completing this lab, you will be able to:

- Create API endpoints to perform Create, Retrieve, Update, and Delete operations on transient data with a Spring Boot Application.
- Create REST API endpoints, and use POSTMAN to test your REST APIs.

Set-up: Create application

1. Open a terminal window by using the menu in the editor: **Terminal > New Terminal**.

2. Change to your project folder, if you are not in the project folder already.

   ```
   cd /home/project
   ```

3. Run the following command to clone the Git repository that contains the starter code needed for this lab, if it doesn't already exist.

```
[ ! -d 'bfvor-sl-microservice-java' ] && git clone https://github.com/ibm-developer-skills-network/bfvor-sl-microservice-java.git
```

4. Change to the directory **bfvor-sl-microservice-java/crud** to start working on the lab.

```
cd bfvor-sl-microservice-java/crud
```

5. List the contents of this directory to see the artifacts for this lab.

```
ls
```

6. Run the following command on the terminal to install the dependecies that are required.

```
mvn clean install
```

Exercise 1: Understand the crud application

1. In the Files Explorer, open the **bfvor-sl-microservice-java/crud** folder and view **Product.java**, which is in `/home/project/bfvor-sl-microservice-java/crud/src/main/java/com/hello/crud/Product.java`.

Open **Product.java** in IDE

This is the Data class for your controller.

2. Now, in the Files Explorer, open the **bfvor-sl-microservice-java/crud** folder and view **ProductController.java**, which is in `/home/project/bfvor-sl-microservice-java/crud/src/main/java/com/hello/crud/ProductController.java`.

Open **ProductController.java** in IDE

Below are the import dependencies required to create rest endpoint.

```
import com.hello.crud.Product;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
```

3. `@RestController` and `@RequestMapping("/products")` are the annotations that map the endpoint of this class to `/products` url. You will be creating methods, which will service all the REST APIs for adding, retrieving, updating, and deleting products.

```
@RestController
@RequestMapping("/products")
public class ProductController {
```

4. The code has precreated products added to the list. These are defined in the following code.

```
private final List<Product> products;
    public ProductController() {
        products = new ArrayList<>();
        // Initialize with sample data
        Product notebook = new Product();
        notebook.setId(143);
        notebook.setName("Notebook");
        notebook.setPrice(5.49);
        Product marker = new Product();
        marker.setId(144);
        marker.setName("Black Marker");
        marker.setPrice(1.99);
        products.add(notebook);
        products.add(marker);
    }
```

5. Now that the server is defined, you will create the REST API endpoints and define the routes or paths, one for each of the following operations.

- Retrieve all the products - `GET` Request Method
- Retrieve a product by its id - `GET` Request Method
- Add a product - `POST` Request Method
- Update a product by its id - `PUT` Request Method
- Delete a product by its id - `DELETE` Request Method

Add the following code to ProductController.java in the space provided at line 37.

```
// GET /products
    @GetMapping
    public List<Product> getProducts() {
        return products;
    }
    // GET /products/{id}
    @GetMapping("/{id}")
    public Product getProduct(@PathVariable Integer id) {
        return products.stream()
                .filter(product -> product.getId().equals(id))
                .findFirst()
                .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "Product not found"));
    }
    // POST /products
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void addProduct(@RequestBody Product product) {
        products.add(product);
    }
    // PUT /products/{id}
    @PutMapping("/{id}")
    public ResponseEntity<Void> updateProduct(@PathVariable Integer id, @RequestBody Product updatedProduct) {
        Optional<Product> existingProduct = products.stream()
                .filter(product -> product.getId().equals(id))
                .findFirst();
        if (existingProduct.isPresent()) {
            Product product = existingProduct.get();
            if(updatedProduct.getName() != null)
            product.setName(updatedProduct.getName());
            if(updatedProduct.getPrice() != null)
            product.setPrice(updatedProduct.getPrice());
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
```

```
    }
    // DELETE /products/{id}
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Integer id) {
        Optional<Product> product = products.stream()
                .filter(p -> p.getId().equals(id))
                .findFirst();
        if (product.isPresent()) {
            products.remove(product.get());
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }
```

Run and test the Spring Boot crud application with cURL

1. In the terminal, run the maven goal to run spring boot application.

```
mvn spring-boot:run
```

The server starts running and listening at port 5000.

2. Open another `Terminal` by clicking the Terminal menu and selecting **New Terminal**.

3. In the new terminal, run the following command to access the **http://localhost:5000/products** API endpoint. `curl` command stands for `Client URL` and is used as command line interfacing with the server serving REST API endpoints. It is, by default, a `GET` request.

```
curl http://localhost:5000/products
```

This returns a JSON with the products that have been preloaded.

4. In the terminal, run the following command to add a product to the list. This will be a POST request to which you will pass the product parameter as a JSON.

```
curl -X POST -H "Content-Type: application/json" \
    -d '{"id": 145, "name": "Pen", "price": 2.5}' \
    http://localhost:5000/products
```

This command will not return any output. It will add the product to the list of products.

5. Verify if the product is added by running the following command.

```
curl http://localhost:5000/products/145
```

This should return the details of the product you just added with a POST request.

Using POSTMAN to test the REST API endpoints

While the `GET` endpoints are easy to test with `curl` using a command line interface, the `POST`, `PUT`, and `DELETE` commands can be cumbersome. To circumvent this problem, you can use Postman, which is a software that is available as a service.

Postman will need remote access to the server as it is an external entity. To get the URL, click the following button.

Launch Application

Copy the URL into a notepad or other text editors. Go to https://www.postman.com/ and sign up before you start using POSTMAN.

▶ Click here for instructions to sign up for Postman services.

1. Go to `My Workscpace` from top menu.

2. Click **Create New** to start creating a request to the REST API endpoint.

3. Choose **HTTP** from the options that you are presented with.

3. Paste the URL that you have copied into the address bar. Change the request type to `POST`. To send input as JSON, choose `raw->body->JSON`.

4. Enter the following JSON object and click the `Send` button. This will add the product to your previous list of products.

```
{
    "id":146,
```

```
    "name":"Laptop Bag",
    "price":45.00
}
```

4. Verify the list of products to ensure that the request has gone through and the product is added. Change the request type to `GET` and remove the JSON object from the request body. Click `Send` and observe the output in the response window.

5. Test the `PUT` endpoint to update product details by changing the price of the item with id 146 to 42.00. Set the request type to `PUT` and add the product id to the end of the URL and add the value to be changed as a JSON body and click `Send`.

```
{
    "price":42.00
}
```

6. Verify the product with id 146 to ensure that the request has gone through and the product is updated. Change the request type to `GET` and remove the JSON object from the request body. Click `Send` and observe the output in the response window.

Practice labs

1. Update product details of product with id 144 by changing its price to 2.50.

▼ Click here for a hint

Go to products/144 endpoint. Set the request type to `PUT` and add the product id to the end of the URL and add the value to be changed as a JSON body and click `Send`.

```
{
    "price":2.50
}
```

▼ Click here for the solution

2. Add the following product using `POST` method.

```
{
    "id":142,
    "name":"Eraser",
    "price":1.50
}
```

▼ Click here for a hint

Refer to step 3 of the previous exercise

3. Delete the product with id 142.

▼ Click here for a hint

Go to products/142 endpoint. Set the request type to `DELETE` and click `Send`.

▼ Click here for the solution

Congratulations! You have completed the lab for CRUD operations with Java Spring Boot.

Summary

In this lab, you have performed CRUD Operations like GET, POST, PUT, and DELETE on a Spring Boot App and tested the above methods using POSTMAN.

Author(s)

Harsh Singh