Spring Boot Microservices API Test using Junit Integration Test

In this lab, we will work with a spring boot microservice that can service the following REST API requests: GET, POST, PUT DELETE API requests to perform CRUD (Create Read Update Delete) operations.

We will test the system as a black box and test the system from outside. We will test for a REST resource and test the following:

- The HTTP response code
- The HTTP response headers
- The payload (JSON)

Open a File explorer to navigate to the directory

C:\apitestlabs\customerservice

This directory contains a simple spring boot customer microservice that handles the following requests:

GET request to get all customers
URL: http://localhost:8090/api/customers

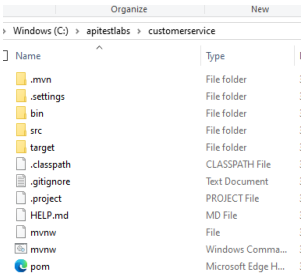
GET request to get a customer using customer id
URL: http://localhost:8090/api/customers/{id}
id is the template variable
For example http://localhost:8090/api/customers/2 will return customer data for a customer id value 2

POST request to create a new customer
URL: http://localhost:8090/api/customers
Post a Json data via request body

PUT request to update an existing customer with an id
URL: http://localhost:8090/api/customers/{id}
id is the template variable
For example http://localhost:8090/api/customers/2

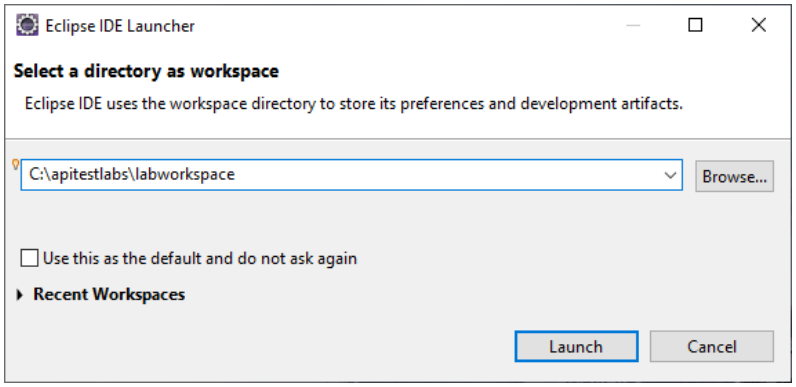
DELETE request to delete an existing customer with an id
URL: http://localhost:8090/api/customers/{id}
id is the template variable
For example http://localhost:8090/api/customers/2

1. Open a File Explorer, Navigate to **C:\apitestlabs\customerservice**



Review the resources in this microservice directory.
You may open this project in Eclipse and review the source code and configuration.

2. Start your eclipse and go to your lab workspace: **C:\apitestlabs\labworkspace**



3. Let us import the customer service project. In Eclipse,

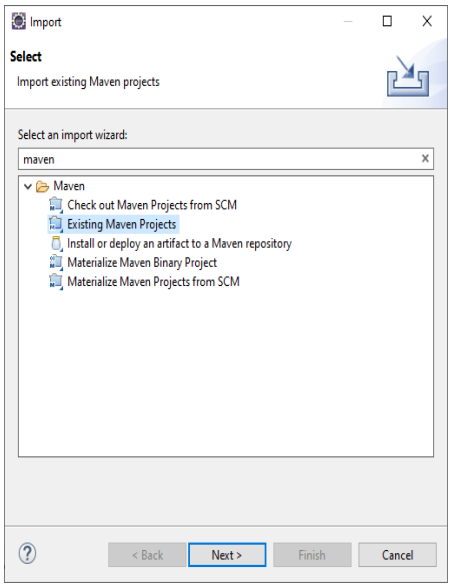
   Click **on File → Import → Maven → Existing maven projects**

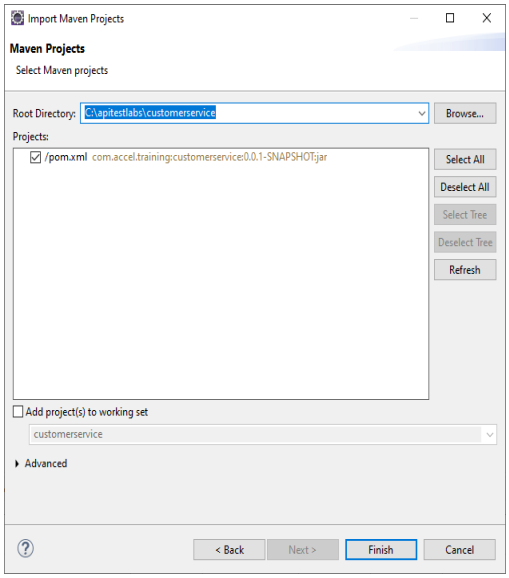
   Click on **Next**.

   Navigate to **C:\apitestlabs\customerservice**

   Select this project.

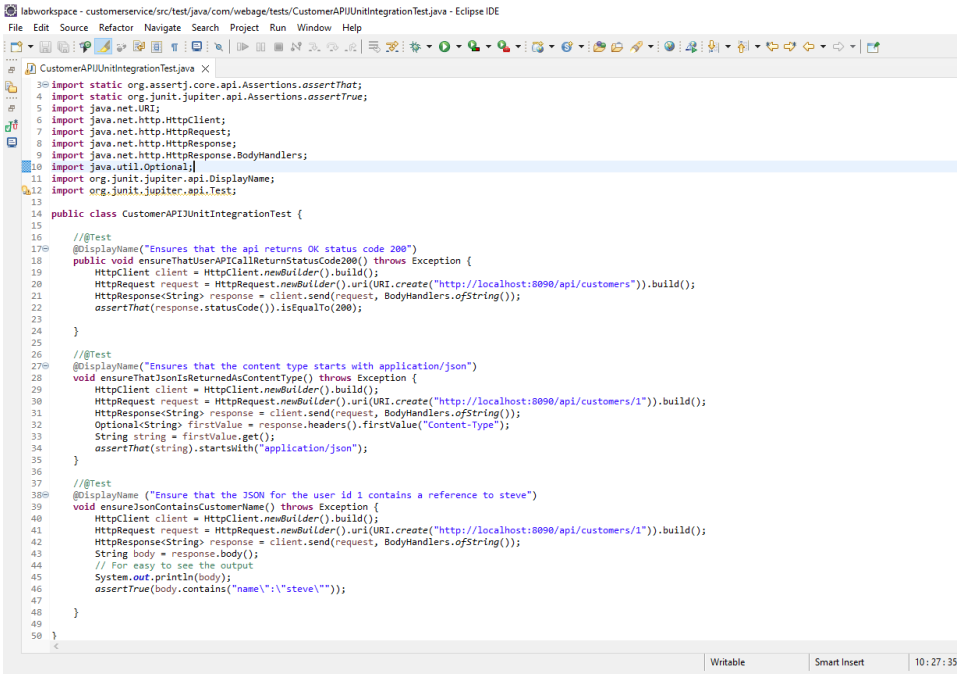
   Click on **Finish.**





4. In Eclipse Project Explorer view, Select the project → right click on customer service project → Maven → Update Project

   In Project Explorer view, click on **src/test/java →com.webage.tests → CustomerAPIJUnitIntegrationTest.java**
   and open this source code.



5. Review the Test code in this file.

6. Let us test our API GET request and returns the status code 200 OK..
   Uncomment the @Test annotation above the following method:
   ensureThatUserAPICallReturnStatusCode200 ()

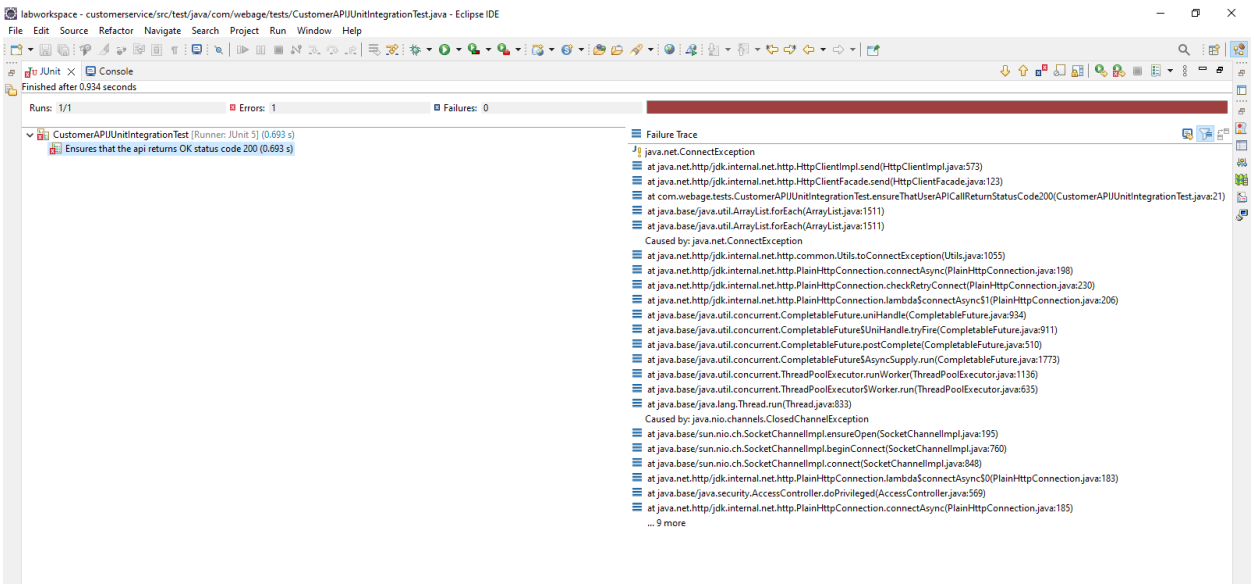
```
@Test
@DisplayName("Ensures that the api returns OK status code 200")
 public void ensureThatUserAPICallReturnStatusCode200() throws Exception {
     HttpClient client = HttpClient.newBuilder().build();
     HttpRequest request =
    HttpRequest.newBuilder().uri(URI.create("http://localhost:8090/api/customers")
    ).build();
     HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
     assertThat(response.statusCode()).isEqualTo(200);

 }
```

   and save the file.

7. Select the method **ensureThatUserAPICallReturnStatusCode200() → right click → Run As → Junit Test**

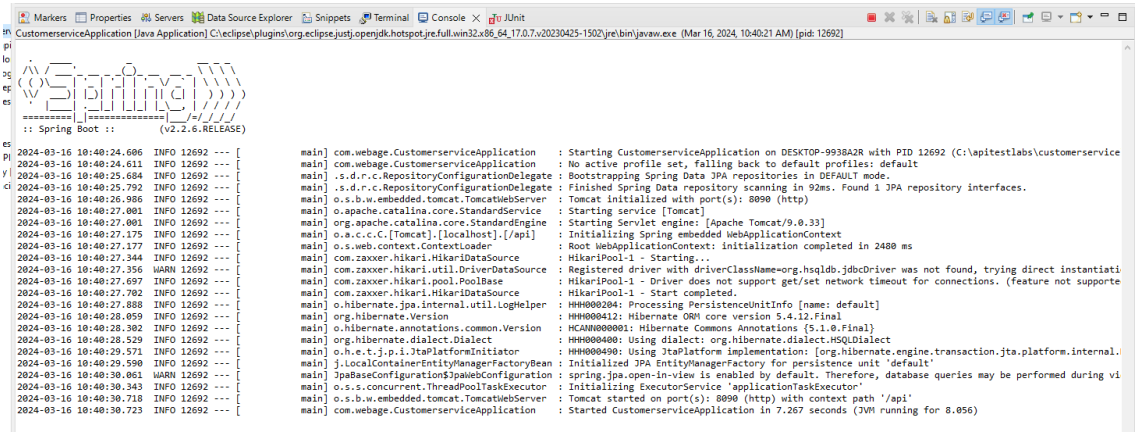
and run the Junit Test.

   What is the result? Pass | Fail ?



   If the API application is not running, this test should fail as shown in the screen shot.

8. Let us start and run the application. Click on **src/main/java ---→ com.webage -→ CustomerServiceApplication.java -→ right click -- > Run As → Java Application**

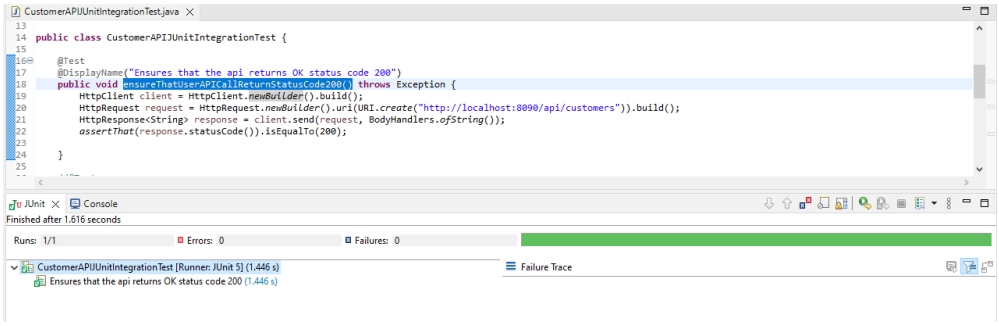


Review the application log messages.

9. Select the method **ensureThatUserAPICallReturnStatusCode200() → right click → Run As → Junit Test**

and run the Junit Test.

What is the result? Pass | Fail ?



10. Check the console view for any log messages.

11. Let us test that the API call returns the JSON type data content. The method **ensurethatJsonIsReturnedAsContentType()** contains code to test this task. Uncomment the @Test annotation above this method and **save the file**.

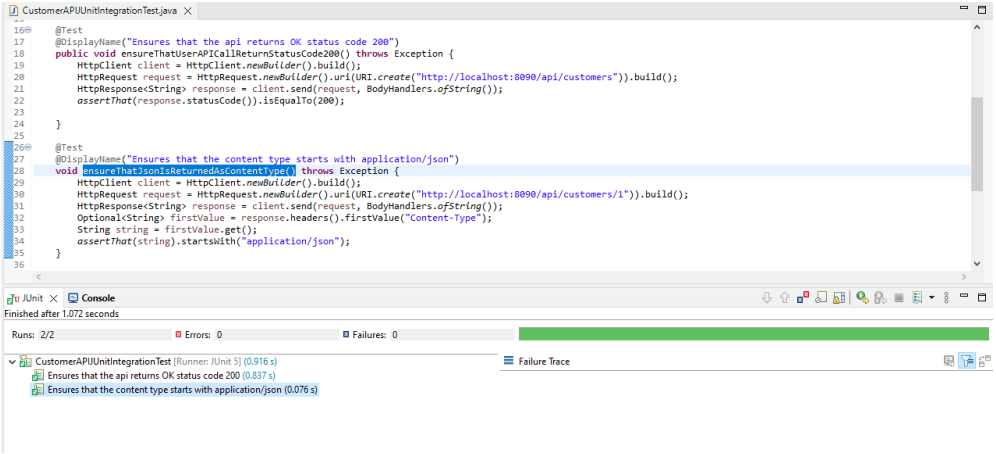
```
    @Test
    @DisplayName("Ensures that the content type starts with application/json")
    void ensureThatJsonIsReturnedAsContentType() throws Exception {
        HttpClient client = HttpClient.newBuilder().build();
        HttpRequest request =
HttpRequest.newBuilder().uri(URI.create("http://localhost:8090/api/customers/1")).bui
ld();
        HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
        Optional<String> firstValue = response.headers().firstValue("Content-Type");
        String string = firstValue.get();
        assertThat(string).startsWith("application/json");
    }
```

12. Select the method → **right click → Run As → Junit Test**

and run the Junit Test.

What is the result? Pass | Fail ?

Review the **console** messages.

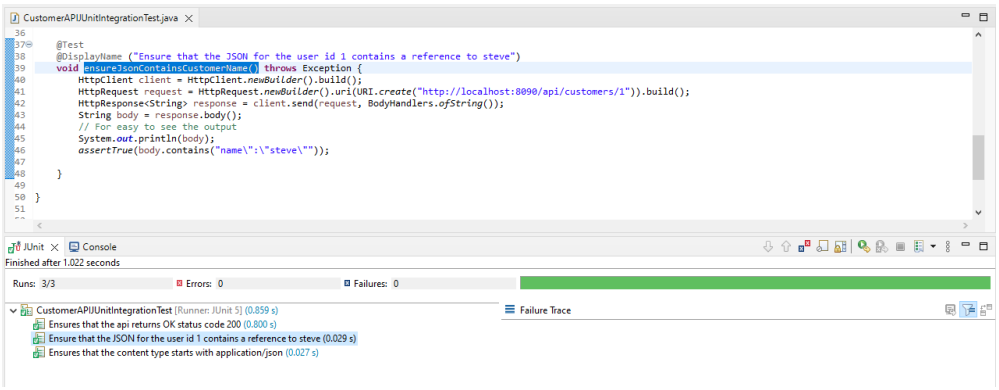


13. Let us test API returns a data that contains a specific value. For example, for the customer with id=1, name must be steve. Let us test that API returns a name steve for customer id=1.
    Uncomment the @Test annotation above the method
    ensureJsonContainsCustomerName()

```
    @Test
    @DisplayName ("Ensure that the JSON for the user id 1 contains a reference to
steve")
    void ensureJsonContainsCustomerName() throws Exception {
        HttpClient client = HttpClient.newBuilder().build();
        HttpRequest request =
HttpRequest.newBuilder().uri(URI.create("http://localhost:8090/api/customers/1")).bui
ld();
        HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
        String body = response.body();
        // For easy to see the output
        System.out.println(body);
        assertTrue(body.contains("name\":\"steve\""));

    }
```

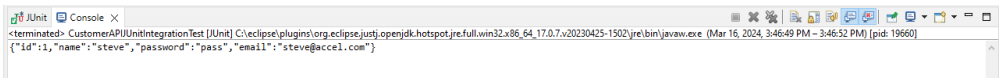
    and save the file.

14. Run the test. Select the method → **right click → Run As → Junit Test** and run the Junit Test.



    What is the result? Pass | Fail ?

    Review the **console** messages.



15. Based on what you have learned, write your own methods and tests.