Lab 3

Run and Debug the API
Overview

In this lab we’ll run through an end to end test, and add unit testing coverage.

Step 1: Full test

In this step, you will go through a use case of creating a product, making an update and finally deleting it. These instructions will assume you are using POSTMAN, but the API Console can be used directly within Anypoint Studio. Along the way you’ll check the state of the product. You will need to:

- Create a product.
- Check for its existence.
- Perform an update.
- Check the change has been applied.
- Delete the product.
- Confirm the product is no longer available.

The following instructions will guide you through the process:

1. Right click the application. Select Run As > Mule Application.
2. The application will start running, and after a few seconds the API console will be available for us to test out the integration.
3. Open up POSTMAN (or an equivalent).
4. Start by creating a product. Select the POST method and set the body using the example from the RAML specification, or the example below:
CAUTION

The productNumber must be unique, so if this record has already been created, try setting a different random value, or ask your instructor to provide a value by querying the database directly to check for existing records.

1. Set the endpoint to be `http://localhost:8081/api/product`, and add a Content-Type header with value `application/json`. 
2. Review the mock body of the operation, which in this case, contains the name, description, category, etc. of the product to be created, and click the **Send** button.

3. If everything was successful, you should receive a 201 response code, and a response body that contains the new product ID.

   **Take note of this ID.** We will use it in the following steps.

4. Open a new tab for the GET method on the /product/{id} resource.

5. Enter the ID from the product you created before, and click **Send**.

   If everything goes **Ok**, you should get a 200 status response code, with the body of the product you created before.
6. You have created a product and confirmed it exists. Now update it. Open a new tab and select PUT.

7. Again, specify the product ID to update in the URL (set the url to http://localhost:8081/api/product/{id} with your product id.), which is the same ID from before, and take a moment to check how you are changing the product (pick a minor update or your choice). Set a Content-Type header with value application/json.

8. After taking note of your update click Send.

9. If everything goes ok, you should get a 204 status response code and no body.

After you update the information, perform a GET operation again under the /product/{id} resource, specifying the ID of your product, and you should obtain the updated information.

10. To end with your basic operations, delete the product. Create a new tab and select DELETE as the HTTP verb. Set the url to http://localhost:8081/api/product/{id} with your product id.
If everything goes Ok, you should get a 204 status response code and no body.

11. To confirm we have deleted it, perform a GET operation under the /product/{id} resource, specifying the ID of your product. You should get no information. In fact, as the product and resource do not exist, you should get a 404 status response code, and the message "Resource not found", which is what we want.

Stop the Mule runtime.

**Step 2: Add MUnit to your API**

MUnit is an application testing framework that allows you to easily build automated tests for your integrations and APIs. It provides a full suite of integration and unit test capabilities, and is fully integrated with Maven and Surefire for integration with your continuous testing and deployment environment.

In this step you will use the MUnit for RAML feature to create a test for a specific flow.

The following instructions will guide you through the process.

1. Inside api-product.xml file, right click on the "APIKit Router" component, go to "MUnit" and select "Create api-product.xml Suite for RAML". This will create a new file called "api-product-apikit-test.xml" on "src/test/munit" folder with a test flow for each response defined in the RAML file.
A new `api-product-apikit-test.xml` file will be generated and a viewer will open.

Concentrate on the test flow called "get:/product:api-product-config-200-application/json-FlowTest". We are going to mock the Database invocation, so you do not have to depend on external resources.

2. Go to the palette and select the **Mock When** component.

3. Drag & drop in the Behavior section
4. Click on the magnifier button icon in the Mock component configuration.

5. In the **General** Panel complete with the following data
   - Processor: `db:select`
   - In the **Then return** combo, select **Edit inline**
   - Check **Payload**

6. Press `fx`

7. Press `fx`

8. Now complete the field **Value** with this representation of a product simulating the Database response:
[
    {
        id: 1,
        name: "ELMO",
        description: "Playskool sesame street play all day Elmo",
        product_number: "UUID34234923934293",
        manufactured: true,
        safety_stock_level: 540,
        standard_cost: 78,
        list_price: 99.9,
        size: "200x300",
        size_unit_measure_code: "in",
        weight_unit_measure_code: "gr",
        weight: 400,
        days_to_manufacture: 10,
        categories: "playskool, sesame street",
        stock: 2800,
        colors: "blue, red, green",
        images: "/wcsstore/HTSStorefrontAssetStore/Attachment/HomePage/Product_eSpots/Play-All-Day-Elmo.png",
        created_date: now(),
        modified_date: now()
    },
    {
        id: 2,
        name: "ELMO",
        description: "Playskool sesame street play all day Elmo",
        product_number: "UUID34234923934293",
        manufactured: true,
        safety_stock_level: 540,
        standard_cost: 78,
        list_price: 99.9,
        size: "200x300",
        size_unit_measure_code: "in",
        weight_unit_measure_code: "gr",
        weight: 400,
        days_to_manufacture: 10,
        categories: "playskool, sesame street",
        stock: 2800,
        colors: "blue, red, green",
        images: "/wcssstore/HTSStorefrontAssetStore/Attachment/HomePage/Product_eSpots/Play-All-Day-Elmo.png",
        created_date: now(),
        modified_date: now()
    }
]
9. Check the assert called "Assert That - Payload is Expected", adding a simple expression that checks that the result is the expected one. In this case you can simply check that the "id" that comes in the JSON response equals "1" (which is what you set it to in the mocked response) extracting it with a DataWeave inline expression.

Set these attributes on the component:

a. Expression: #[payload[0].id]
b. Is: #[MunitTools::equalTo(1)]
c. Message: The response payload is not correct!

10. To run the test, right click on the flow name and select "Run MUnit test". This will open the MUnit Tab and shows the execution.
11. If everything goes OK, you will see the test executed successfully.

If you have any problems, call over the instructor to assist.

You can download a complete project from exchange.

**Summary**

In this module we ran through a full set of tests, and added unit testing into the API.

- **Overview**

We saw how easy it is to use MUnit to provide unit test coverage for the API.

See the MUnit documentation for more information.