Lab 2

Design the Omni Channel API
Overview

In the first lab of this module we learned how to discover the Omni Channel Experience API published in Anypoint Exchange. But of course, somebody had to design this API in the first place. For the purposes of this exercise, let's ignore the fact that the Omni Channel Experience API already exists. Instead, let's take on the role of the API designer and explore how to design the experience API using a Design First approach with MuleSoft’s API-led Connectivity methodology. The goal of our Design First approach is to be able to design APIs that are easy to use for their intended target audience.

Anypoint Platform provides first-class tooling to support the needs of the API Designer. Anypoint Design Center provides a robust RAMl editor for designing the specifications of the API and standing up a "mock" service to accelerate development efforts. It is also tightly integrated with Anypoint Exchange to simplify the process of discovering and referencing other API design-oriented assets like common data types, traits, schemas, and data examples.

For the purposes of this workshop we will design new API's using REST. RESTful API's are very popular right now and support a variety of patterns that help solve many reliability, scalability and integration challenges you might face. We will use an API-first design approach using the RESTful API Modeling Language (RAML) standard. We will follow REST best practices to promote adoption to your consumers. This lab will also introduce the main concepts of resource oriented design and will show how RAML is used to bring APIs to life. For more information on RAML, please see: www.raml.org.

In this lab, we will start designing a new REST API and test it before implementing it. We will use Anypoint Platform’s Design Center to design the API. We will then use Anypoint Exchange to search and find data definitions and examples published across the organization that enable us to standardize on common data elements. Lastly, we will utilize the Mocking Service to unlock resource dependencies of your API design. This significantly cuts down the time spent building the omni-channel application by turning the RAML design over to the omni-channel developers immediately and adopting a rapid prototyping style. Developers can utilize the mocked up API capabilities to produce a "working" application that calls the Mocking Service.
Understanding Resource Oriented Design (ROD)

Designing REST interfaces is a bit of a paradigm change compared to SOAP services. SOAP services design was thinking in actions and methods whereas ROD wants you to think along the lines of your business entities that should be modeled as resources. And resources might be everything from a document, a video, your favorite device or an order process.

Not long ago we modeled interfaces by making Java methods externally available, having action oriented or (insert your programming language) method naming, e.g. getProducts(String name) or the equivalent getter method setProduct(Product product).

Now there is a maturity model that measures the alignment of your Web APIs with the REST architecture style. Martin Fowler provides an excellent explanation of Richardson's model.

Reaching level 2 and maybe 3 is desirable and a ROD targets to achieve that. ROD will help you create stable, mature and scalable APIs that are equipped to evolve nicely, even within Use Cases you can't foresee currently.

The quintessence on ROD is:

1. **Use standard methods** (e.g. HTTP GET, PUT, POST, DELETE, HEAD) and notice their specifications - e.g. GET, HEAD, DELETE and PUT are idempotent methods
2. **Use standard response codes** for success and error; use existing patterns e.g. not every successful call should return a 200 OK - often a 201 Created is appropriate after a POST /products or a 202 Accepted when a long running task was started by the server
3. **Use standard header parameters**
4. Design and craft resources in your business language - here is where RAML comes into play
5. **Interlink** as many of your resources as you can; e.g. when you return a collection of products, have every product in the result carry its own link so the client can easily access it

NOTE This is not a mandate to do REST purist's HATEOAS but to start creating a web of resources and avoid clients to know URI patterns, which results in unnecessary and avoidable coupling

Collection Resources

Typically resources are designed as either collection or instance resources. Collection resources hold a collection of resources that can be queried, ordered, extended and most importantly linked.

Let's take a typical example of a product resource. Most likely there are several of them and you want to be able to create new products also.

By designing a **/products** resource a list of products is available. Typically the methods

- POST (create new list instance)
are available on the list.

A further abstraction is to move the list query onto a \texttt{/products/search} resource that one can then nicely scale independently (e.g. by caching) and offer all the tailored search functionality needed like specific query parameters.

**Instance Resources**

Instance resources contain the details of a specific resource, i.e. the instantiation of a resource. For our product example that might include a product id and the name of the product. Highly valuable are then links to other products that might be part of that product.

Instance resources must be uniquely identify-able and typically look like this \texttt{/products/product/{instance_id}}. Typically the methods

- GET (retrieve product information)
- PUT (update product information)
- DELETE (delete product)

are available on the product resource.

Now we understand resources let's explore how we can work with the resources.

**Creating an Instance Resource**

An instance resource is created by \texttt{POST}ing against a collection resource. There are multiple ways to implement that. One could argue that a \texttt{POST} request against the list resource creating an instance should already carry all instance details in the body and then returning the created resource.

This is one way, sadly it is not the most reliable when something goes wrong and the response fails. Then a new \texttt{POST} needs to be issued which creates a new instance on the server side.

A more reliable and recommended pattern is the following. First a client issues a \texttt{POST} on a list resource. But instead of the server returning a complete instance already, it returns the link to the instance in the Location header carried with a \texttt{201 Created} response. Now the client can poll the instance resource using an idempotent GET or PUT until it is successful. In case something went wrong the client can simply continue polling without creating a new instance every time as a get request is idempotent by specification.

**Step 1: Review the E-Commerce Application Requirements**

You have a new requirement for a new e-commerce application that customers can use for shopping and ordering products through a web portal or mobile device. The e-commerce
application has the following requirements we are asked to fulfill. It is written in the language of the e-commerce developer and we will derive the RAML specification from that.

**Authentication/Registration:**

- **Login**
  - Mail
  - Password
  - Returns all user information

- **Signup**
  - Name
  - Mail
  - Password
  - Returns all user information

**Functionalities**

- **Products:**
  - List all the products available
    - Name
    - Price
    - Stock
    - Id

- **Orders**
  - Create a new order
    - Returns order id
  - Add product to order:
    - Order id
    - Product id
    - Quantity
  - Remove item from order
  - Confirm order
    - Shipping address data
    - Credit card data
  - List orders of the user
    - Full data

Let’s design these now in RAML.
Step 2: Create the Omni Channel Experience API

In this step, you will create an API and design it using the Anypoint Design Center.

1. You can access Anypoint Design Center from the home page of Anypoint Platform:

   ![Anypoint Platform UI](image1)

   Alternatively you can use the "hamburger menu" from any page within Anypoint Platform portal to navigate to Design Center:

   ![Exchange UI](image2)
2. **Anypoint Design Center** is used to design your API's. When you arrive at the Design Center landing page you will see a list of API's that have already been designed by your organization.

3. To create a new API, Click on the **Create** button. Select "API Specification".

4. Select "Create API Specification".
5. Set the following values in the Add API form:

- **API name**: `<username> - Omni Channel Experience API`.
- **Visual Editor**: Select the Visual editor radio button.

*Note: Since we are working in a shared organization, please give your API a unique name. For the purposes of this exercise please prepend your API with your Anypoint account username.*

6. Click **Create Specification**. This will create the basic structure of your RAML-based API specification. Note that for full RAML support, you should start with the first option (API Designer), which is the standard RAML-based API Designer.
Step 3: Create the Orders List Resource

In order to provide order handling functionality for the e-commerce application, we will design a list resource in our API using RAML.

**Anypoint Design Center** will provide a visual API editor to start with a step-by-step tutorial to guide you through designing your API visually.

1. Let’s create our API and skip the tutorial for now. Click Hide this tips. (Feel free to go through the tutorial later on).

2. Let’s add some general details about the API.
Name the API by setting the title to `<username>` Omni Channel Experience API.

Since this is our first design let’s designate this API as version 1.0. Set version to 1.0.

Set the media type to application/json.

Most API specifications will designate an API endpoint Base URI. This tells the API consumer where they will access the API. Set the URI to `http://localhost:8081/api`

You should now see the following in your API design:

![API Design Interface](image.png)

**NOTE**

While you are visually creating the API, Anypoint Design Center presents the real-time RAML and OAS Definition on the right panel. This will continue as we add resources to your API. You can switch between RAML and OAS view with the bottom right tabs.

3. Now you need to add a resource. A resource is the definition of an entity that your API will manage. Click the + to the right of Resources, then choose Add resource.
4. Rename the resource from `/new-resource` to `/order`. Select the POST option and under documentation, give it a description: *Creates a new Order instance* as shown. Notice the generated RAML on the right.

Congratulations! You have completed creation of an /order resource with a basic description.

**Step 4: Create the Order Instance Resource**

In the previous step we created an Order collection resource and POST method to enable the e-commerce application to create a new Order instance. What if we wanted to let the e-commerce application get the status of a single order? We need to create a new resource underneath the Order
instance and use the GET method.

1. Hover the mouse cursor over the `/order` resource. You will notice that a + appears in the resource panel.

2. Press the + button to create a new, nested resource. Notice that a default name of "new-resource" has been created. In addition, the GET method has been selected by default.

3. Rename `/order/new-resource` to `/order/{order_id}`. Curly brackets indicate dynamic variables where a value will need to be passed when called.

4. As we did in the previous step, enter a description: *Order instance resource allowing to retrieve an order*
5. Under the GET method click on the **Responses** tab, then click on the **Add New Response** button.

6. You should now see a response with a 200 OK response code.
7. Under the response, click **Add Body** to add a JSON response to the GET method.

8. Notice the RAML specification on the right updated with the body.
9. Let’s add an example JSON to the response. We need to expand the “bodies” section by clicking on “Details”.

10. Once expanded, you will see the “Example” section.
11. Under “Example”, click the Edit tab and copy/paste the following JSON code
{  
"order_id": "order_22",  
"tracking_code": "12310391209318",  
"rel_self": "http://alc.mulesoft.com/api/orders/order/order_22",  
"creation_date": "2015/12/31 10:00:00",  
"cost": {  
  "price": 12,  
  "tax": 1,  
  "final_price": 13  
},  
"shipping": {  
  "shipping_address_id": "1122",  
  "rel_address": "http://alc.mulesoft.com/api/users/user/vip_user_8/address/1122"  
},  
"items": [  
{  
  "product_id": "121",  
  "quantity": 1  
},  
{  
  "product_id": "122",  
  "quantity": 2  
]  
}
Step 5: Enable the Mocking Service

A very useful feature during the design phase is having the ability for your consumers to interact with your API without having to code anything. The API editor provides a mocking service that will read the RAML file, create the API/service and return example data responses. The mocking service allows users to interact with the API as if it was built and deployed. This feature allows you to rapidly iterate the API design with its consumer to finalize the contract.

1. Look at the baseUri in the RAML Viewer. Notice it is set based on what we entered in the API Summary.

2. Enable the mocking service by switching the Mocking Service to ON on top of the RAML viewer:

   ![Mocking Service Enable](image)

   After a few seconds, the mocking service will start, creating a new URL for the baseUri:

   ![Mocking Service URL](image)

   Once the Mocking service is enabled, you can click Try It at the bottom of the RAML viewer to submit an API request and receive the example response you defined. Let's try to retrieve an Order using the Mock service.

3. Click Try It.
You will see the following page
4. Enter any value in the **order_id** text and press send.

5. Scroll down and you should see a 200 response with the Order record in a JSON format that adheres to the example you set in the API definition.
Step 6: Switching between RAML and OAS view

Inside the Anypoint Platform you can work with RAML and OAS seamlessly. You can check that going to the right panel, where the specification code is shown, and selecting on the bottom the OAS tab.
Mulesoft supports OAS 2.0. The specification can be either a YAML (.yaml) or JSON (.json).

Click the RAML tab to continue with the next step.

**Step 7: Edit the RAML for more advanced RAML capabilities**

In the Step 3 and Step 4 of this lab we created resources and methods for our Omni Channel Experience API utilizing Anypoint Visual API Designer. This is good when you are a beginner. As you get used to define your RAMLs, you would like to edit them by yourself.

So first we need to change the view.

1. Click the Edit RAML button to go to the RAML Editor mode and confirm that you will be using the code editor instead of the visual editor.
NOTE Once you have switched to editing RAML directly, you will not be able to switch back to the Visual Editor.

2. On the popup window, please select **Yes, send me to the code editor**.

3. Press **Continue**

You should see the RAML editor.
4. Place the cursor at the end of the RAML specification.

The screenshot and table below gives an overview of the various sections of the API Designer that you will use.

**TIP**
To see the shelf options, you need to be in a new line to see the suggested commands.

<table>
<thead>
<tr>
<th><strong>Project Explorer</strong></th>
<th>Shows list of all files in your API design project. File types supported include RAML, JSON, and any other custom file format.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAML Editor</strong></td>
<td>Where you define your API</td>
</tr>
<tr>
<td><strong>Shelf</strong></td>
<td>Where you have suggested commands available to use depending on where you are. This is contextual. You need to go to a new line to see it.</td>
</tr>
<tr>
<td><strong>API Summary</strong></td>
<td>This is a live preview of your API. As you modify your RAML, the changes will be reflected on this pane. This is what users of your API will see and use to understand how to use the API.</td>
</tr>
</tbody>
</table>

**Step 8: Reusing an Asset from Exchange**

In the Step 4 of this lab we created a GET methods for our Omni Channel Experience API and defined the response with a hard-coded JSON example. Wouldn’t it be great if we could provide to developers an a reusable fragment that would be returned from this API instead of hard-coding? Well, that is exactly what the new version of Anypoint Exchange allows you to do.

1. Let’s provide an example of what the e-commerce application should expect when invoking the GET method on the /order/{order_id) resource. There are several ways to declare the example response in RAML, but let’s see how easy it is to use Exchange to search for and incorporate a **RAML fragment** into our project.
2. You should now see the project dependency section of your API specification. RAML allows API designers to not only define the resources, methods, and attributes of the API, but it also allows you to reference external files containing fragments of RAML. These fragments can define commonly used data definition, examples, traits, and resourceTypes, making the API design process easier and more consistent across the organization.

3. Click on the + button to add an Exchange dependency:

4. You should see a list of RAML fragments.
5. For the purposes of this workshop we have created a simple example of an order in Exchange. Use the search bar to find the asset titled **Order Instance Example** and add it to your project:

6. Add the RAML fragment to your project by ticking the checkbox and pressing the **Add** button.
7. The API Designer will now pull the **Order Instance Example** RAML fragment from Exchange.

8. To see these files, click on the **Files** icon at the top left of the API Designer:

9. Expand **exchange_modules** and select **order-instance-example.raml**. Your Project Explorer should look something like this:
The above RAML fragment on the right defines an example of what an Order instance might look like. By defining this example using a format-neutral language like RAML (instead of a JSON object which is what is expected to be returned) we give the API designer flexibility to change the method's data format (i.e. from JSON to XML) and still be able to create a valid example.

10. Remove the hardcoded example we wrote on Step 4
At this point we have only referenced the RAML fragment from our project. The last step is to designate the Order example as the example: data for the /order/{order_id}: resource.

Let’s start by copy the Resource complete path into the clipboard.

11. Click on the ellipsis menu:
12. Now, return to the bottom of our main API specification file and insert the following text:

```plaintext
examples: !include
```

**NOTE** Notice that we are changing example by examples.

The `!include` directive allows you to create a relative path reference to other files in your RAML project.

13. Paste the pathname next to the `!include` directive in the RAML file.

14. Take a look at the API Summary. You could see the Get resources under `/{order_id}`
15. Click on the **Get** resource and you will see the description:

- Operation description
- URI Parameters
- Response examples

**NOTE**  Note that you will have to scroll down to see the entire API Summary
16. Scroll to the bottom of Get resource summary. Notice there is a bottom scroll bar as well.
17. Scroll to the top right of **Get** resource summary. Notice the bottom scroll bar disappears.
Note that the Try it button is in a different location when using the RAML Editor instead of the Visual API Editor.

18. Press Try it button and test the application again. You will see the following page:
19. Enter any value in the **order_id** text and press send.
20. Scroll down and you should see a 200 response with the Order record in JSON format. Notice that the data displayed is from the example file you included.
Summary

In this lab, you completed the following steps:

- Step 1: Review the E-Commerce Application Requirements
- Step 2: Create the Omni Channel Experience API
- Step 3: Create the Orders List Resource
- Step 4: Create the Order Instance Resource
- Step 5: Enable the Mocking Service
- Step 6: Switching between RAML and OAS view
- Step 7: Edit the RAML for more advanced RAML capabilities
- Step 8: Reusing an Asset from Exchange

We easily created and designed a new experience API for our e-commerce order entry application, providing the ability to submit orders, get orders and check order status. We leveraged RAML for a design first approach.
We saw how the mock service can be utilized to provide application developers an API mock up they can build their applications on. This significantly speeds up end to end development.

We highly recommend you take the time to explore the API Specification for the full Omni Channel Experience API located in Design Center. It provides a comprehensive example of an experience API specification and can be used as a proper example when designing new API's.

To learn more about RAML follow this tutorial.

See Designing Your API for more information on API design.

Also review the following free ebooks on API design.

- Undisturbed REST: A Guide to Designing the Perfect API
- 7 Habits of Effective API & Service Mgmt
- 7 Steps to Building Your API Blueprint
- Ben Kepes on Why APIs are Critical to Business

Congratulations! You have completed Lab 2.

Please proceed to Lab 3