# Lab 6: Generate the API Project

[Lab 6: Generate the API Project](#_86maahp7gk5r)

[Overview](#_2w1bfxad0hk)

[Step 1: Export the Order Entry API RAML](#_d0062bjx8kmk)

[Step 2: Create a new Mule Project from the RAML](#_z9opaf6efk5z)

[Step 3: Test the API on the Mule run-time](#_me8wfn9gqott)

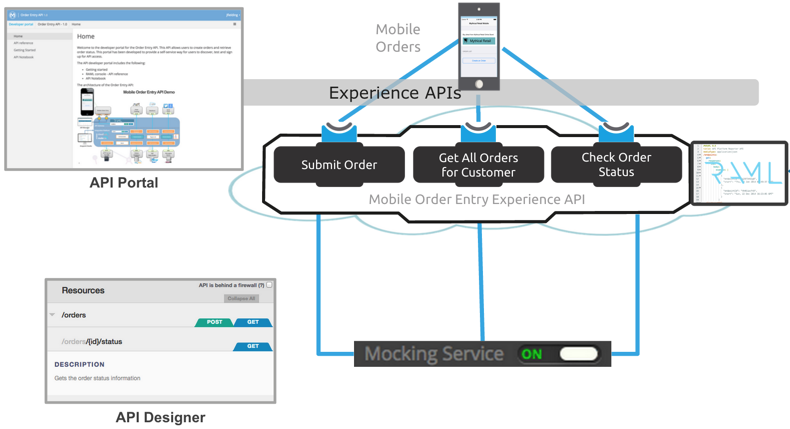
[Step 4: Test the API from a Mobile Application](#_2zop2p90jeqo)

[Summary](#_eauu08fs6i0p)

## Overview

You have completed the design of your Order Entry API and now you are ready to implement it. In this lab, we will create the implementation of our API that will process REST requests and orchestrate calls to internal systems.

You will use Anypoint Studio to create a Mule application where there will be one flow for each method of each resource (i.e. GET orders). Additionally you will use APIKit (as part of your Mule application) to process REST requests, transform them to messages to be handled and processed by each flow.



The implementation will consist of a few steps

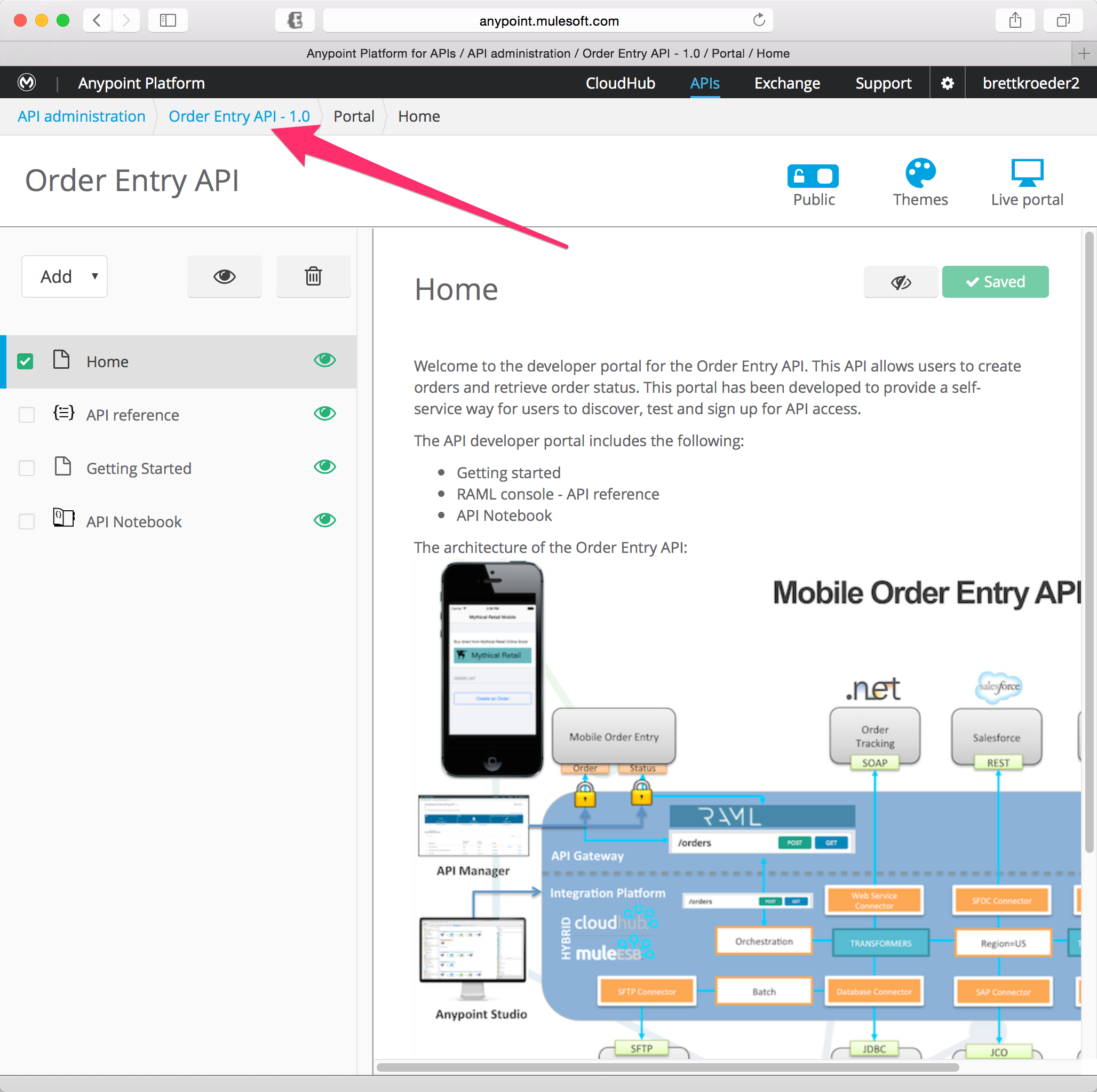
1. Export the Order Entry API RAML from API Designer
2. Create a new Mule Project in Anypoint Studio from the Order Entry API RAML
3. Run the Mule app from Anypoint Studio

Then we will test this new API using the API Console and a Mobile application.

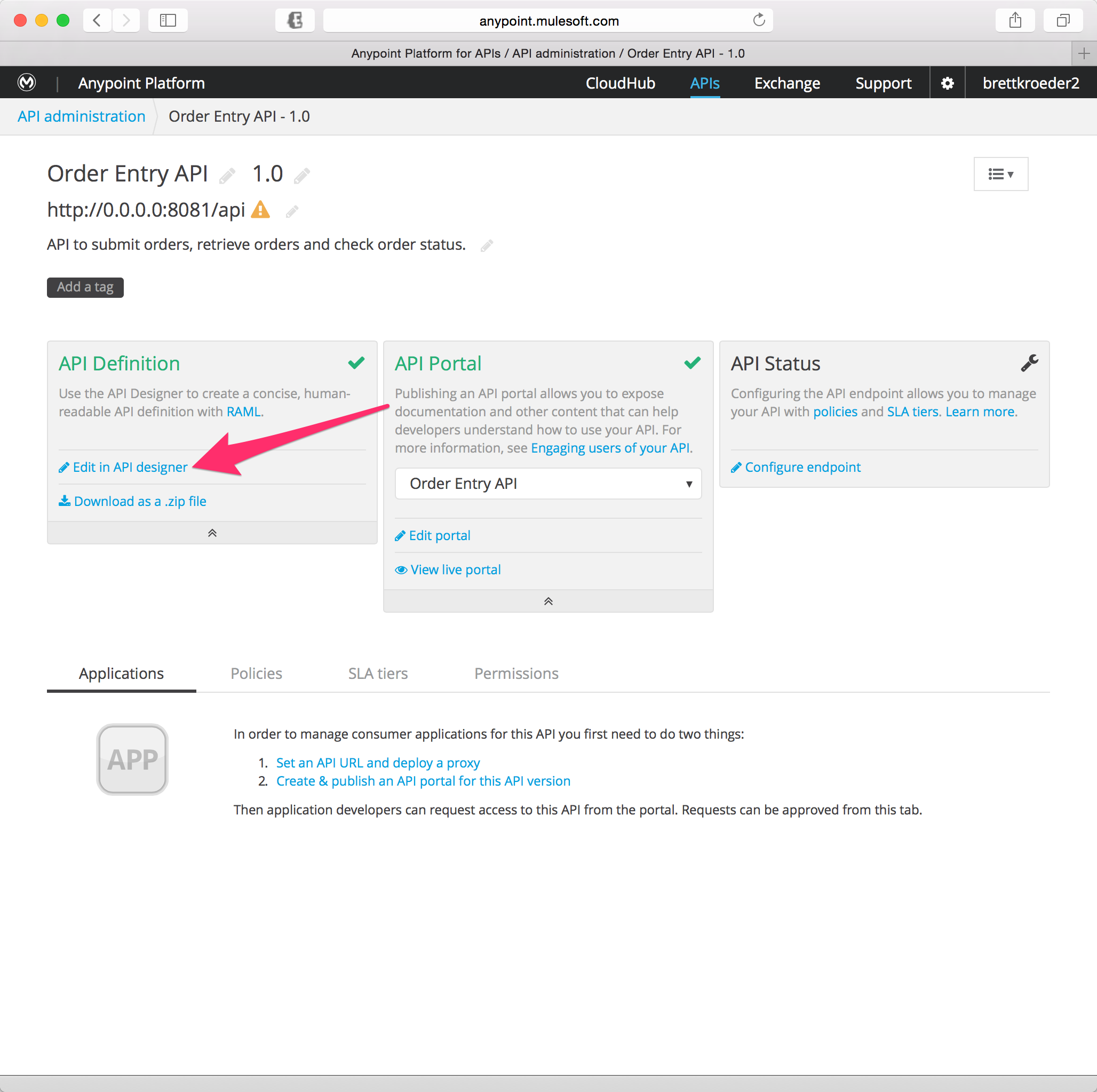
In this application we will submit a new order, then retrieve existing orders, and finally get the state of a specific order.

## Step 1: Export the Order Entry API RAML

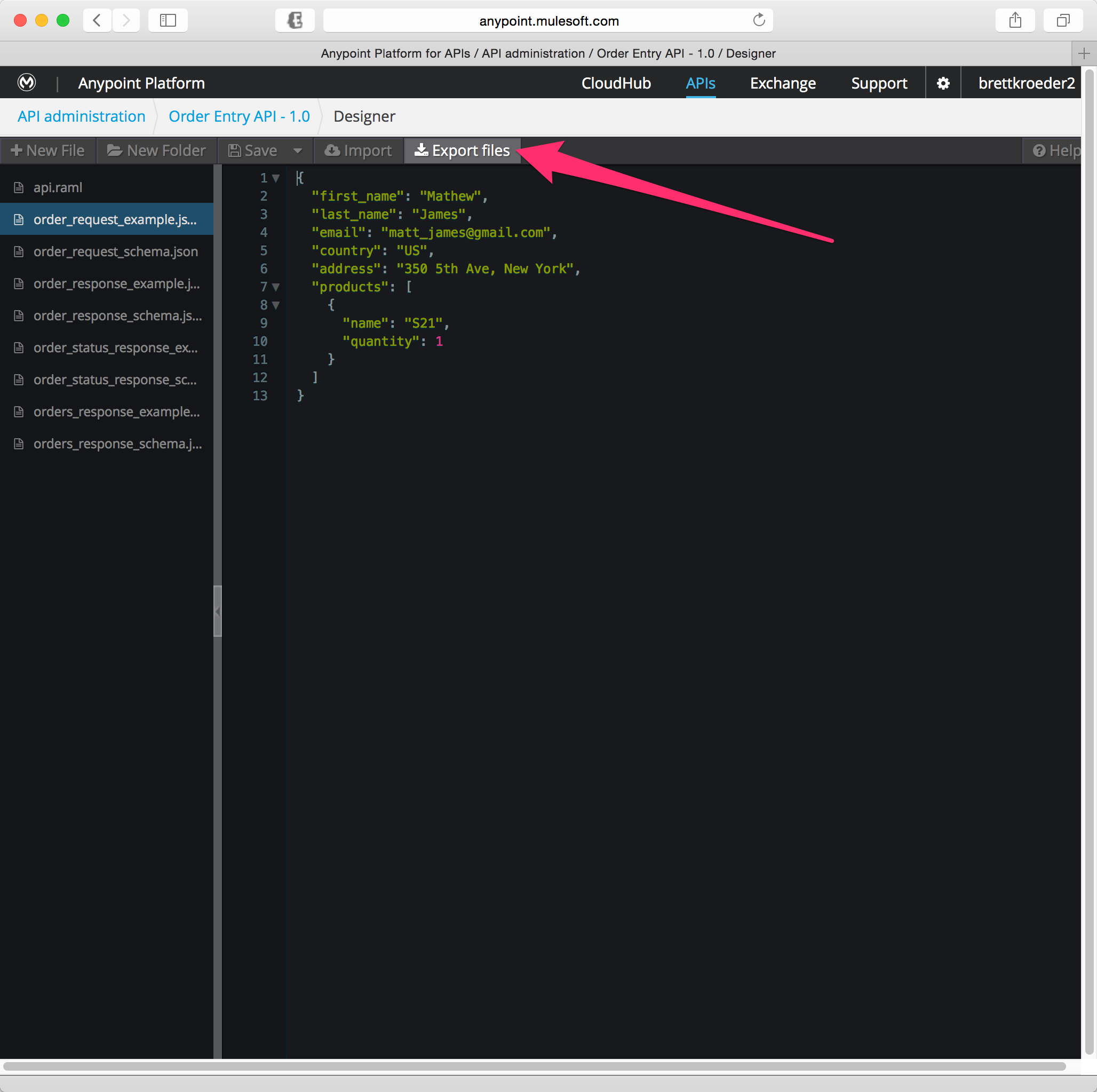
In this step, we will use our API definition to create an implementation of our API. In the previous lab we exported the API definition.



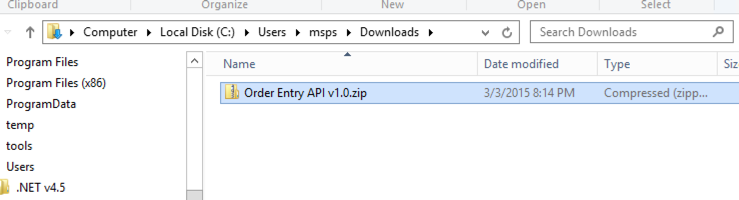
1. Click on the *Order Entry API - 1.0* breadcrumb to navigate back to the API definition.



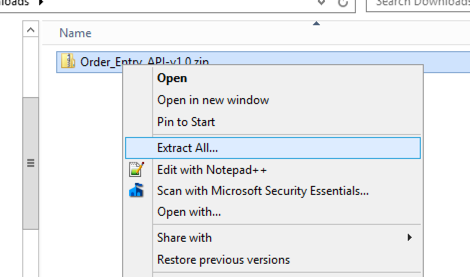
1. Click on *Edit in API Designer*.
2. Go to the and select **Export files**.

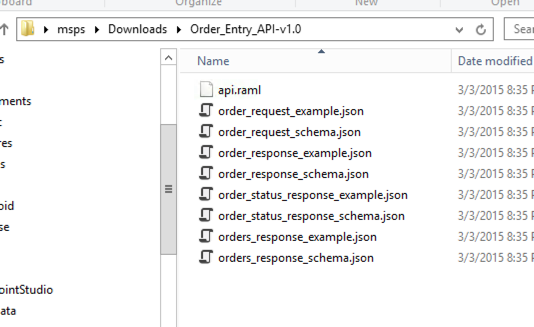


1. Click on *Export files*. This will export the API definition files into one zip file.



1. Save the export file on your desktop or in any directory on your workstation.
2. Extract the file and examine the contents.





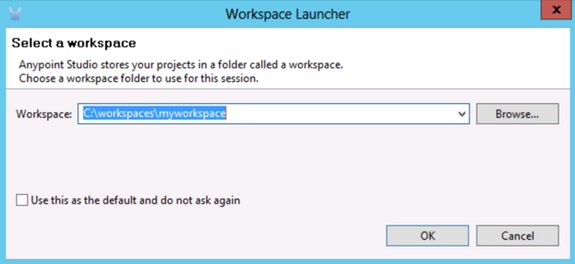
The RAML file contains the definition of your API, and the JSON files contain the schemas for the request and response objects. With these, we will create our API implementation.

## Step 2: Create a new Mule Project from the RAML

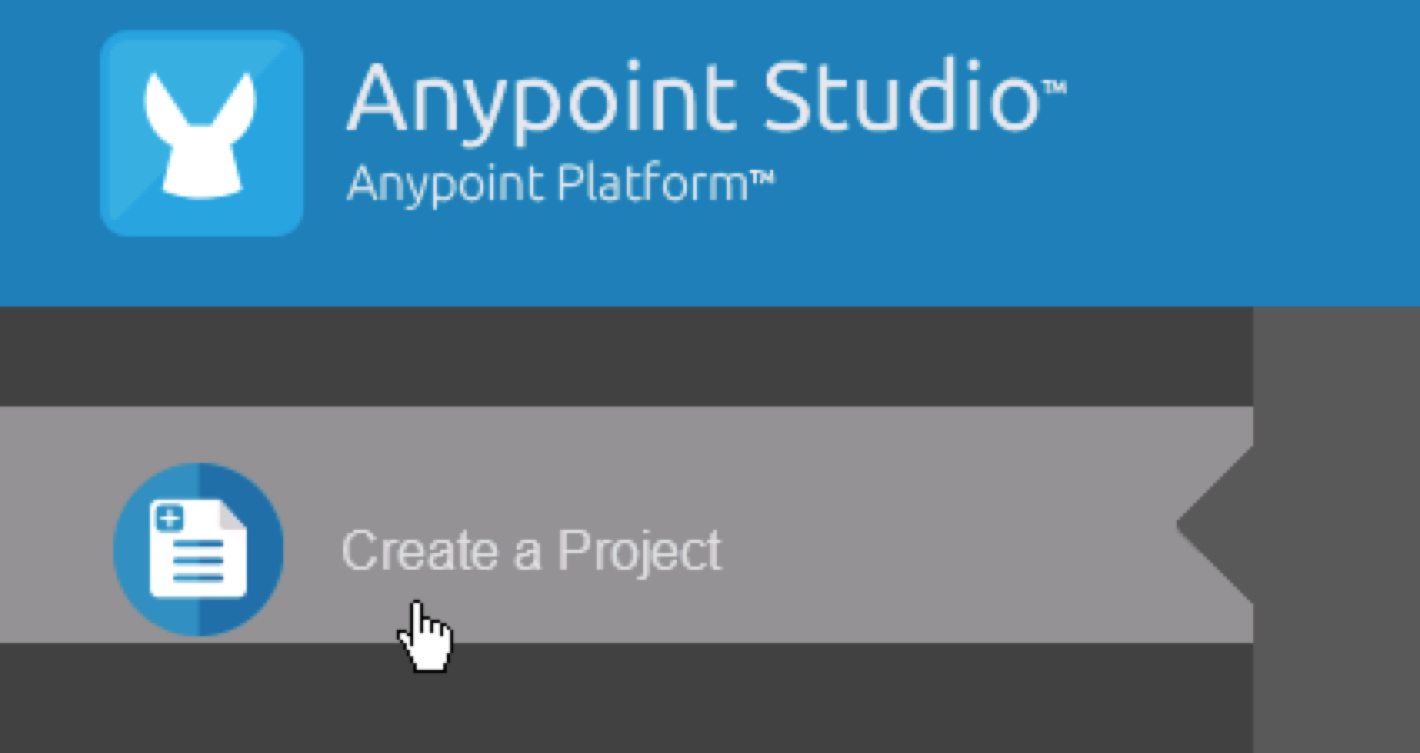
In this step we will create a new Mule application in Anypoint Studio from the Order Entry API RAML. This will be the implementation of our REST API.



1. Start Anypoint Studio.

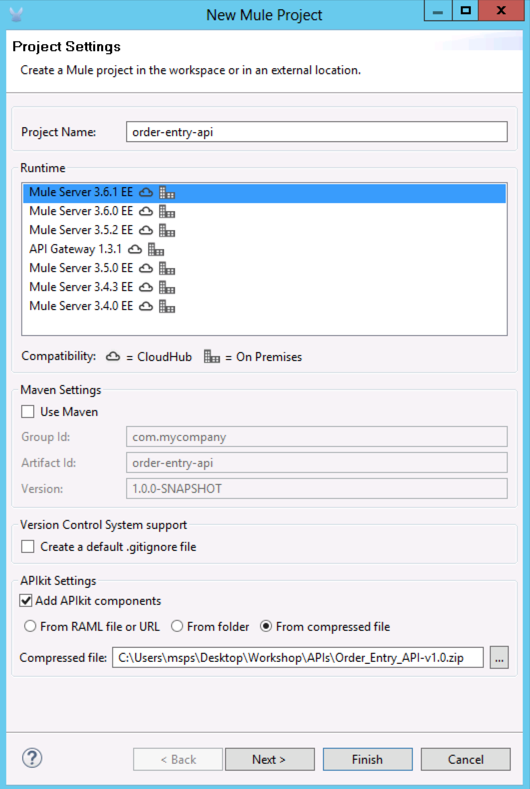


1. In the Workspace Launcher specify a new workspace such as C:\workspaces\myworkspace.



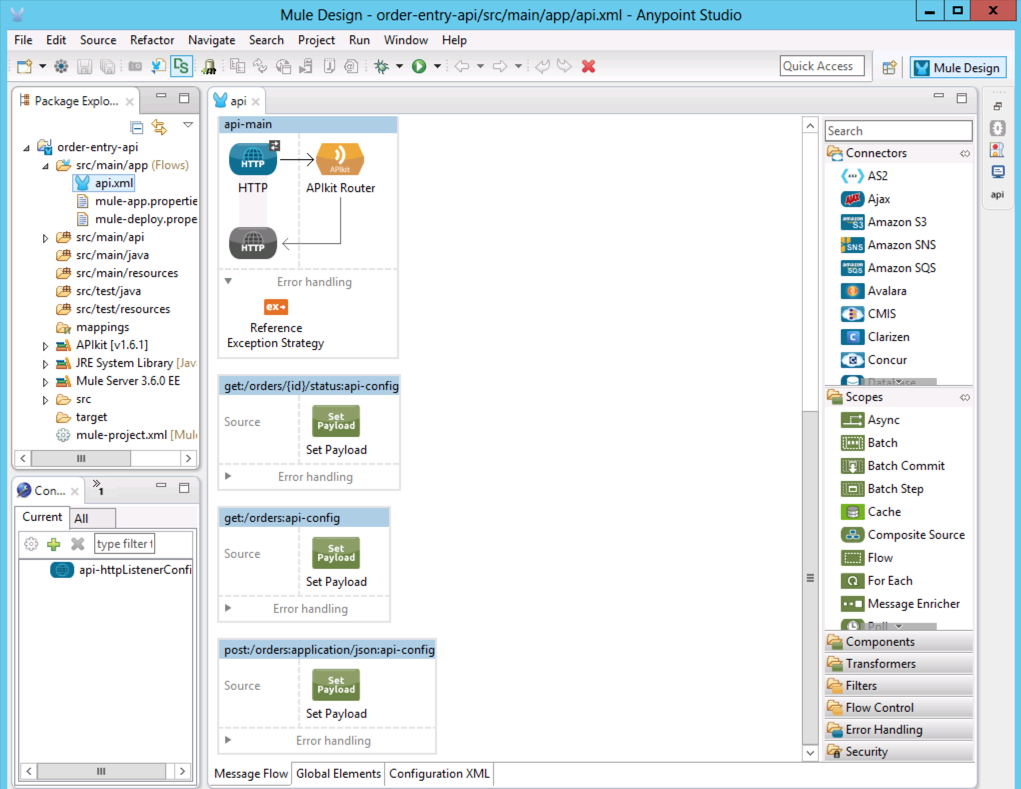
From Anypoint Studio,

1. Choose Create a Project from the Welcome Screen, or select **File** > **New** > **Mule Project**.
2. A Window will pop-up to define the details of this new application.



1. Give the project the name **order-entry-api**
2. Select the **Mule Server 3.6.1 EE** run-time.
3. Under APIkit Settings check **Add APIkit components**
4. Choose **From compressed file**
5. Click the … to select the file
6. Select the zip file that you downloaded: **C:\Users\msps\Downloads\Order\_Entry\_API-v1.0.zip**
7. Click **Open**
8. Click **Finish**. This will create a skeleton project that implements your API.

|  |
| --- |
| **NOTE**: It will take 1-2 minutes to generate the project. |

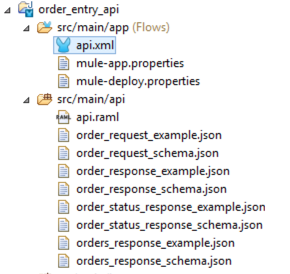


Let’s explore what was automatically generated.

1. Scroll down to the bottom of the diagram and you will see a flow called **api-main** with three sub-flows, one for each method defined in the RAML.

|  |  |
| --- | --- |
|  | **api-main**  This is the main flow. It exposes an HTTP service and processes the requests using the APIKit Router.  The HTTP request will be converted to a mule message, and redirected to the requested flow by the APIKit router.  If you take a look at the HTTP configuration, you will see that its listening for requests on **http://localhost:8081/api** |
|  | **get:/orders/{id}/status**  This flow processes the requests to "GET" an order's status by an order ID. |
|  | **get:/orders**  This flow processes the requests to "GET" all items from the order collection. |
|  | **post:/orders**  This flow processes requests to "POST" or create a new order in the orders collection. |

The sub-flows above are defined by your API in the RAML file. Typically, there will be flows that look like this get:/resource, post:/resource, put:/resource, etc. Note that the name of the flow is very important for the APIkit router to be able to route the request to the appropriate flow - you don’t want to change these.

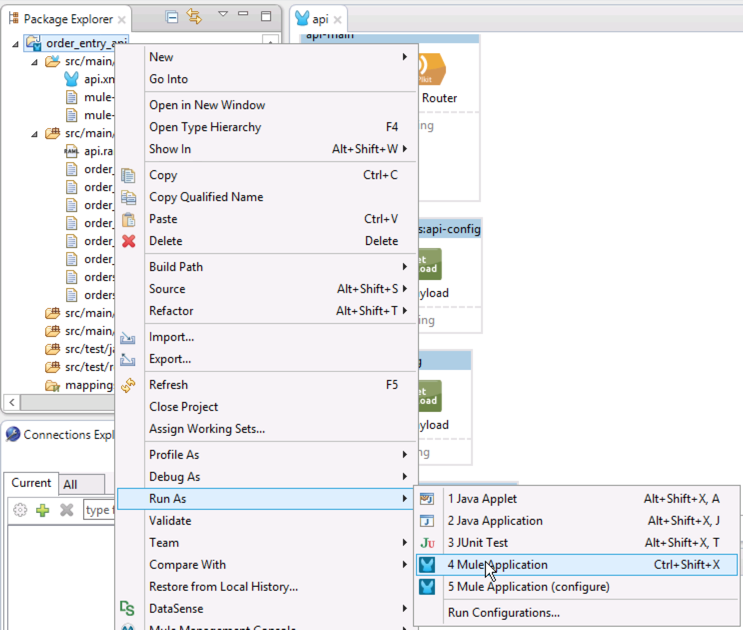


When APIkit detects example data in the response of a method in the RAML it inserts a **Set Payload Transformer** into the flow which returns the static response specified by the example data reference.

|  |
| --- |
| **NOTE:** If you do not see the **src/main/api** folder with the api.raml files and corresponding JSON files (as shown in the screenshot above), the project generation was not successful. Try deleting/re-generating your project or alert your instructor to troubleshoot. |

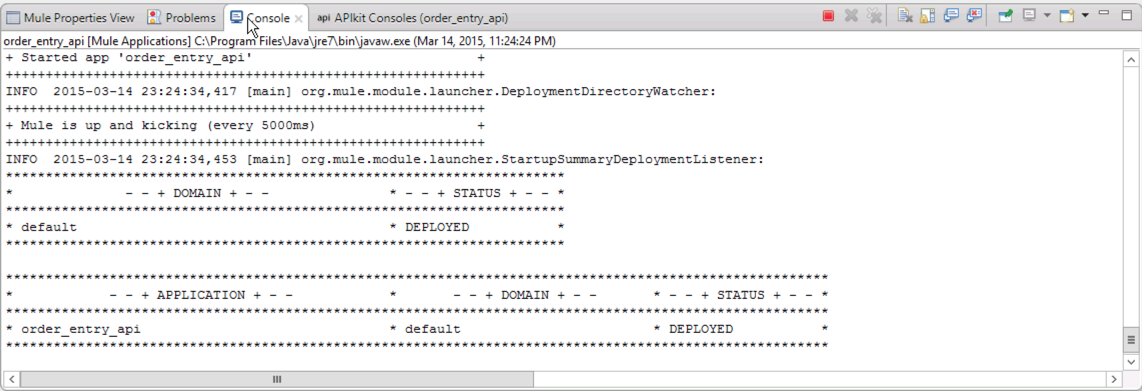
The static response returned by the auto-generated **Set Payload Transformer** allows you to to test the stub immediately after generation. Obviously these flows can be enhanced to provide more advanced mock services as well as evolve into the full API implementation.

## Step 3: Test the API on the Mule run-time

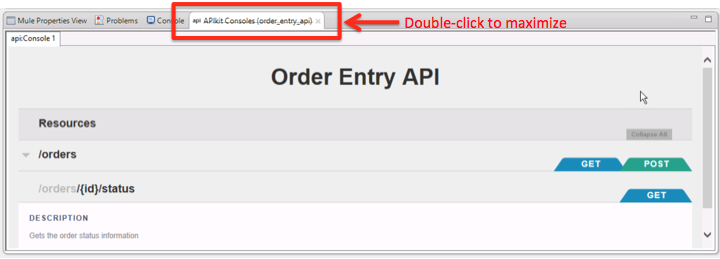


To test the API, let’s run it within Studio first.

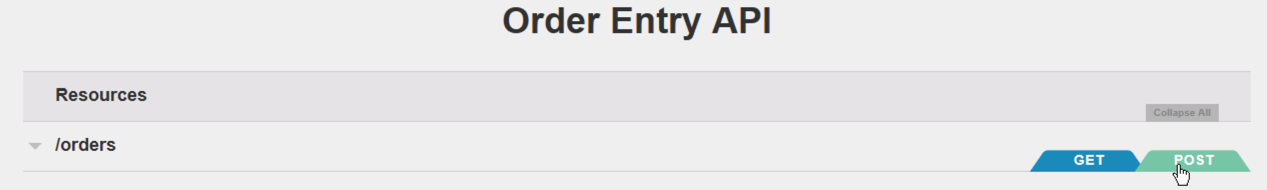
1. Right click the application.
2. Select **Run As > Mule Application**.



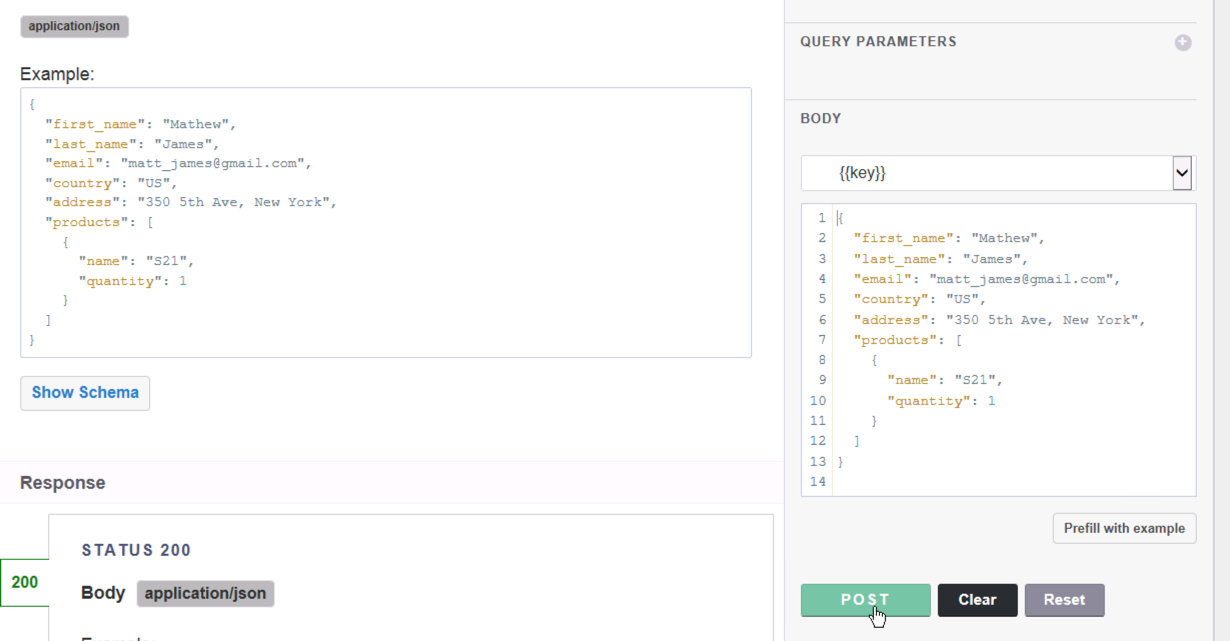
|  |
| --- |
| **NOTE**: Anypoint Studio deployed this application to an embedded Mule run-time. There is no need to deploy to a separate Mule server environment. The developer will be able to develop and test the application locally until it's ready to be deployed to a shared development or QA environment. |



1. Test the application using the same API console you worked with in API Designer and the API Portal. To maximize the console just double-click the APIKit Console tab.



1. Click the POST tab to test the **Submit an Order** method.

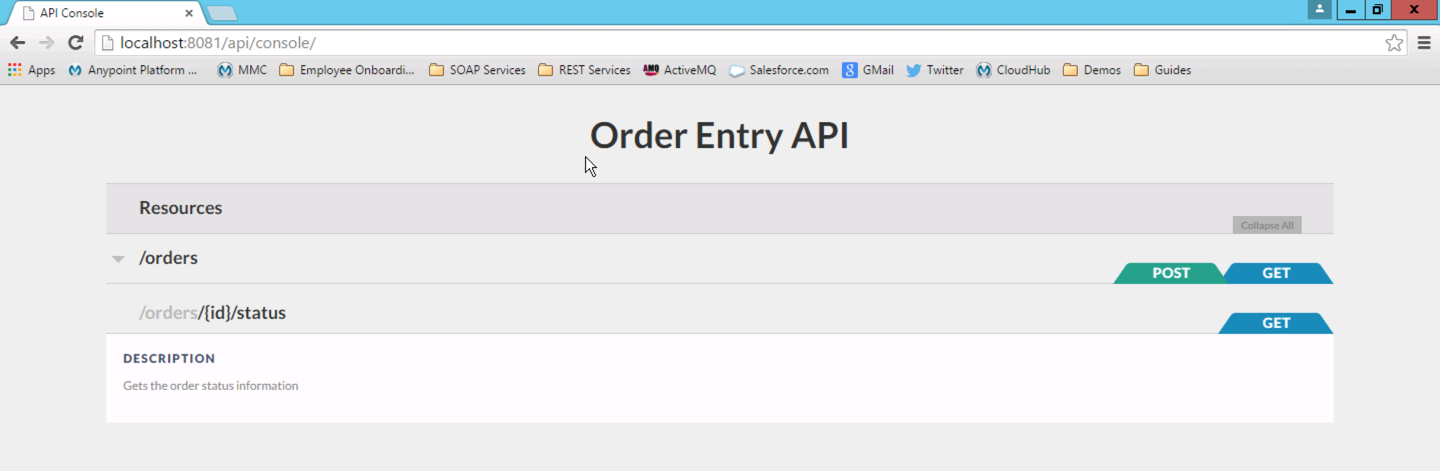


1. Scroll down below, you’ll see the request body
2. Click **Try It**.
3. Click the POST button.



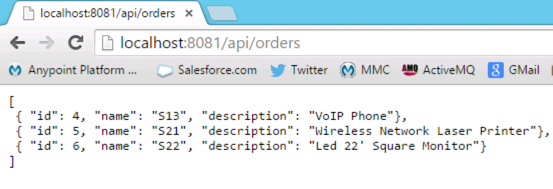
1. You should see the response above.

|  |
| --- |
| **Remember...** this is no longer a mocking service responding. It is a live application deployed to the Mule run-time server. You just tested your end-to-end API running in your environment. |

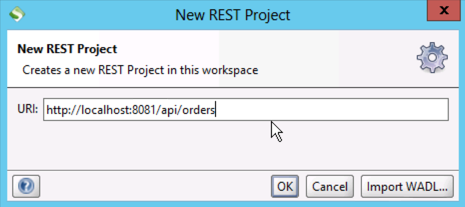


1. Access the console from your web browser by entering the URL <http://localhost:8081/api/console> and run the same test.

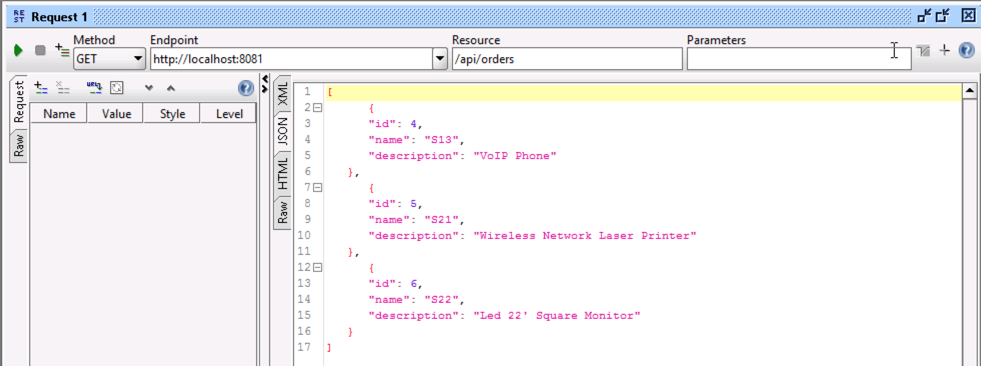
|  |
| --- |
| **NOTE:** We added /console to the URL. APIKit router exposes the RAML console by default at <api url>/console. This behavior can be modified to use a different URL or to use a completely different HTTP endpoint. |



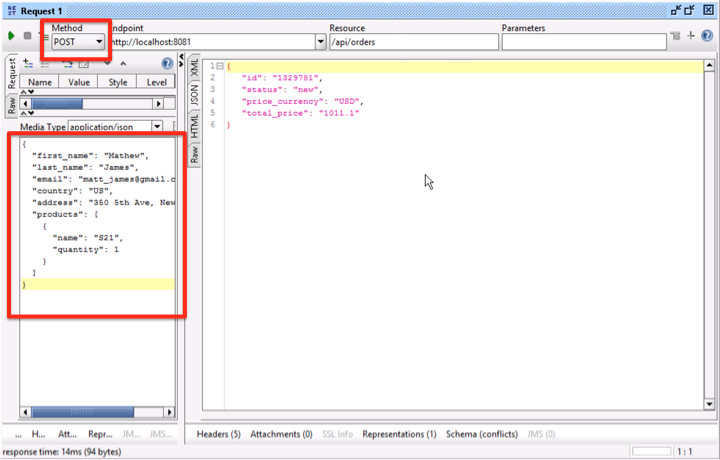
1. Access the GET /orders using your browser by entering the URL [**http://localhost:8081/api/orders**](http://localhost:8081/api/orders)**.** You should see a list of orders.



1. Access the GET /orders using from SOAP-UI by creating a new REST Project with the URL [**http://localhost:8081/api/orders**](http://localhost:8081/api/orders)**.**



1. Click the icon and you should see a list of orders returned.
2. Click the JSON tab on the response to see the data returned in JSON format.



Lastly, test the POST /orders in SOAP-UI, same as what you did earlier to test the mocking service.

1. Open the **order\_request\_example.json** file
2. Copy the JSON
3. Paste the Order example into SOAP-UI

{

"first\_name": "Mathew",

"last\_name": "James",

"email": "matt\_james@gmail.com",

"country": "US",

"address": "350 5th Ave, New York",

"products": [

{

"name": "S21",

"quantity": 1

}

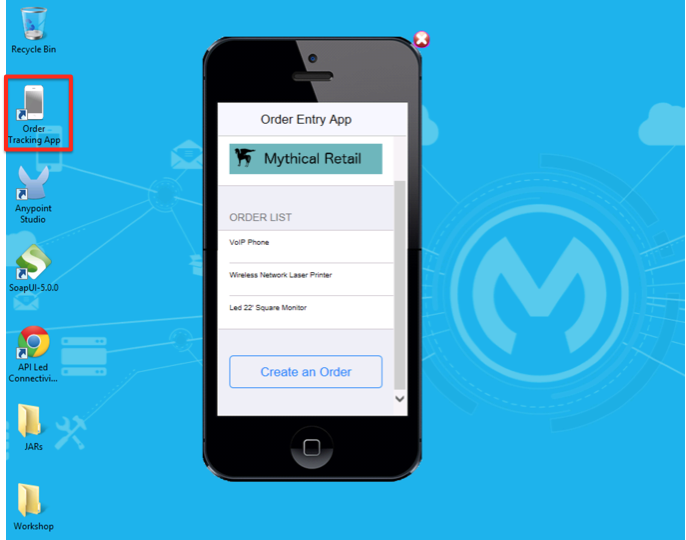
]

}

1. You should see the response in the above SOAP-UI window.

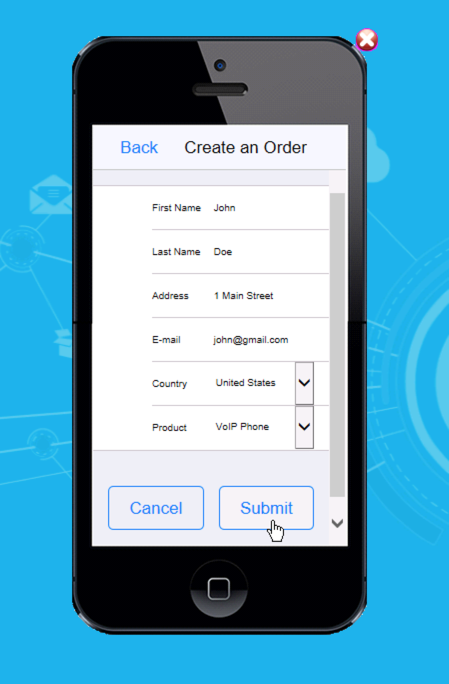
## Step 4: Test the API from a Mobile Application

Your API application is running on the embedded Mule server, lets try to hit it using a mobile application. We have developed a mobile application emulator to test our APIs. This mobile emulator has a mobile app that invokes the REST API located at <http://localhost:8081/api>.

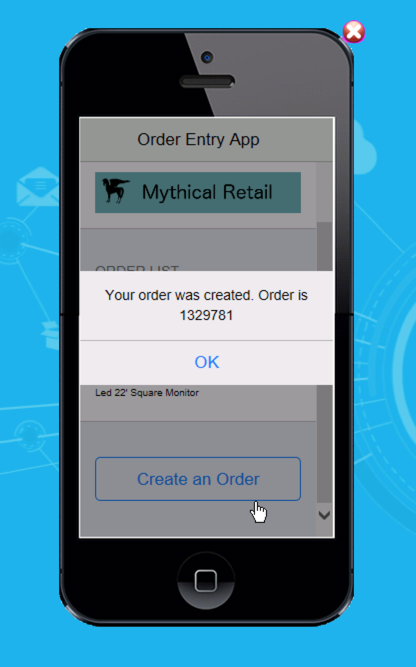




1. On the Desktop there is a shortcut called "**Order Tracking App**", double click on that link and the mobile simulator will open. Alternatively, there is an icon on the Windows task bar that you can use.   
     
   Notice the orders that show up under ORDER LIST. That’s coming from your API.



1. Click on **Create an Order**, you will the order creation screen.
2. Enter your first name, last name, email, address
3. Choose United States
4. Select a product.
5. Click **Submit**.

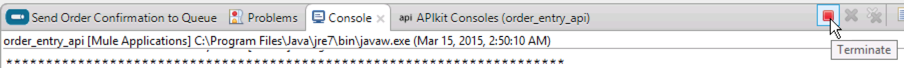


You will get an order confirmation number back. This was returned by your API. Nothing really happened behind the scenes because we haven't fully implemented the **API** yet. In the next lab we will implement this by passing the order information to our back-end systems.

1. Close the Mobile App simulator by clicking the  icon on the top right corner of the phone.

You have completed the initial testing of a deployed version of the Order Entry API. Let’s shutdown the embedded server for now.

1. In the Console window, click the  icon to terminate the server.



## Summary

In this lab you demonstrated how you completed the following steps:

[Lab 6: Generate the API Project](#_86maahp7gk5r)

[Overview](#_2w1bfxad0hk)

[Step 1: Export the Order Entry API RAML](#_d0062bjx8kmk)

[Step 2: Create a new Mule Project from the RAML](#_z9opaf6efk5z)

[Step 3: Test the API on the Mule run-time](#_me8wfn9gqott)

[Step 4: Test the API from a Mobile Application](#_2zop2p90jeqo)

[Summary](#_eauu08fs6i0p)

In this lab, we

* used the RAML specification to define our API
* auto-generated a skeleton project to implement the API

This lab shows how APIkit quickly allows developers to import the API designed REST API (RAML and supporting artifacts) to enable MuleSoft flow development. The APIkit also deploys the interactive console with the MuleSoft flows. The lab shows the extent to which a developer can deploy an on-premise and interactive API from the design spec in RAML.

It is important to note that the implementation of an API is a real Mule application and allowed us to test the end-to-end API. Building integration applications and exposing APIs do not require additional knowledge making it easy for Mule developers to work on both integrations and APIs.

See the [APIkit doc](http://www.mulesoft.org/documentation/display/current/APIkit) for more information.

Congratulations! You have finished Lab 6.

Please update the spreadsheet indicating you have completed Lab 6.