MuleSoft®

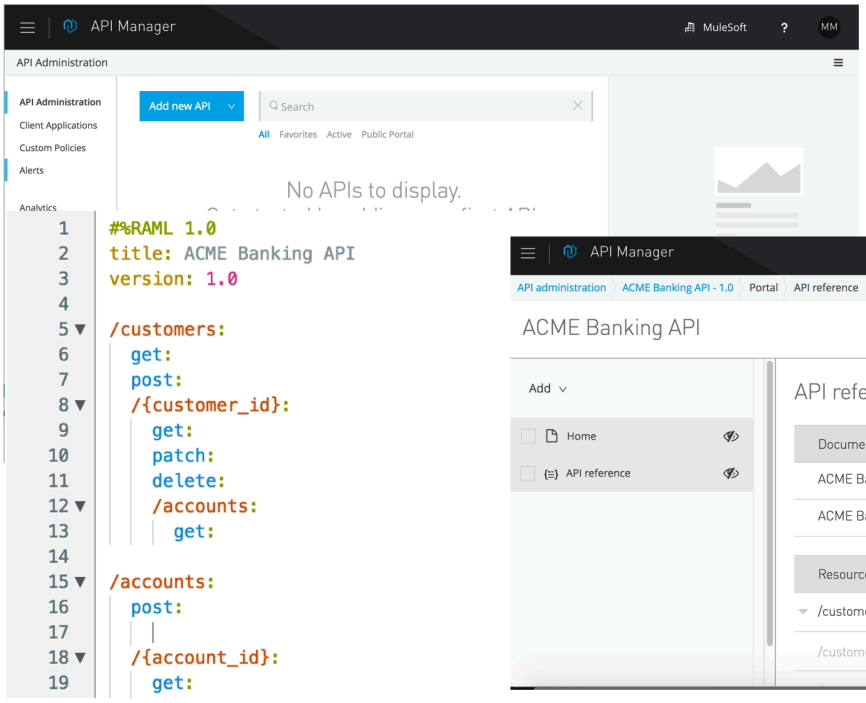
Anypoint Platform Development: API Design

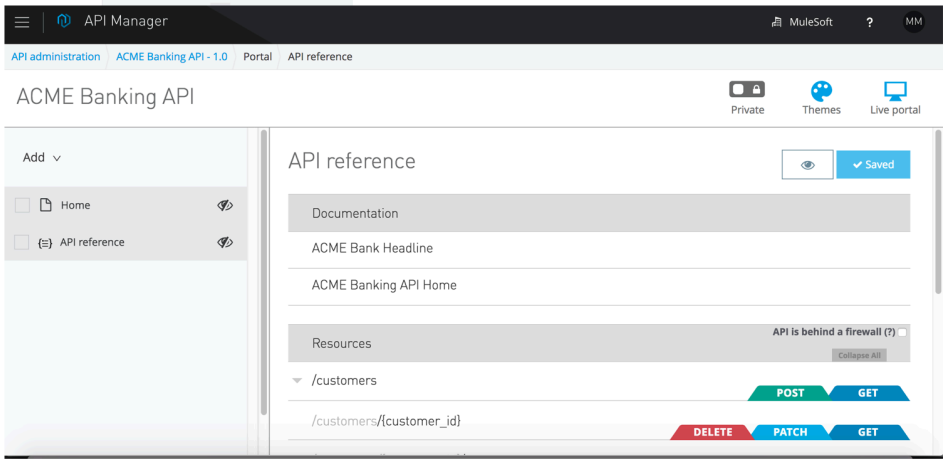
Student Manual

May 5, 2017

Table of Contents

Introducing the Course





Objectives:

- Learn about the course format.
- Download the course files.
- Make sure your computer is set up for class.
- Review the course outline.

MuleSoft

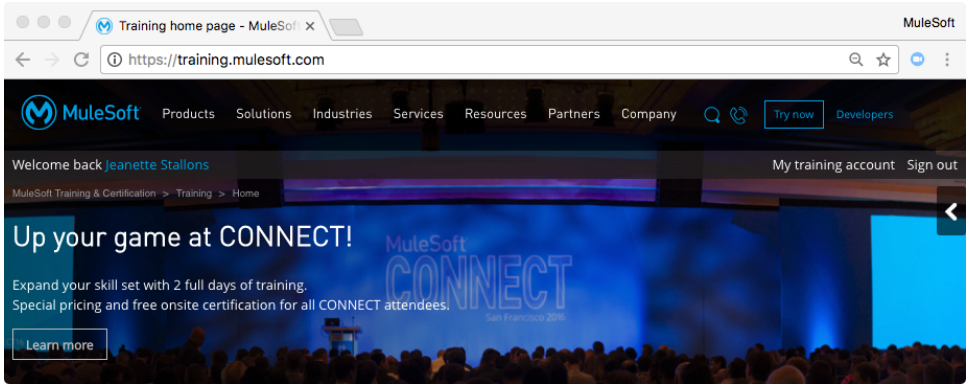
Walkthrough: Set up your computer for class

In this walkthrough, you make sure your computer is set up correctly so you can complete the class exercises. You will:

- Download the course files from the MuleSoft Training Learning Management System.

Download student files

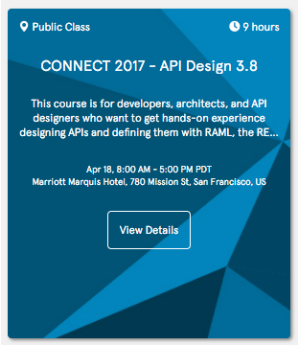
1. In a web browser, navigate to http://training.mulesoft.com.
2. Click the My training account link.



3. Log in to your MuleSoft training account using the email that was used to register you for class.

   Note: If you have never logged in before and do not have a password, click the Forgot your password link, follow the instructions to obtain a password, and then log in.

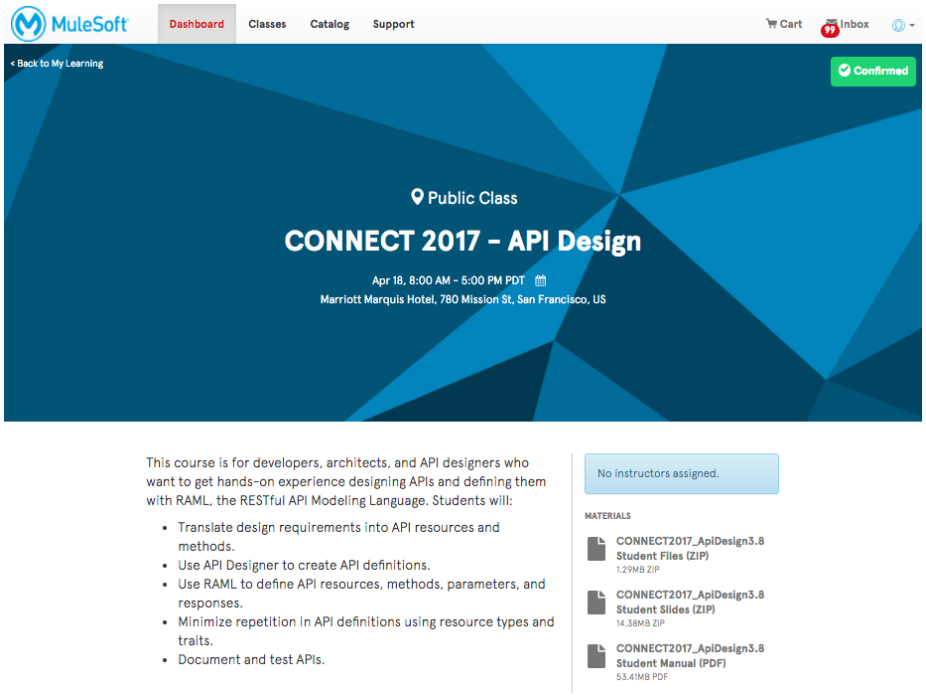
4. On the My Learning page, locate the box for your class.



   Note: If you do not see your event, locate the Current and Completed buttons under the My Learning tab, click the Completed button, and look for your event here.

5. Click the event's View Details button.

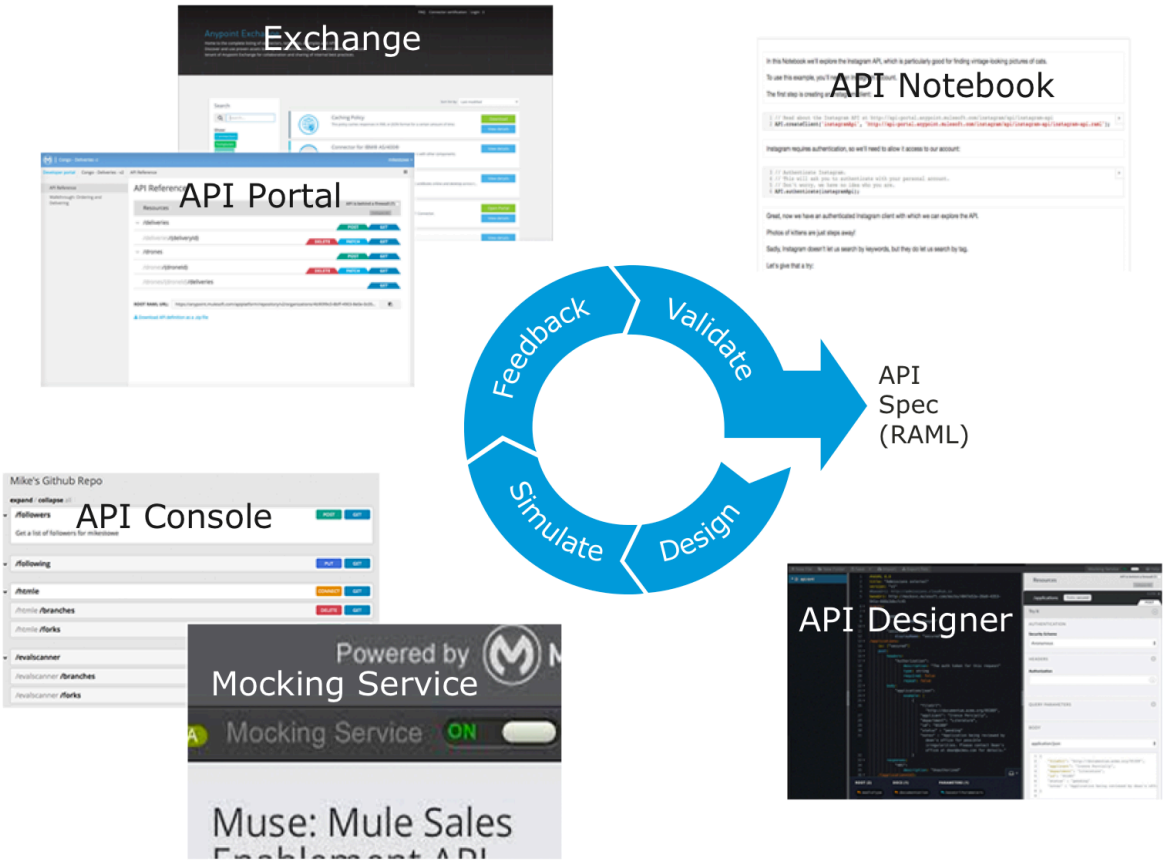
6. Locate the list of course materials on the right-side of the page.



7. Click the student files link to download the files.
8. Click the student manual link to download the manual.
9. Click the student slides link to download the slides.
10. On your computer, locate the student files ZIP and expand it.
11. Open the course snippets.txt file.

    Note: Keep this file open. You will copy and paste text from it during class.

PART 1: Designing APIs



Objectives:

- Describe REST API architecture.
- Describe API development lifecycle.
- Translate functional requirements for APIs into resources and HTTP methods.
- Navigate Anypoint Platform.

Module 1: Introducing RESTful API Design

SOAP

RPC

```
POST /SendUserMessage HTTP/1.1
Host: api.example.com
Content-type: application/json

{"userId": 501, "message": "Hello!"}
```

REST

```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
  post:
  /{customer_id}:
    get:
    patch:
```

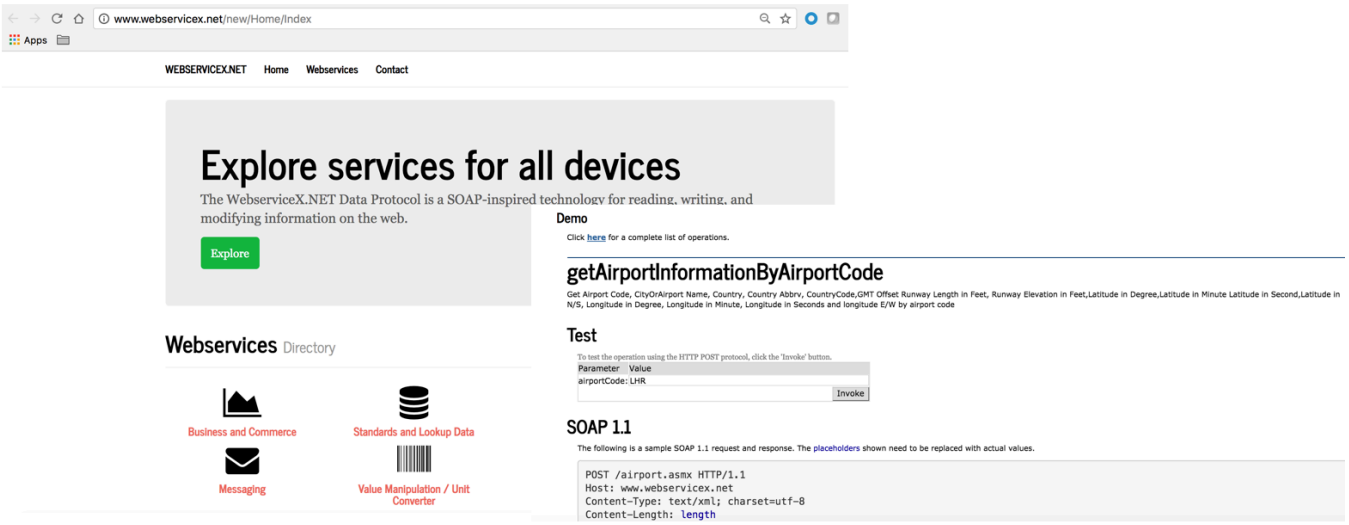
Objectives:

- Describe the common web API formats including SOAP, RPC and REST.
- Describe REST API architecture.
- List the rules for retaining REST principles in APIs.
- Describe design-first approach for REST APIs.

Walkthrough 1-1: Review and make a call to a Public SOAP API

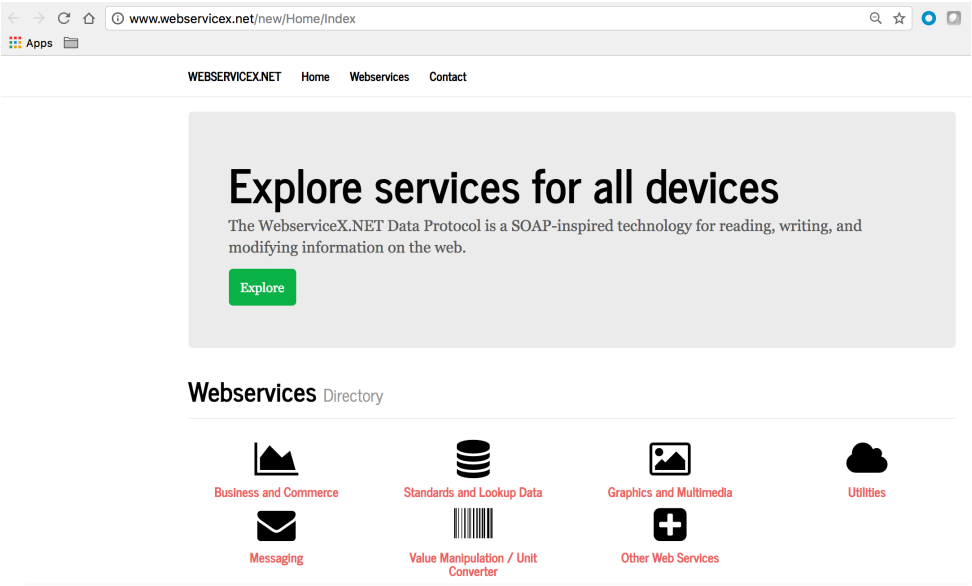
In this walkthrough, you explore and understand a SOAP API. You will:

- Examine an example SOAP API.
- Make a call to a SOAP API endpoint to retrieve information.

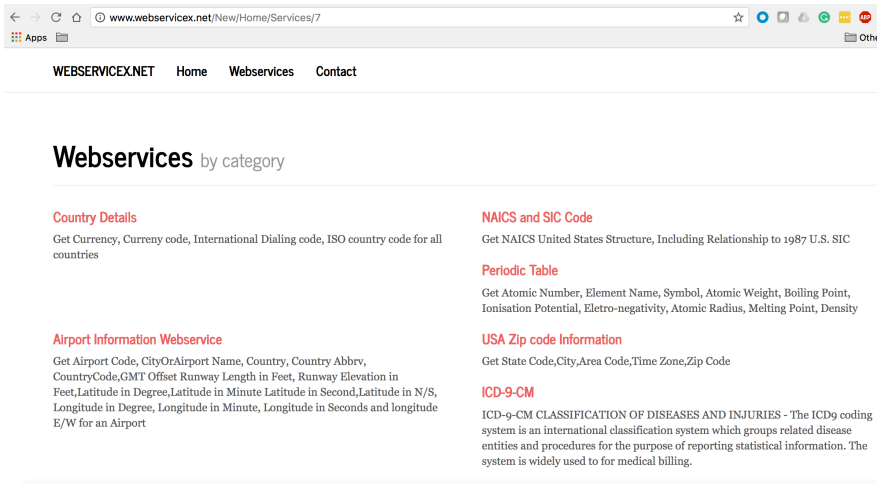


Examine the Airline Information SOAP API

1. Return to the course snippets.txt file.
2. Copy the URL for the SOAP API: http://www.webservicex.net/new/Home/Index.
3. In a web browser, navigate to that URL
4. In the Webservices Directory, click Standards and Lookup Data.

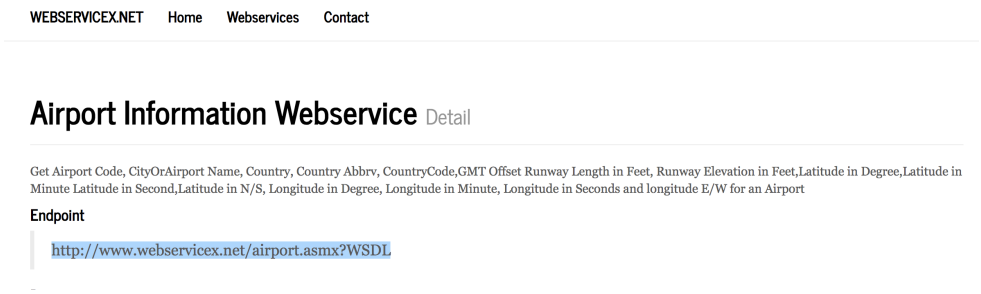


5. In the list of Webservices by category, click the Airport Information Webservice.

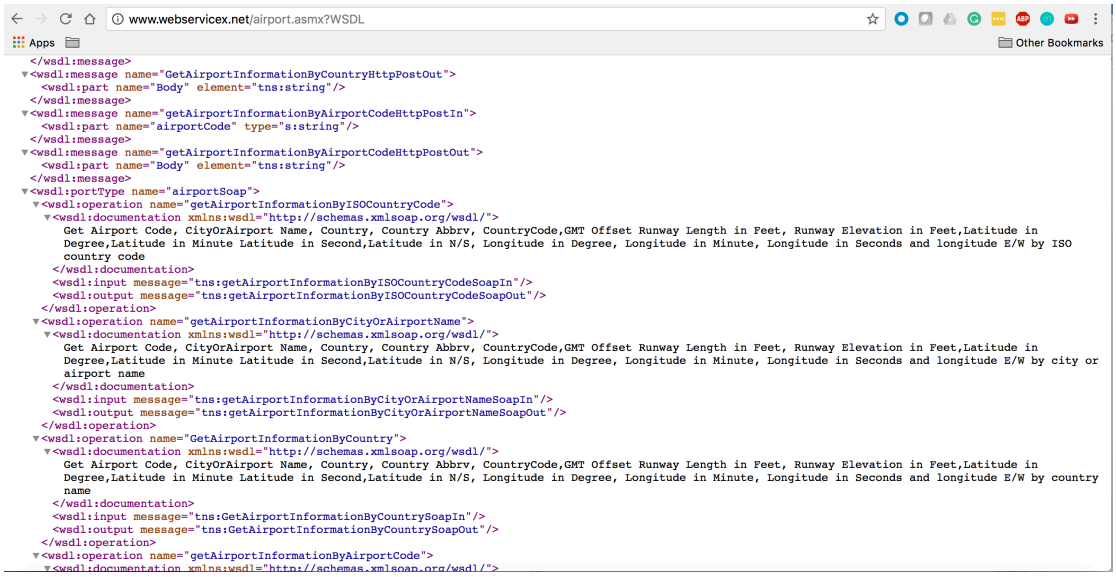


View the Airport Information Webservice endpoint WSDL

6. In the Airport Information Webservice Detail page, copy the link specified for the Endpoint.

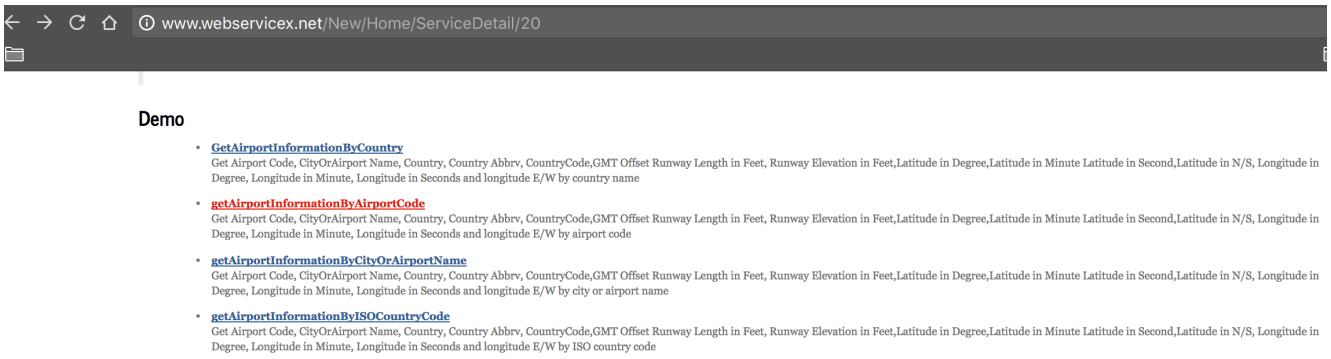


7. In a new tab in the web browser, navigate to the Endpoint URL.
8. Locate the operations and HTTP methods supported by the webservice.

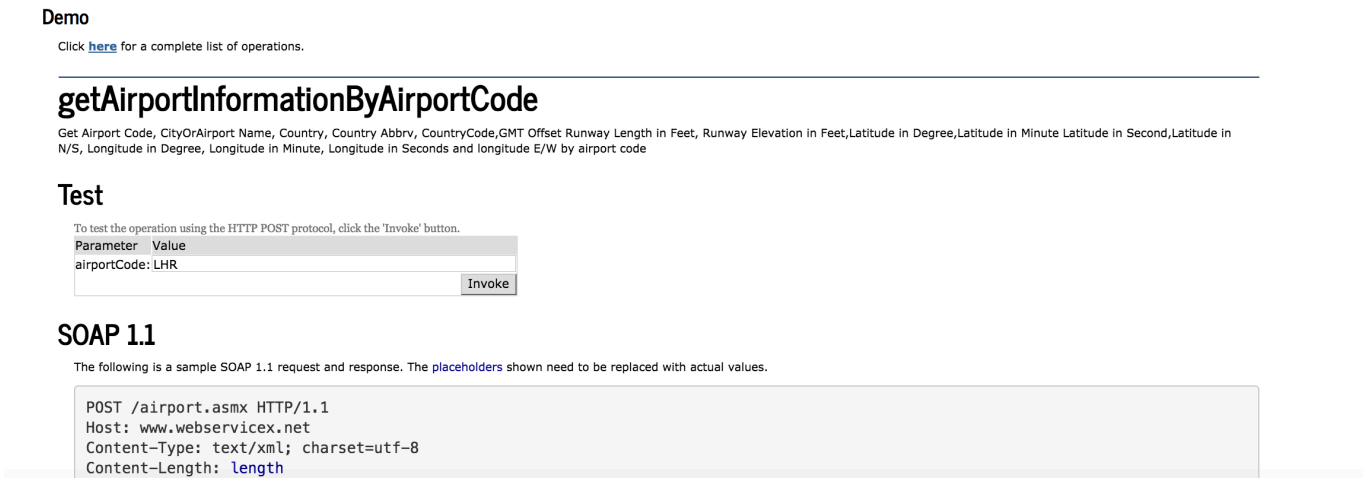


Make a call to retrieve data from an Airport Information Webservice operation

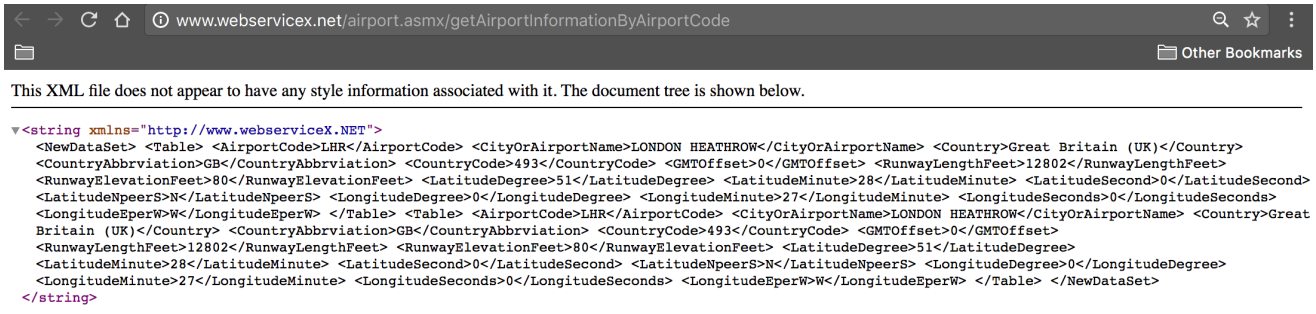
9. Go back to the Airport Information Webservice Detail page.
10. In the list of operations under Demo, click getAirportInformationByAirportCode.



11. In the Test parameter dialog box, type the airportCode parameter value as LHR.
12. Click Invoke.

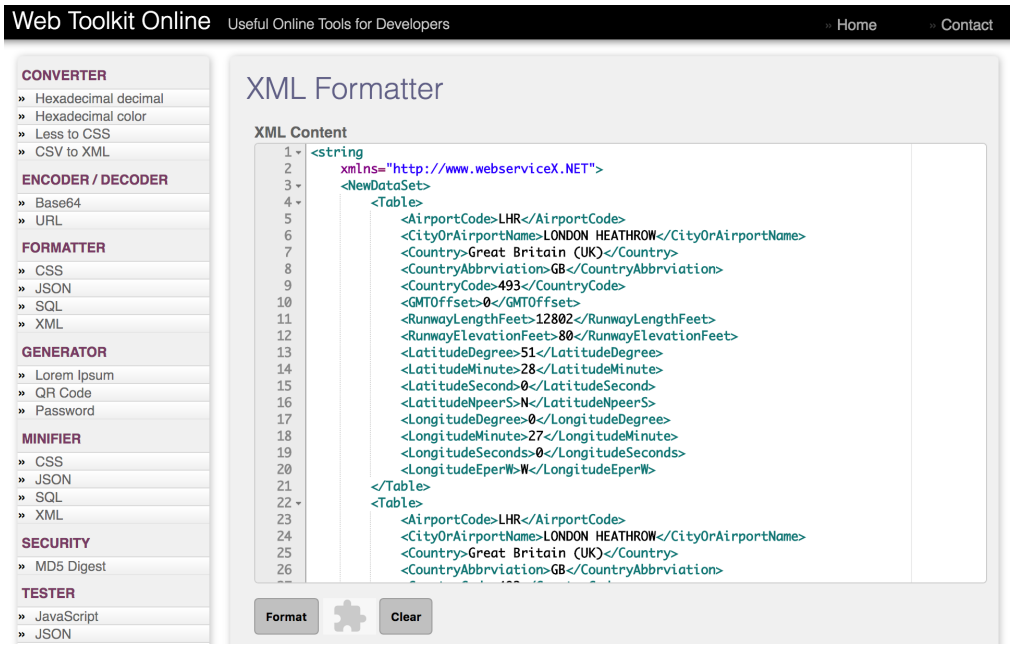


13. Verify that a new tab opens that displays the response received from the SOAP webservice operation.



14. Copy the content after the horizontal line in the response webpage.

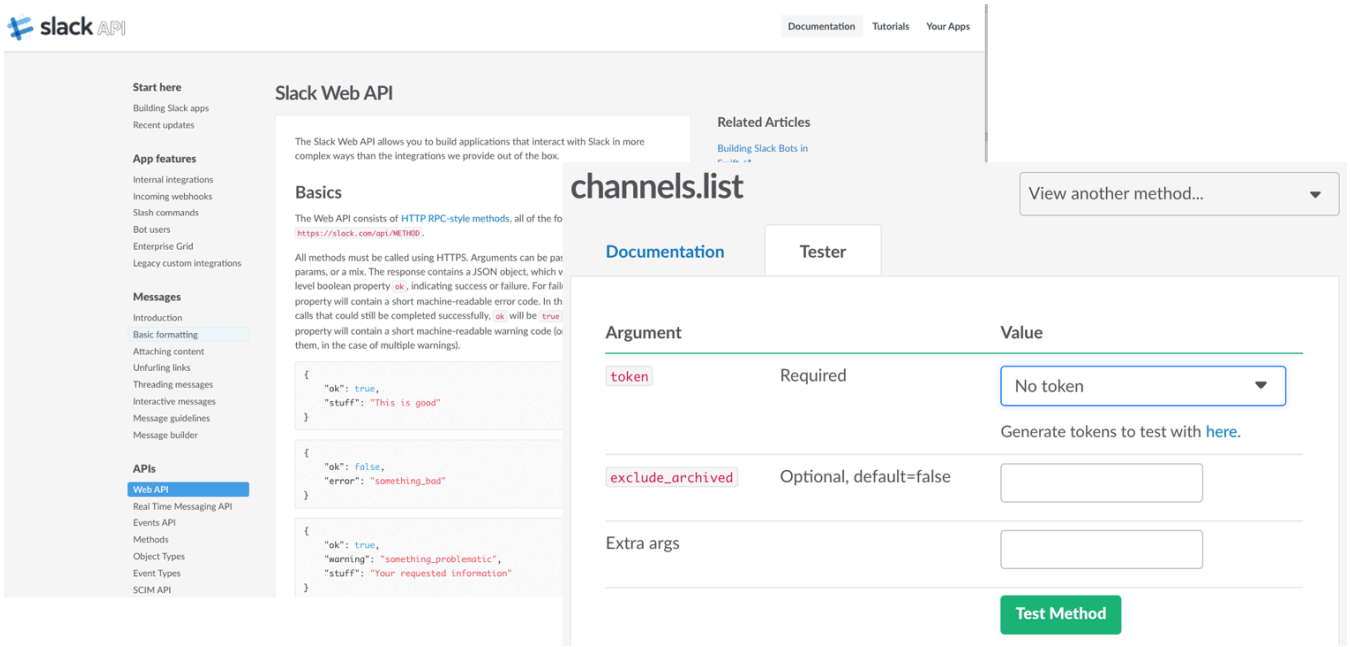
15. Return to the course snippets.txt file and copy the URL for the Web Toolkit Online http://www.webtoolkitonline.com/xml-formatter.html.
16. In a new tab in the web browser, navigate to that URL.
17. Paste the content in the XML Content area.
18. Click Format.



Walkthrough 1-2: Review and make a call to an RPC API

In this walkthrough, you explore an RPC API. You will:

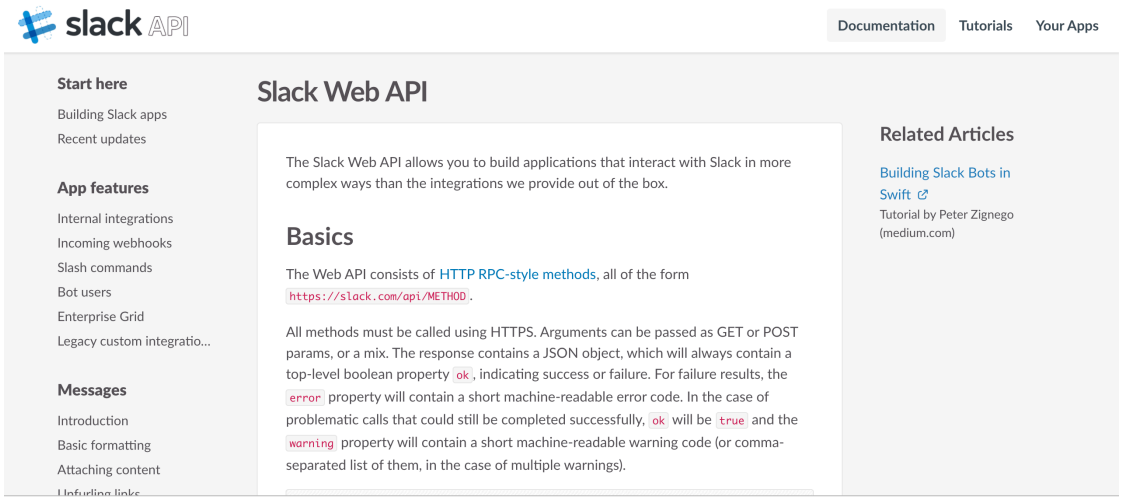
- Examine the Slack RPC API.
- Make a call to an RPC endpoint.



Examine the Slack Web API

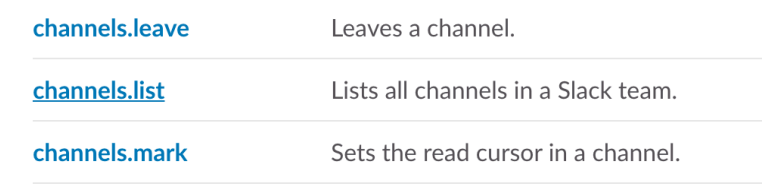
1. In a web browser, navigate to https://api.slack.com/web.

   Note: This URL is also located in the course snippets.txt file.

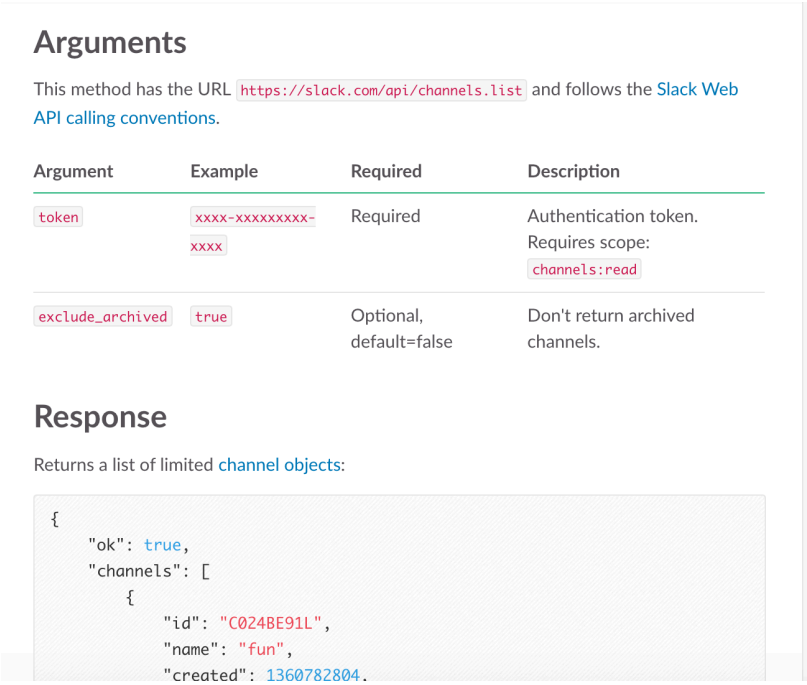


2. Click the HTTP RPC style methods link in the middle of the page.

3. Scroll down to the list of methods that describe the actions that can be performed with the channels.
4. Click the channels.list method.

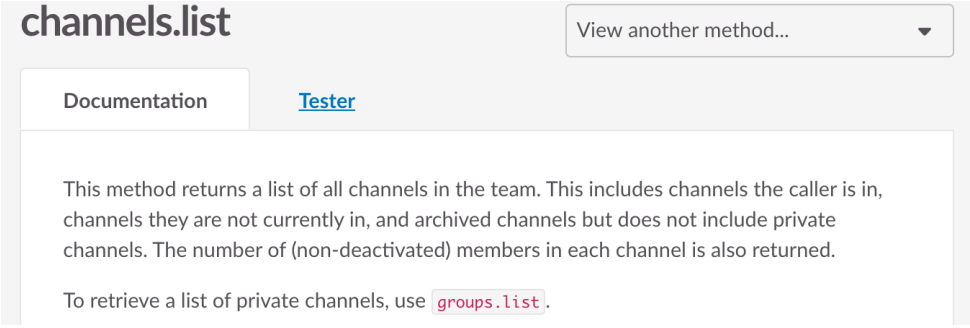


5. View the documentation for the channels.list method with the arguments and response information.

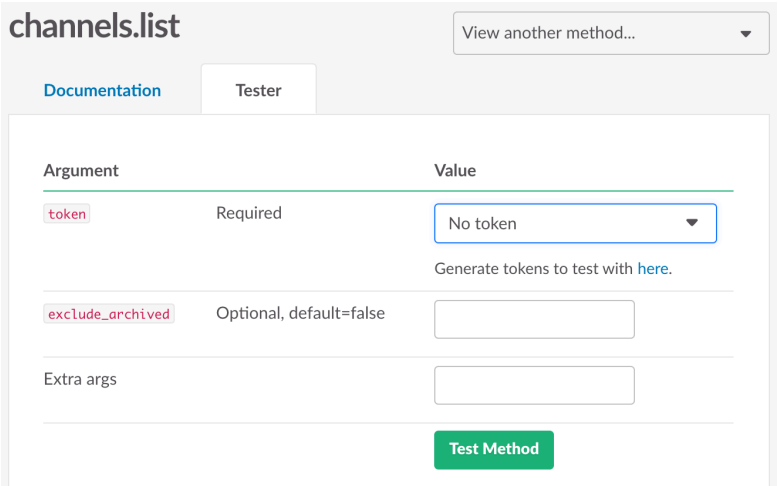


Make a call to an endpoint

6. Scroll back up and click the Tester tab.



7. In the Tester, select the No token or Invalid token option for the token attribute and click the Test Method button.



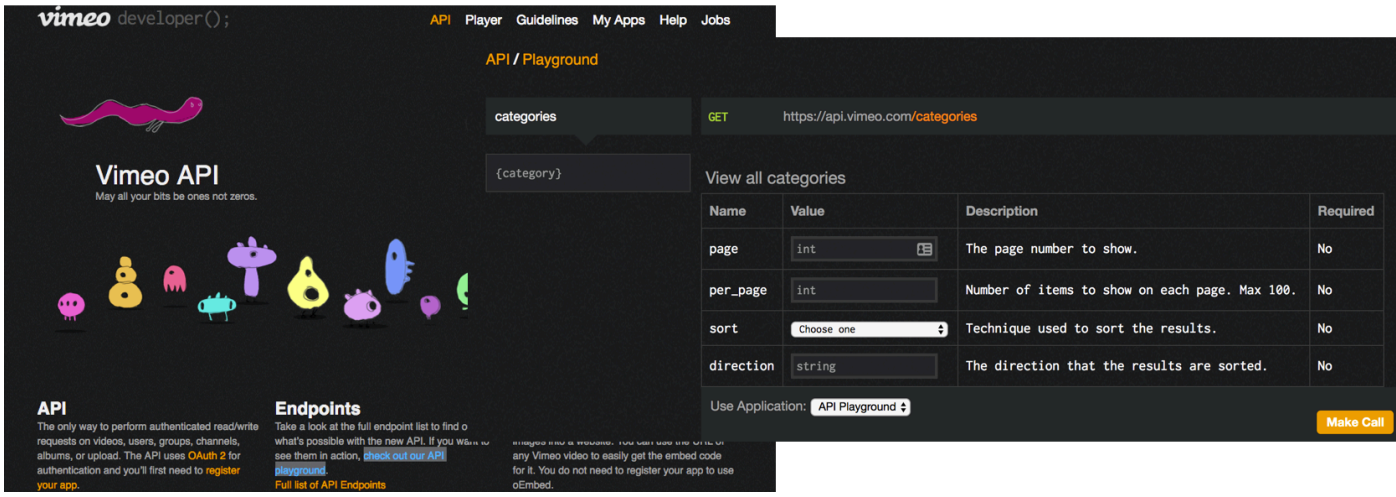
8. Verify that you see the response as not authorized or invalid authorization.

Note: To obtain a valid token, you need a Slack account and to belong to an organization with channels and members. If you do have an account, you can generate tokens by following the information in the link below the token attribute value in the tester.

Walkthrough 1-3: Review and make a call to a REST API

In this walkthrough, you explore a REST API. You will:

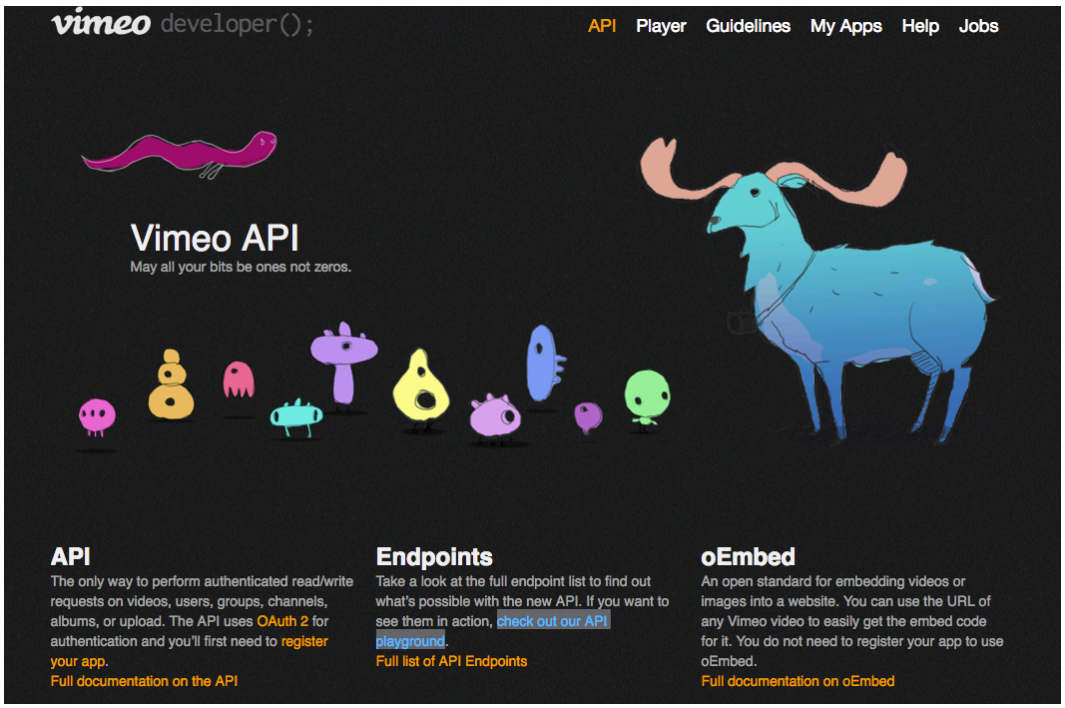
- Examine the Vimeo REST API.
- Make a call to a REST API endpoint to retrieve information.



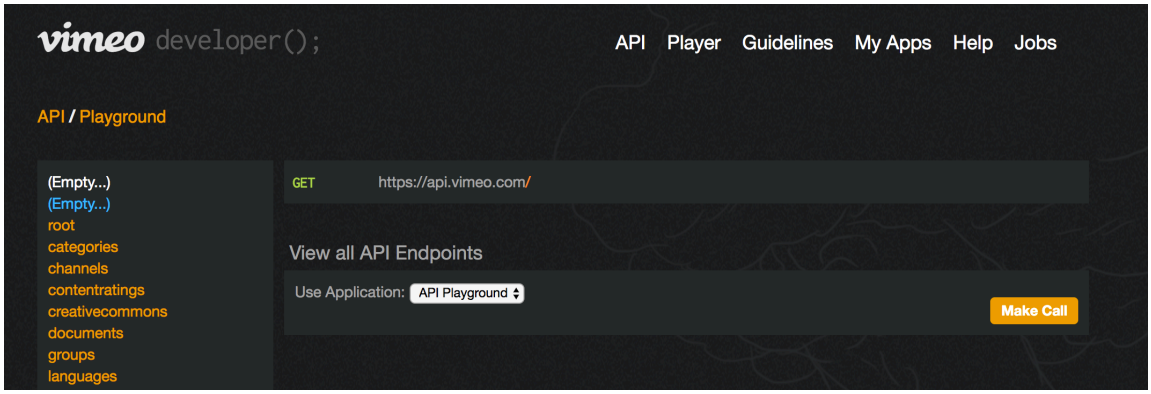
Examine the Vimeo REST API

1. In a web browser navigate to https://developer.vimeo.com/api.

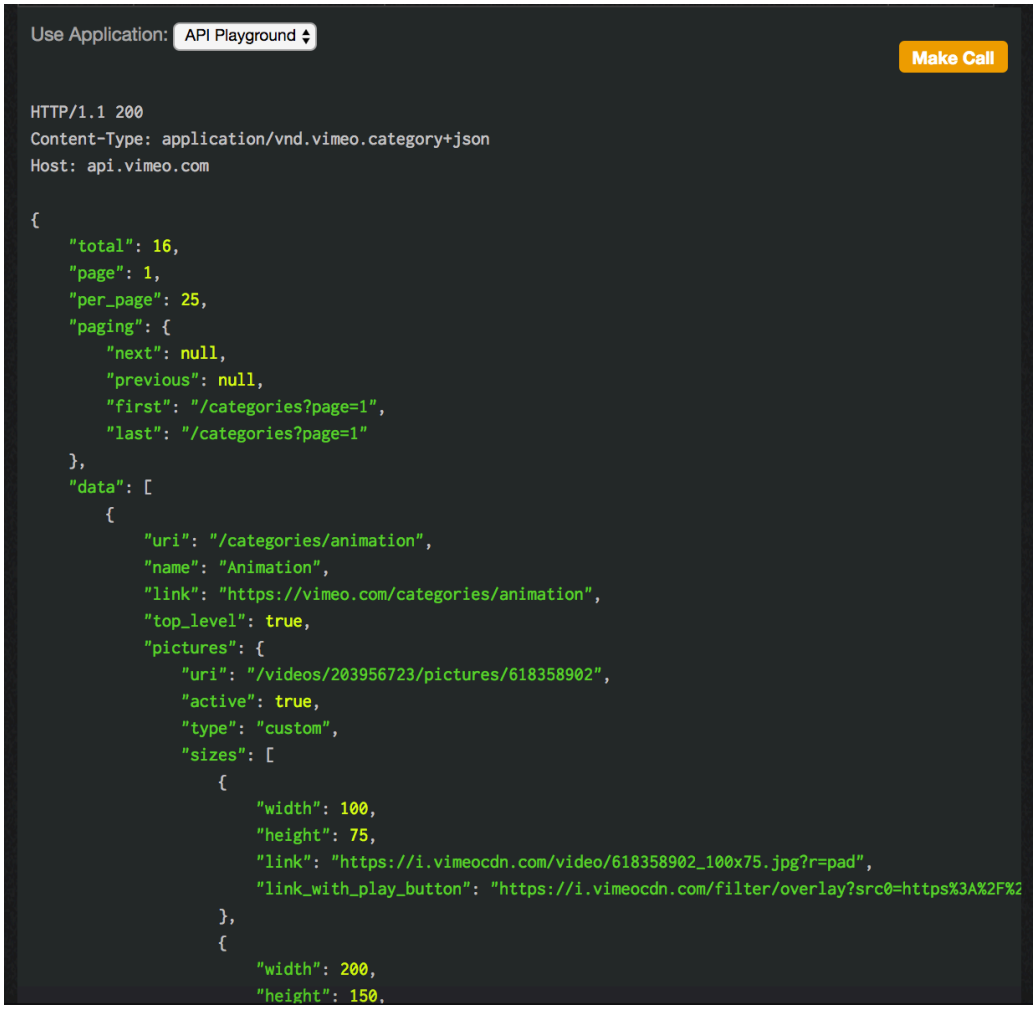
   Note: This URL is also located in the course snippets.txt file.



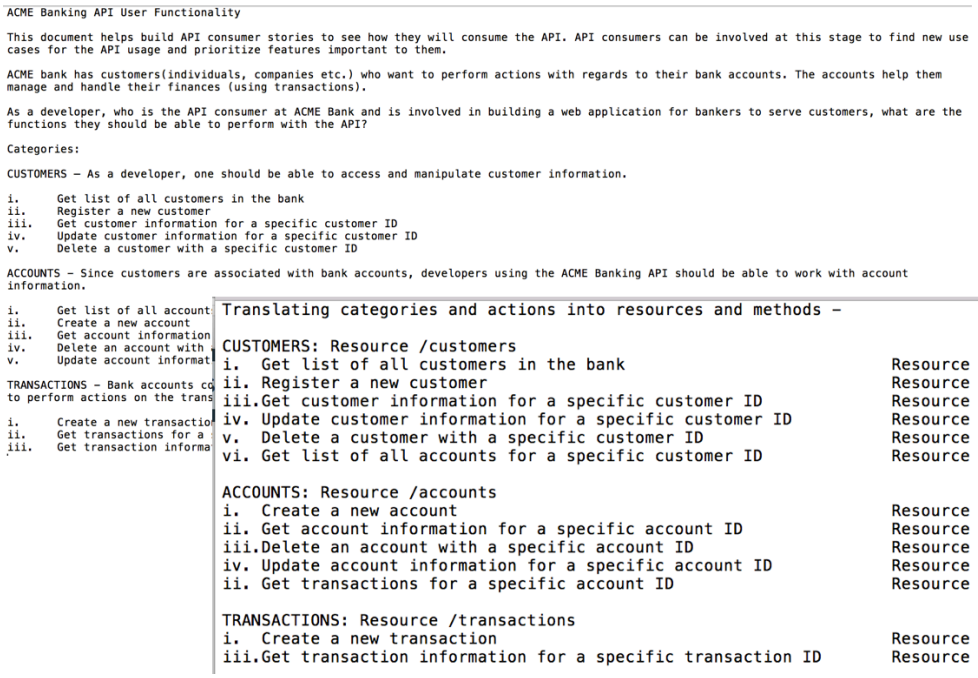
2. Under the Endpoints heading on the Vimeo developer page, click check out our API playground.
3. On the API/Playground page, click (Empty…) on the left-hand side.



4. In the expanded list of resources, click categories.
5. Click the Make Call button.
6. Verify the response has a status code 200.
7. Scroll the page and view the data for the Animation and Arts & Design categories.



Module 2: Translating Functional Requirements for APIs



ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at this stage to find new use cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies etc.) who want to perform actions with regards to their bank accounts. The accounts help them manage and handle their finances (using transactions).

As a developer, who is the API consumer at ACME Bank and is involved in building a web application for bankers to serve customers, what are the functions they should be able to perform with the API?

Categories:

CUSTOMERS – As a developer, one should be able to access and manipulate customer information.

i. Get list of all customers in the bank
ii. Register a new customer
iii. Get customer information for a specific customer ID
iv. Update customer information for a specific customer ID
v. Delete a customer with a specific customer ID

ACCOUNTS – Since customers are associated with bank accounts, developers using the ACME Banking API should be able to work with account information.

i. Get list of all account
ii. Create a new account
iii. Get account information
iv. Delete an account with
v. Update account informat

TRANSACTIONS – Bank accounts c
to perform actions on the trans

i. Create a new transactio
ii. Get transactions for a
iii. Get transaction informa

Translating categories and actions into resources and methods –

CUSTOMERS: Resource /customers

i. Get list of all customers in the bank	Resource /customers	Method GET
ii. Register a new customer	Resource /customers	Method POST
iii.Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method GET
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}	Method DELETE
vi. Get list of all accounts for a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET

ACCOUNTS: Resource /accounts

i. Create a new account	Resource /accounts	Method POST
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method GET
iii.Delete an account with a specific account ID	Resource /accounts/{account_id}	Method DELETE
iv. Update account information for a specific account ID	Resource /accounts/{account_id}	Method PUT
ii. Get transactions for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET

TRANSACTIONS: Resource /transactions

i. Create a new transaction	Resource /transactions	Method POST
iii.Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET

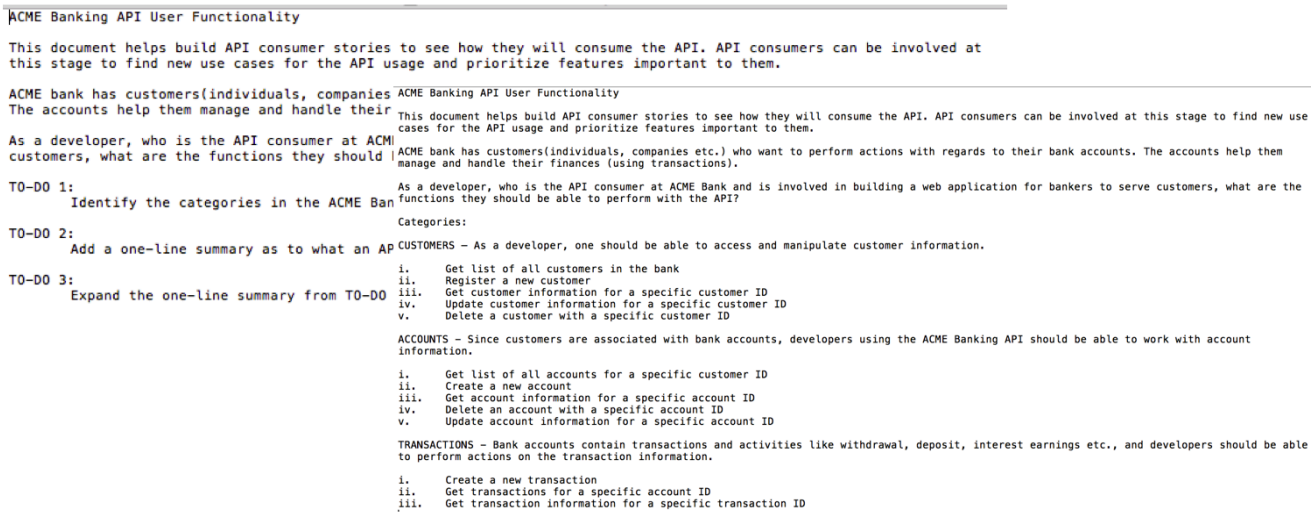
Objectives:

- Identify the different categories and actions for a REST API.
- Convert categories to resources.
- Select HTTP methods to support the actions on categories.

Walkthrough 2-1: List the categories and actions for an API

In this walkthrough, you list out the functional requirements for an API. You will:

- Identify the categories for a banking API.
- Define actions for the categories to decide how users will consume the API.



```
ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at
this stage to find new use cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies
The accounts help them manage and handle their

As a developer, who is the API consumer at ACM
customers, what are the functions they should

TO-DO 1:
       Identify the categories in the ACME Ban

TO-DO 2:
       Add a one-line summary as to what an AP

TO-DO 3:
       Expand the one-line summary from TO-DO
```

```
ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at this stage to find new use
cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies etc.) who want to perform actions with regards to their bank accounts. The accounts help them
manage and handle their finances (using transactions).

As a developer, who is the API consumer at ACME Bank and is involved in building a web application for bankers to serve customers, what are the
functions they should be able to perform with the API?

Categories:

CUSTOMERS - As a developer, one should be able to access and manipulate customer information.

i.    Get list of all customers in the bank
ii.   Register a new customer
iii.  Get customer information for a specific customer ID
iv.   Update customer information for a specific customer ID
v.    Delete a customer with a specific customer ID

ACCOUNTS - Since customers are associated with bank accounts, developers using the ACME Banking API should be able to work with account
information.

i.    Get list of all accounts for a specific customer ID
ii.   Create a new account
iii.  Get account information for a specific account ID
iv.   Delete an account with a specific account ID
v.    Update account information for a specific account ID

TRANSACTIONS - Bank accounts contain transactions and activities like withdrawal, deposit, interest earnings etc., and developers should be able
to perform actions on the transaction information.

i.    Create a new transaction
ii.   Get transactions for a specific account ID
iii.  Get transaction information for a specific transaction ID
```

Create user stories to define functional requirements for a banking API

1. Navigate to the studentFiles folder on your computer.
2. Navigate to the module02 directory and open the WT2-1 User functionality text file in a text editor.

```
ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at
this stage to find new use cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies etc.) who want to perform actions with regards to their bank accounts.
The accounts help them manage and handle their finances (using transactions).

As a developer, who is the API consumer at ACME Bank and is involved in building a web application for bankers to serve
customers, what are the functions they should be able to perform with the API?

TO-DO 1:
       Identify the categories in the ACME Banking API

TO-DO 2:
       Add a one-line summary as to what an API consumer be able to do with these categories

TO-DO 3:
       Expand the one-line summary from TO-DO 2 into a detailed list of actions for each of the categories
```

3. Read the file.

Brainstorm for categories and actions

4. Brainstorm with the class about possible categories and actions.

   Note: The instructor may break you out into smaller groups to first brainstorm with some of your peers.

List categories

5. In the WT 2-1 User functionality text file, add the following categories.

    - CUSTOMERS
    - ACCOUNTS
    - TRANSACTIONS

List detailed actions

6. In the WT 2-1 User functionality text file, add a detailed list of actions that developers should be able to perform with the API for each of the categories.

    CUSTOMERS –

    i. Get list of all customers in the bank
    ii. Get customer information for a specific customer ID
    iii. Register a new customer
    iv. Update customer information for a specific customer ID
    v. Delete a customer with a specific customer ID

    ACCOUNTS –

    vi. Get list of all accounts for a specific customer ID
    vii. Get account information for a specific account ID
    viii. Create a new account
    ix. Update account information for a specific account ID
    x. Delete account with a specific account ID

    TRANSACTIONS –

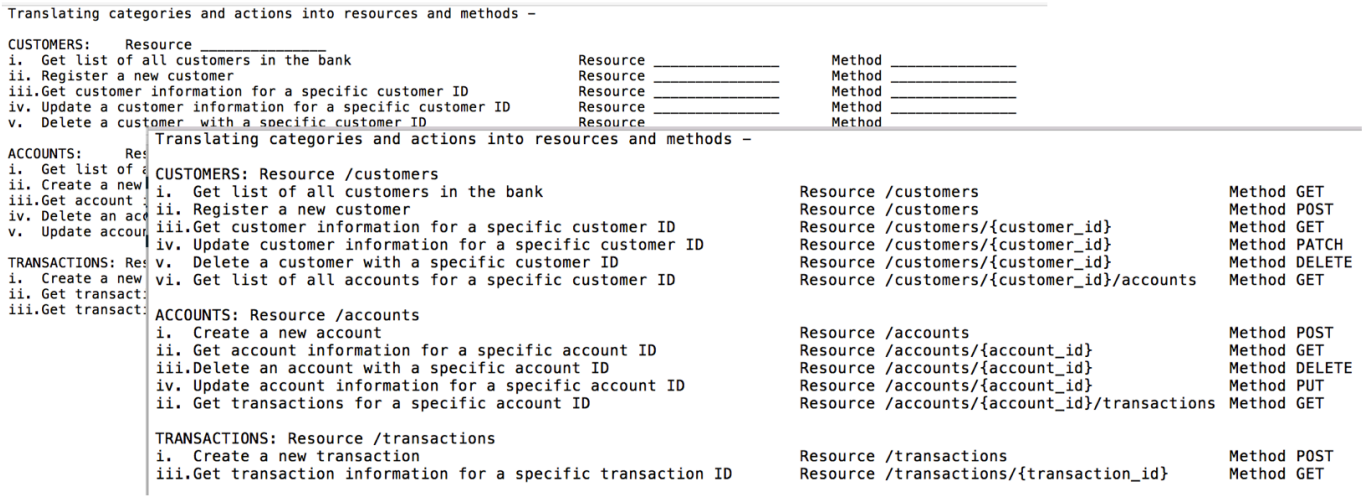
    xi. Get transactions for a specific account ID
    xii. Get transaction information for a specific transaction ID
    xiii. Create a new transaction

7. Save the file.

Walkthrough 2-2: Translate categories and actions into resources and methods

In this walkthrough, you translate the categories and actions identified in the last walkthrough into resources and methods. You will:

- Translate categories into resources.
- Identify the HTTP methods for actions.



```
Translating categories and actions into resources and methods –

CUSTOMERS:    Resource _____________
i.  Get list of all customers in the bank                    Resource _____________   Method _____________
ii. Register a new customer                                  Resource _____________   Method _____________
iii.Get customer information for a specific customer ID      Resource _____________   Method _____________
iv. Update a customer information for a specific customer ID Resource _____________   Method _____________
v.  Delete a customer  with a specific customer ID           Resource _____________   Method _____________

ACCOUNTS:     Res
i.  Get list of a
ii. Create a new
iii.Get account
iv. Delete an ac
v.  Update accou

TRANSACTIONS: Res
i.  Create a new
ii. Get transact
iii.Get transact
```

```
Translating categories and actions into resources and methods –

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                    Resource /customers                          Method GET
ii. Register a new customer                                  Resource /customers                          Method POST
iii.Get customer information for a specific customer ID      Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID   Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID            Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID      Resource /customers/{customer_id}/accounts   Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                     Resource /accounts                           Method POST
ii. Get account information for a specific account ID        Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID             Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID     Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID               Resource /accounts/{account_id}/transactions Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                 Resource /transactions                       Method POST
iii.Get transaction information for a specific transaction ID Resource /transactions/{transaction_id}    Method GET
```

Translate categories into resources

1. Return to the module02 folder in the studentFiles folder on your computer.
2. Open the WT2-2 Resources and methods file in a text editor.

```
CUSTOMERS:    Resource _____________
i.  Get list of all customers in the bank                    Resource _____________   Method _____________
ii. Get customer information for a specific customer ID      Resource _____________   Method _____________
iii.Register a new customer                                  Resource _____________   Method _____________
iv. Update a customer information for a specific customer ID Resource _____________   Method _____________
v.  Delete a customer  with a specific customer ID           Resource _____________   Method _____________

ACCOUNTS:     Resource _____________
i.  Get list of all accounts for a specific customer ID      Resource _____________   Method _____________
ii. Get account information for a specific account ID        Resource _____________   Method _____________
iii.Create a new account                                     Resource _____________   Method _____________
iv. Delete an account with a specific account ID             Resource _____________   Method _____________
v.  Update account information for a specific account ID     Resource _____________   Method _____________

TRANSACTIONS: Resource _____________
i.  Get transactions for a specific account ID               Resource _____________   Method _____________
ii. Get transaction information for a specific transaction ID Resource _____________   Method _____________
iii.Create a new transaction                                 Resource _____________   Method _____________
```

3. Review the file.

Brainstorm about possible resources and methods

4. Brainstorm with the class about possible resources and methods.

   Note: The instructor may break you out into smaller groups to first brainstorm with some of your peers.

Specify resources

5. In the WT2-2 Resources and methods file, specify resources for the Customers entity.
   - CUSTOMERS: Resource /customers
     i. Get list of all customers - Resource /customers
     ii. Register a new customer - Resource /customers
     iii. Get customer information for a specific customer ID - Resource /customers/{customer_id}
     iv. Update customer information for a customer ID - Resource /customers/{customer_id}
   - Delete a customer with a specific customer ID - Resource /customers/{customer_id}
6. After a discussion, fill out the resources for Accounts entity.
   - ACCOUNTS: Resource /accounts
     i. Get list of all accounts for a specific customer ID – Resource /customers/{customer_id}/accounts
     ii. Create a new account – Resource /accounts
     iii. Get account information for a specific account ID – Resource /accounts/{account_id}
     iv. Delete an account with a specific account ID – Resource /accounts/{account_id}
     v. Update account information for a specific account ID – Resource /accounts/{account_id}
7. Specify resources for the Transactions entity.
   - TRANSACTIONS: Resource /transactions
     i. Create a new transaction – Resource /transactions
     ii. Get transactions for a specific account ID – Resource /accounts/{account_id}/transactions
     iii. Get transaction information for a specific transaction ID – Resource /transactions/{transaction_id}

Specify HTTP methods for the actions

8. Specify methods for the identified resources.

    - Get list of all customers – Method GET
    - Register a new customer – Method POST
    - Get customer information for a specific customer ID – Method GET
    - Update customer information for a customer ID – Method PATCH
    - Delete a customer with a specific customer ID – Method DELETE

    - Get list of all accounts for a specific customer ID – Method GET
    - Create a new account – Method POST
    - Get account information for a specific account ID – Method GET
    - Delete an account with a specific account ID – Method DELETE
    - Update account information for a specific account ID – Method PUT

    - Create a new transaction – Method POST
    - Get transactions for a specific account ID – Method GET
    - Get transaction information for a specific transaction ID – Method GET

Rearrange resources in the order of invocation

9. In the ACCOUNTS category, move the first line that contains the resource and method to get list of accounts for a specific customer inside the CUSTOMERS category.

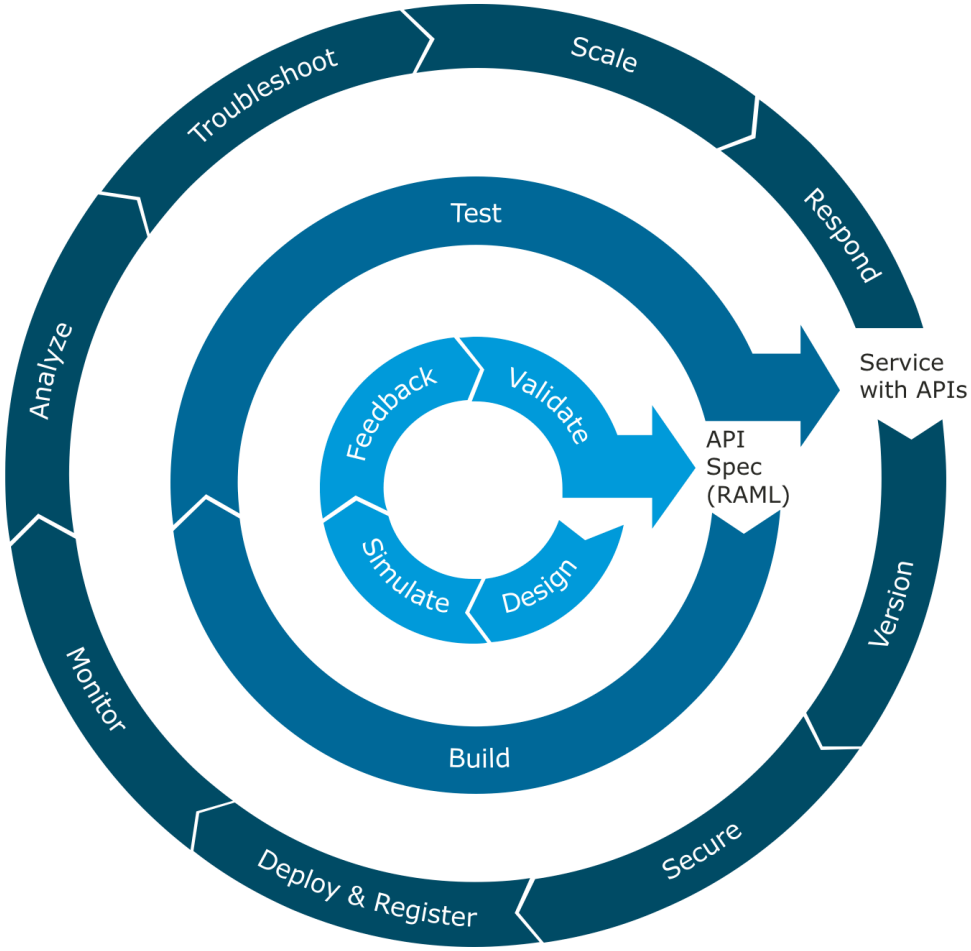
```
CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                    Resource /customers                          Method GET
ii. Register a new customer                                  Resource /customers                          Method POST
iii.Get customer information for a specific customer ID      Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID   Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID            Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID      Resource /customers/{customer_id}/accounts   Method GET
```

10. In the TRANSACTIONS category, move the second line that contains the resource and method to get transactions for a specific account inside the ACCOUNTS category.

```
ACCOUNTS: Resource /accounts
i.  Create a new account                                     Resource /accounts                           Method POST
ii. Get account information for a specific account ID        Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID             Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID     Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID               Resource /accounts/{account_id}/transactions Method GET
```

11. Save the text file.

Module 3: Introducing API-Led Connectivity and the API Lifecycle



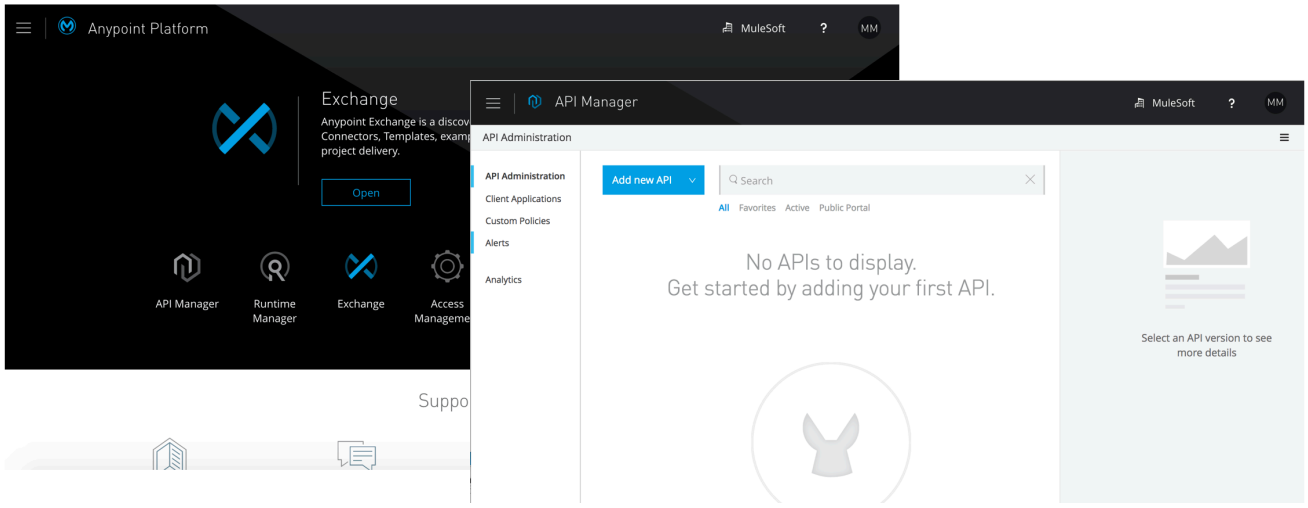
Objectives:

- Describe the API development lifecycle.
- Explain MuleSoft's API-led connectivity approach.
- Navigate Anypoint Platform.
- Describe the API development lifecycle with Anypoint Platform.

Walkthrough 3-1: Explore API Manager in Anypoint Platform

In this walkthrough, you get familiar with Anypoint Platform and its features. You will:

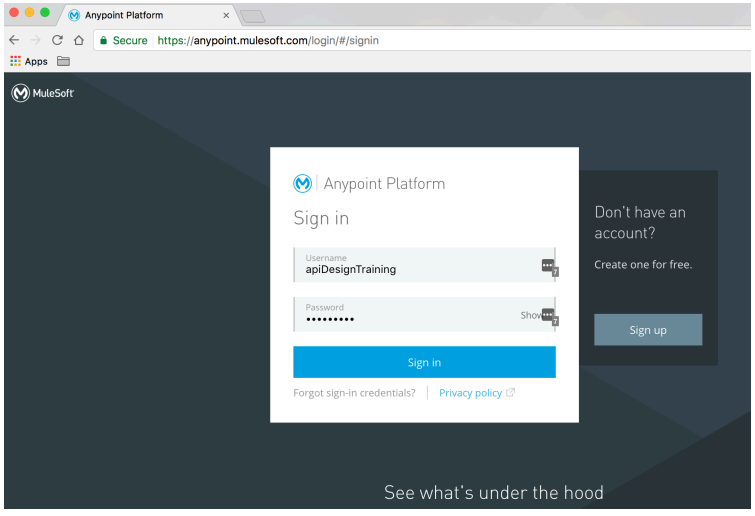
- Log in to Anypoint Platform.
- Explore API Manager in Anypoint Platform.



Log in to Anypoint Platform

1. In a web browser, navigate to https://anypoint.mulesoft.com.
2. Log in to Anypoint Platform.

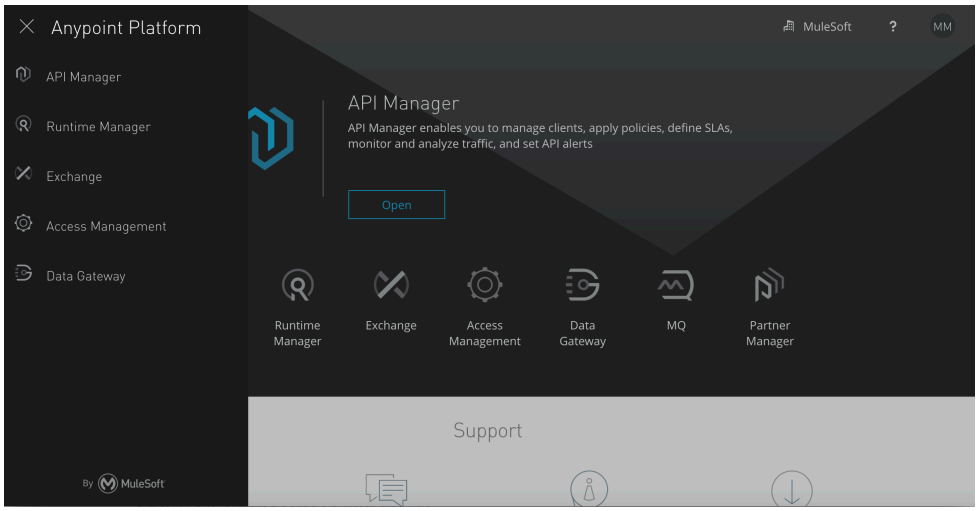
   Note: You can use a trial account or your company account (if you already have one). If you do not have an account, sign up for a free, 30-day trial account now.



Explore API Manager in Anypoint Platform

3. Click the menu button located in the upper-left in the main menu bar.
4. Review the menu that appears that includes links to your Anypoint Platform entitlements.

   Note: You can also open this menu by pressing your keyboard's Escape key.

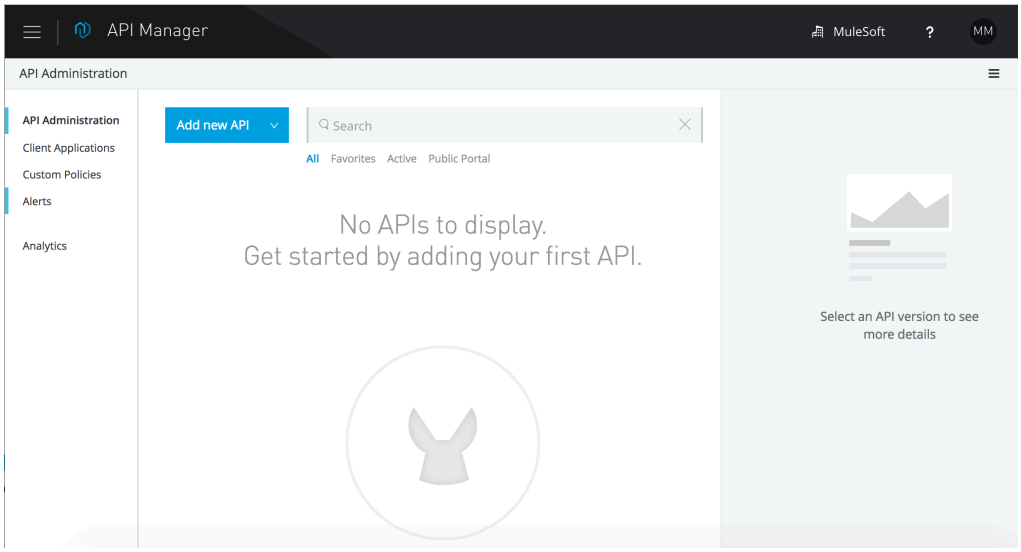


5. From the menu, select API Manager.

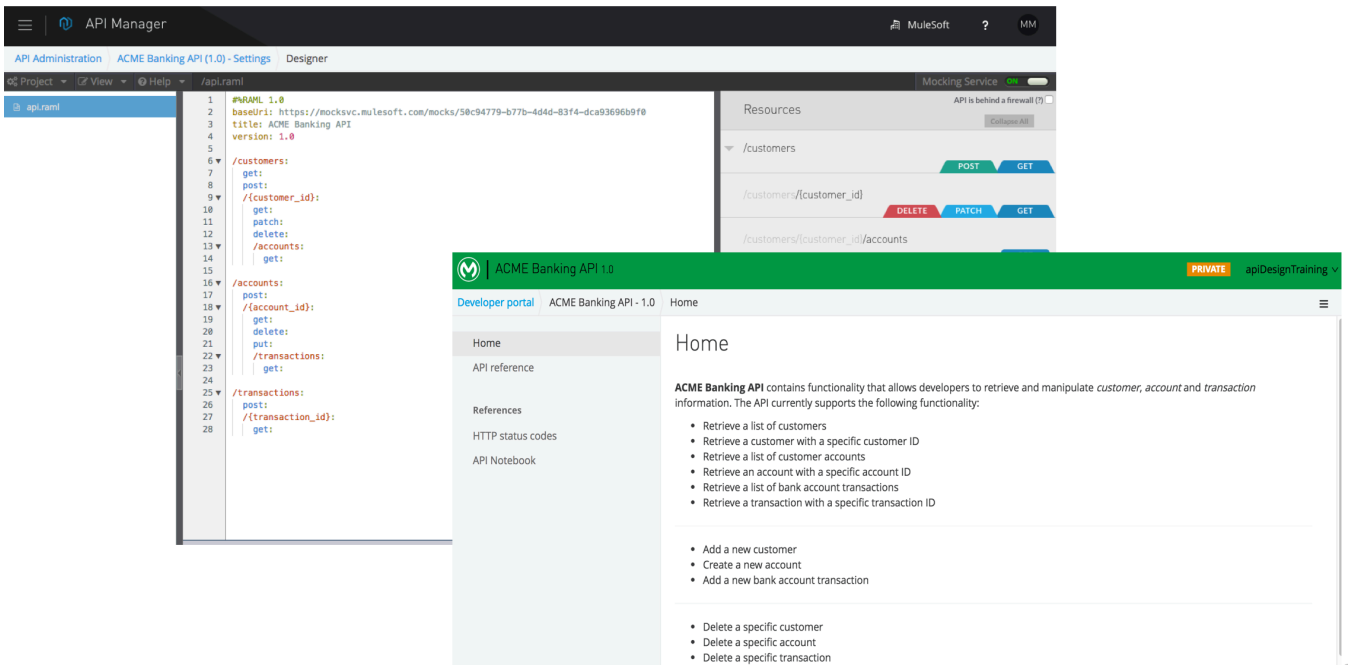
   Note: This will be called the main menu from now on.

6. In the web page for API Manager, verify that you can see the following functionality:

   - A Button to add A new API (with an option to import an API using the downward arrow next to it).
   - A menu in the upper-right corner below the username initials to help work with developer portals for APIs, API Analytics, uploading custom policies, and more.



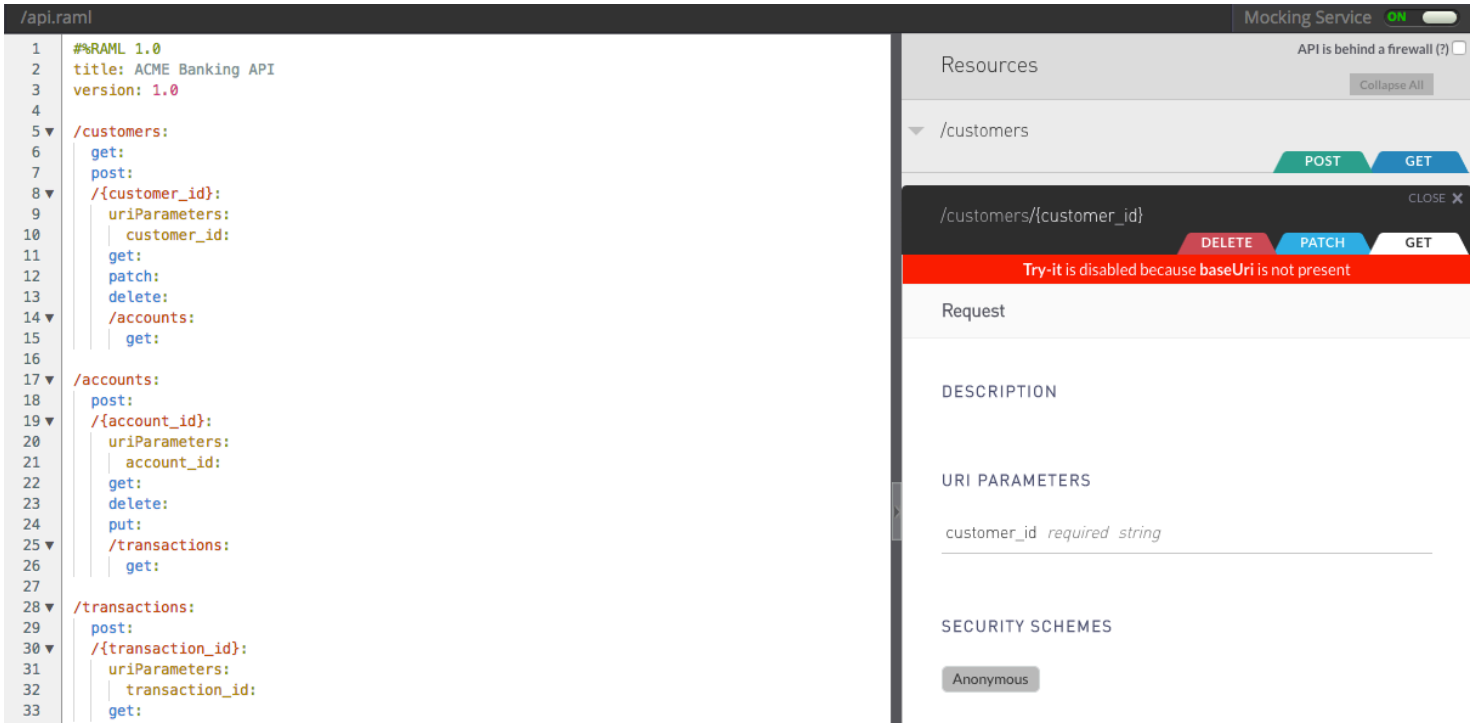
PART 2: Defining APIs with the RESTful API Modeling Language (RAML)



Objectives:

- Create API definitions with RAML 1.0.
- Use patterns to refactor and modularize API definitions.
- Specify security schemes to secure resources in APIs.
- Add documentation to RAML API definitions.
- Make APIs discoverable through API Portals and Anypoint Exchange.
- Test APIs through the API Console.
- Add state specific responses to promote hypermedia.
- Learn when and how to version APIs.

Module 4: Defining API Resources and Methods



Objectives:

- Use API Designer to create API definitions with RAML 1.0.
- Define resources and methods in RAML API definitions.
- Specify URI parameters for resource methods.

Walkthrough 4-1: Create an API and define resources in RAML 1.0

In this walkthrough, you create an API definition for the ACME Banking use case. You will:

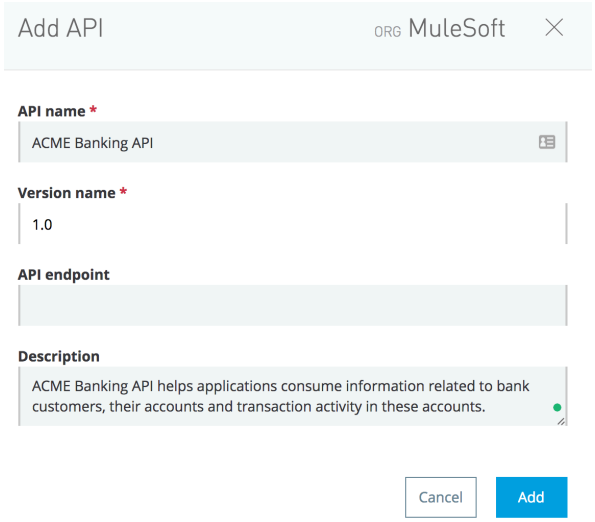
- Create the ACME Banking API in API Designer.
- Define the resources and nested resources identified for the API.

```
Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                    Resource /customers
ii. Register a new customer                                  Resource /customers
iii.Get customer information for a specific customer ID      Resource /customers/{customer_id
iv. Update customer information for a specific customer ID   Resource /customers/{customer_id
v.  Delete a customer with a specific customer ID            Resource /customers/{customer_id
vi. Get list of all accounts for a specific customer ID      Resource /customers/{customer_id

ACCOUNTS: Resource /accounts
i.  Create a new account                                     Resource /accounts
ii. Get account information for a specific account ID        Resource /accounts/{account_id}
iii.Delete an account with a specific account ID             Resource /accounts/{account_id}
iv. Update account information for a specific account ID     Resource /accounts/{account_id}
ii. Get transactions for a specific account ID               Resource /accounts/{account_id}/

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                 Resource /transactions
iii.Get transaction information for a specific transaction ID Resource /transactions/{transact
```

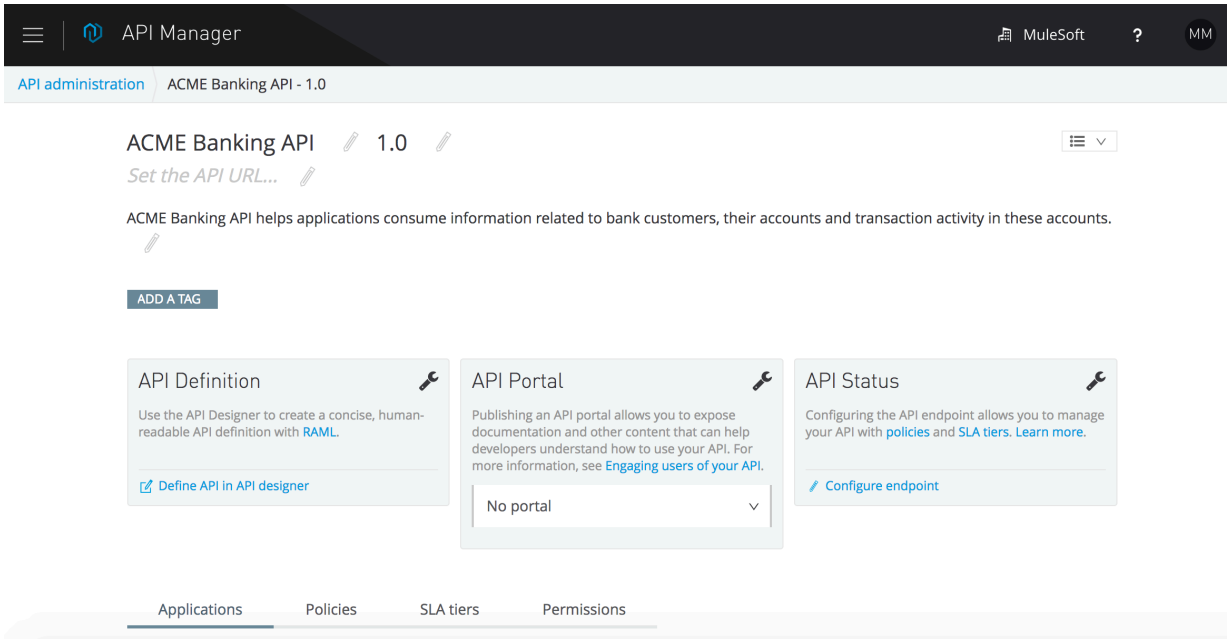
```
1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  /customers:
6    /{customer_id}:
7      /accounts:
8
9  /accounts:
10   /{account_id}:
11     /transactions:
12
13 /transactions:
14   /{transaction_id}:
```

Create a new API

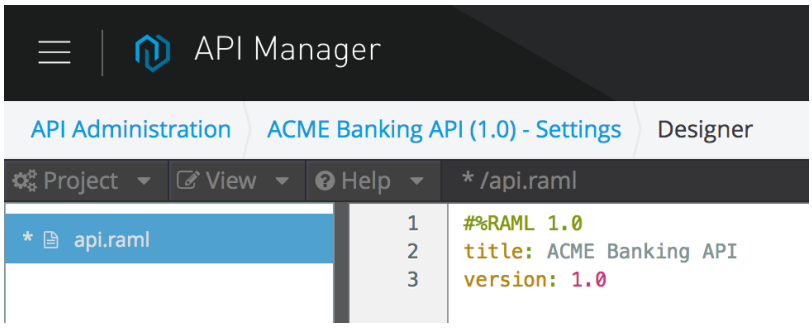
1. Return to API Manager in Anypoint Platform.
2. Click the Add new API button.
3. In the Add API dialog box, enter the following information:
   - API name: ACME Banking API
   - Version name: 1.0
   - Description: This API provides endpoints to view and manipulate information related to customers, their accounts, and transaction activity.



4. Click Add.
5. Look at the different sections and links for the API on the API version details page for ACME Banking API.



6. In the API Definition section, click the Define API in API designer link; the API Designer should open.
7. Examine the different parts of API Designer:
   - The file browser: The section on the left
   - The editor: The section in the middle with the RAML
   - The shelf: The gray section below the editor with tools to help write the RAML
   - The API Console: The section on the right for getting information about and testing APIs
8. In the editor, change the RAML version from 0.8 to 1.0.



Note: To change the background color from black to white, select View > Toggle Background Color or press Ctrl+Shift+T.

Review the resources and nested resources identified for the API

9. Return to the WT 2-2 Resources and methods file in a text editor.

```
Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                        Resource /customers                            Method GET
ii. Register a new customer                                      Resource /customers                            Method POST
iii.Get customer information for a specific customer ID          Resource /customers/{customer_id}              Method GET
iv. Update customer information for a specific customer ID       Resource /customers/{customer_id}              Method PATCH
v.  Delete a customer with a specific customer ID                Resource /customers/{customer_id}              Method DELETE
vi. Get list of all accounts for a specific customer ID          Resource /customers/{customer_id}/accounts     Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                         Resource /accounts                             Method POST
ii. Get account information for a specific account ID            Resource /accounts/{account_id}                Method GET
iii.Delete an account with a specific account ID                 Resource /accounts/{account_id}                Method DELETE
iv. Update account information for a specific account ID         Resource /accounts/{account_id}                Method PUT
ii. Get transactions for a specific account ID                   Resource /accounts/{account_id}/transactions   Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                     Resource /transactions                         Method POST
iii.Get transaction information for a specific transaction ID    Resource /transactions/{transaction_id}        Method GET
```

10. Review the resources.

Define the resources

11. Return to API Designer.
12. Place your cursor after the version number and press the Enter key two times to add two new lines.
13. Add the /customers resource.

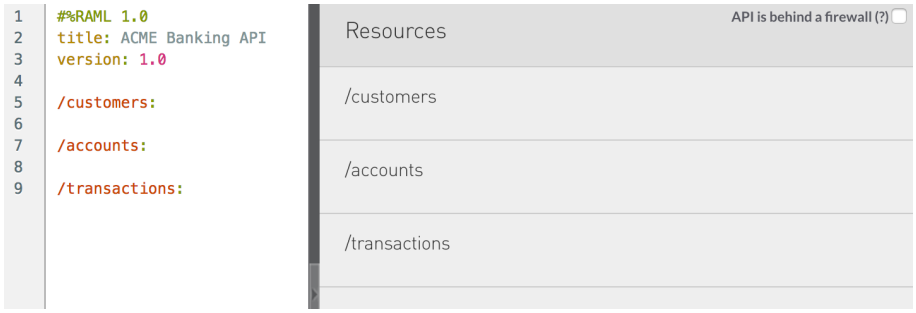
    ```
    /customers:
    ```

14. Press enter twice and add the /accounts resource.
15. Add the /transactions resource.

```
1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  /customers:
6
7  /accounts:
8
9  /transactions:
```

16. Look at the API Console; you should see the three resources.

    Note: If you do not see the API Console, click the arrow located in the middle of the right edge of the web browser window.



Define the nested resources

17. Go to the end of the /customers resource line, press enter to add a new line.

    Note: To go to the end of a line, click anywhere in the line and press Ctrl+E.

18. Indent by pressing the Tab key.
19. Add a nested resource called {customer_id} that will require a URL parameter called customer_id to be pass do it.

    Note: You will specify URI parameters in the next walkthrough.

```
/customers:
  /{customer_id}:
```

20. On a new line, indent and add a nested resource called accounts.

```
/customers:
  /{customer_id}:
    /accounts:
```

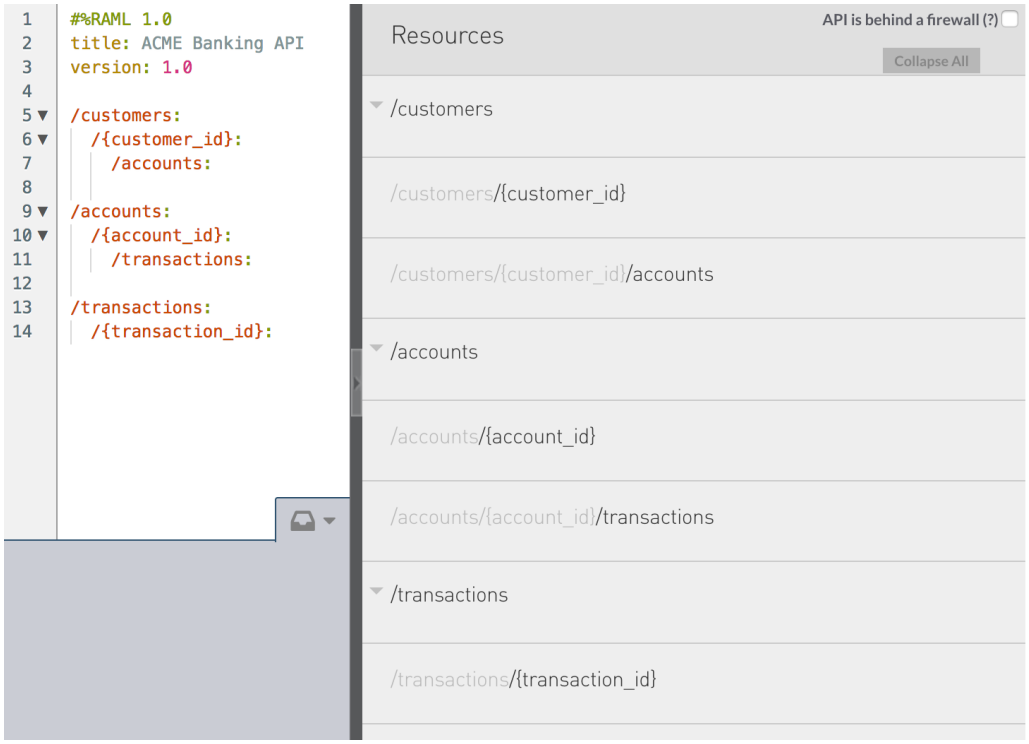
21. Go to a new line after the main /accounts resource.
22. Indent and add a nested resource called {account_id}.
23. Add a nested resource called transactions.

```
/customers:
  /{customer_id}:
    /accounts:

/accounts:
  /{account_id}:
    /transactions:
```

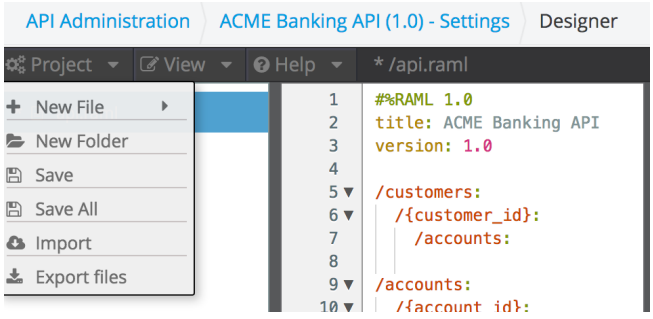
24. Go to a new line after the main /transactions resource.
25. Indent and add a nested resource called {transaction_id}.

```
/customers:
  /{customer_id}:
    /accounts:

/accounts:
  /{account_id}:
    /transactions:

/transactions:
  /{transaction_id}:
```

26. Look at the API Console; you should see the resources and nested resources displayed as noted in the WT 2-2 Resources and methods file.



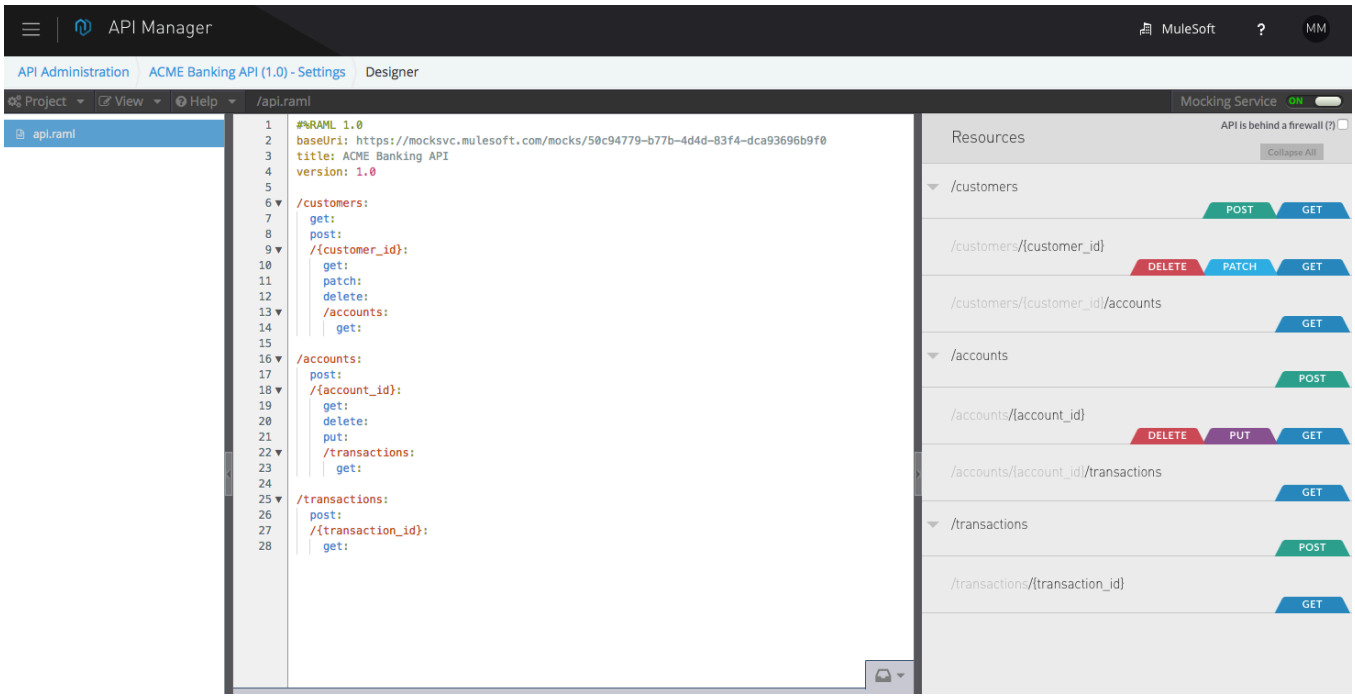
27. From the Project menu in the upper-left, click Save.



Walkthrough 4-2: Define methods for the resources

In this walkthrough, you add methods to the defined resources to retrieve, modify, or delete data. You will:

- Define HTTP methods for resources to retrieve collection and item data.
- Define HTTP methods for resources to modify and delete data.



Review the get methods identified for the API

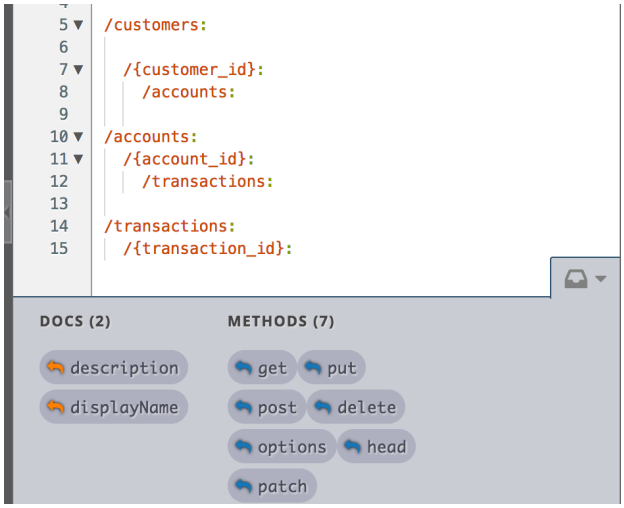
1. Return to the WT 2-2 Resources and methods file in a text editor.

```
Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                      Resource /customers                          Method GET
ii. Register a new customer                                    Resource /customers                          Method POST
iii.Get customer information for a specific customer ID        Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID     Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID              Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID        Resource /customers/{customer_id}/accounts   Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                       Resource /accounts                           Method POST
ii. Get account information for a specific account ID          Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID               Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID       Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID                 Resource /accounts/{account_id}/transactions Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                   Resource /transactions                       Method POST
iii.Get transaction information for a specific transaction ID  Resource /transactions/{transaction_id}      Method GET
```

2. Review the resources that have get methods.

Define HTTP methods for resources to retrieve collection and item data

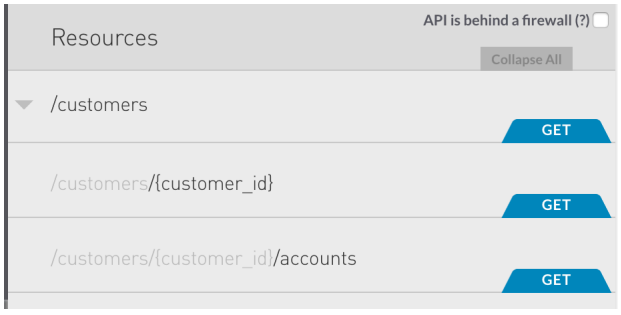
3. Return to API Designer.
4. Add a new line between the /customers and /{customer_id} resources.
5. In the shelf below the RAML editor, click the get method.



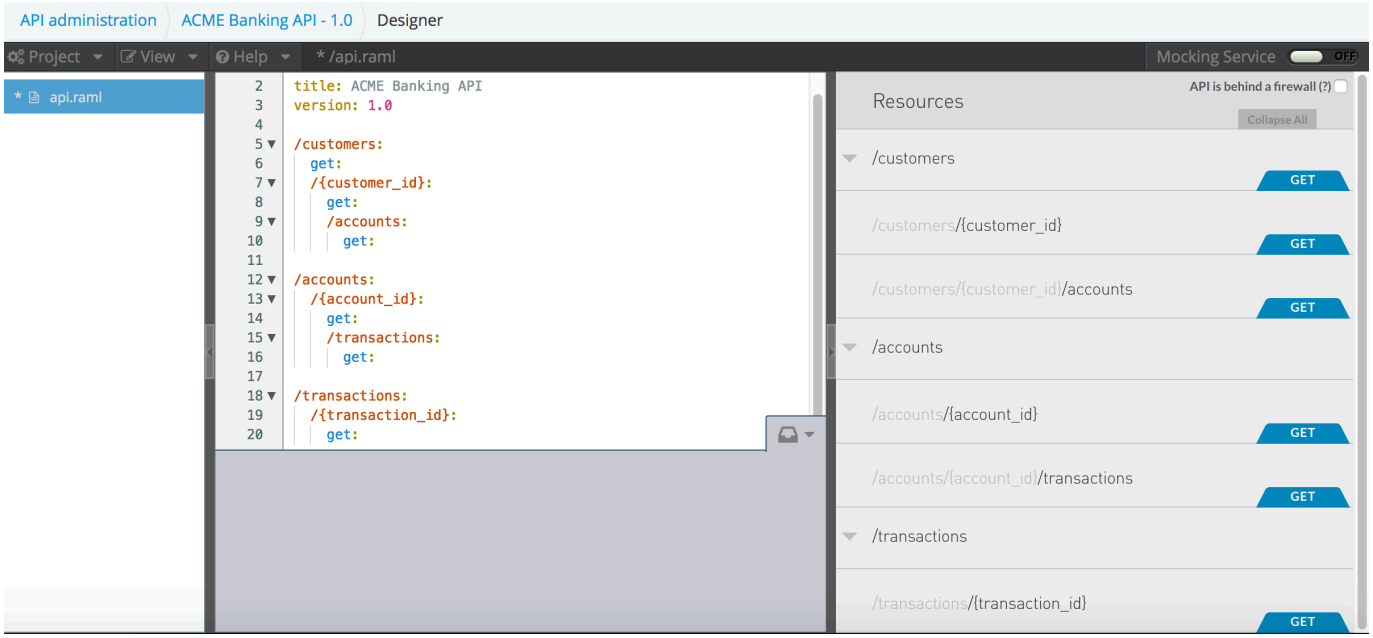
6. Delete the new line that is automatically added below the get method.
7. Similarly, add get methods to the nested /{customer_id} and /accounts resources.
8. Delete any extra lines of code.

```
/customers:
  get:
  /{customer_id}:
    get:
    /accounts:
      get:

/accounts:
  /{account_id}:
    /transactions:

/transactions:
  /{transaction_id}:
```

9. Look at the API Console; you should see tabs for the get methods.



10. For the /accounts resource in the editor, add get methods to the nested /{account_id} and /transactions resources.
11. For the /transactions resource, add a get method to the nested /{transaction_id} resource.



Review the methods identified for the /customers resource and nested resources

12. Return to the WT 2-2 Resources and methods file in a text editor.

```
Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                        Resource /customers                          Method GET
ii. Register a new customer                                      Resource /customers                          Method POST
iii.Get customer information for a specific customer ID          Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID       Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID                Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID          Resource /customers/{customer_id}/accounts   Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                         Resource /accounts                           Method POST
ii. Get account information for a specific account ID            Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID                 Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID         Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID                   Resource /accounts/{account_id}/transactions Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                     Resource /transactions                       Method POST
iii.Get transaction information for a specific transaction ID    Resource /transactions/{transaction_id}      Method GET
```

13. Review the methods for the /customers resource and nested resources.

Define methods for the /customers resource to add, modify, and delete data

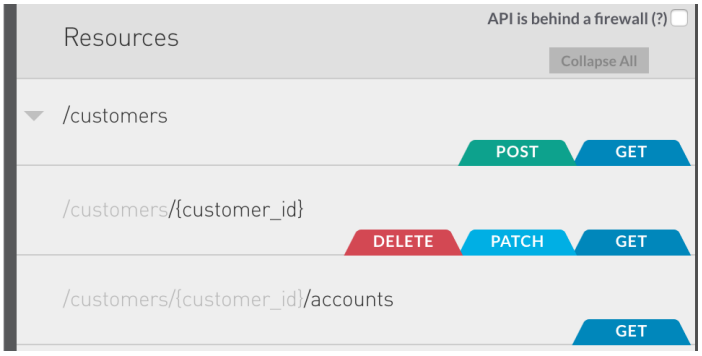
14. Return to API Designer.
15. Type or use the shelf to add a post method to the /customers resource.

16. Add patch and delete methods to the /{customer_id} nested resource.

```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
  post:
  /{customer_id}:
    get:
    patch:
    delete:
    /accounts:
      get:

/accounts:
  post:
    |
  /{account_id}:
    get:
```

17. Look at the API Console; you should see tabs for the new methods.

```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
  post:
  /{customer_id}:
    get:
    patch:
    delete:
    /accounts:
      get:
```



Review the methods identified for the /accounts resource and nested resources

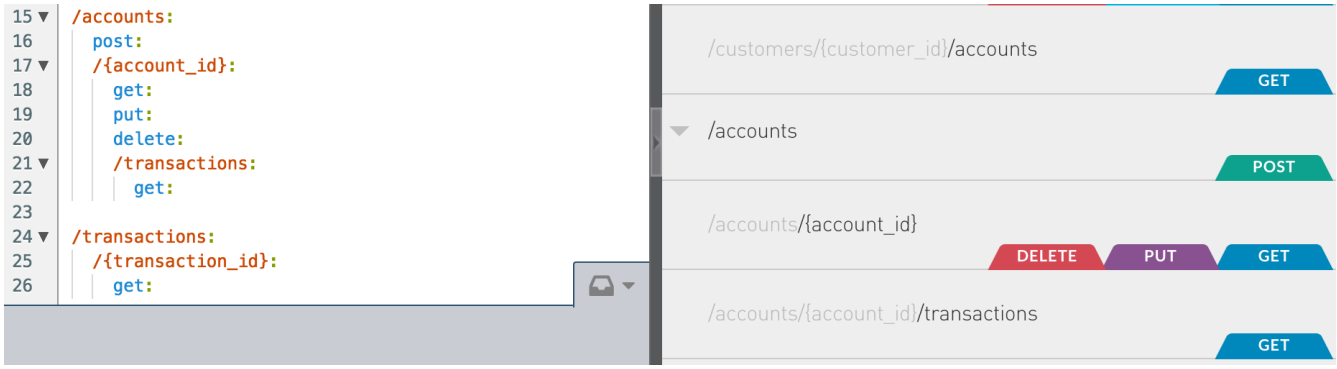
18. Return to the WT 2-2 Resources and methods file in a text editor.

```
Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                    Resource /customers                          Method GET
ii. Register a new customer                                  Resource /customers                          Method POST
iii.Get customer information for a specific customer ID      Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID   Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID            Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID      Resource /customers/{customer_id}/accounts   Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                     Resource /accounts                           Method POST
ii. Get account information for a specific account ID        Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID             Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID     Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID               Resource /accounts/{account_id}/transactions Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                 Resource /transactions                       Method POST
iii.Get transaction information for a specific transaction ID Resource /transactions/{transaction_id}     Method GET
```

19. Review the methods for the /accounts resource and nested resources.

Define methods for the /accounts resource to add, modify, and delete data

20. Return to API Designer.
21. Type or use the shelf to add a post method to the /accounts resource.
22. Add put and delete methods to the /{account_id} nested resource.
23. Look at the API Console; you should see tabs for the new methods.



Review the methods identified for the /transactions resource and nested resources

24. Return to the WT 2-2 Resources and methods file in a text editor.

```
Translating categories and actions into resources and methods –

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                        Resource /customers                          Method GET
ii. Register a new customer                                      Resource /customers                          Method POST
iii.Get customer information for a specific customer ID          Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID       Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID                Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID          Resource /customers/{customer_id}/accounts   Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                         Resource /accounts                           Method POST
ii. Get account information for a specific account ID            Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID                 Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID         Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID                   Resource /accounts/{account_id}/transactions Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                     Resource /transactions                       Method POST
iii.Get transaction information for a specific transaction ID    Resource /transactions/{transaction_id}      Method GET
```

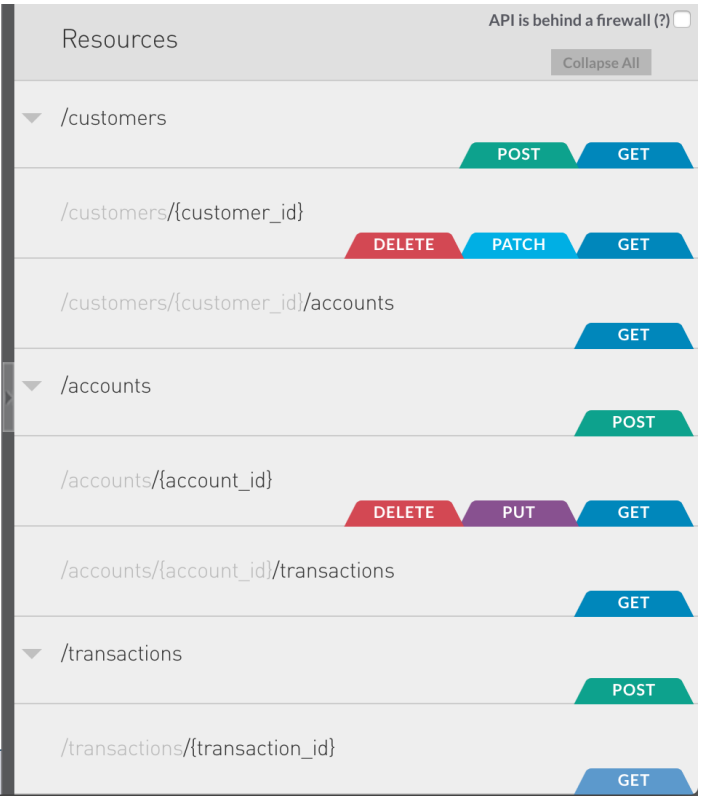
25. Review the methods for the /transactions resource and nested resources.

Define methods for the /transactions resource to add, modify and delete data

26. Return to API Designer.
27. Type or use the shelf to add a post method to the /transactions resource.
28. Save the project.

29. Look at the API Console; you should see tabs for the new methods.

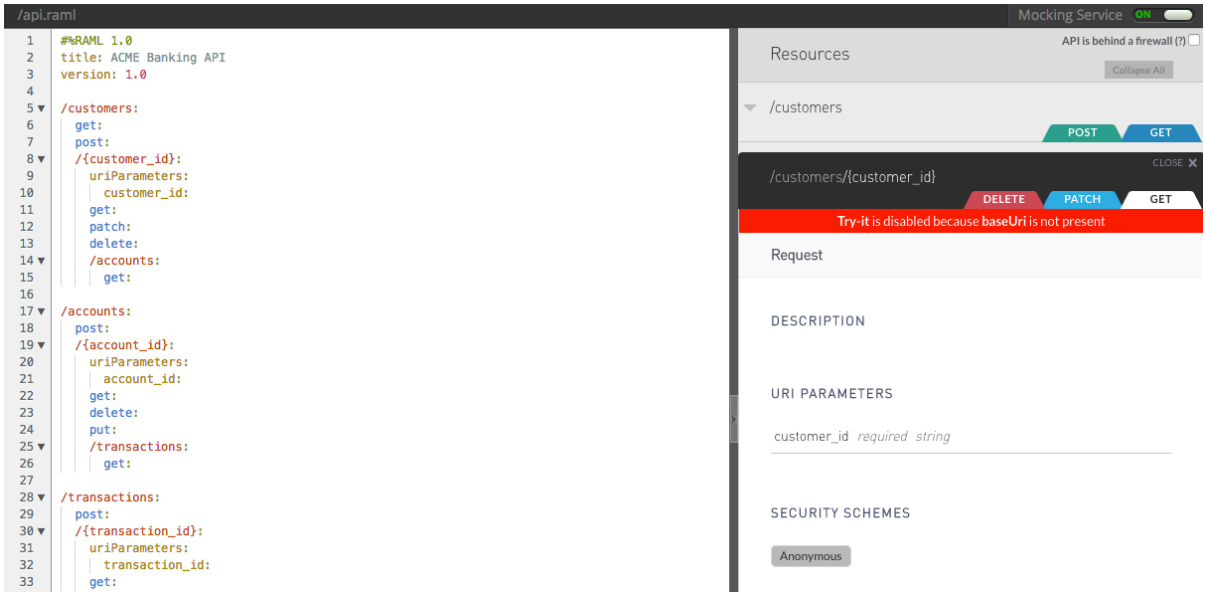
```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
  post:
  /{customer_id}:
    get:
    patch:
    delete:
    /accounts:
      get:

/accounts:
  post:
  /{account_id}:
    get:
    put:
    delete:
    /transactions:
      get:

/transactions:
  post:
  /{transaction_id}:
    get:
```



Walkthrough 4-3: Specify URI parameters for necessary resource methods

In this walkthrough, you specify URI parameters for resources that require it. You will:

- Identify resources that require URI parameters.
- Define URI parameters in the RAML definition.



Identify resources that require URI parameters

1. Return to the WT2-2 Resources and methods file in a text editor.
2. Identify the nested resources that require an ID passed to the URI as a parameter.

```
Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                    Resource /customers                          Method GET
ii. Register a new customer                                  Resource /customers                          Method POST
iii.Get customer information for a specific customer ID      Resource /customers/{customer_id}            Method GET
iv. Update customer information for a specific customer ID   Resource /customers/{customer_id}            Method PATCH
v.  Delete a customer with a specific customer ID            Resource /customers/{customer_id}            Method DELETE
vi. Get list of all accounts for a specific customer ID      Resource /customers/{customer_id}/accounts   Method GET

ACCOUNTS: Resource /accounts
i.  Create a new account                                     Resource /accounts                           Method POST
ii. Get account information for a specific account ID        Resource /accounts/{account_id}              Method GET
iii.Delete an account with a specific account ID             Resource /accounts/{account_id}              Method DELETE
iv. Update account information for a specific account ID     Resource /accounts/{account_id}              Method PUT
ii. Get transactions for a specific account ID               Resource /accounts/{account_id}/transactions Method GET

TRANSACTIONS: Resource /transactions
i.  Create a new transaction                                 Resource /transactions                       Method POST
iii.Get transaction information for a specific transaction ID Resource /transactions/{transaction_id}     Method GET
```

Note: The {customer_id}, {account_id}, and {transaction_id} resources are the ones that require URI parameters with the respective values. These URLs are invalid when called without an ID value in the URI.

Define a customer_id URI parameter

3. Return to API Designer.
4. Add a new line below the /{customer_id} nested resource above the get method.
5. Make sure the cursor in the new line is aligned to the get method in the line below.
6. Locate uriParameters in the shelf and click it.

   Note: Make sure you have the correct indentation. If you don't you will not see the correct keywords at that level of indentation.

7. Make sure the cursor is indented on the new line below uriParameters.
8. In the shelf, click customer_id:.

9. Delete the new line that is automatically added below the customer_id parameter.

```
5 /customers:
6   get:
7   post:
8   /{customer_id}:
9     uriParameters:
10      customer_id:
11    get:
12    patch:
13    delete:
14  /accounts:
15    get:
```

Define account_id and transaction_id URI parameters

10. Similarly, add a uriParmeters parameter to the nested /{account_id} resource of /accounts and name it account_id.

```
/accounts:
  post:
  /{account_id}:
    uriParameters:
      account_id:
    get:
    put:
    delete:
    /transactions:
      get:
```

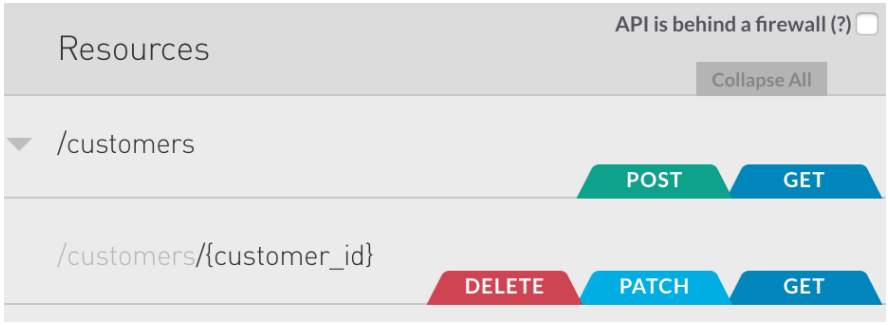
11. Add a uriParmeters parameter to the nested /{transaction_id} resource of /transactions and name it transaction_id.

```
/transactions:
  post:
  /{transaction_id}:
    uriParameters:
      transaction_id:
    get:
```

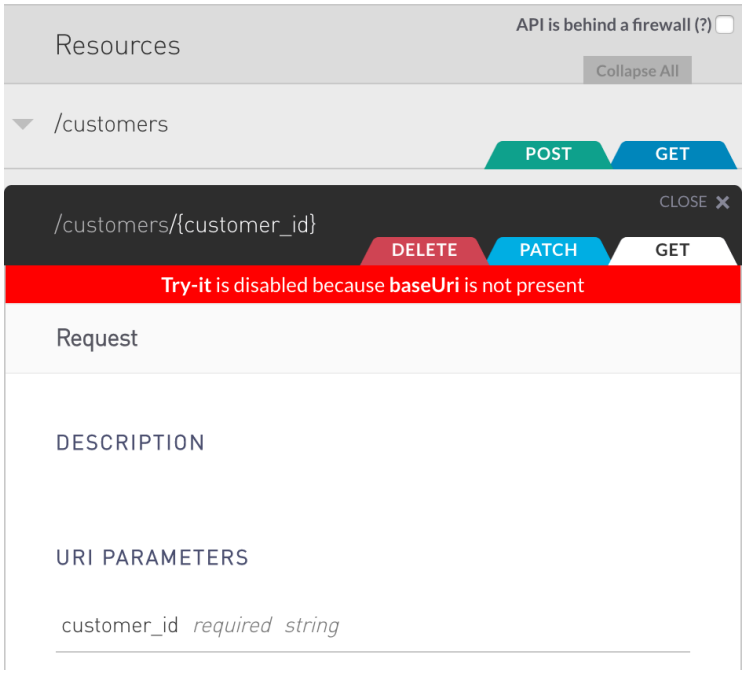
12. Save the project.

View the generated API documentation

13. In the API Console, locate the /customers/{customer_id} resource and click its GET tab.



14. Locate customer_id listed under URI PARAMETERS.



Note: We will be passing in values to the URI parameter and testing the methods in the later modules after we input examples in the API definition. URI parameters are necessary part of the resource being called.

15. In the upper-right corner, click CLOSE X.

Module 5: Specifying Responses

```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
    headers:
      Accept?:
    responses:
      200:
        headers:
          Cache-Control:
          Expires:
            type: datetime-only
        body:
          application/json:
          application/xml:
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

Objectives:

- Describe the structure of HTTP method responses.
- Use status codes in HTTP responses.
- Add error handling and caching information to HTTP responses.
- Select and specify the types of content returned in HTTP responses.

Walkthrough 5-1: Add HTTP 2xx responses to resource methods

In this walkthrough, you add HTTP 2xx response bodies to all resource methods. You will:

- Add an HTTP 200 response body to all GET methods indicating success of the HTTP request.
- Add an HTTP 201 response body to all POST methods indicating success of the HTTP request.
- Add an HTTP 200 response body to all DELETE methods indicating success of the HTTP request.
- Add a response body with both HTTP 200 and 201 status codes to the PUT method.
- Add an HTTP 204 response body to PATCH methods indicating request success.

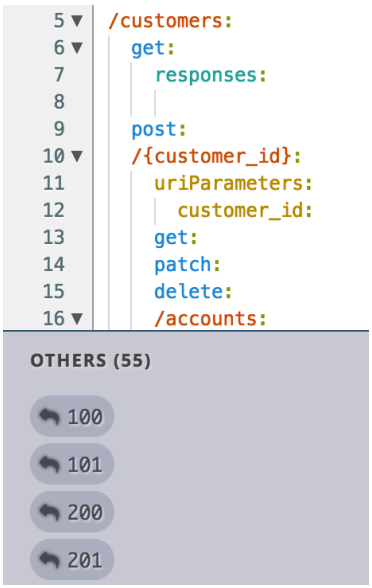
```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
    responses:
      200:
        body:
          application/json:
  post:
    responses:
      201:
        headers:
          Location:
        body:
          application/json:
  /{customer_id}:
    uriParameters:
      customer_id:
    get:
      responses:
        200:
          body:
            application/json:
    patch:
      responses:
        204:
    delete:
      responses:
        200:
          body:
            application/json:
```

Add an HTTP 200 response body to GET methods indicating request success

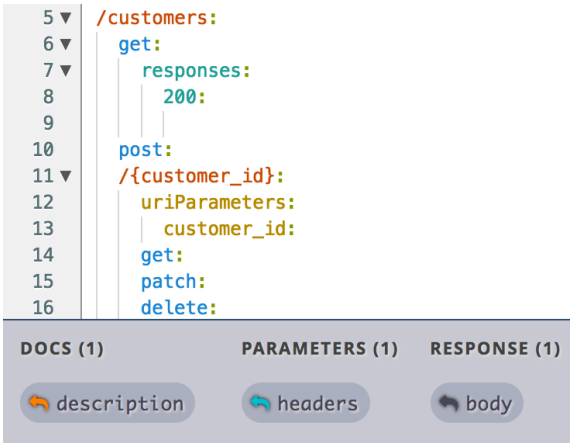
1. Return to API Designer.
2. In the editor, add a new line under the get method of the /customers resource.
3. Press the tab key to indent.

4. In the shelf, click responses.

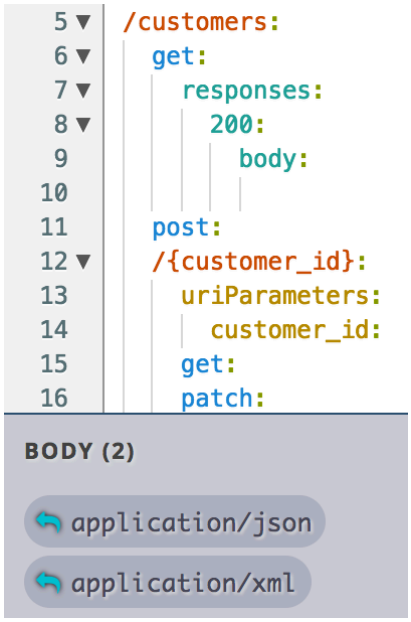
5. In the shelf, click 200.



6. In the shelf, click body.



7. In the shelf, click application/json .



8. Delete the new line created below the application/json line.

```
5 /customers:
6   get:
7     responses:
8       200:
9         body:
10          application/json:
11    post:
```

9. Similarly, in the /{customer_id} nested resource, add an HTTP 200 response body to the get method.

```
12 /{customer_id}:
13   uriParameters:
14     customer_id:
15   get:
16     responses:
17       200:
18         body:
19           application/json:
```

10. Repeat the process to add the same HTTP 200 response body to the get methods of the following resources:

    - /customers/{customer_id}/accounts
    - /accounts/{account_id}
    - /accounts/{account_id}/transactions
    - /transactions/{transaction_id}

```
/customers:
  get:
    responses: ⋯
  post:
  /{customer_id}:
    uriParameters:
      customer_id:
    get:
      responses:
        200:
          body:
            application/json:
    patch:
    delete:
    /accounts:
      get:
        responses:
          200:
            body:
              application/json:

/accounts:
  post:
  /{account_id}:
    uriParameters:
      account_id:
    get:
      responses:
        200:
          body:
            application/json:
    delete:
    put:
    /transactions:
      get:
        responses:
          200:
            body:
              application/json:

/transactions:
  post:
  /{transaction_id}:
    uriParameters:
      transaction_id:
    get:
      responses:
        200:
          body:
            application/json:
```

Add an HTTP 201 response body to POST methods indicating request success

11. Add an HTTP 201 response body to the post method of the /customers resource.

```
/customers:
  get:
    responses:
      200:
        body:
          application/json:
  post:
    responses:
      201:
        body:
          application/json:
```

12. Add a new line above the body and indent to the same level as body.
13. In the shelf, click the headers parameter.
14. In the shelf, scroll and click the Location header.
15. Delete the new line created below the Location line.

```
  post:
    responses:
      201:
        headers:
          Location:
        body:
          application/json:
```

16. Repeat the process to add the same HTTP 201 response body to the post methods of the following resources:

    - /accounts
    - /transactions

```
/accounts:
  post:
    responses:
      201:
        headers:
          Location:
        body:
          application/json:
  /{account_id}:
```

```
/transactions:
  post:
    responses:
      201:
        headers:
          Location:
        body:
          application/json:
```

Add an HTTP 200 response body to the DELETE methods indicating request success

17. Add an HTTP 200 response body to the delete methods of the following resources:

    - /customers/{customer_id}
    - /accounts/{account_id}

```
/{customer_id}:
  uriParameters:
    customer_id:
  get:
    responses:
      200:
        body:
          application/json:
  patch:
  delete:
    responses:
      200:
        body:
          application/json:
```

```
/{account_id}:
  uriParameters:
    account_id:
  get:
    responses:
      200:
        body:
          application/json:
  delete:
    responses:
      200:
        body:
          application/json:
  put:
```

Add a response body with both HTTP 200 and 201 responses to the PUT method

18. Add an HTTP 200 response body to the put method of the /{account_id} nested resource.
19. Add a second status code to the response with a value of 201.

```
  put:
    responses:
      200:
        body:
          application/json:
      201:
        body:
          application/json:
```

Add an HTTP 204 response body to PATCH methods indicating request success

20. Add a new line of code below the patch method of the /customers resource and indent.
21. In the shelf below, click responses.
22. In the shelf, click 204.

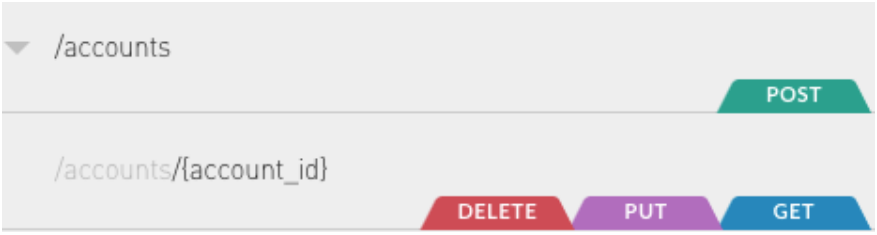
23. Delete the new line created below the 204 status code line.

```
25                application/json:
26 ▼        patch:
27            responses:
28              204:
29          delete:
```

24. Save the project.

View the generated API documentation

25. In the API Console, locate the /accounts/{account_id} resource and click its PUT tab.



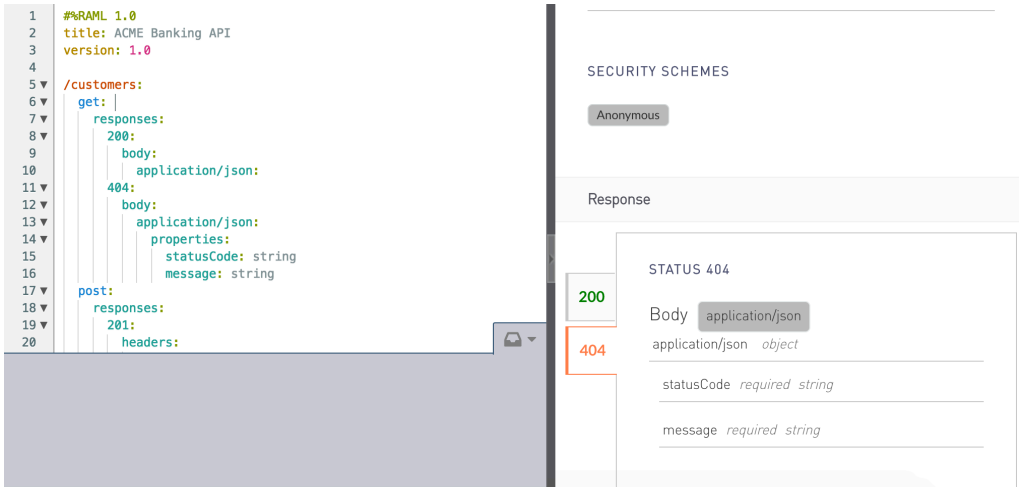
26. Locate the two HTTP status code responses under Response.

27. Scroll up to the /accounts/{account_id} resource name and click the CLOSE X button.

Walkthrough 5-2: Add responses bodies to return error status codes and messages for client-side errors

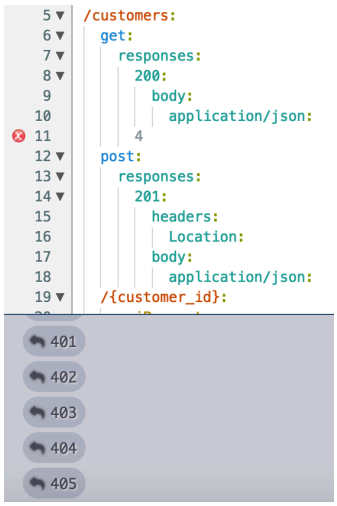
In this walkthrough, you add response bodies to return error status codes and error messages for client-side errors. You will:

- Add HTTP 4xx status code responses to all GET and DELETE methods.
- Specify error messages to be returned.



Add a 404 Not Found HTTP status code response to a GET method

1. Return to API Designer.
2. In the /customers resource, add a new line under application/json in the get method.
3. Press backspace or delete two times to go back two tab spaces so the cursor is at the same indent level as the 200.
4. Type 4, then scroll through the shelf to locate 404 and click it.

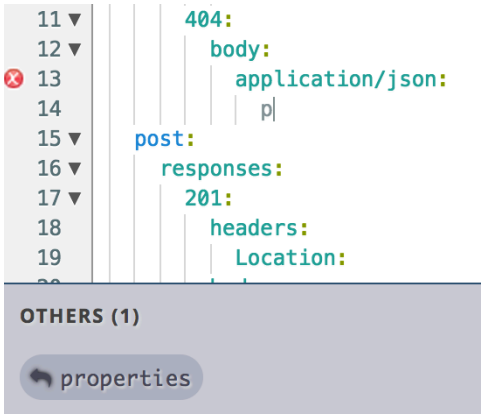


5. In the shelf, click body.
6. In the shelf, click application/json.

```
/customers:
  get:
    responses:
      200:
        body:
          application/json:
      404:
        body:
          application/json:

  post:
```

Add a custom error message for the response

7. In the new line below application/json, type the letter p and in the shelf, click properties.



8. In the new line below properties, type statusCode:.
9. Press spacebar and scroll down in the shelf and click string.

```
      404:
        body:
          application/json:
            properties:
              statusCode: string
```

10. Press enter and type message: in the new line.

11. Press spacebar and scroll down in the shelf and click string.

```
/customers:
  get:
    responses:
      200:
        body:
          application/json:
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

Add the HTTP 404 response body to all GET and DELETE methods

12. Copy the 404 response body you just created.

```
/customers:
  get:
    responses:
      200:
        body:
          application/json:
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

13. Add the 404 response body to the get methods of the following resources:

- /customers/{customer_id}
- /accounts
- /accounts/{account_id}
- /accounts/{account_id}/transactions
- /transactions/{transaction_id}

```
  /{customer_id}:
    uriParameters:
      customer_id:
    get:
      responses:
        200:
          body:
            application/json:
        404:
          body:
            application/json:
              properties:
                statusCode: string
                message: string
```

14. Add the 404 response body to the delete methods of the following resources:

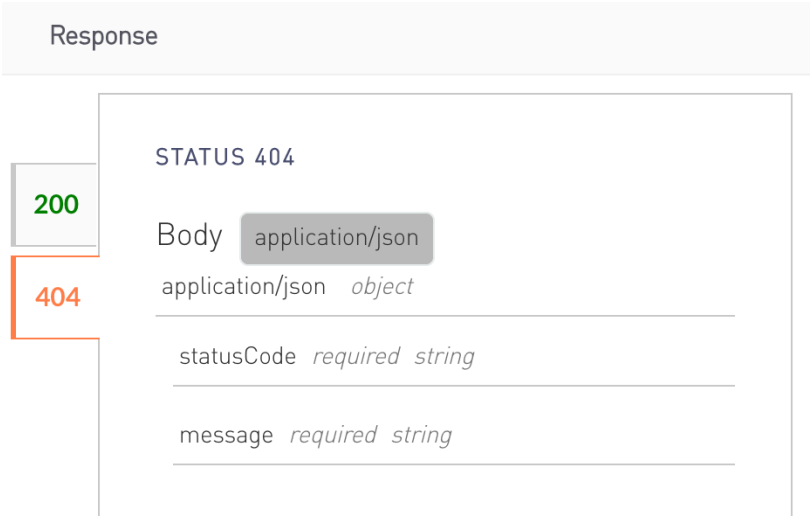
    - /customers/{customer_id}
    - /accounts/{account_id}

```
/{customer_id}:
  uriParameters:
    customer_id:
  get:
    responses:
      200:
        body:
          application/json:
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
  patch:
    responses:
      204:
  delete:
    responses:
      200:
        body:
          application/json:
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

View the API documentation

15. In the API Console, click the GET for the /customers resource.

16. Scroll down to the Response section and click the 404 tab; you should see the custom error message properties listed under the application/json object.

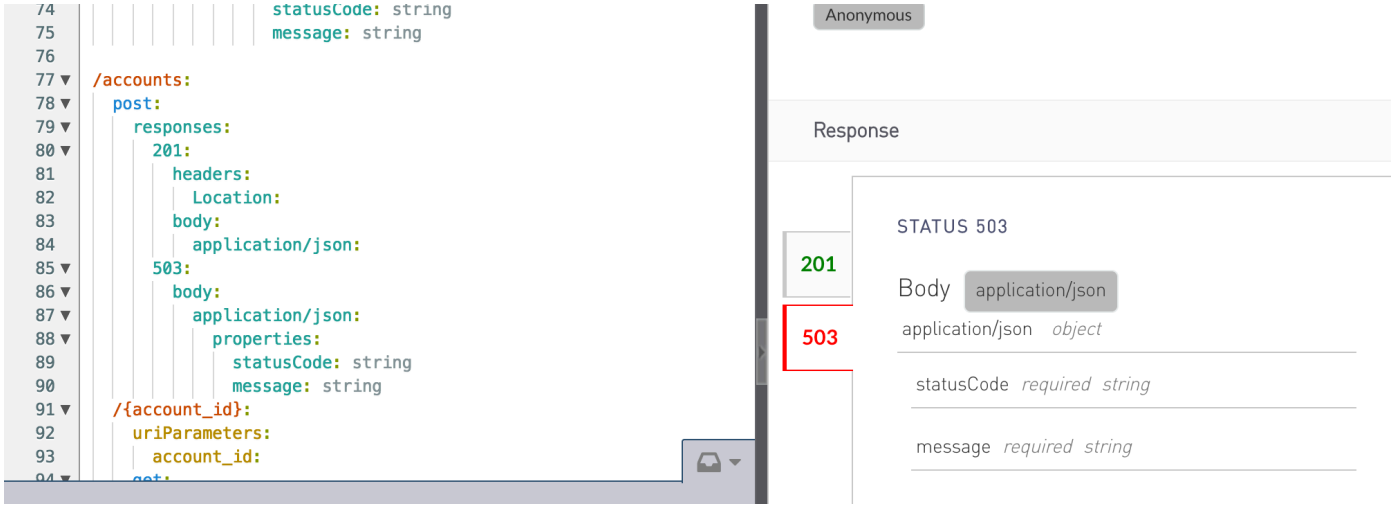


17. Scroll up and click CLOSE X.

Walkthrough 5-3: Add response bodies to return error status codes and messages for server-side errors

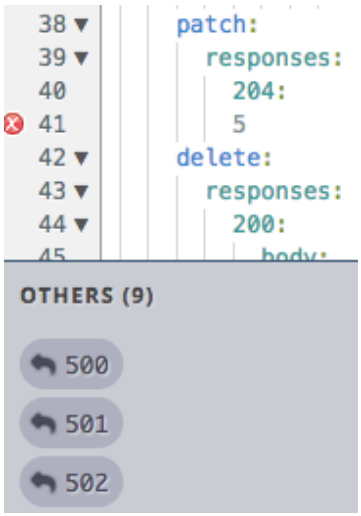
In this walkthrough, you add response bodies to return error status codes and error messages for server-side errors. You will:

- Add necessary HTTP 5xx status code responses to all PATCH, PUT and POST methods.
- Specify error messages to be returned.



Add a 501 Not Implemented HTTP status code response to the PATCH method

1. Return to API Designer.
2. In the /customers resource, add a new line under 204 in the patch method.
3. Type 5, then scroll through the shelf to locate 501 and click it.



4. In the shelf, click body.

5. In the shelf, click application/json.

```
38 ▼      patch:
39 ▼        responses:
40            204:
41 ▼          501:
42 ▼            body:
43              application/json:
44
45 ▼      delete:
```

Add a custom error message for the response

6. In the new line below application/json, type the letter p and in the shelf, click properties.
7. In the new line below properties, type statusCode:.
8. Press spacebar and scroll down in the shelf and click string.

```
38 ▼      patch:
39 ▼        responses:
40            204:
41 ▼          501:
42 ▼            body:
43 ▼              application/json:
44                properties:
45                  statusCode: string
```

9. Press enter and type message: in the new line.
10. Press spacebar and scroll down in the shelf and click string.

```
38 ▼      patch:
39 ▼        responses:
40            204:
41 ▼          501:
42 ▼            body:
43 ▼              application/json:
44 ▼              properties:
45                  statusCode: string
46                  message: string
```

Add the HTTP 501 response body to the PUT method

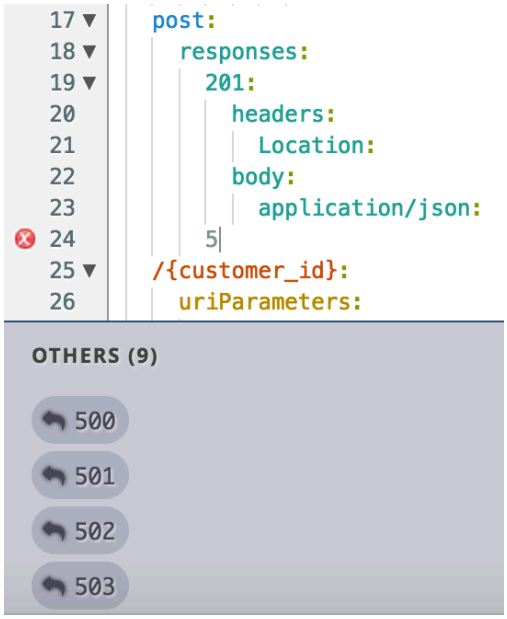
11. Copy the 501 response body you just created.

12. Add the 501 response body to the put method of the /accounts/{account_id} resource.

```
98 ▼    put:
99 ▼      responses:
100 ▼       200:
101           body:
102             application/json:
103 ▼       201:
104           body:
105             application/json:
106 ▼       501:
107 ▼         body:
108 ▼           application/json:
109 ▼             properties:
110                 statusCode: string
111                 message: string
112 ▼   /transactions:
```

Add the 503 Service Unavailable HTTP status code response to a POST method

13. In the /customers resource, add a new line under application/json in the post method.
14. Press backspace or delete two times to go back two tab spaces so the cursor is at the same indent level as the 201.
15. Type 5, then click 503 in the shelf.



16. In the shelf, click body.
17. In the shelf, click application/json.
18. In the shelf, click properties.
19. In the new line below the properties line, type statusCode:.

20. Press spacebar and scroll down the shelf and select string.
21. In the new line press enter and type message:.
22. Press spacebar and scroll down the shelf and select string.
23. Press enter and type message: in the new line.
24. Press spacebar and scroll down in the shelf and click string.

```
post:
  responses:
    201:
      headers:
        Location:
      body:
        application/json:
    503:
      body:
        application/json:
          properties:
            statusCode: string
            message: string
```

Add the HTTP 503 response body to all POST methods

25. Copy the 503 response body you just created.
26. Add the 503 response body to the post methods of the following resources:

- /accounts
- /transactions

```
/accounts:
  post:
    responses:
      201:
        headers:
          Location:
        body:
          application/json:
      503:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

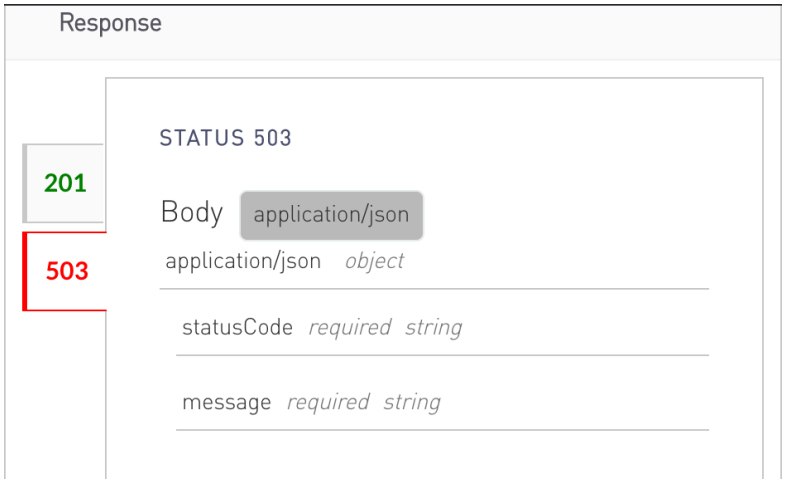
```
/transactions:
  post:
    responses:
      201:
        headers:
          Location:
        body:
          application/json:
      503:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

27. Save the project.

View the API documentation

28. In the API Console, click POST for the /customers resource.

29. Scroll down to the Response section and click the 503 tab; you should see the custom error message properties listed under the application/json object.

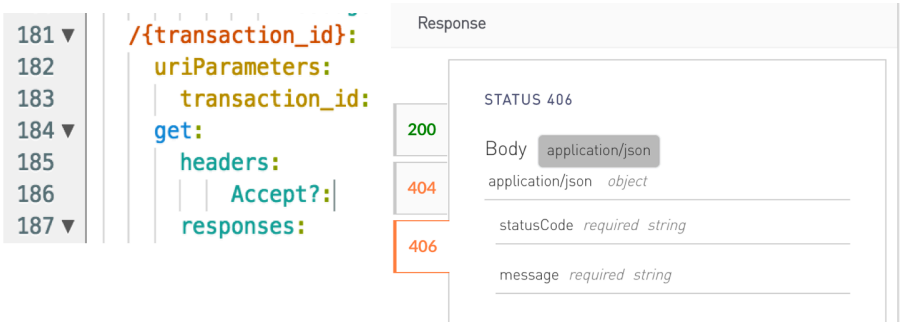


30. Scroll up and click CLOSE X.

Walkthrough 5-4: Add flexible content-types to HTTP responses

In this walkthrough, you add flexible content-types to the HTTP responses for retrieving the customer information including accounts and transactions. You will:

- Add XML body type to the HTTP 200 responses for all GET methods.
- Add an optional accept header to specify the type of response body requested.
- Add HTTP status code errors for when unsupported content types are requested.



Add XML body type to the HTTP 200 responses body for all GET methods

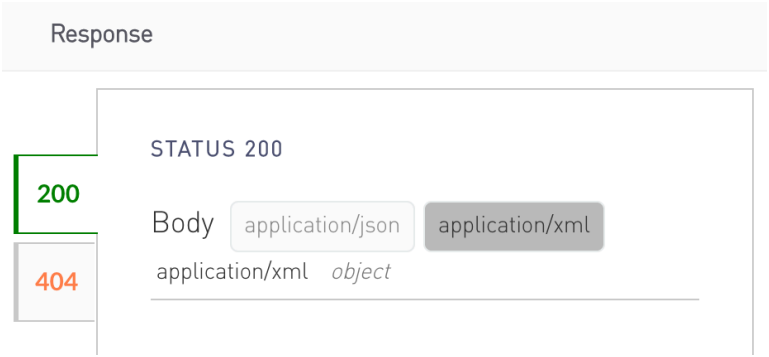
1. Return to API Designer.
2. In the /customers resource, add a new line below application/json in the 200 response of the get method.
3. In the shelf, click application/xml.
4. Delete the new line created below application/xml.

```
5  /customers:
6    get:
7      responses:
8        200:
9          body:
10           application/json:
11           application/xml:
12       404:
```

5. Add the application/xml body type to the HTTP 200 responses of the get methods of the following resources:
    - /customers/{customer_id}
    - /customers/{customer_id}/accounts
    - /accounts/{account_id}
    - /accounts/{account_id} /transactions
    - /transactions/{transaction_id}

View the API documentation

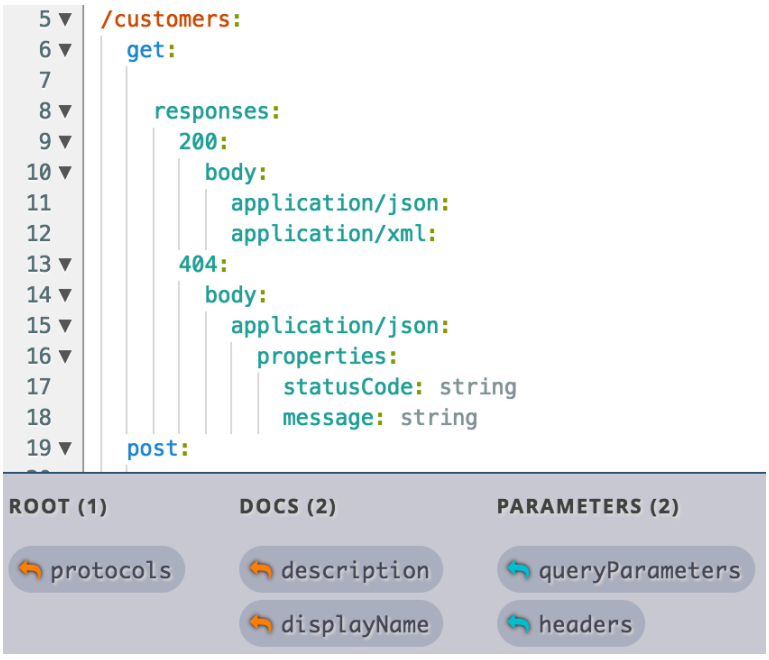
6. In the API Console, click GET for the /customers resource.
7. Scroll down to the Response section and click the 503 tab; you should see both application/json and application/xml listed as possible return types.
8. Click application/XML.



9. Scroll up and click CLOSE X.

Add an optional accept header to specify the type of response body requested

10. In the /customers resource, add a new line below the get line of the get method.
11. In the shelf, click headers.



12. In the shelf, click Accept.

13. Delete the new line created below the Accept header property.
14. Go to the line with the Accept header and change it to an optional parameter by adding a question mark at the end of it.

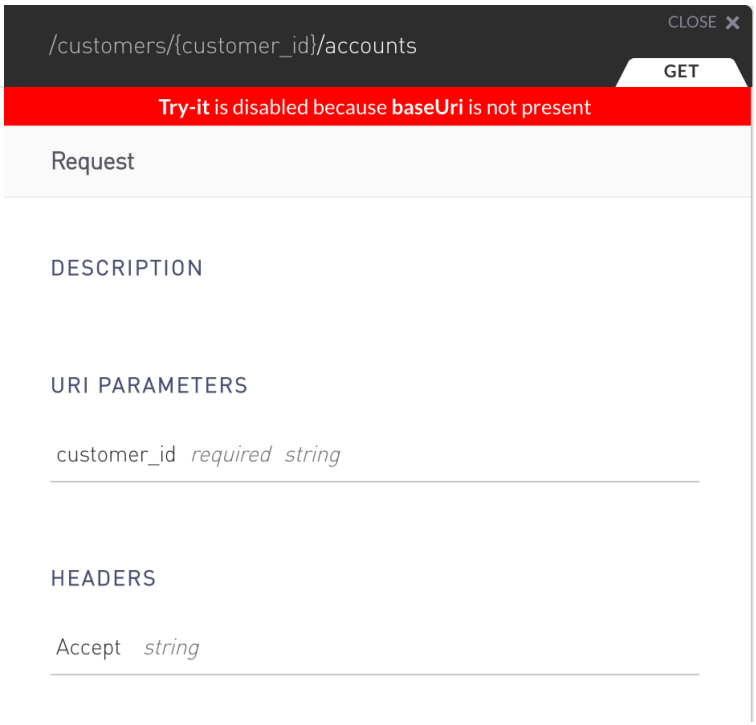
Note: When you make the Accept header optional, the API can be implemented to return the response body in JSON by default when the header is not specified.

15. Add the optional Accept header to the get methods of the following resources:

- /customers/{customer_id}
- /customers/{customer_id}/accounts
- /accounts /{account_id}
- /accounts /{account_id}/transactions
- /transactions /{transaction_id}

View the API documentation

16. In the API Console, click GET for the /customers/{customer_id}/accounts resource.
17. In the Headers section for the request, locate the Accept header.



18. Click CLOSE X.

Add HTTP status code errors for when unsupported content types are requested

19. In the /customers resource, add a new line below the message property in the HTTP 404 response body.
20. Press backspace or delete four times to go four tab spaces back.

21. Type 4 and in the shelf, scroll down and click 406.

22. In the shelf, click body.
23. In the shelf, click application/json.
24. In the shelf, click properties.
25. Add a property called statusCode of type string.
26. Add a property called message of type string.

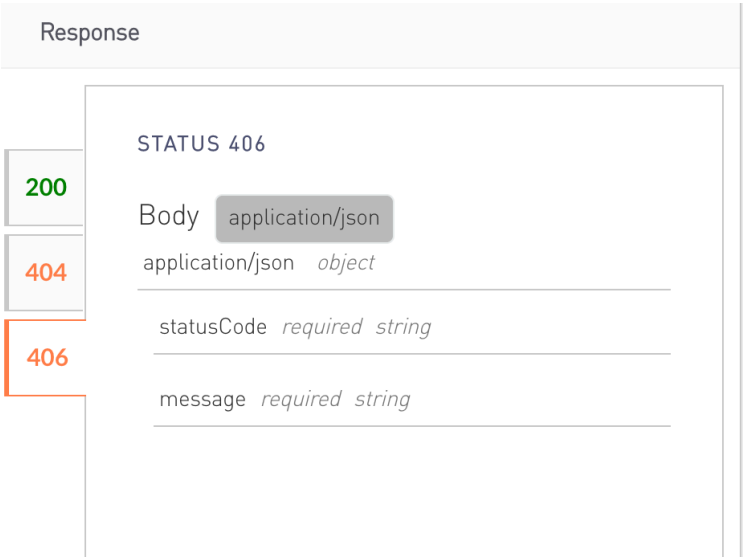
```
/customers:
  get:
    headers:
      Accept?:
    responses:
      200:
        body:
          application/json:
          application/xml:
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
      406:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

27. Copy the 406 response body you just created.

28. Add the 406 response body to the post methods of the following resources:
    - /customers/{customer_id}
    - /customers/{customer_id}/accounts
    - /accounts /{account_id}
    - /accounts /{account_id}/transactions
    - /transactions /{transaction_id}
29. Save the project.

View the API documentation

30. In API Console, click GET for the /customers resource.
31. In the Response section, click the 406 tab to view the HTTP 406 response body.



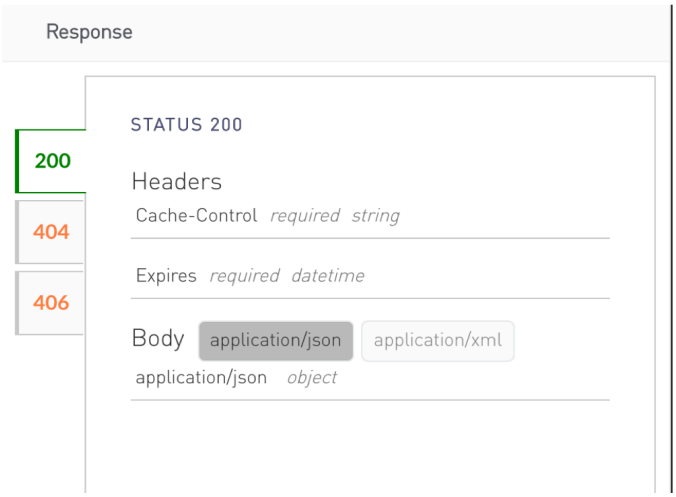
32. Click CLOSE X.

Walkthrough 5-5: Add caching information to HTTP methods

In this walkthrough, you indicate that a resource is cacheable by using cache-control headers. You will:

- Add a Cache-Control header to a GET response to enable caching.
- Add an Expires header to a GET response to set the date when the cached resource becomes invalid.

```
/customers:
  get:
    headers:
      Accept?:
    responses:
      200:
        headers:
          Cache-Control:
          Expires:
            type: datetime
        body:
          application/json:
          application/xml:
```



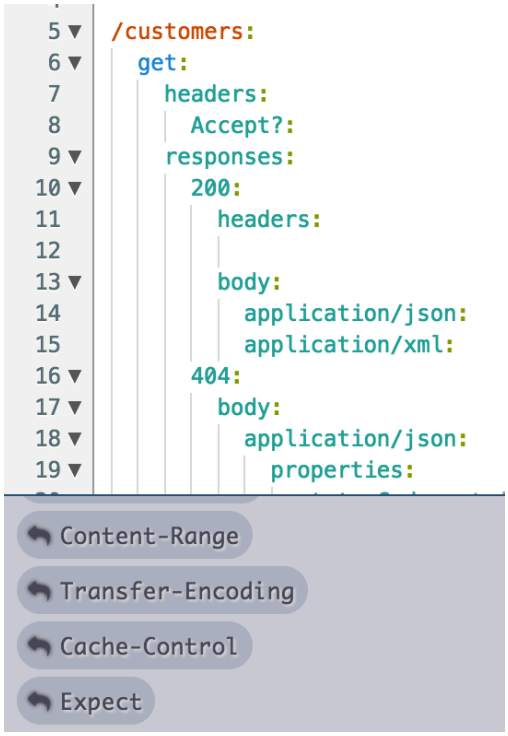
Add a Cache-Control header to a GET response to enable caching

1. Return to API Designer.
2. In the get method for the /customers resource, add a new line below the line of code with 200.
3. In the shelf, click headers.

```
/customers:
  get:
    headers:
      Accept?:
    responses:
      200:

        body:
          application/json:
          application/xml:
      404:
        body:
          application/json:
            properties:
              statusCode: string
```

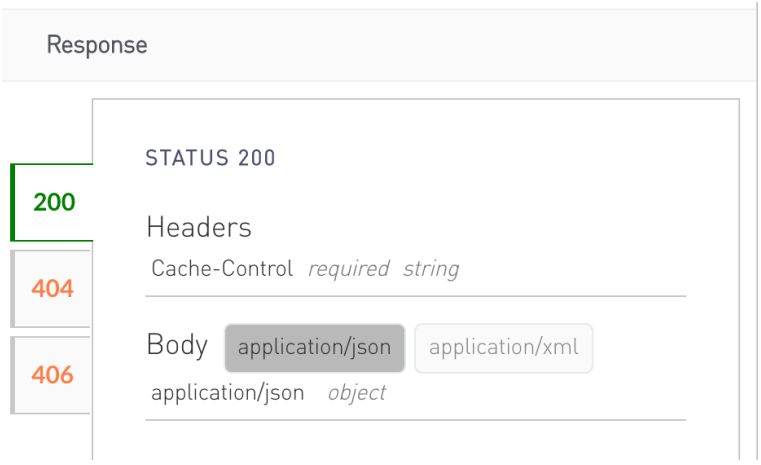
DOCS (1) description | PARAMETERS (1) headers | RESPONSE (1) body

4. In the shelf, scroll down and click Cache-Control.



View the API documentation

5. In the API Console, click GET for the /customers resource.
6. In Response section, locate the Cache-Control header .



Add an Expires headers to a GET response to set the date when the cached resource becomes invalid

7. In the editor, go to the new line after the Cache-Control header.

8. Align the cursor below the start of the Cache-Control line.

```
5 ▼ /customers:
6 ▼   get:
7       headers:
8         Accept?:
9 ▼     responses:
10 ▼      200:
11 ▼        headers:
12            Cache-Control:
13
```

9. In the shelf, scroll down and click Expires.
10. In the shelf, click type in the OTHERS section.
11. In the shelf, scroll down and click datetime.

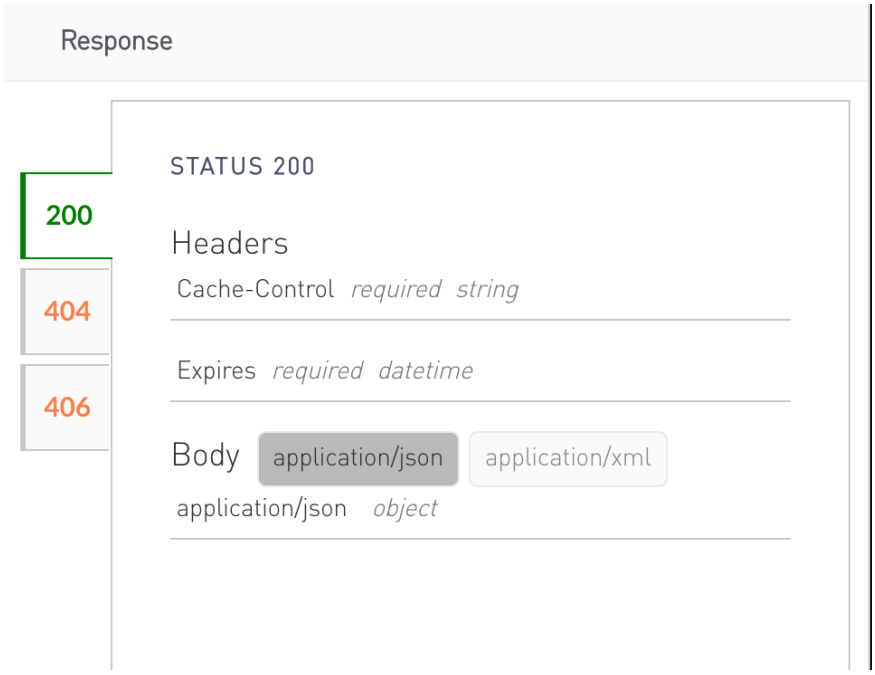
```
5 ▼ /customers:
6 ▼   get:
7       headers:
8         Accept?:
9 ▼     responses:
10 ▼      200:
11 ▼        headers:
12            Cache-Control:
13            Expires:
14              type: datetime
15 ▼        body:
16            application/json:
17            application/xml:
```

12. Save the Project.

View the API documentation

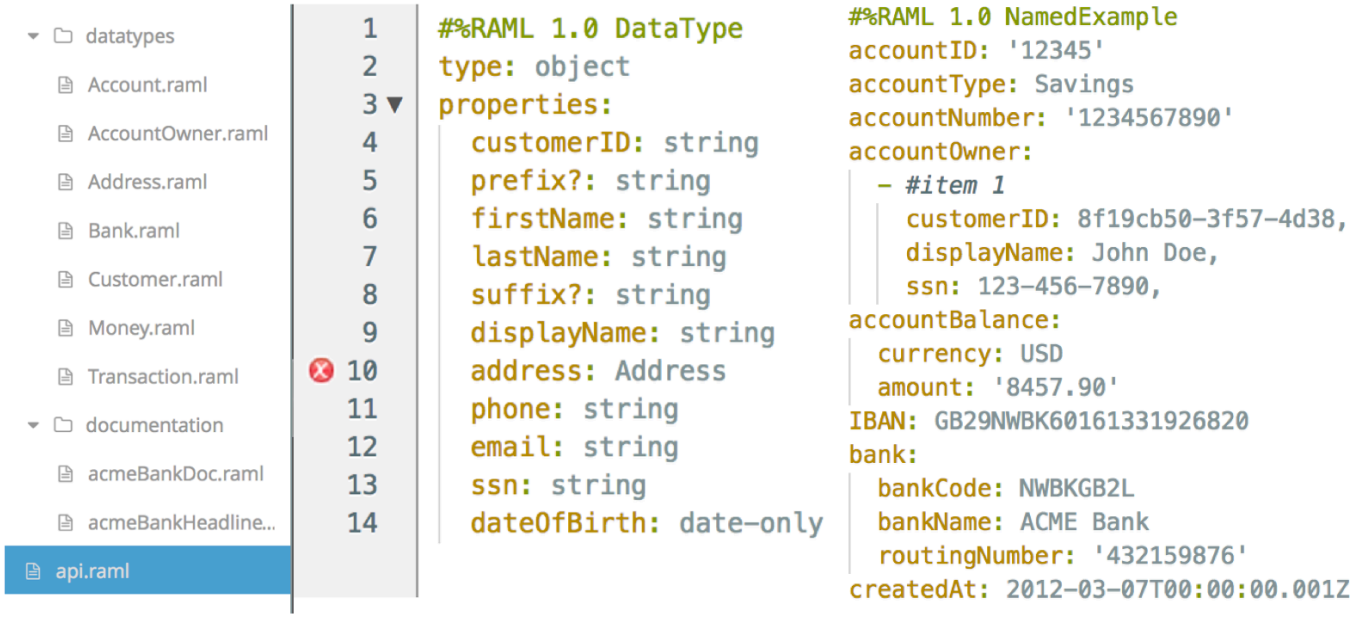
13. In the API Console, click GET for the /customers resource.

14. In the Response section, locate the Expires header for the HTTP Status 200 response.



15. Scroll up and click CLOSE X.

Module 6: Modelling Data



Objectives:

- List datatypes and their attributes to be returned from or sent to resource methods.
- Create datatype fragments.
- Set request and response body types to datatypes.
- Create examples for datatype fragments.
- Include examples in request and response bodies.

Walkthrough 6-1: List datatypes and their attributes for an API

In this walkthrough, you identify datatypes and their attributes to be used in HTTP request and response bodies for the ACME Banking API. You will:

- List the datatypes required for the resource methods.
- Identify the attributes for each datatype.
- Create necessary additional datatypes to simplify the identified datatypes.
- Identify optional attributes in datatypes.

Attributes for Customer datatype along with each attribute type:

- customerID	type: string
- prefix?	type: string
- firstName	type: string
- middleName?	type: string
- lastName	type: string
- suffix?	type: string
- displayName	type: string
- address	type: Address
- phone	type: string
- email	type: string
- ssn	type: string
- dateOfBirth	type: date-only

Attributes for Account datatype along with each attribute type:

- accountID	type: string
- accountType	type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber	type: string
- accountOwner	type: AccountOwner[]
- accountBalance	type: Money (since it should include currency and an amount)
- IBAN	type: string
- bank	type: Bank
- interestRate?	type: number
- createdAt	type: datetime
- modifiedAt?	type: datetime

Attributes for Transaction datatype along with each attribute type:

- transactionID	type: string
- fromAccount	type: Account
- toAccount	type: Account
- transactionType	type: enum[atm, check, deposit, cashWithdrawal, onlineTransfer, sepa]
- transactionName?	type: string
- transactionAmount	type: Money
- newAccountBalance	type: Money
- postedAt	type: datetime
- completedAt?	type: datetime

Attributes for Address datatype:

- addressLine1	type: string
- addressLine2?	type: string
- city	type: string
- state	type: string
- country	type: string
- zipCode	type: string

List out the base datatypes required for the resource methods

1. Navigate to the studentFiles folder on your computer.
2. Navigate to the module08 directory and open the WT8-1 Datatypes and attributes text file in a text editor.

Identify the attributes for each datatype

3. Brainstorm with the class about possible attributes for each datatype.

   Note: The instructor may break you out into smaller groups to first brainstorm with some of your peers.

4. After a brief discussion, type the following attributes and their types for the Customer datatype:

   - customerID type: string
   - prefix type: string
   - firstName type: string
   - middleName type: string
   - lastName type: string
   - suffix type: string
   - displayName type: string
   - addressLine1 type: string
   - addressLine2 type: string
   - city type: string
   - state type: string
   - country type: string
   - zipCode type: string
   - phone type: string
   - email type: string
   - ssn type: string
   - dateOfBirth type: date-only

5. After a brief discussion, type the following attributes and their types for the Account datatype:

   - accountID type: string
   - accountType type: enum [Checking, Savings, Overdraft Savings, Credit Card]
   - accountNumber type: string
   - accountOwner type: Customer[]
   - accountBalance type: Money (since it should include currency and an amount)
   - IBAN type: string
   - bankCode type: string
   - routingNumber type: string
   - bankName type: string

- swiftBIC type: string
- interestRate type: number
- createdAt type: datetime
- modifiedAt type: datetime

6. After a brief discussion, type the following attributes and their types for the Transaction datatype:

- transactionID type: string
- fromAccount type: Account
- toAccount type: Account
- transactionType type: enum [atm, check, deposit, cashWithdrawal, online, sepa]
- transactionName type: string
- transactionAmount type: Money
- newAccountBalance type: Money
- postedAt type: datetime
- completedAt type: datetime

Create necessary additional user-defined datatypes as per the attribute definition

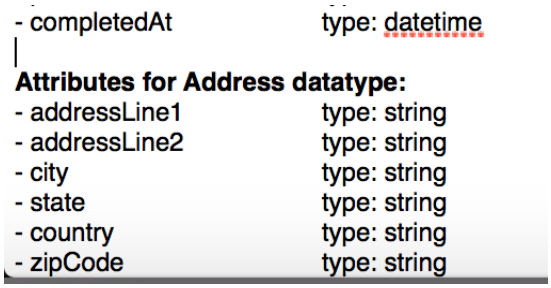
7. In the Customer datatype, select attributes from addressLine1 to zipCode and cut the lines and paste them at the end of the document.

```
- lastName              type: string
- suffix                type: string

- phone                 type: string
- email                 type: string
- ssn                   type: string
- dateOfBirth           type: date-only

Attributes for Account datatype along with each attribute type:
- ID                    type: string
- accountType           type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber         type: string
- accountOwner          type: Customer[]
- accountBalance        type: Money (since it should include currency and an amount)
- IBAN                  type: string
- bankID                type: string
- swiftBIC              type: string
- interestRate          type: number
- createdAt             type: datetime
- modifiedAt            type: datetime

Attributes for Transaction datatype along with each attribute type:
- ID                    type: string
- fromAccount           type: Account
- toAccount             type: Account
- transactionType       type: enum[atm, check, deposit, cashWithdrawal, onlineTransfer]
- transactionName       type: string
- transactionAmount     type: Money
- newAccountBalance     type: Money
- postedAt              type: datetime
- completedAt           type: datetime


- addressLine1          type: string
- addressLine2          type: string
- city                  type: string
- state                 type: string
- country               type: string
- zipCode               type: string
```

8. In the empty line, in between the Customer datatype attributes, type:

   - address type: Address

9. For the attributes that were to the end of the document, add a header to indicate them as attributes for Address datatype.



10. Similarly, create a new datatype Money with the following attributes:

    - currency type: string
    - amount type: string

11. In the Account datatype, go to the line that contains the accountOwner attribute and change the type to AccountOwner[].

Attributes for Account datatype along with each attribute type:

- ID type: string
- accountType type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber type: string
- accountOwner type: AccountOwner[]
- accountBalance type: Money (since it should include currency and an amount)
- IBAN type: string

Note: Although using Customer data type for accountOwner attribute is valid, we create an AccountOwner datatype because you only need a subset of information (name, ID and ssn) about the customer to add them as an account owner. This way the entire customer datatype and their attributes are not necessary.

12. Create a new datatype AccountOwner with the following attributes:

    - customerID type: string
    - ssn type: string
    - displayName type: string

13. In the Account datatype, select the attributes bankCode, bankName and swiftBIC and move them to end of the document and name them as attributes for Bank datatype.

Attributes for Account datatype along with each attribute type:

- ID	type: string
- accountType	type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber	type: string
- accountOwner	type: AccountOwner[]
- accountBalance	type: Money (since it should include currency and an amount)
- IBAN	type: string
- bankCode	type: string
- bankName	type: string
- swiftBIC	type: string
- interestRate	type: number
- createdAt	type: datetime
- modifiedAt	type: datetime

14. Reference the bank datatype within the Account datatype as:

- bank type: Bank.

Identify optional attributes in the datatypes

15. Brainstorm with the class about the attributes that could be optional for each datatype.
16. Identify the optional attributes and add a ? right after the attribute name.

Attributes for Customer datatype along with each attribute type:

- ID	type: string
- prefix?	type: string
- firstName	type: string
- middleName?	type: string
- lastName	type: string
- suffix?	type: string
- displayName	type: string
- address	type: Address
- phone	type: string
- email	type: string
- ssn	type: string
- dateOfBirth	type: date-only

Attributes for Account datatype along with each attribute type:

- ID	type: string
- accountType	type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber	type: string
- accountOwner	type: AccountOwner[]
- accountBalance	type: Money (since it should include currency and an amount)
- IBAN	type: string
- bank	type: Bank
- interestRate?	type: number
- createdAt	type: datetime
- modifiedAt?	type: datetime

Attributes for Transaction datatype along with each attribute type:

- ID type: string
- fromAccount type: Account
- toAccount type: Account
- transactionType type: enum[atm, check, deposit, cashWithdrawal, onlineTransfer, sepa]
- transactionName? type: string
- transactionAmount type: Money
- newAccountBalance type: Money
- postedAt type: datetime
- completedAt? type: datetime

Attributes for Address datatype:

- addressLine1 type: string
- addressLine2? type: string
- city type: string
- state type: string
- country type: string
- zipCode type: string

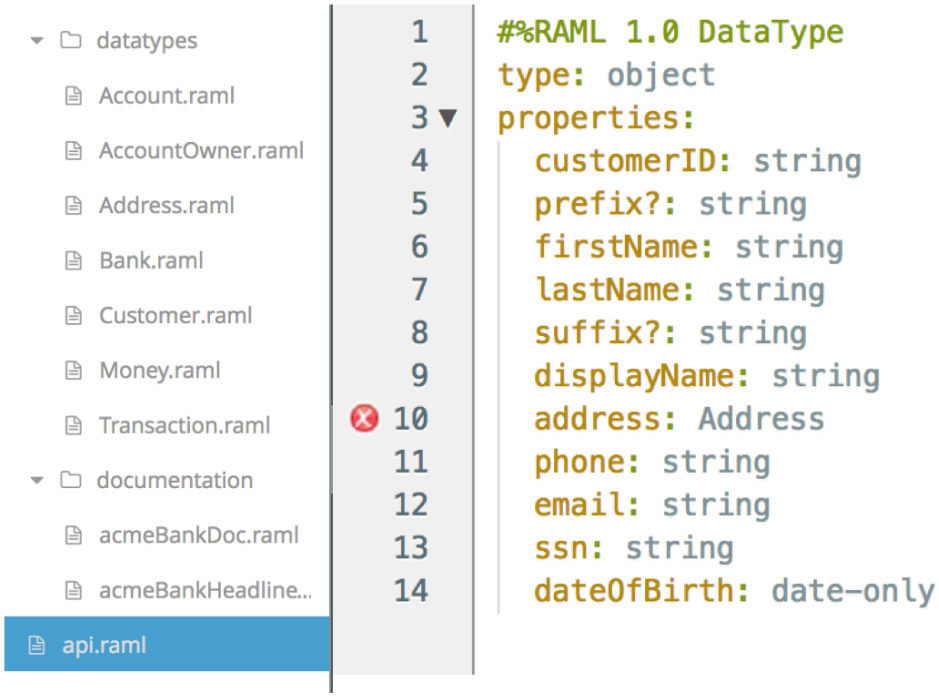
Attributes for the Bank datatype:

- bankCode type: string
- bankName type: string
- swiftBIC? type: string
- routingNumber? type: string

Walkthrough 6-2: Create datatype fragments

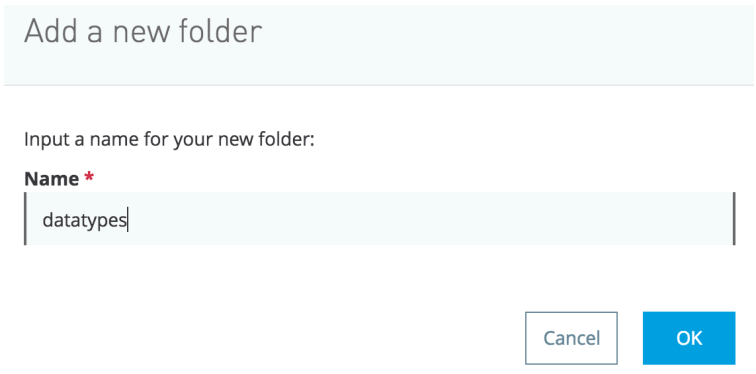
In this walkthrough, you define datatype fragments for the identified datatypes for the ACME Banking API. You will:

- Create individual datatype fragment files for the identified datatypes.
- Define the datatypes with required and optional attributes.



Refactor the custom error message object to use a datatype

1. Return to API Designer.
2. Click the Project menu and select New Folder.
3. In the Add a new folder dialog box, set the name to datatypes.



4. Click OK.

5. In the file browser, click api.raml and go to the /customers resource get method HTTP 404 response body and copy the lines from properties up until the message string.

```
16            application/json:
17            application/xml:
18          404:
19            body:
20              application/json:
21                properties:
22                  statusCode: string
23                  message: string
```

6. In the file browser section, right-click the datatypes folder and select New File > RAML 1.0 > Type.
7. In the Add new Type file dialog box, set the name to CustomErrorMessage.raml and click OK.
8. Go to the end of the first line in the new file and press enter.
9. Type the following:

```
type: object
```

10. Press enter and paste the copied lines below.

```
#%RAML 1.0 DataType
type: object
properties:
      statusCode: string
      message: string
```

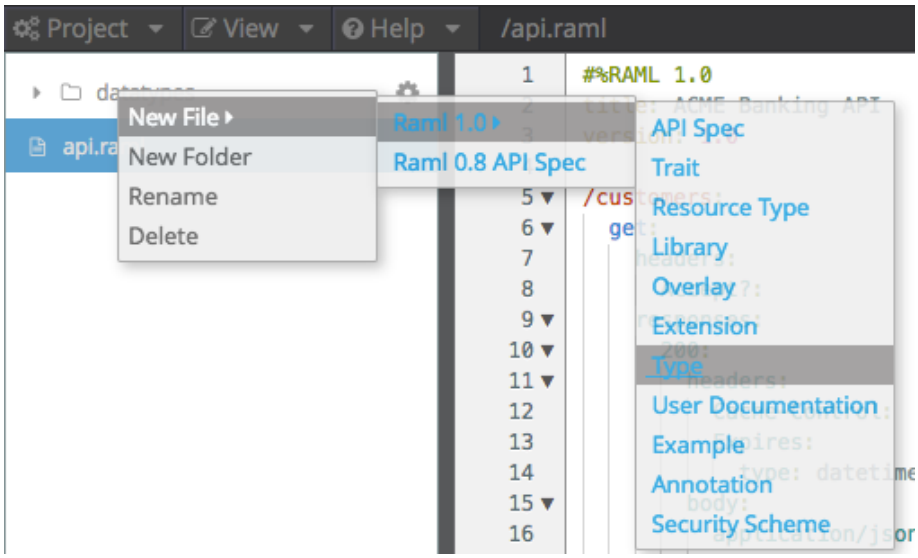
11. Select the statusCode and message properties line and press Shift+Tab to indent the lines properly.

```
#%RAML 1.0 DataType
type: object
properties:
  statusCode: string
  message: string
```

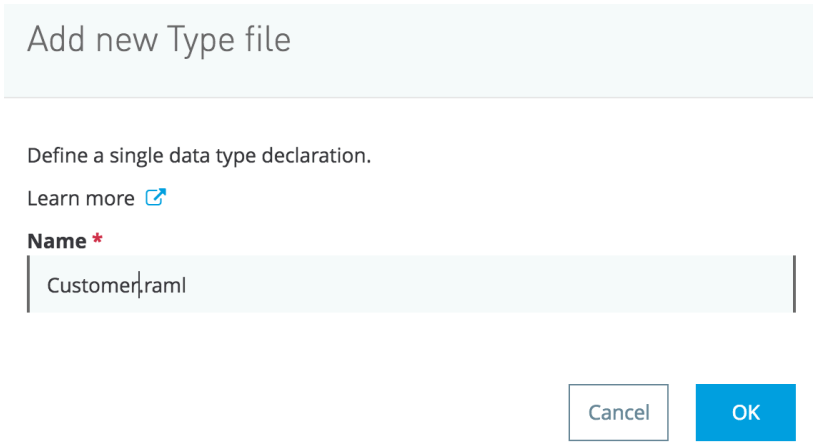
12. Save the file.

Create a Customer datatype fragment

13. In the file browser section, right-click the datatypes folder and select New File > RAML 1.0 > Type.

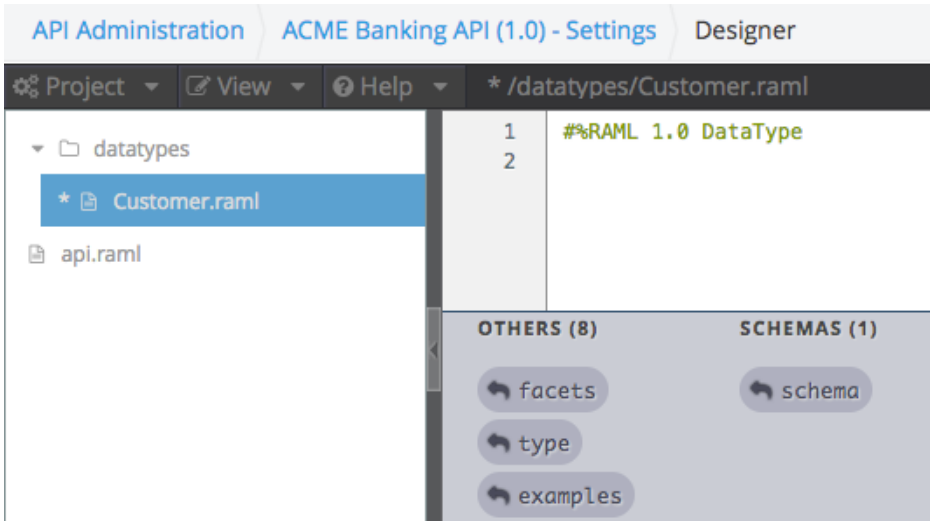


14. In the Add new type dialog box, set the name to Customer.raml.

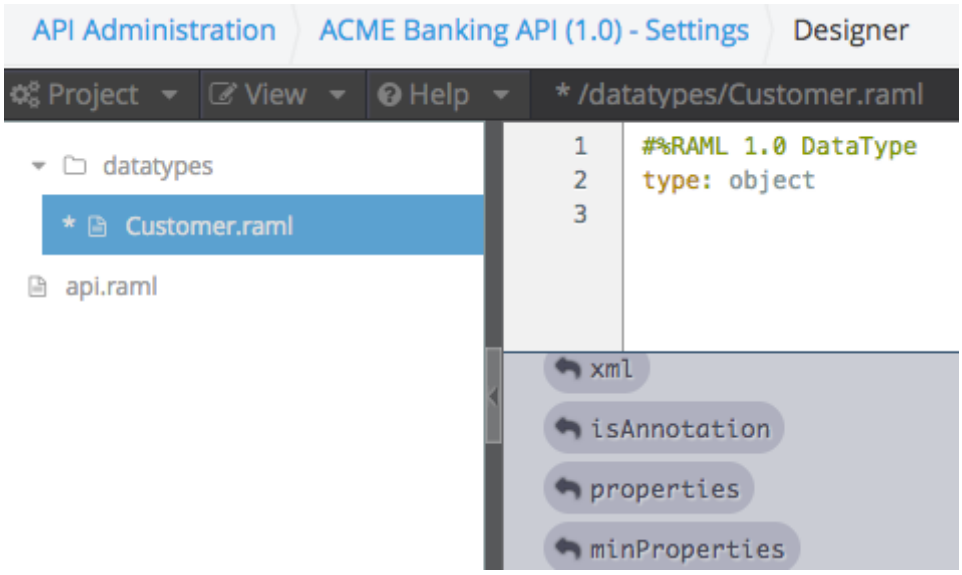


15. Click OK.
16. In the RAML editor, go to the end of the first line and press enter to add a new line.

17. In the shelf, click type.



18. In the shelf, click object and press enter.
19. In the shelf, click properties.



20. In your text editor, return to the WT 6-1 Datatypes and attributes file.
21. Review the attributes of the Customer datatype.
22. Return to API Designer.
23. Add the attributes for the Customer one below the other; for each, specify the attribute name, followed by a colon, and then the attribute type.

24. Add a ? after the prefix and suffix attribute names to specify them as optional attributes.

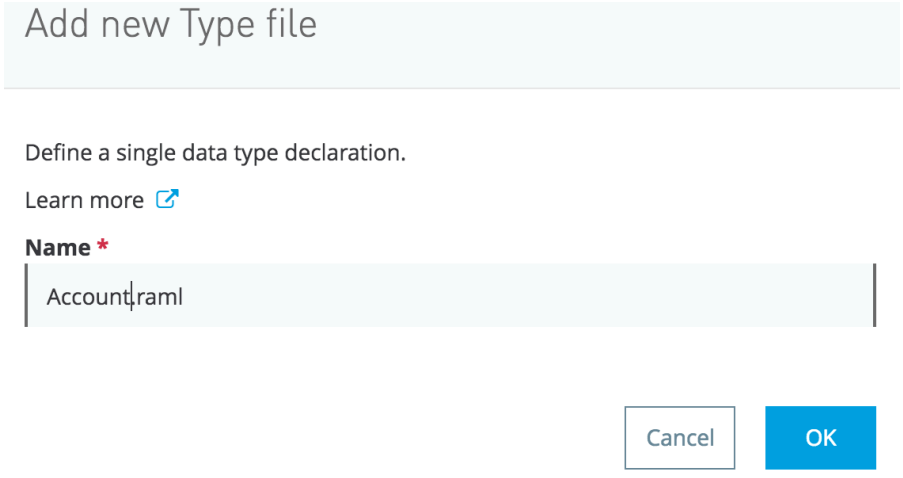
```
1  #%RAML 1.0 DataType
2  type: object
3  properties:
4    customerID: string
5    prefix?: string
6    firstName: string
7    lastName: string
8    suffix?: string
9    displayName: string
10   address: Address
11   phone: string
12   email: string
13   ssn: string
14   dateOfBirth: date-only
```

Note: The red cross in the address attribute line can be ignored. While linking the datatype fragments to the api.raml the errors won't persist in the main RAML definition or the API Console.

25. Save the project.

Create an Account datatype fragment

26. Similarly, add another new type file and set its name to Account.raml.



27. In the RAML editor, add required and optional attributes for the Account datatype.

```
#%RAML 1.0 DataType
type: object
properties:
  accountID: string
  accountType:
    enum: [Checking, Savings, Overdraft Savings, Credit Card]
  accountNumber: string
  accountOwner: AccountOwner[]
  accountBalance: Money
  IBAN: string
  bank: Bank
  interestRate?: number
  createdAt: datetime
  modifiedAt?: datetime
```

Note: Ignore the red errors in the three lines that inherit types from user-defined attributes that are yet to be defined.

28. Save the project.

Create a Transaction datatype fragment

29. Similarly, create another new Type file called Transaction.raml and type the required and optional attributes for the Transaction datatype.
30. Use the enum datatype to specify the transactionType has possible values of atm, deposit, cashWithdrawal, sepa, and online.

Note: The enum datatype should be defined in a line below the attribute as a key-value pair.

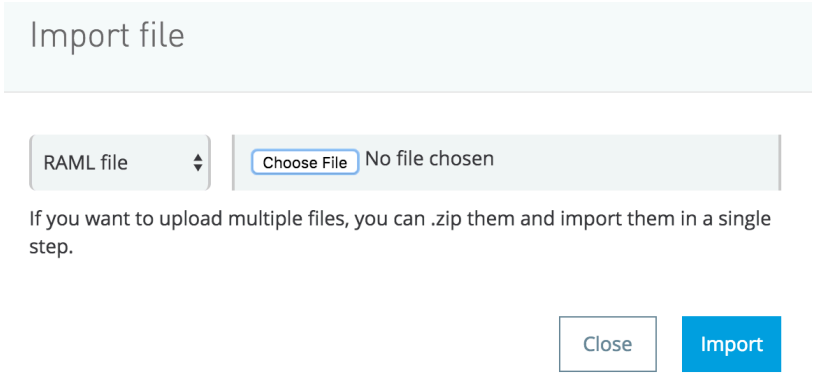
```
#%RAML 1.0 DataType
type: object
properties:
  transactionID: string
  fromAccount: Account
  toAccount: Account
  transactionType:
    enum: [atm, deposit, cashWithdrawal, sepa, online]
  transactionName?: string
  transactionAmount: Money
  newAccountBalance: Money
  postedAt: datetime
  completedAt?: datetime
```

Note: Ignore the red errors in some of the user-defined datatypes reference. Once the files are referenced in the main RAML file, the errors won't affect the API design specification.

31. Save the project.

Import other datatype fragment files into API Designer

32. In API Designer, click the Project menu and click Import.
33. In the Import file dialog box, click the Choose File button.

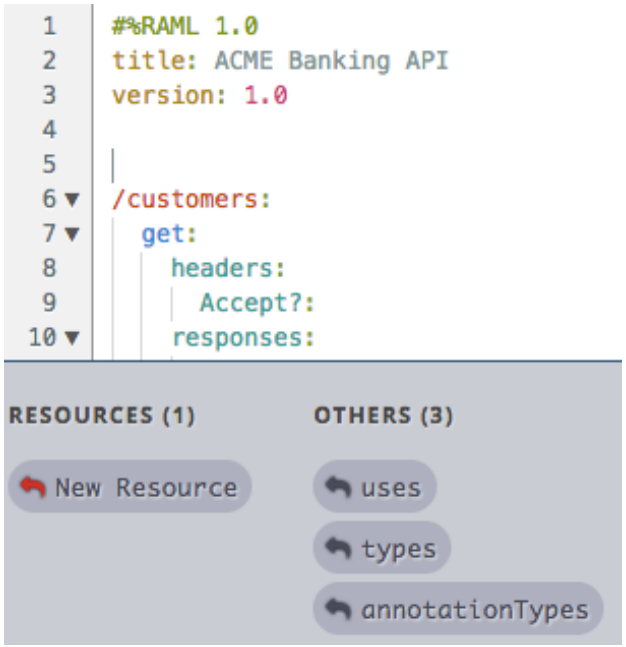


34. In the File browser, locate the studentFiles directory, open the module08 folder, and double-click Address.raml.
35. Click Import.
36. In the left-side navigation, click Address.raml and drag-and-drop it inside the datatypes folder.
37. Similarly, import Money.raml, AccountOwner.raml and Bank.raml and drag them into the datatypes directory.
38. Verify the project structure has the datatype files inside the datatypes folder, the documentation files in the documentation folder, and the api.raml file in the root location.

Include the datatype fragments in the main RAML API definition

39. In the file browser section, select api.raml.
40. In the RAML editor, go to the end of the documentation line that includes the acmeBankDoc.raml documentation file and press enter to add a new line below it.
41. Press enter and place the cursor at the beginning of the line.

42. In the shelf, scroll down and click types.

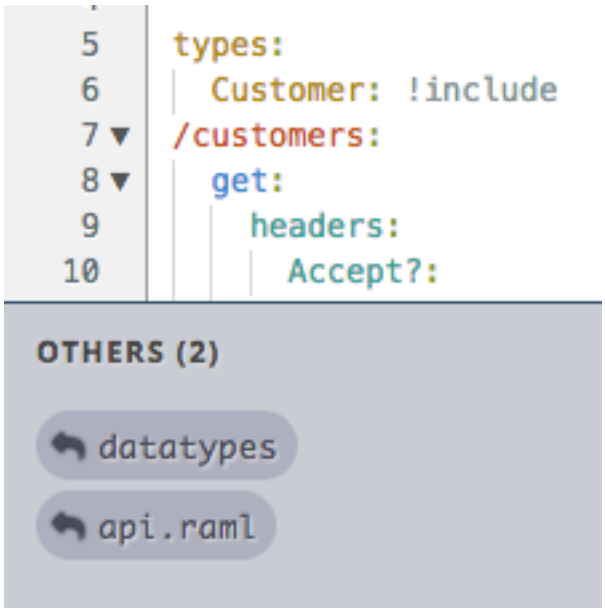


43. In the new line type,

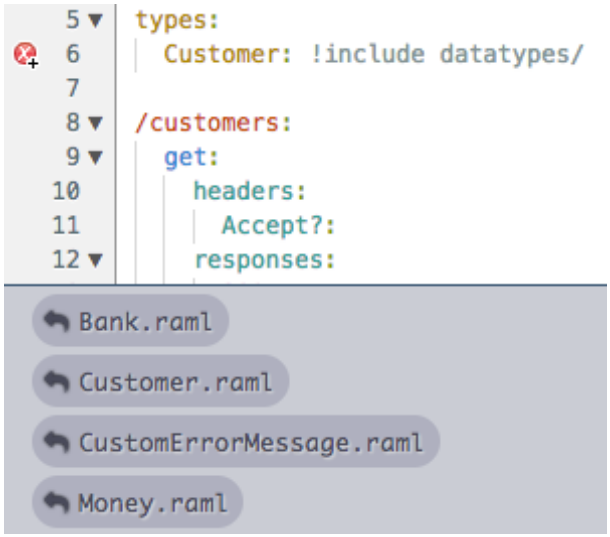
    Customer: !i

    Note: Make sure there is a space between the colon after Customer and the !.

44. In the shelf, click include.
45. Press spacebar and in the shelf, click datatypes.



46. Type / and in the shelf, click Customer.raml.



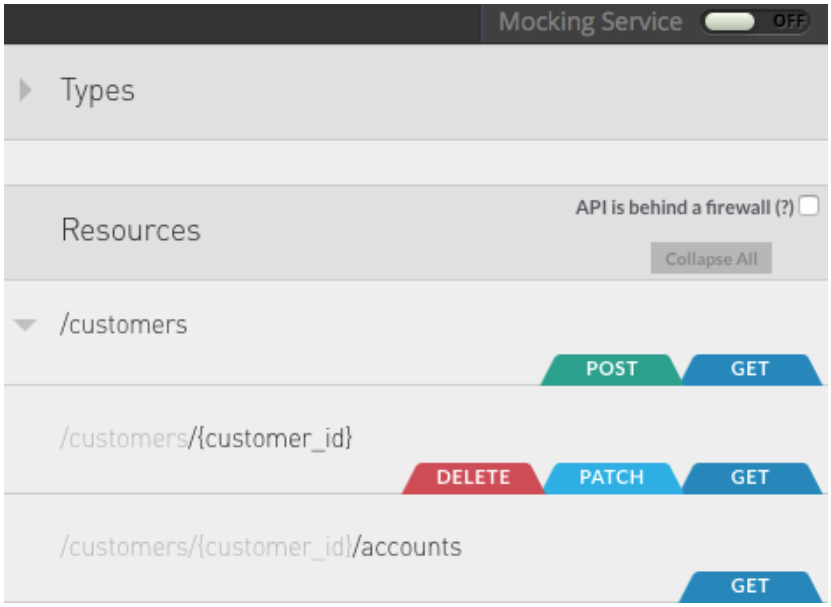
47. Similarly, include rest of the datatypes in the api.raml file.
48. Press enter to add an empty line in between the types and the /customers resource.

```
#%RAML 1.0
title: ACME Banking API
version: 1.0

types:
  Customer: !include datatypes/Customer.raml
  Account: !include datatypes/Account.raml
  Transaction: !include datatypes/Transaction.raml
  Address: !include datatypes/Address.raml
  AccountOwner: !include datatypes/AccountOwner.raml
  Money: !include datatypes/Money.raml
  Bank: !include datatypes/Bank.raml
  CustomErrorMessage: !include datatypes/CustomErrorMessage.raml

/customers:
```

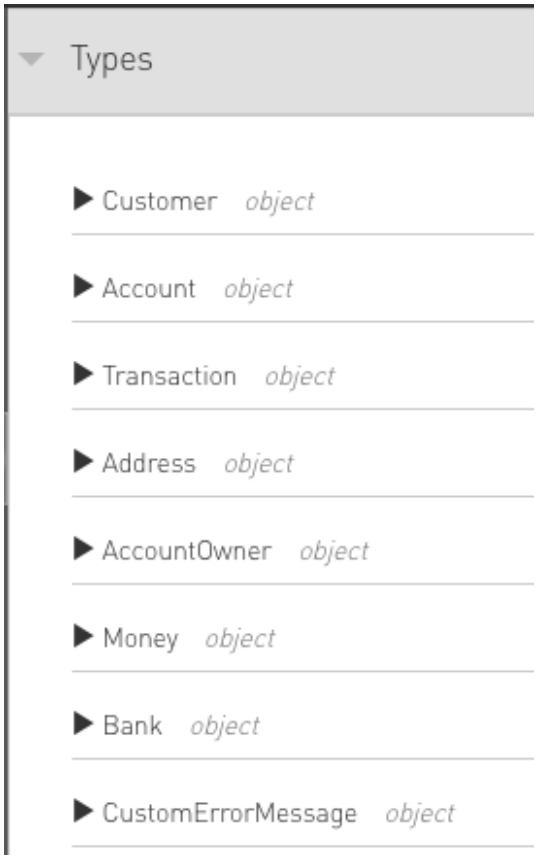
49. Save the project.

View the documentation

50. In the API Console, click the arrow in the Types section to expand it.



51. Verify that you see all the datatypes listed.

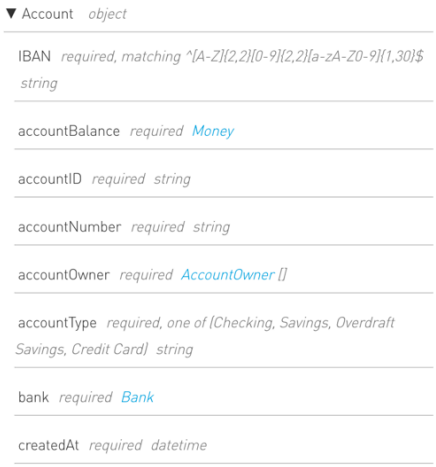


Walkthrough 6-3: Validate datatype attribute values using patterns

In this walkthrough, you add patterns and format values to validate necessary datatype attributes. You will:

- Specify attribute patterns and format values for certain datatype attributes.

```
1  #%RAML 1.0 DataType
2  type: object
3  properties:
4    currency:
5      type: string
6      pattern: ^[A-Z]{3,3}$
7    amount:
8      type: string
9      pattern: ^[+|-]?\d*\.\d{2}$
```



Specify attribute patterns and format values for necessary datatype attributes

1. Return to API Designer.
2. In the file browser section, expand the datatypes folder and select Account.raml.
3. Go to the line that contains the IBAN attribute and delete the type string from the line.
4. Press enter and press tab.

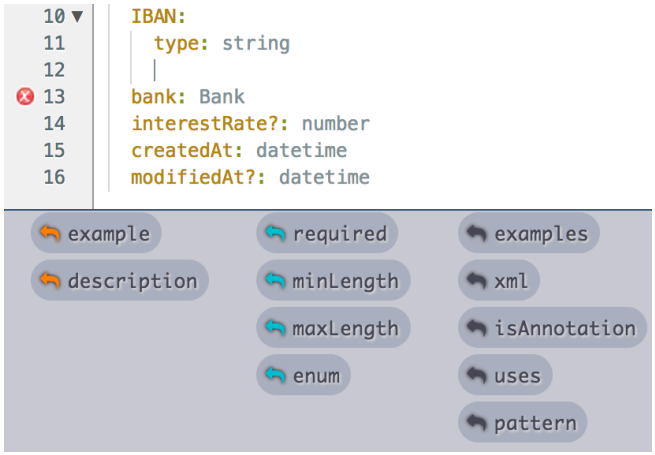
```
accountNumber: string
accountOwner: AccountOwner[]
accountBalance: Money
IBAN:
  |
bank: Bank
interestRate?: number
createdAt: datetime
modifiedAt?: datetime
```

5. Type:

   type: string

   Note: Make sure you add a space between the colon after type and the value string.

6. In the shelf, click pattern.

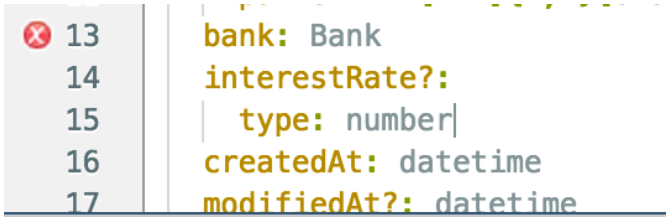


7. Return to or open the course snippets.txt file.
8. Locate the Module 8 section and copy the pattern for IBAN.
9. Return to API Designer and paste the text in the value of the pattern node.

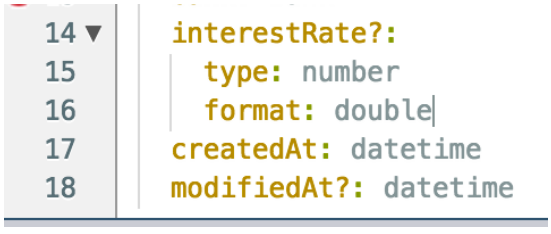
```
IBAN:
  type: string
  pattern: ^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}$
```

10. Go to the line that contains the interestRate attribute and delete number attribute type from the line.
11. Press enter and press tab.
12. Set the type to number and press enter.

```
type: number
```



13. In the shelf, click format and specify the format value as double.

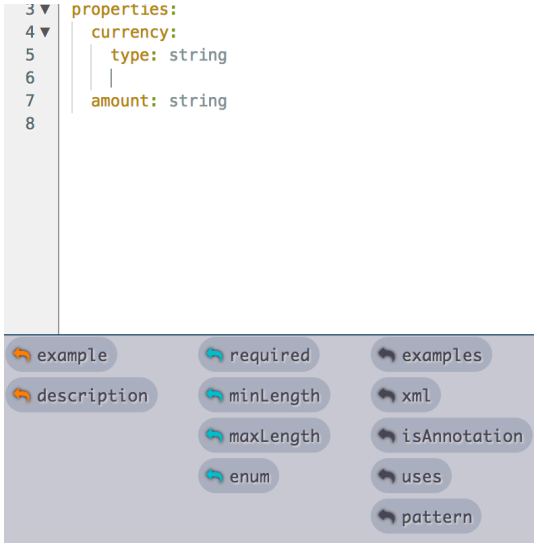


14. Save the project.

15. In the file browser section, select Money.raml inside the datatypes folder.
16. In the editor, go to the line that contains currency attribute and delete the attribute type string from the line.
17. Press enter and press tab.
18. Set the type to string and press enter.

```
type: string
```

19. In the shelf, click pattern.



20. Return to the course snippets.txt file and copy the pattern value for currency and paste it in the editor.

```
properties:
  currency:
    type: string
    pattern: ^[A-Z]{3,3}$
  amount: string
```

21. Go to the line that contains amount attribute and delete the attribute type string from the line.
22. Press enter and press tab.
23. Set the type to string and press enter.

```
type: string
```

24. In the shelf, click pattern.

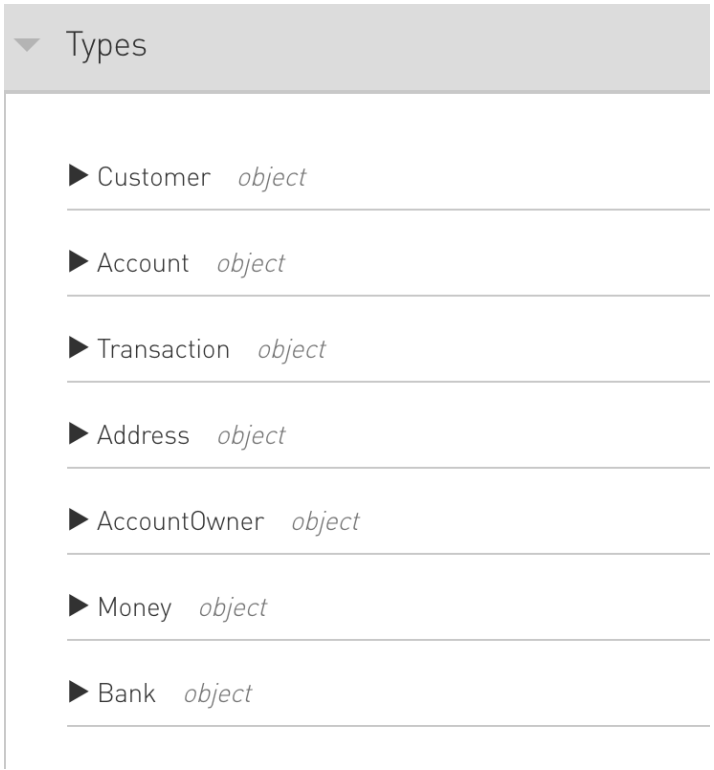
25. Return to the course snippets.txt file and copy the line that contains the pattern value for amount and paste it in the RAML editor.

```
#%RAML 1.0 DataType
type: object
properties:
  currency:
    type: string
    pattern: ^[A-Z]{3,3}$
  amount:
    type: string
    pattern: ^[+|-]?\d*\.\d{2}$
```

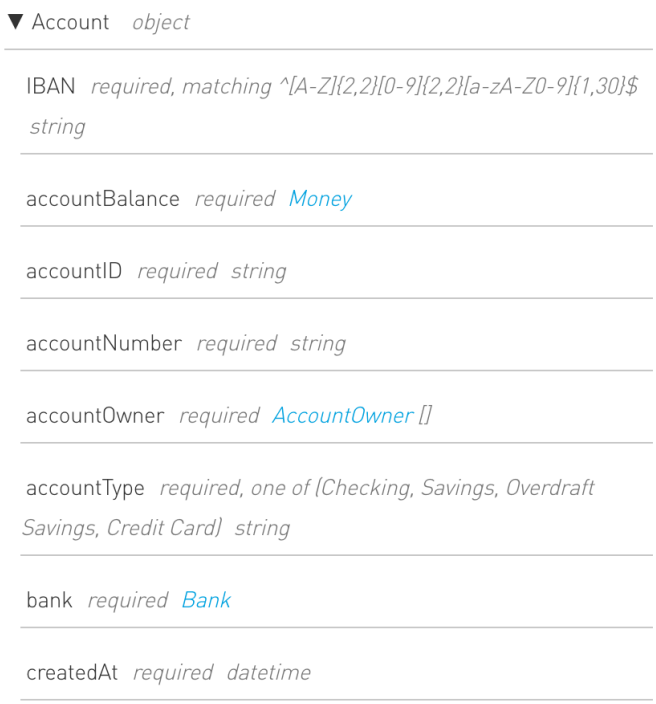
26. Save the project.

View the API documentation

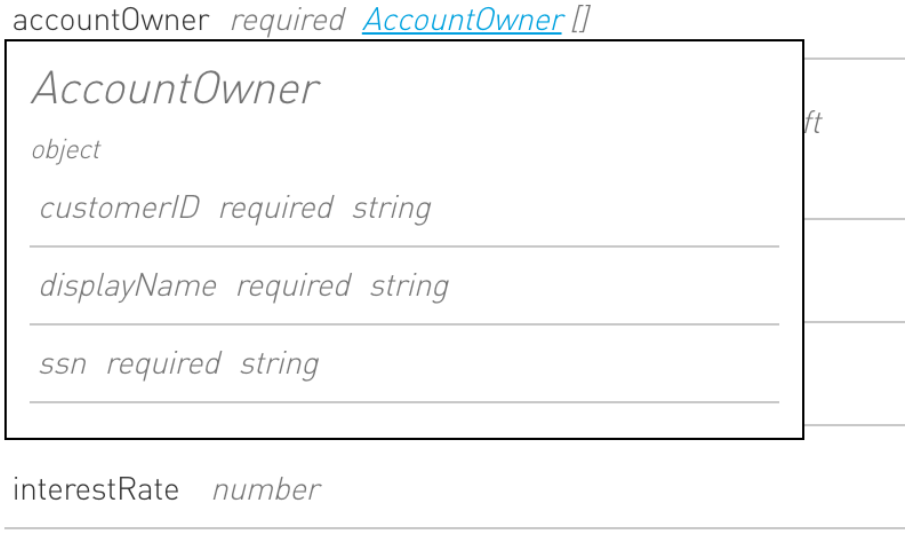
27. In the API Console, expand the Types section; you should see all the datatypes listed.



28. Expand the Account datatype and see the attributes of the Account datatype listed.



29. Click the blue AccountOwner datatype link in the accountOwner attribute and verify that you can see the attributes of the AccountOwner datatype in a pop-up dialog box.



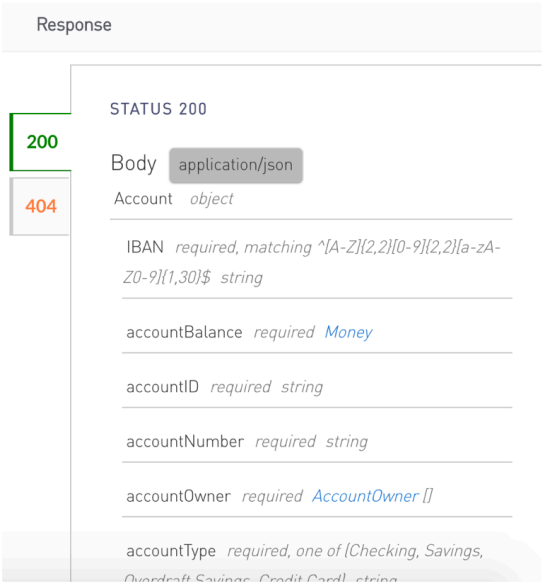
30. Click the Types section to collapse the list of datatypes.

Walkthrough 6-4: Specify datatypes in resource methods

In this walkthrough, you specify the type of data to be sent in the request or received from the response of a resource method. You will:

- Add the type parameter with a reference to the datatype for all resource method request and response bodies.

```
181 ▼    put:
182        description: Update a specific account information
183 ▼      body:
184          application/json:
185            type: Account
186 ▼      responses:
187 ▼        201:
188            body:
189              application/json:
190          204:
```



Add type parameters for the appropriate method requests and responses

1. Return to API Designer.
2. In the file browser section, click api.raml.
3. Locate the /customers resource and in the get method HTTP 200 response body, add a new line below the response body type application/json.
4. Press tab and in the shelf, click type.

```
42 ▼          body:
43              application/json:
44
45              application/xml:
46 ▼        404:
47 ▼          body:
48 ▼            application/json:
```

displayName default facets
example type
description examples

5. In the shelf, click Customer and add [] (square brackets) to denote an array of Customer objects as the response body.

```
40                  example: Tue, 18 Ap
41                  format: rfc2616
42 ▼            body:
43                application/json:
44                  type: Customer[]
45                application/xml:
```

6. Copy the line type: Customer[] and go to the end of the next line that contains the application/xml response body type.
7. Press enter and press tab.
8. Paste the copied line.

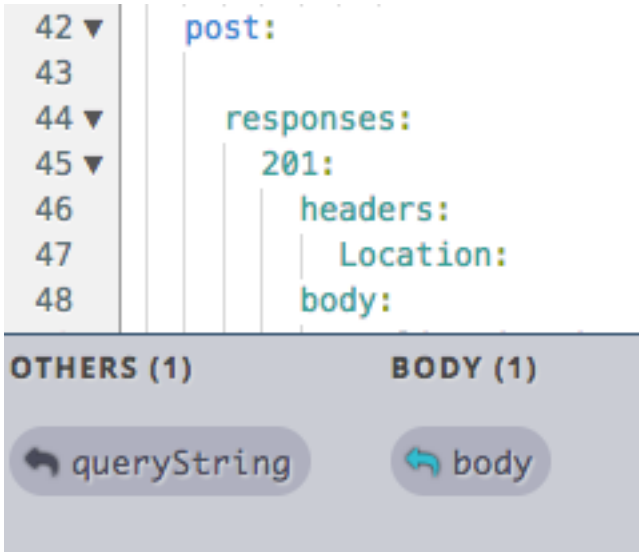
```
42 ▼            body:
43                application/json:
44                  type: Customer[]
45                application/xml:
46                  type: Customer[]
47 ▼          404:
```

9. Similarly, in the /accounts nested resource get method and the /transactions resource get method, add the respective Account and Transaction array datatypes in the HTTP 200 response body.

```
118 ▼          responses:
119 ▼            200:
120 ▼              body:
121                  application/json:
122                    type: Account[]
123                  application/xml:
124                    type: Account[]
125 ▼            404:
197 ▼      /transactions:
198 ▼        get:
199            description: Retrieve a list of tra
               account
200 ▼          responses:
201 ▼            200:
202 ▼              body:
203                  application/json:
204                    type: Transaction[]
```

10. In the /customers resource post method, add a new line below the post method node.

11. In the shelf, scroll down and click body.



12. In the shelf, click application/json.
13. In the shelf, click type.
14. In the shelf, click Customer.

15. Add the request body for the /accounts post method, /{account_id} put method and the /transactions post method with the respective datatypes in the request.

```
124
125 ▼ /accounts:
126 ▼   post:
127 ▼     body:
128         application/json:
129           type: Account
130 ▼     responses:
```

```
176 ▼     put:
177 ▼       body:
178           application/json:
179             type: Account
180 ▼       responses:
```

```
217 ▼ /transactions:
218 ▼   post:
219 ▼     body:
220         application/json:
221           type: Transaction
222 ▼     responses:
223 ▼       201:
```

16. Go to the /{customer_id} nested resources get method.
17. Add a new line below the HTTP 200 response body type application/json.
18. Press tab and type:

    type: Customer

19. Add a new line below the application/xml body type and type:

    type: Customer

```
61 ▼     get:
62         headers:
63           Accept?:
64 ▼       responses:
65 ▼         200:
66 ▼           body:
67               application/json:
68                 type: Customer
69               application/xml:
70                 type: Customer
```

20. Add the type node for the /{account_id} and /{transaction_id} get method with the value as the respective datatypes.

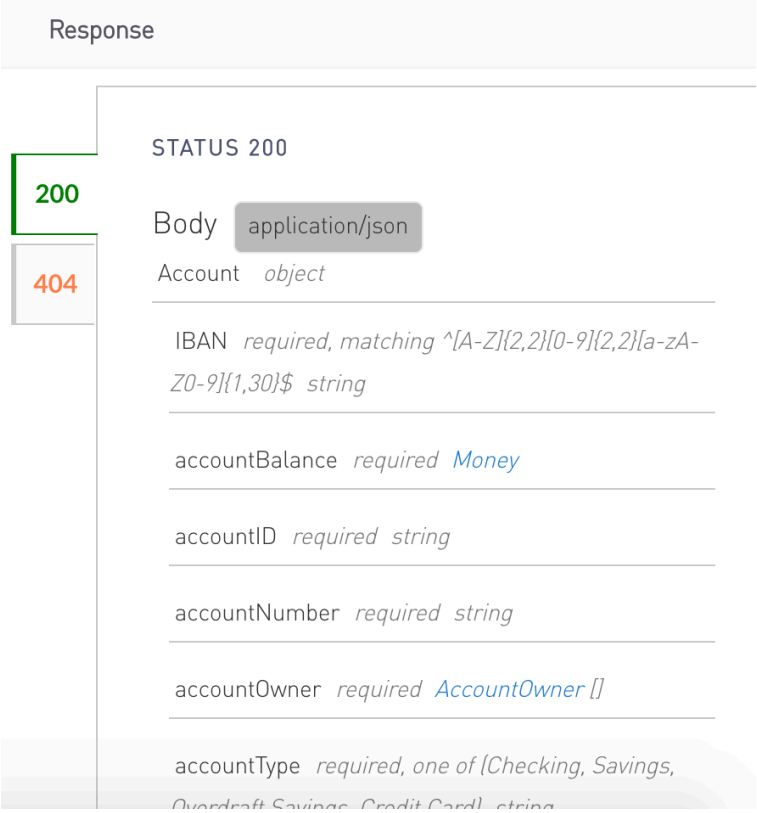
```
/{account_id}:
  uriParameters:
    account_id:
  get:
    headers:
      Accept?:
    responses:
      200:
        body:
          application/json:
            type: Account
          application/xml:
            type: Account
      404:
/{transaction_id}:
  uriParameters:
    transaction_id:
  get:
    headers:
      Accept?:
    responses:
      200:
        body:
          application/json:
            type: Transaction
          application/xml:
            type: Transaction
      404:
```

21. Save the project.

View the documentation

22. In the API Console, click the GET tab for the /accounts/{account_id} resource.

23. Verify that you can see the Account object listed in the 200 Response section of the resource.



24. Click CLOSE X.

Reference the custom error message datatype in the necessary resource method responses

25. Go to the /customers resource get method HTTP 404 response body and delete the lines that contain the properties.
26. Set the datatype to CustomErrorMessage.

```
type: CustomErrorMessage
```

27. Modify all the other error response bodies in the RAML definition to be of this datatype.

```
30 ▼    404:
31 ▼      body:
32          application/json:
33            type: CustomErrorMessage
34 ▼    406:
35 ▼      body:
36          application/json:
37            type: CustomErrorMessage
```

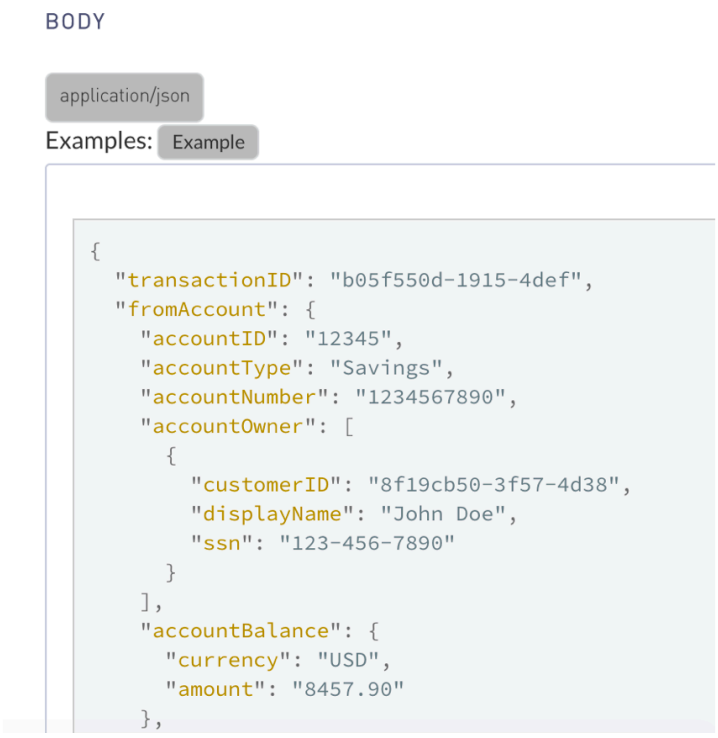
28. Save the project.

Walkthrough 6-5: Define example fragments for datatypes

In this walkthrough, you create example fragments for each of the datatypes. You will:

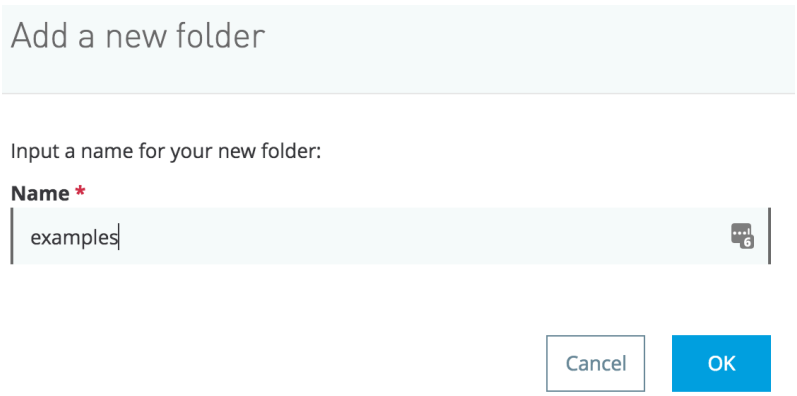
- Create example fragments for the datatypes.
- Include example fragments in request and response bodies.

```
#%RAML 1.0 NamedExample
accountID: '12345'
accountType: Savings
accountNumber: '1234567890'
accountOwner:
  - #item 1
    customerID: 8f19cb50-3f57-4d38,
    displayName: John Doe,
    ssn: 123-456-7890,
accountBalance:
  currency: USD
  amount: '8457.90'
IBAN: GB29NWBK60161331926820
bank:
  bankCode: NWBKGB2L
  bankName: ACME Bank
  routingNumber: '432159876'
createdAt: 2012-03-07T00:00:00.001Z
```

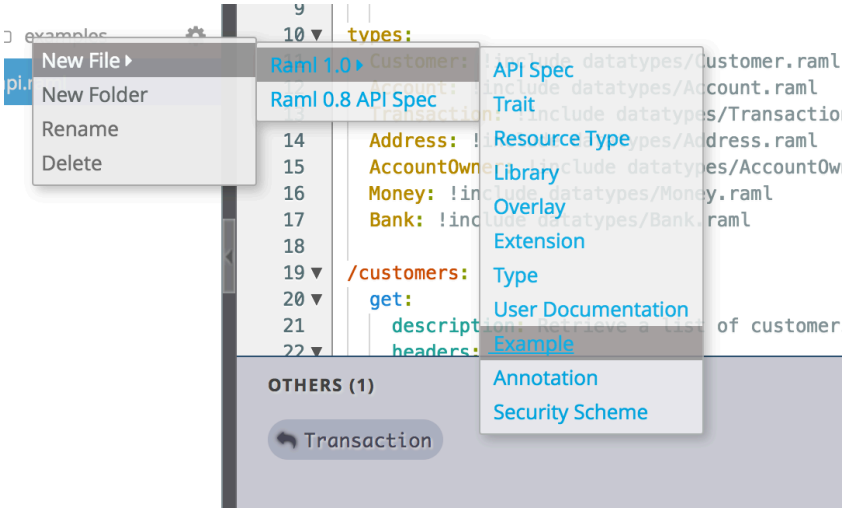


Create example fragment for the Customer datatype

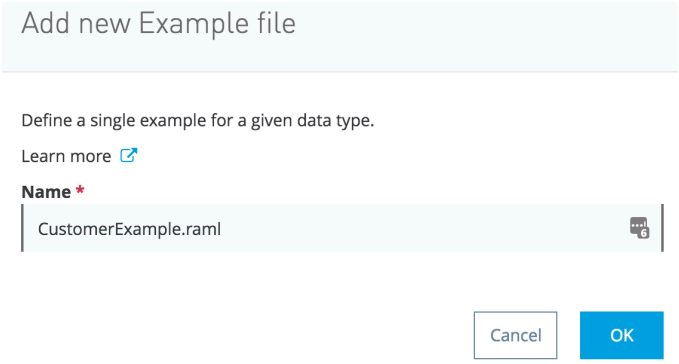
1. Return to API Designer.
2. Click the Project menu and select New Folder.
3. In the Add a new folder dialog box, set the name of the folder to examples.



4. Click OK.
5. Right-click the examples folder in the file browser section and select New File > RAML 1.0 > Example.



6. In the Add new Example file dialog box, set the name to CustomerExample.raml and click OK.



7. In the CustomerExample.raml file, delete the line that contains the value node.

8. Return to the course snippets.txt file and copy the snippet for the CustomerExample.raml content values and paste it in the second line.

```
#%RAML 1.0 NamedExample
customerID: 8f19cb50-3f57-4d38
firstName: John
lastName: Doe
displayName: John Doe
address:
  addressLine1: 1234 Lane
  addressLine2: Apt.#620
  city: San Francisco
  state: California
  zipCode: 94108
phone: 415-000-0000
email: johndoe@example.com
ssn: 123-456-7890
dateOfBirth: 1983-01-01
```

9. Save the project.

Create example fragment for the Account datatype

10. Create a new example file AccountExample.raml and copy the example data from the course snippets.txt file.

```
#%RAML 1.0 NamedExample
accountID: 12345
accountType: Savings
accountNumber: 1234567890
accountOwner:
  customerID: 8f19cb50-3f57-4d38
  displayName: John Doe
  ssn: 123-456-7890
accountBalance:
  currency: USD
  amount: 8457.90
IBAN: B29NWBK60161331926820
bank:
  bankCode: NWBKGB2L
  bankName: ACME Bank
  routingNumber: 432159876
createdAt: 2012-03-07T00:00:00.001Z
```

11. Save the project.

Create example fragment for the Transaction datatype

12. Create a new example file called TransactionExample.raml and copy the example data from the course snippets.txt file.

```
#%RAML 1.0 NamedExample
transactionID:
fromAccount:
  accountID: 12345
  accountType: Savings
  accountNumber: 1234567890
  accountOwner:
    customerID: 8f19cb50-3f57-4d38
    displayName: John Doe
    ssn: 123-456-7890
  accountBalance:
    currency: USD
    amount: 8457.90
  IBAN: B29NWBK60161331926820
  bank:
    bankCode: NWBKGB2L
    bankName: ACME Bank
    routingNumber: 432159876
  createdAt: 2012-03-07T00:00:00.001Z
toAccount:
  accountID: 56437
  accountType: Credit Card
  accountNumber: 4321987650
  accountOwner:
    customerID: 8f19cb50-3f57-4d38
    displayName: John Doe
    ssn: 123-456-7890
  IBAN: B29NWBK60161331926820
  bank:
    bankCode: NWBKGB2L
    bankName: ACME Bank
    routingNumber: 432159876
  createdAt: 2012-03-07T00:00:00.001Z
transactionType: onlineTransfer
transactionAmount:
  currency: USD
  amount: 70.00
newAccountBalance:
  currency: USD
  amount: 8387.90
```

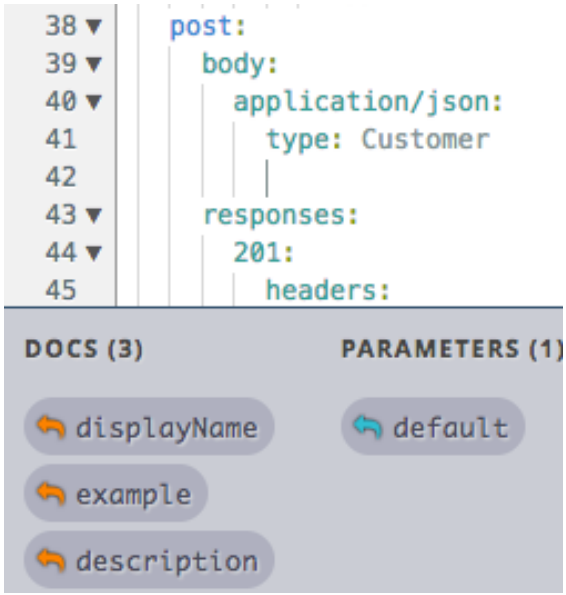
13. Save the project.

Include example fragments in request and response bodies

14. In the file browser section, select api.raml.
15. In the /customers resource post method, add a new line below the line that contains the HTTP request body Customer datatype.

```
38 ▼   post:
39 ▼     body:
40 ▼       application/json:
41           type: Customer
42
43 ▼     responses:
```

16. Press enter and in the shelf, click example.



17. In the example node, type the value as:

```
!include examples/CustomerExample.raml
```

```
38 ▼   post:
39 ▼     body:
40 ▼       application/json:
41           type: Customer
42           example: !include examples/CustomerExample.raml
43 ▼     responses:
```

18. Copy the example node line.
19. In the /{customer_id} get method response body, add a new line below the line that contains the type node Customer.

20. Paste the copied line for both media types (application/json and application/xml).

```
/{customer_id}:
  uriParameters:
    customer_id:
  get:
    headers:
      Accept?:
    responses:
      200:
        body:
          application/json:
            type: Customer
            example: !include examples/CustomerExample.raml
          application/xml:
            type: Customer
            example: !include examples/CustomerExample.raml
      404:
```

21. Similarly, add the AccountExample.raml file example node in the /accounts resource post request body as well as in /{account_id} nested resource get and put method response body.

```
/{account_id}:
  uriParameters:
    account_id:
  get:
    headers:
      Accept?:
    responses:
      200:
        body:
          application/json:
            type: Account
            example: !include examples/AccountExample.raml
          application/xml:
            type: Account
            example: !include examples/AccountExample.raml
      404:
```

```
  put:
    body:
      application/json:
        type: Account
        example: !include examples/AccountExample.raml
    responses:
      200:
```

22. Similarly, add the TransactionExample.raml file example node in the /transactions resource post request body as well as in /{transaction_id} nested resource get method response body.

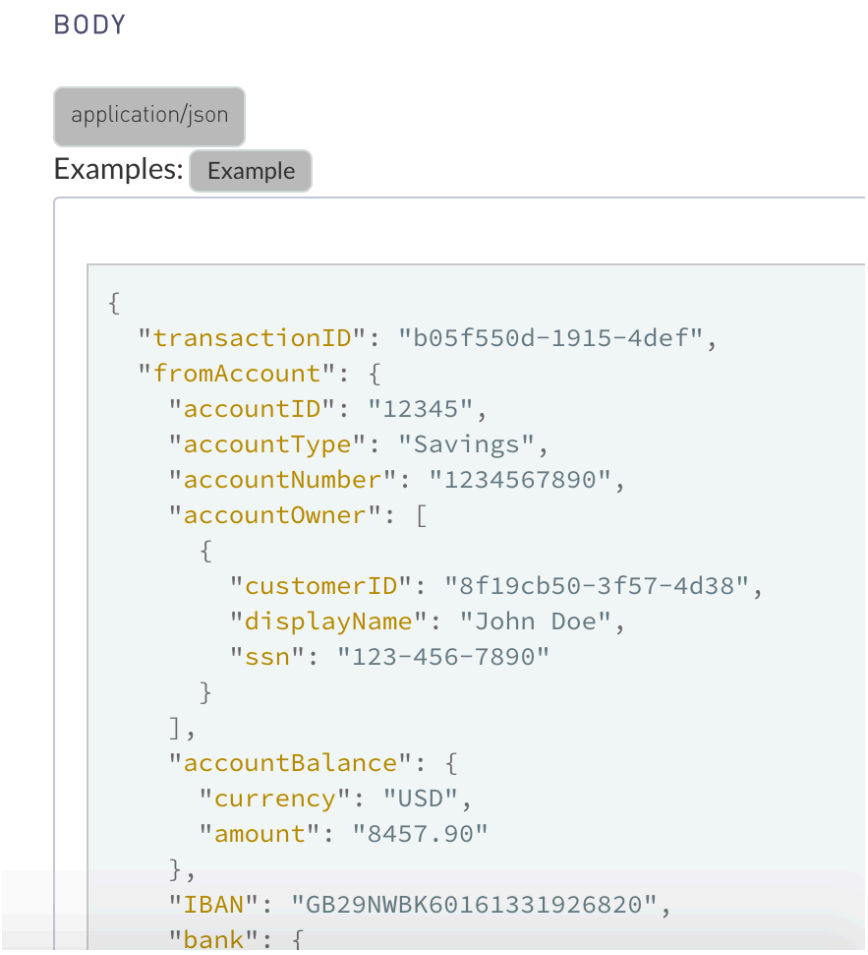
```
196 /transactions:
197   post:
198     body:
199       application/json:
200         type: Transaction
201         example: !include examples/TransactionExample.raml
202     responses:
```

```
212   /{transaction_id}:
213     uriParameters:
214       transaction_id:
215     get:
216       headers:
217         Accept?:
218       responses:
219         200:
220           body:
221             application/json:
222               type: Transaction
223               example: !include examples/TransactionExample.raml
224             application/xml:
225               type: Transaction
226               example: !include examples/TransactionExample.raml
227         404:
```

23. Save the project.

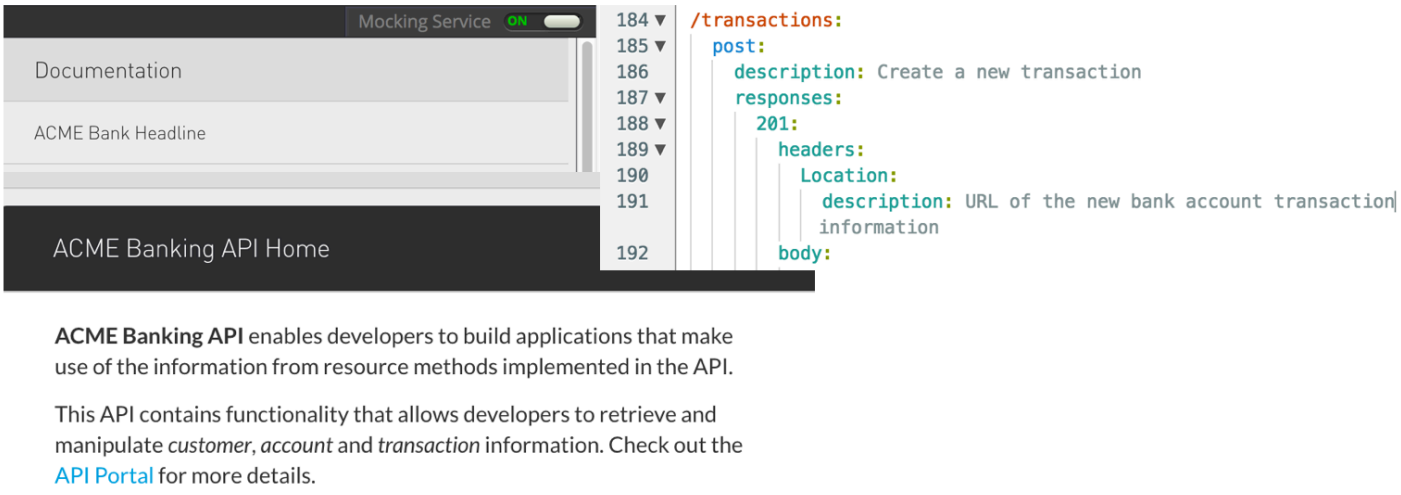
View the documentation

24. In the API Console, click the POST tab for the /transactions resource.
25. Verify that you see the TransactionExample in the request body of the method.



26. Scroll back up and click CLOSE X.

Module 7: Documenting and Testing APIs



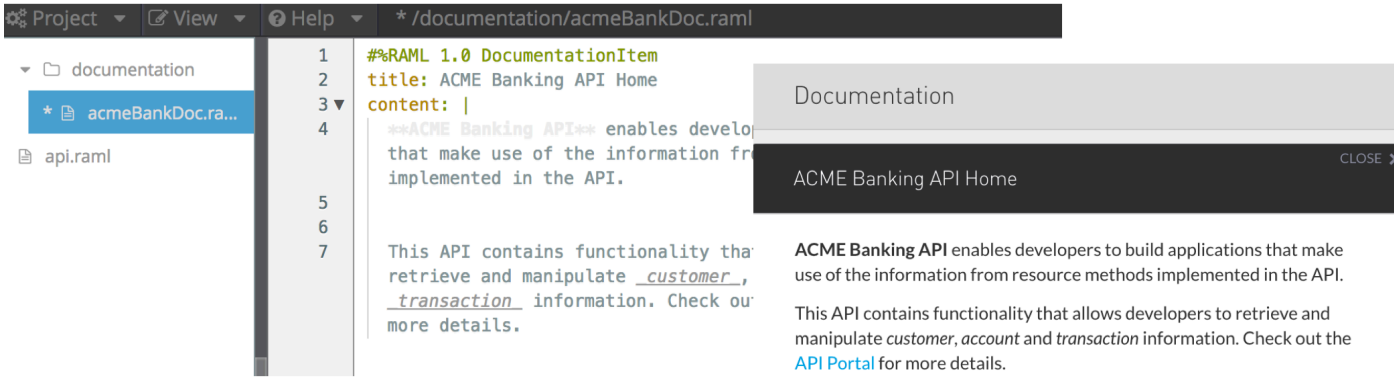
Objectives:

- Add documentation and description nodes to a RAML definitions.
- Use the mocking service to create API endpoints.
- Use the API Console to test API endpoints.

Walkthrough 7-1: Add a documentation fragment to a RAML API definition

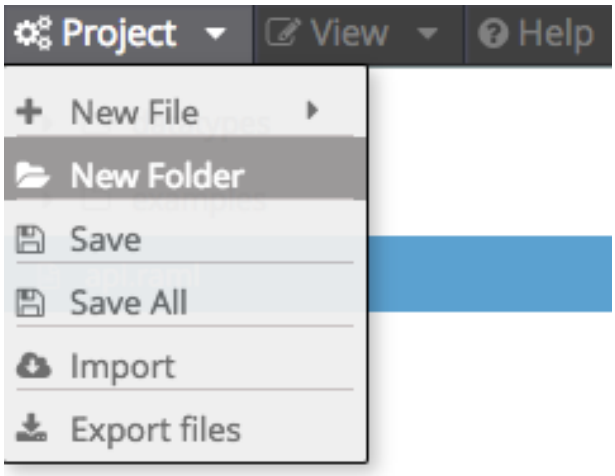
In this walkthrough, you add documentation for the ACME Banking API RAML definition. You will:

- Create a documentation fragment.
- Link the documentation fragment to the main RAML definition file.
- Link multiple documentation fragments to the documentation node in the RAML API definition.

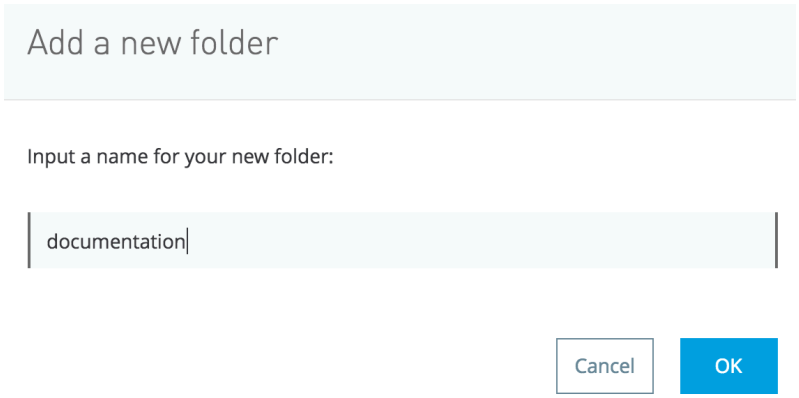


Create a documentation fragment and link it to the main RAML definition file

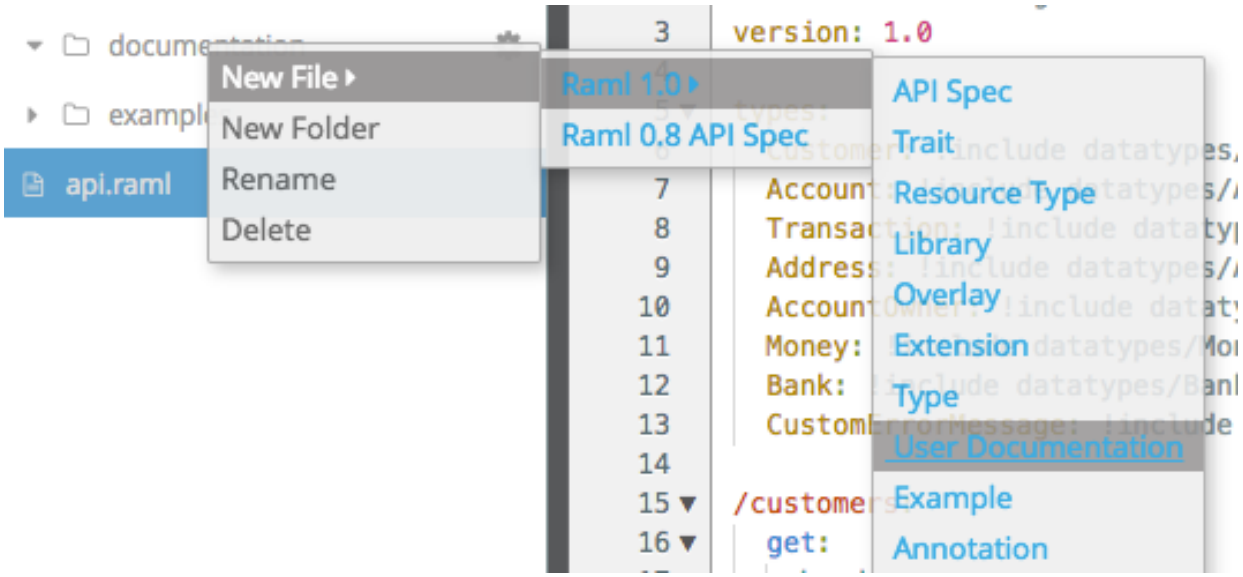
1. Return to API Designer.
2. Click the Project menu and select New Folder.



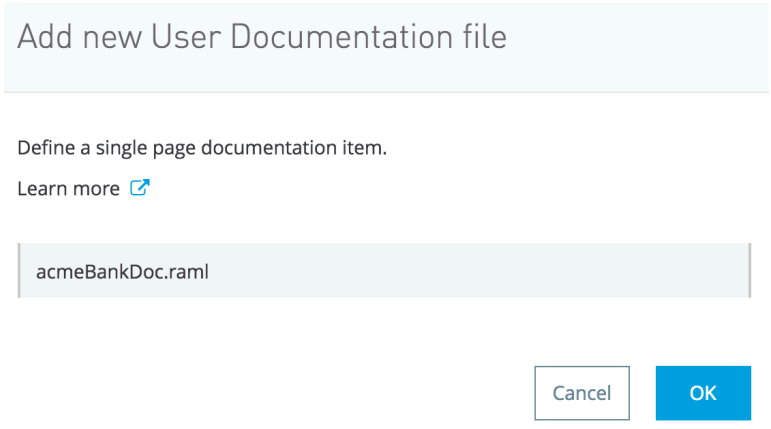
3. In the Add a new folder dialog box, set the folder name to documentation.



4. Click OK.
5. In the file browser section, right-click the documentation folder and select New File > RAML 1.0 > User Documentation.

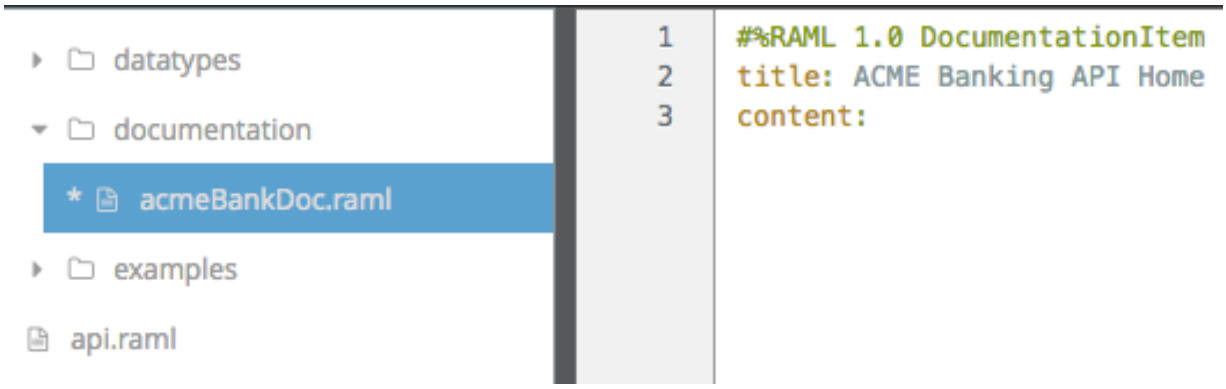


6. In the Add a new User Documentation dialog box, set the file name to acmeBankDoc.raml.



7. Click OK.
8. In the editor, set the title property to ACME Banking API Home.

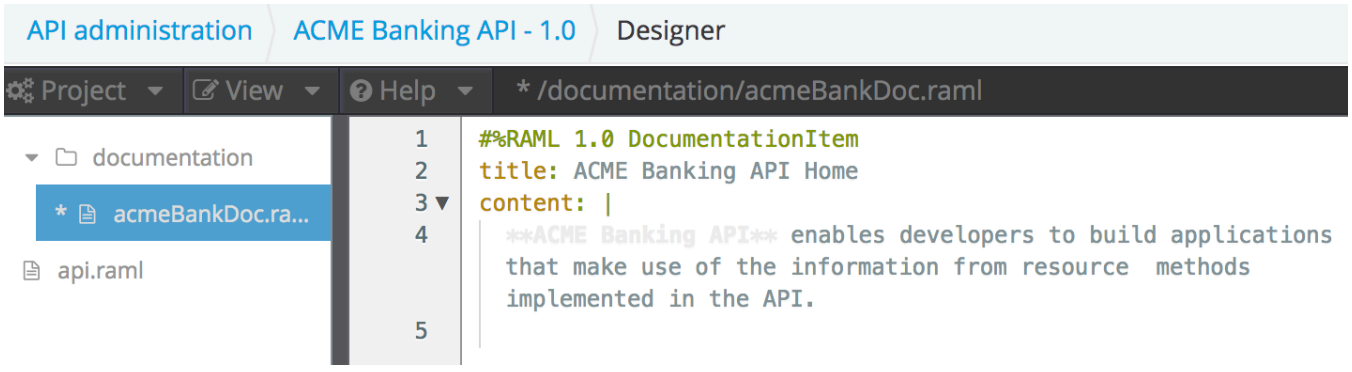
   Note: Be sure to put a space after the colon.



9. After the colon for the content property, press the spacebar and add a | (the pipe operator).
10. Return to the course snippets.txt file.
11. Locate Module 7 and copy the text under Snippet #1.

```
-------Module 7--------

Documentation for acmeBankDoc.raml file -

Snippet #1
**ACME Banking API** enables developers to build applications that make use of the information
from resource  methods implemented in the API.

Snippet #2
This API contains functionality that allows developers to retrieve and manipulate _customer_,
_account_ and _transaction_ information. Check out the [API Portal]() for more details.

Documentation for acmeBankHeadline.raml file -
```

12. Return to API Designer.
13. In acmeBankDoc.raml, on a new line after the content property, paste the text.

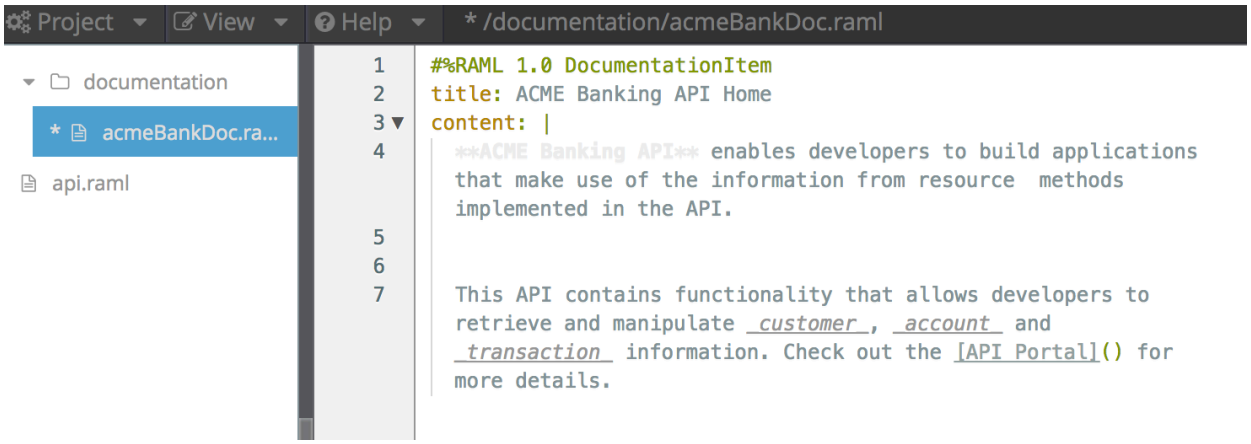


14. Press enter two times to add new empty lines.

15. Return to the course snippets.txt file and copy Snippet #2.

```
-------Module 7--------

Documentation for acmeBankDoc.raml file -

Snippet #1
**ACME Banking API** enables developers to build applications that make use of the information
from resource  methods implemented in the API.

Snippet #2
This API contains functionality that allows developers to retrieve and manipulate _customer_,
_account_ and _transaction_ information. Check out the [API Portal]() for more details.

Documentation for acmeBankHeadline.raml file -
```

16. Return to acmeBankDoc.raml in API Designer and paste the text.

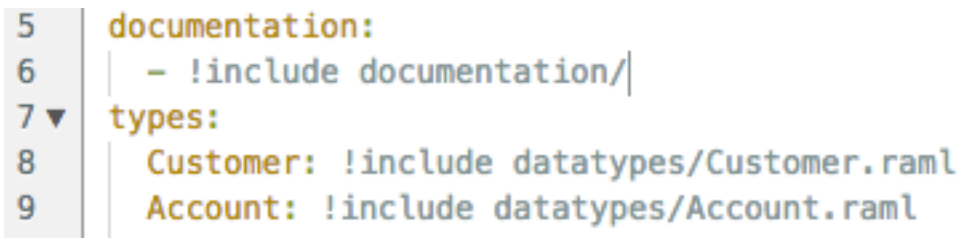


17. Save the project.
18. In the API Designer file browser, click api.raml.
19. In api.raml, go to the line of code before the types node and press enter to add a new line.
20. In the shelf, click documentation.
21. In the new line created below the documentation node, type the following:

```
- !include
```

Note: Be sure to put a space between the - and the !.

22. Stay on the same line and click documentation in the shelf.
23. After the word documentation word, type /.

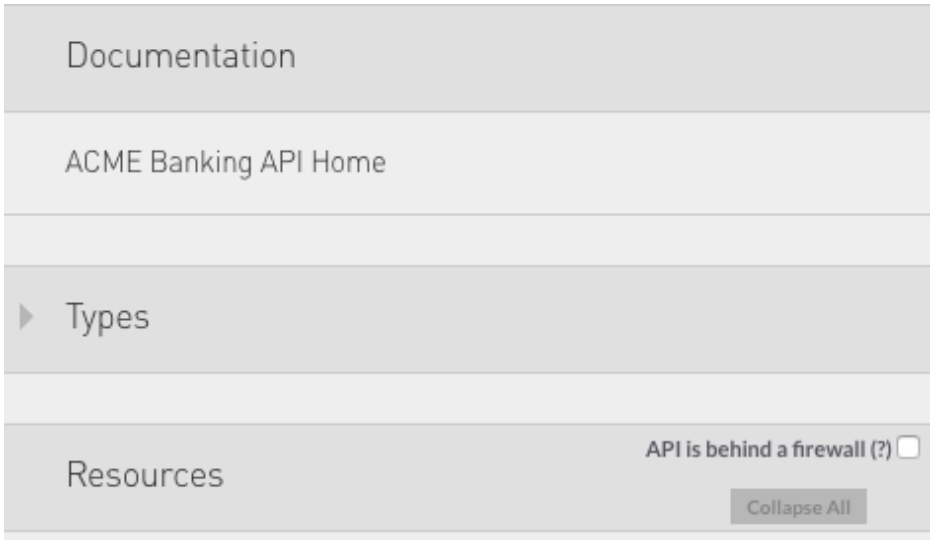


24. In the shelf, click acmeBankDoc.raml.
25. Add a new line.

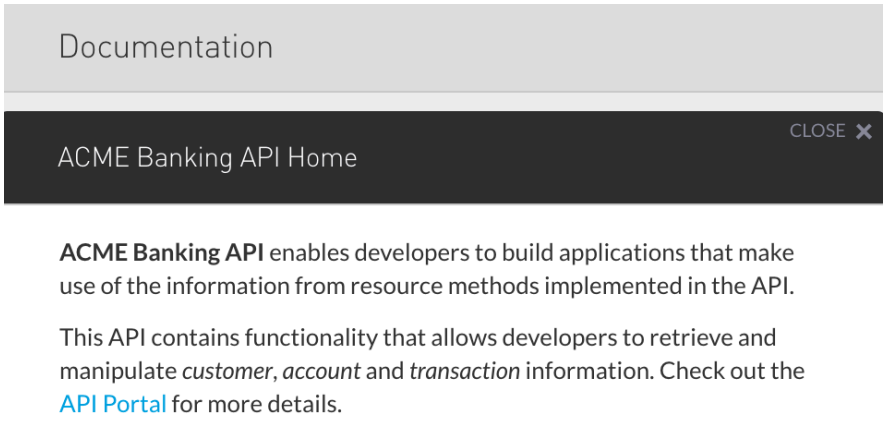
```
documentation:
  - !include documentation/acmeBankDoc.raml

types:
  Customer: !include datatypes/Customer.raml
  Account: !include datatypes/Account.raml
```

View the API documentation

26. In the API Console, locate the new Documentation section and click ACME Banking API Home.



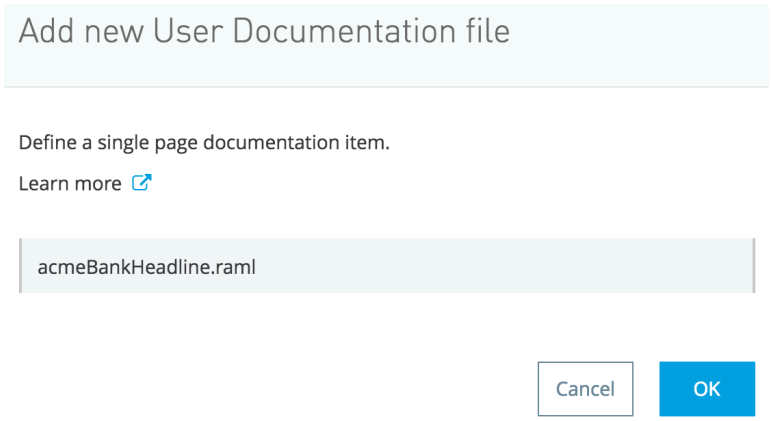
27. Browse the documentation content for the ACME Banking API Home in the console.



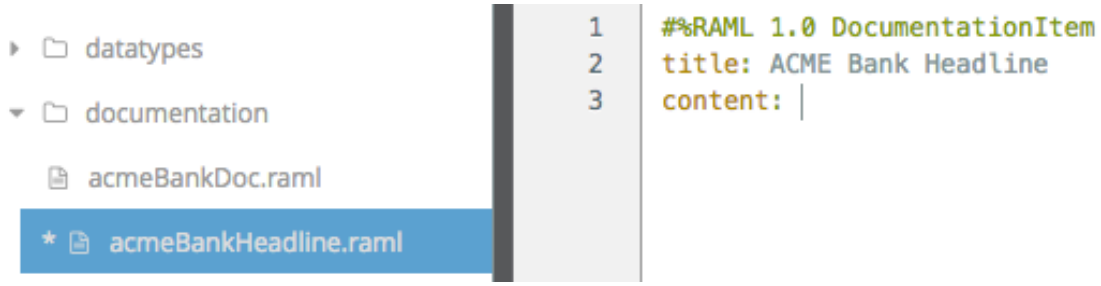
28. Click CLOSE X in the ACME Banking API Home header.

Link multiple documentation fragments to the documentation node

29. In the API Designer file browser, right-click the documentation folder and select New File > RAML 1.0 > User Documentation.
30. In the Add new User Documentation file dialog box, set the file name to acmeBankHeadline.raml.



31. Click OK.
32. In the editor, set the type property to ACME Bank Headline.
33. In the line that contains the content property, press spacebar after the colon and type | (pipe operator) and press enter.

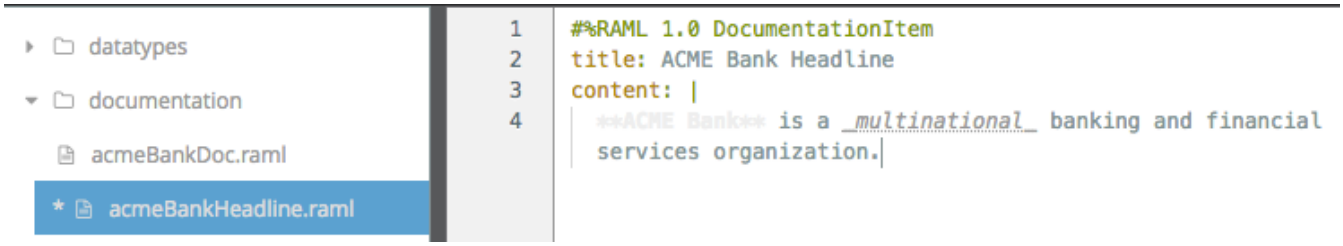


34. Return to the course snippets.txt file and copy Snippet #3.

```
Documentation for acmeBankHeadline.raml file -

Snippet #3
**ACME Bank** is a _multinational_ banking and financial services organization.
```

35. Return to acmeBankHeadline.raml and paste the text on the new line.
36. Save the project.

37. In the API Designer file browser, click api.raml to open the file in the editor.



38. At the end of the line that defines the documentation node, press enter.
39. Add code to include the acmeBankHeadline.raml file:

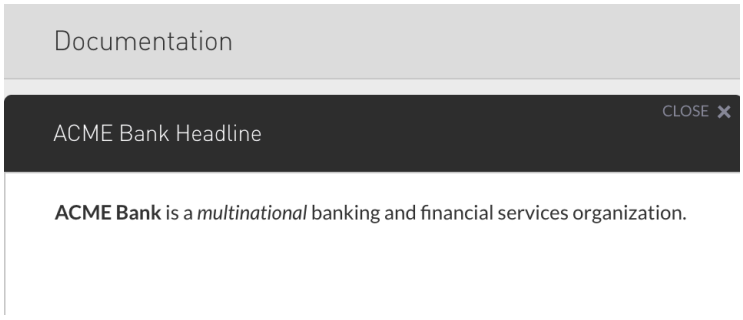
```
- !include documentation/acmeBankHeadline.raml
```

```
documentation:
  - !include documentation/acmeBankHeadline.raml
  - !include documentation/acmeBankDoc.raml

types:
```

40. Save the project.

View the API documentation

41. In the Documentation section of API Console, click ACME Bank Headline.
42. View the documentation content in the Console window.

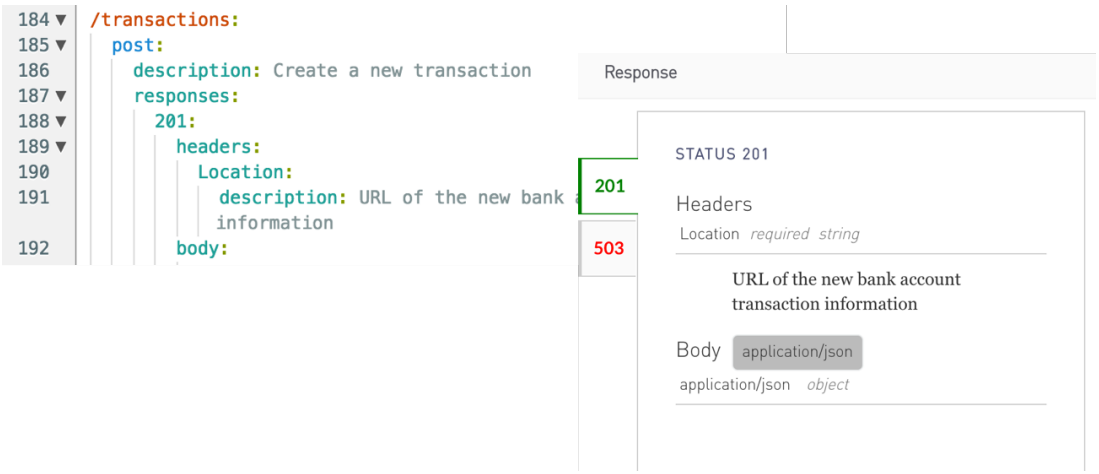


43. Click CLOSE X in the ACME Bank Headline header.

Walkthrough 7-2: Add description nodes to a RAML API definition

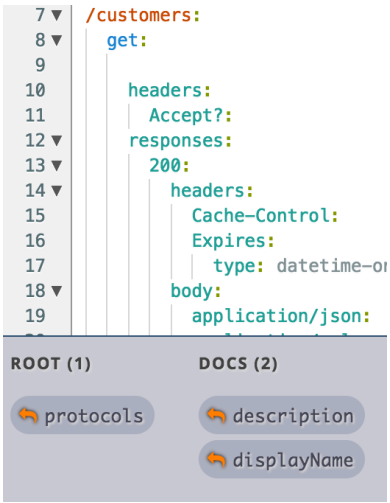
In this walkthrough, you add documentation for the ACME Banking API RAML definition. You will:

- Add descriptions to resource methods.
- Add descriptions to URI parameters.
- Add description and example nodes to request and response headers.



Add a description to a get method

1. Return to api.raml in API Designer.
2. In the /customers resource, create a new line after the line that contains the get method.
3. In the shelf, click description.



4. Set the description to Retrieve a list of customers.

```
/customers:
  get:
    description: Retrieve a list of customers
    headers:
```

Add descriptions to other methods

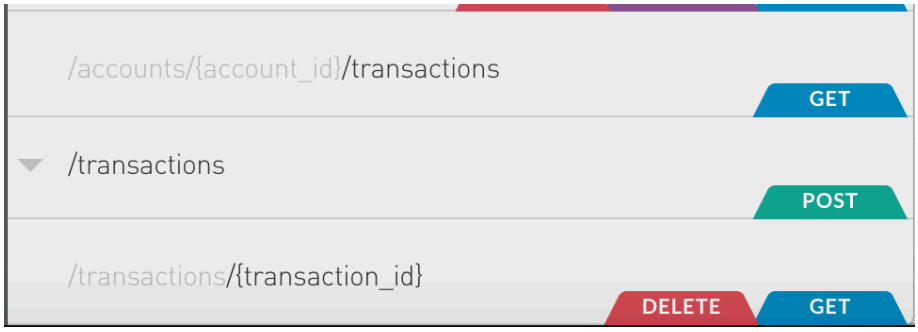
5. Add some or all the following descriptions to the specified methods for the following resources:

• /customers	get	Add a new customer
• /customers /{customer_id}	get	Retrieve a customer with a specific customer ID
• /customers /{customer_id}	patch	Update a specific customer info
• /customers /{customer_id}	delete	Delete a specific customer
• /accounts	get	Retrieve a list of accounts for a specific customer
• /accounts	post	Create a new bank account
• /accounts/{account_id}	get	Retrieve an account info with a specific account ID
• /accounts/{account_id}	delete	Delete an account with a specific account ID
• /accounts/{account_id}	put	Update a specific account info
• /transactions	get	Retrieve a list of transactions for a specific account
• /transactions	post	Create a new transaction
• /transactions/{transaction_id}	get	Retrieve a transaction with a specific transaction ID

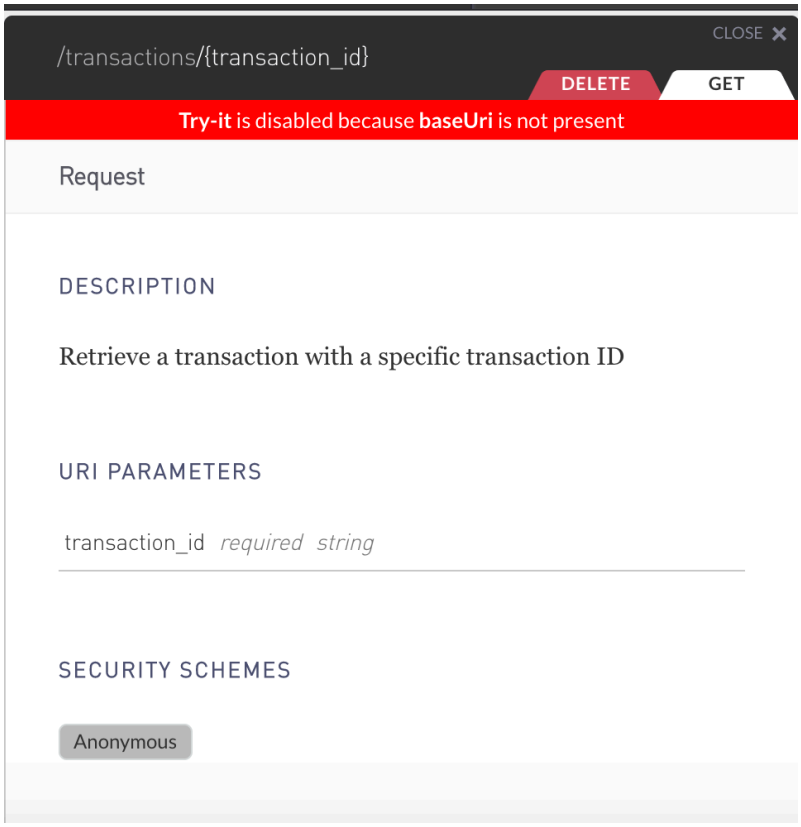
6. Save the project.

View the API documentation

7. In the API Console, click GET for the /transactions/{transaction_id} resource.



8. View the Description in the Request section of the GET method.



9. Click CLOSE X.

Add description nodes to URI parameters

10. In the /customers resource, locate the customer_id URI parameter.
11. Add a description to it and set it to ID of the customer.

```
48 ▼  /{customer_id}:
49 ▼    uriParameters:
50        customer_id:
51          description: ID of the customer
52 ▼    get:
```

12. Similarly, add a description for the account_id URI parameter equal to ID of the account.
13. Add a description for the transaction_id URI parameter equal to ID of the transaction.

Add description and example nodes to a method's request and response headers

14. In the /customers resource, locate the optional Accept header.

15. Add a description to it and set it to Specify the media type of the response.

```
/customers:
  get:
    description: Retrieve a list of customers
    headers:
      Accept?:
        description: Specify the media type of the response
        |
```

16. Go to a new line and click example in the shelf.
17. Set the example to application/xml.

```
/customers:
  get:
    description: Retrieve a list of customers
    headers:
      Accept?:
        description: Specify the media type of the response
        example: application/xml
    responses:
```

18. Similarly, add the description and example nodes to the Accept headers for the following resource methods:

    - /customers/{customer_id} get method
    - /customers/{customer_id}/accounts get method
    - /accounts/{account_id} get method
    - /accounts/{account_id}/transactions get method
    - /transactions/{transaction_id} get method

19. Similarly, add a description node to the Cache-Control header of the response.
20. On the description node line, type | and press enter.
21. Return to the course snippets.txt file.

22. Locate the Description for Cache-Control header and copy the lines of text.

```
Documentation for acmeBankDoc.raml file -

Snippet #1
**ACME Banking API** enables developers to build applications that make use of the information from
resource  methods implemented in the API.

Snippet #2
This API contains functionality that allows developers to retrieve and manipulate _customer_, _account_
and _transaction_ information. Check out the [API Portal]() for more details.

Documentation for acmeBankHeadline.raml file -

Snippet #3
**ACME Bank** is a _multinational_ banking and financial services organization.

Description for Cache-Control header:

Activates caching and defines cache behavior through cache response directives.
Usually defines public or private (cacheable by proxy or not) and max-age for resource.
See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.

Description for Expires header:

Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid.
If both the Expires header and max-age in the Cache-Control header are set, max-age will take
precedence.
```

23. Return to api.raml.
24. Paste the text in the new empty line.

```
15 ▼    responses:
16 ▼      200:
17 ▼        headers:
18 ▼          Cache-Control:
19              description: |
20                Activates caching and defines cache behavior through
                  cache response directives.
21  Usually defines public or private (cacheable by proxy or not) and
    max-age for resource.
22 ▼ See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more
    information.
```

25. Go to the line below the line number that contains the red cross error.
26. Select the lines to before the Expires header and press tab 7 times to indent it in line with the description node.

```
18 ▼          Cache-Control:
19              description: |
20                Activates caching and defines cache behavior through
                  cache response directives.
21  Usually defines public or private (cacheable by proxy or not) and max-
    age for resource.
22 ▼ See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more
    information.
23 ▼            Expires:
```

27. Go to the end of the last line of the description node and press enter.
28. Press backspace or delete to go back one tab space.
29. In the shelf, click example.

```
18 ▼          Cache-Control:
19 ▼            description: |
20                Activates caching and defines cache behavior through
                  cache response directives.
21                Usually defines public or private (cacheable by proxy or
                  not) and max-age for resource.
22                See http://www.w3.org/Protocols/rfc2616/rfc2616-
                  sec14.html for more information.
23
24          Expires:
25            type: datetime
```

30. Set the example node to private, max-age=31536000.

```
18 ▼          Cache-Control:
19 ▼            description: |
20                Activates caching and defines cache behavior through
                  cache response directives.
21                Usually defines public or private (cacheable by proxy or
                  not) and max-age for resource.
22                See http://www.w3.org/Protocols/rfc2616/rfc2616-
                  sec14.html for more information.
23              example: private, max-age=31536000
24          Expires:
25            type: datetime
```

31. Similarly, set the following documentation nodes for the Expires header:

- description: The text for Description for Expires header text in the course snippets.txt file
- example: Tue, 18 Apr 2017 09:30:41 GMT
- format: rfc2616

Note: The default datetime format follows rfc3339 specification.

```
24 ▼          Expires:
25 ▼            description: |
26                Sets a date in RFC 1123 format from which the cached
                  resource should no longer be considered valid.
27                If both the Expires header and max-age in the Cache-
                  Control header are set, max-age will take precedence.
28              type: datetime
29              example: Tue, 18 Apr 2017 09:30:41 GMT
30              format: rfc2616
```

Add documentation to other request and response headers

32. Add some or all of the following descriptions to the Location headers for the specified resource methods:

    - /customers post URL of the new customer information
    - /accounts post URL of the new bank account information
    - /transactions post URL of the new bank account transaction information

```
184 ▼ /transactions:
185 ▼   post:
186       description: Create a new transaction
187 ▼     responses:
188 ▼       201:
189 ▼         headers:
190             Location:
191               description: URL of the new bank account transaction
                  information
192           body:
```

33. Scroll up to the get method of the /customers resource and copy the Accept header's description and example lines.

```
 8 ▼ /customers:
 9 ▼   get:
10       description: Retrieve a list of customers
11 ▼     headers:
12 ▼       Accept?:
13           description: Specify the media type of the response
14           example: application/xml
15 ▼     responses:
16 ▼       200:
```

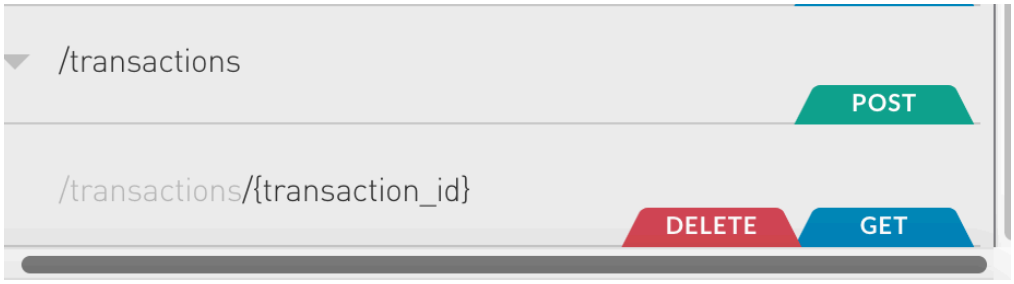
34. Scroll down to the get method of the /accounts nested resource add the description and example to the optional Accept header.

```
 94 ▼     /accounts:
 95 ▼       get:
 96           description: Retrieve a list of accounts for a specific
              customer
 97 ▼         headers:
 98             Accept?:
 99               description: Specify the media type of the response
100           example: application/xml
101 ▼         responses:
```

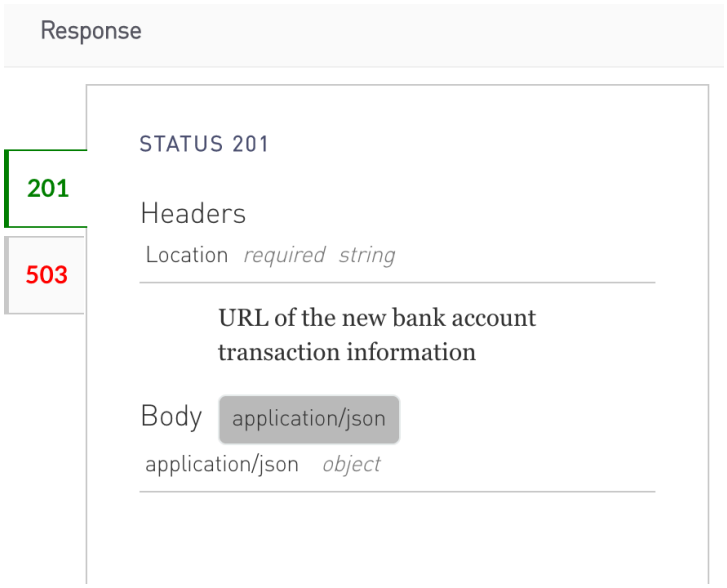
35. Indent the example node.
36. Save the project.

View the API documentation

37. In the API Console, click POST for the /transactions resource.



38. Locate the Location header description in the 201 Response section.

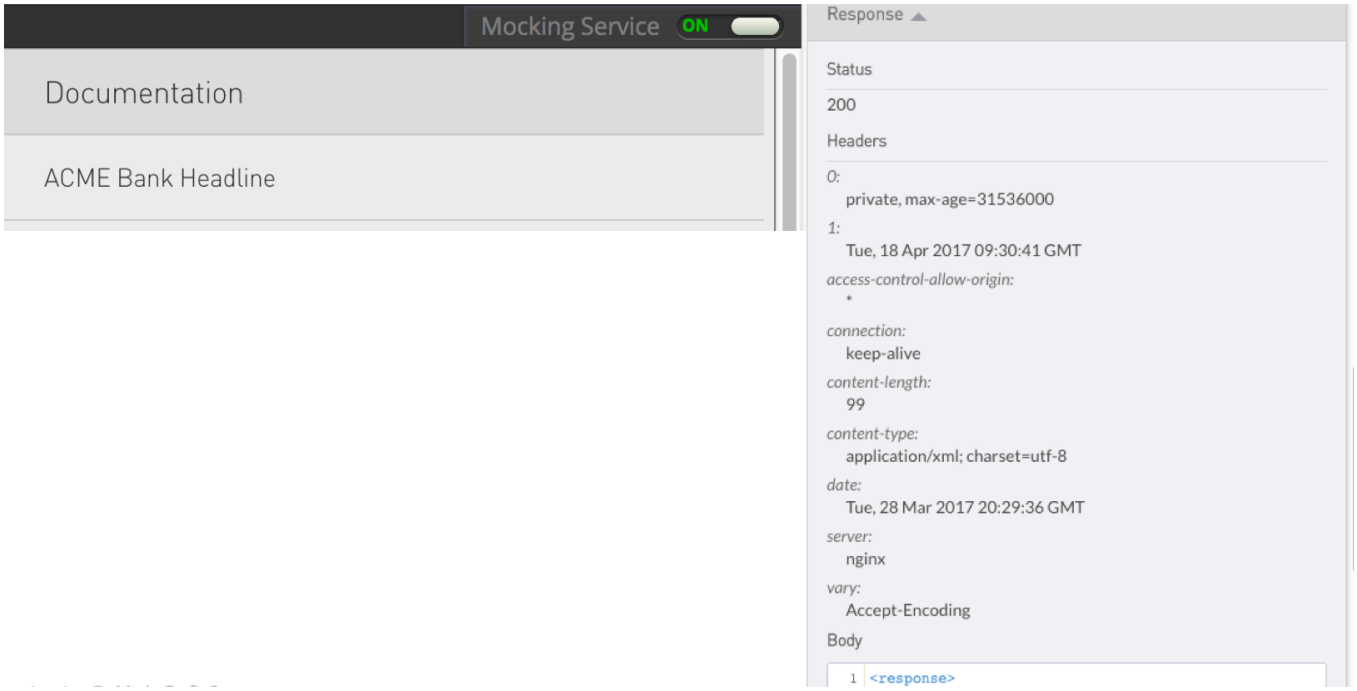


39. In the /transactions resource banner, click CLOSE X.

Walkthrough 7-3: Use the mocking service in API Console to test an API

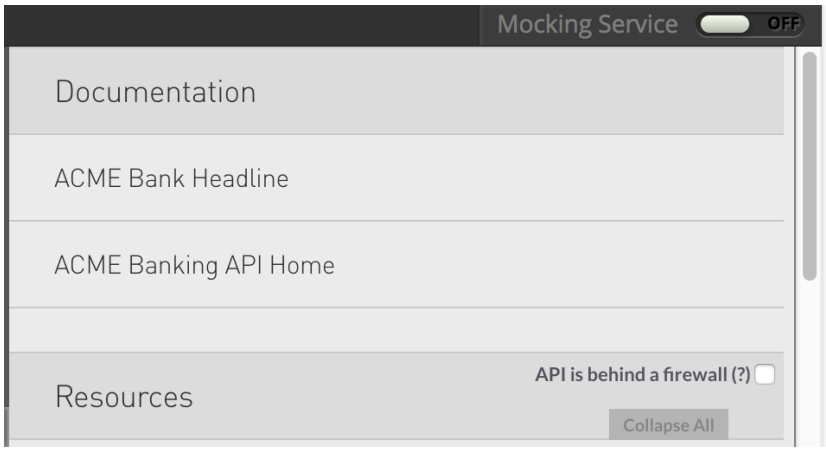
In this walkthrough, you will test your API by enabling mocking service. You will:

- Enable the mocking service in the API Console.
- Test a few methods in the API Console to see the response information.

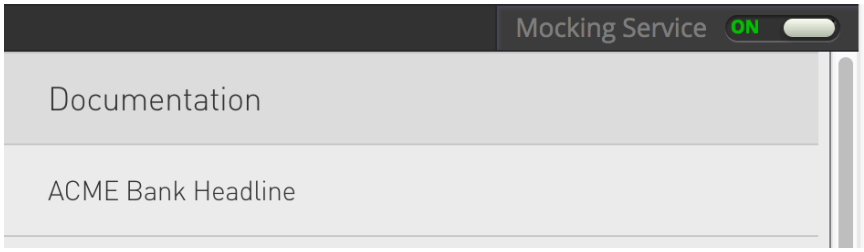


Enable the mocking service in the API Console

1. Return to API Designer.
2. In the top-right corner of the API Console, click the Mocking Service toggle button.



3. Verify that the toggle button now contains green ON text.



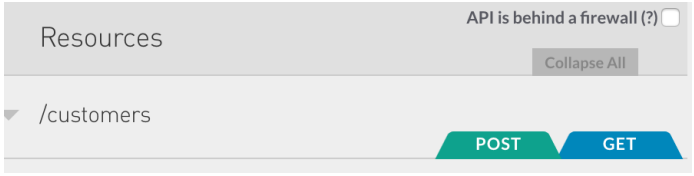
4. Look at api.raml; you should now see a new baseUri parameter was added above the title node.

```
1   #%RAML 1.0
2   baseUri: https://mocksvc.mulesoft.com/mocks/2dd91fa5-7a3b-49a7-
    8926-4adb73dcbe3c
3   title: ACME Banking API
4   version: 1.0
5
6 ▼ documentation:
```

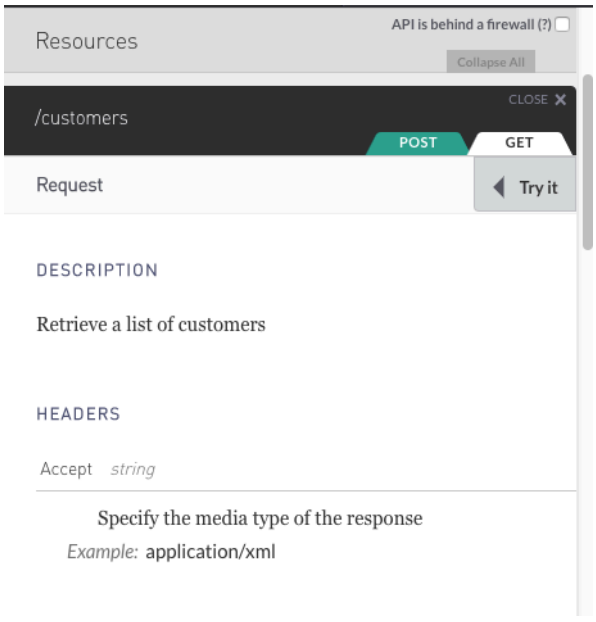
5. Save the project.

Test a method in the API Console to see the response information

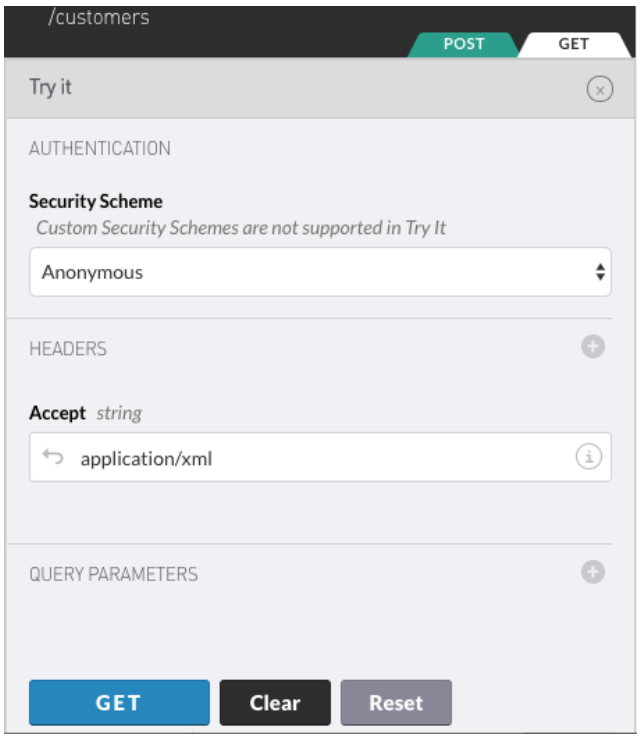
6. In the API Console, click GET the /customers resource.



7. Click Try It.

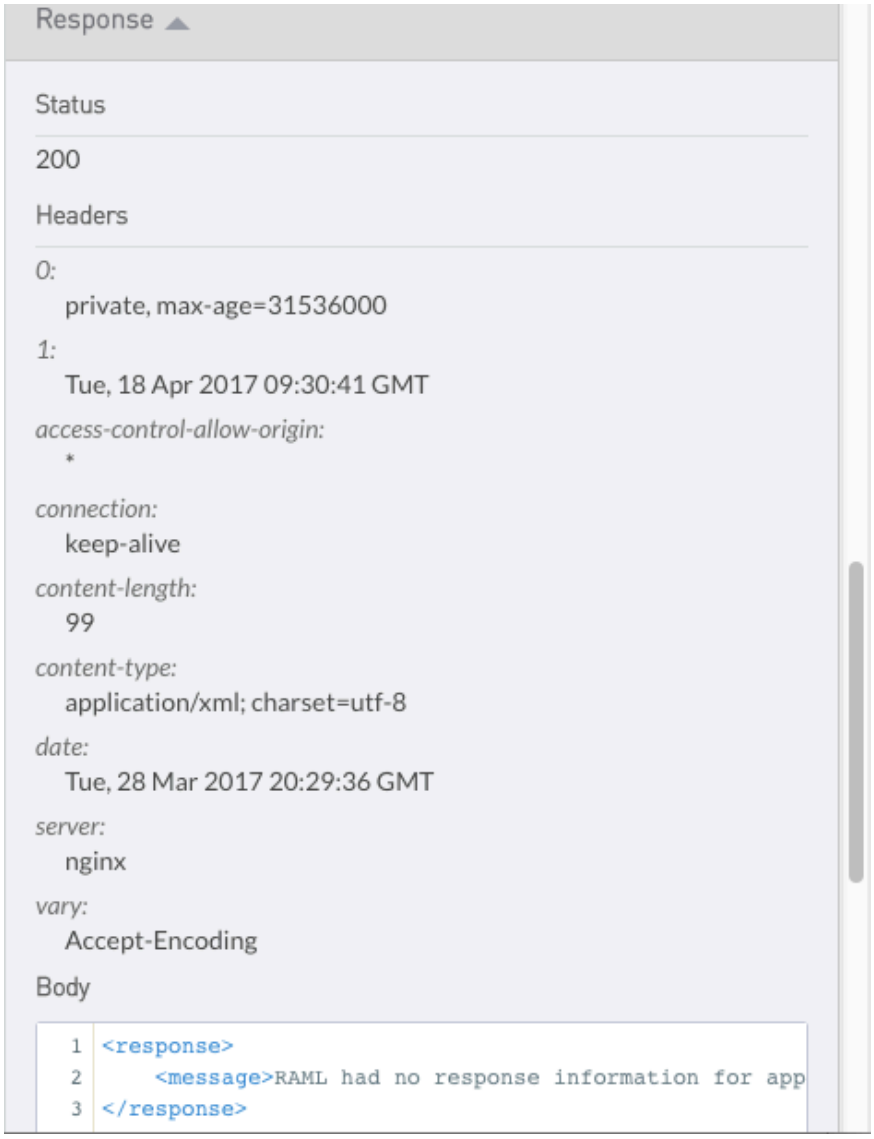


8. In the Try It section, scroll down to locate the GET button and click it.



9. Verify that you can see the Request information (Request URL and the Accept header information).

10. Review the Response status and information displayed right below the request information.

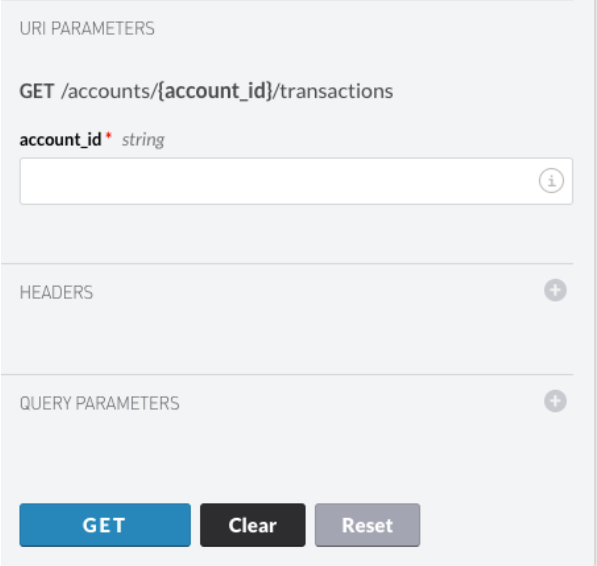


11. Scroll up and click CLOSE X.

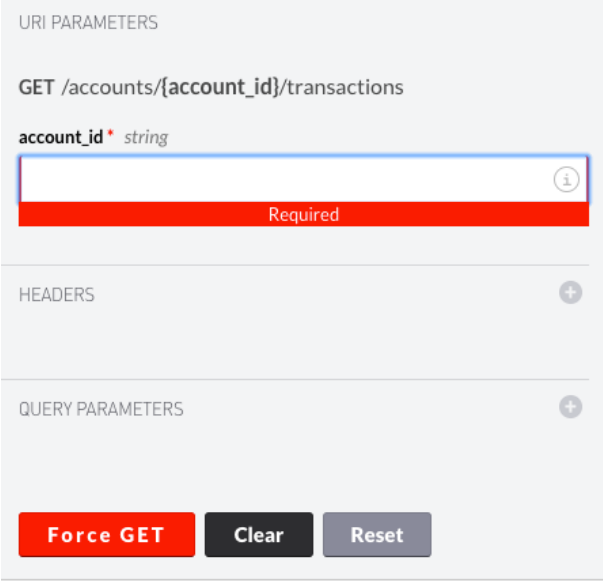
Test another method in the API Console to see the response information

12. Click the GET tab for the /accounts/{account_id}/transactions resource.
13. Click Try It.

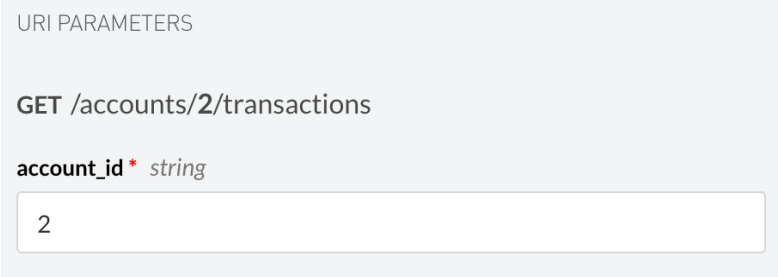
14. In the Try It section, scroll down to locate the GET button and click it.



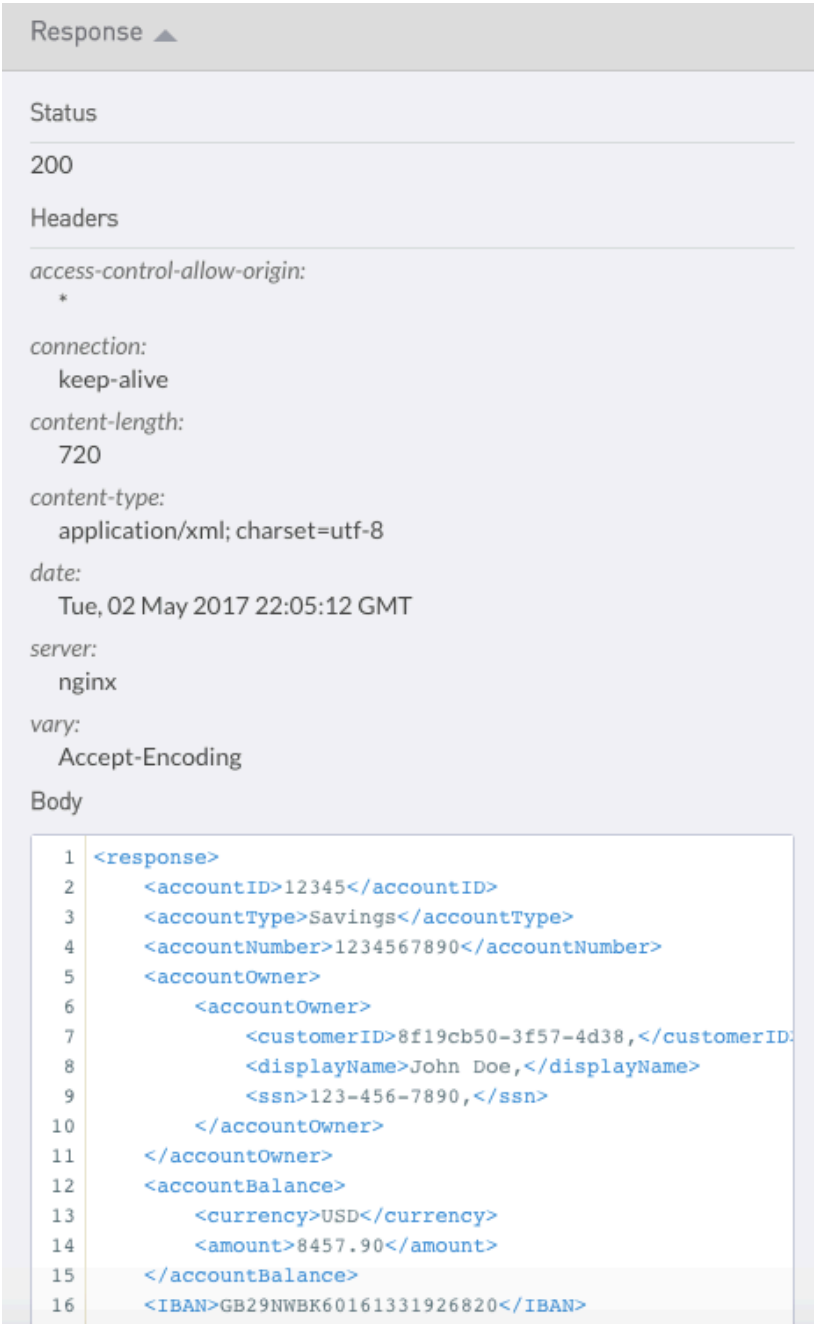
15. Verify that the account_id URI parameter has a red banner denoting it is required before calling the GET method.



16. Enter an account_id of 2 and click the GET button again.

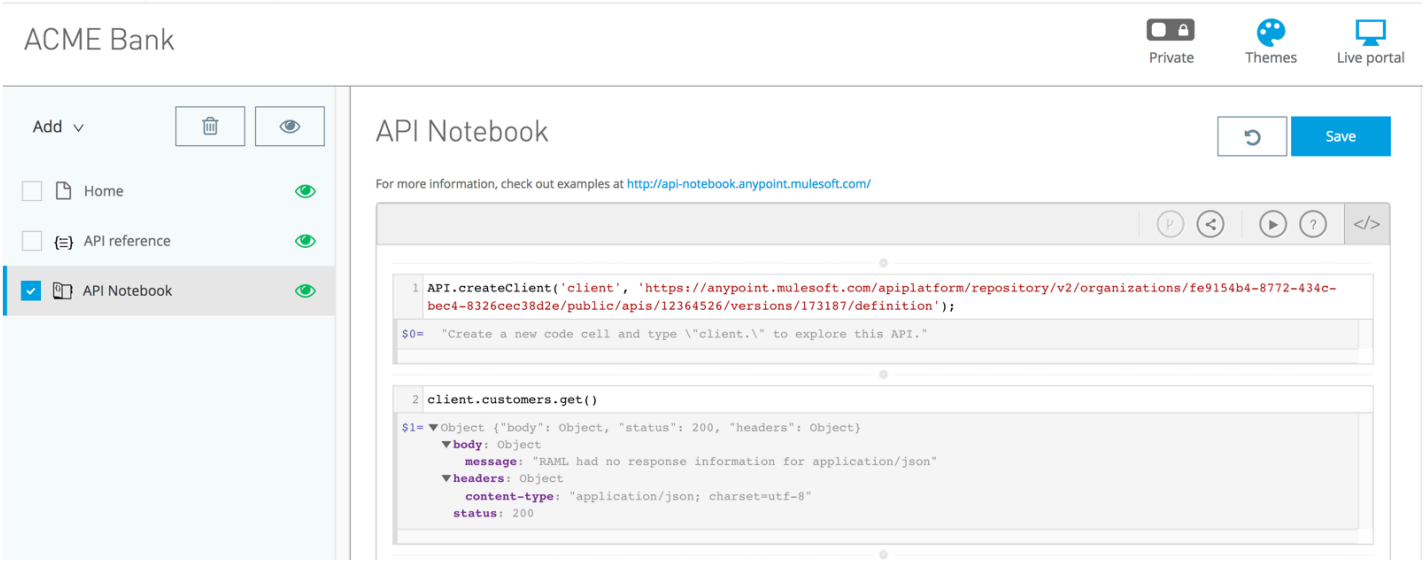


17. Scroll down and review the Response status and information displayed right below the request information.



18. Click CLOSE X in the /accounts/{account_id}/transactions banner.

Module 8: Making APIs Discoverable



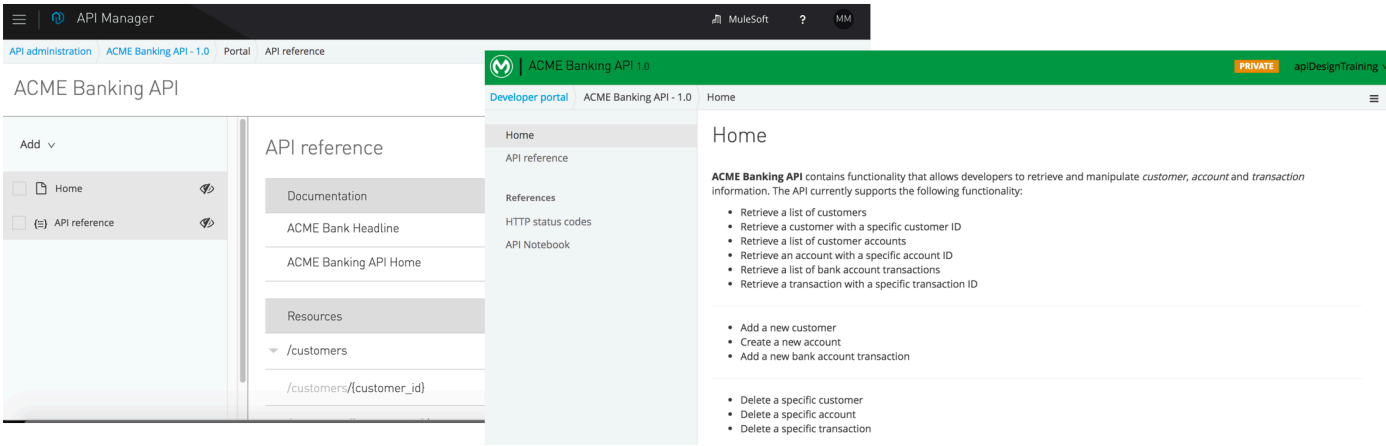
Objectives:

- Create API Portals for learning about and testing APIs.
- Customize API Portals with themes.
- Create a sample use case with API Notebook inside the API Portal.
- Publish API definitions to the Anypoint Exchange for discovery.
- Gather feedback from API consumers.

Walkthrough 8-1: Create and customize an API Portal

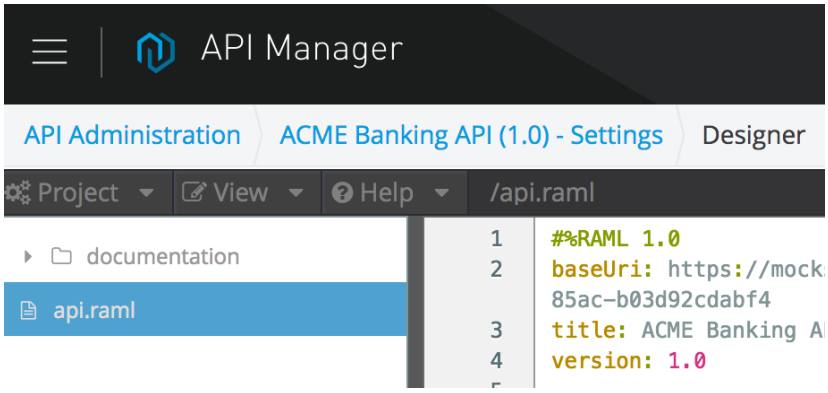
In this walkthrough, you will create an API Portal documenting information about the ACME Banking API. You will:

- Create an API Portal.
- Add various types of content to an API Portal.
- Define a color scheme.
- Publish and view the resulting API Portal.

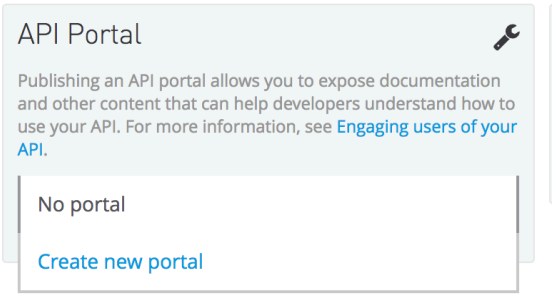


Create an API Portal

1. Return to API Designer.
2. Click the ACME Banking API (1.0) - Settings link above the editor.



3. In the API Portal section on the API version details page, click the drop-down arrow and select Create new portal from the drop-down menu; the API Portal Designer should open.

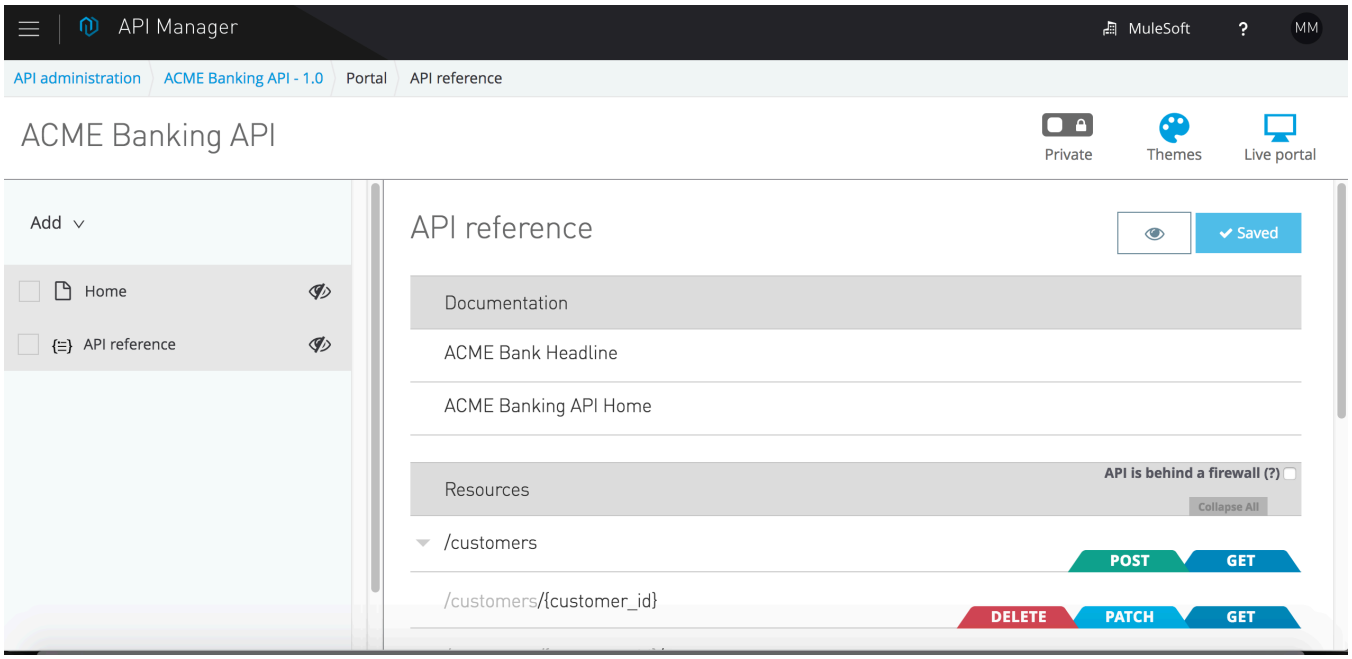


4. In the API Portal Designer, locate the two pages that were automatically created: Home and API reference.
5. In the left-side navigation, click the Home page link.
6. Examine the Home page editor.

   Note: The Home page is empty and has icons in the editor to add text that supports markdown, include images and files.

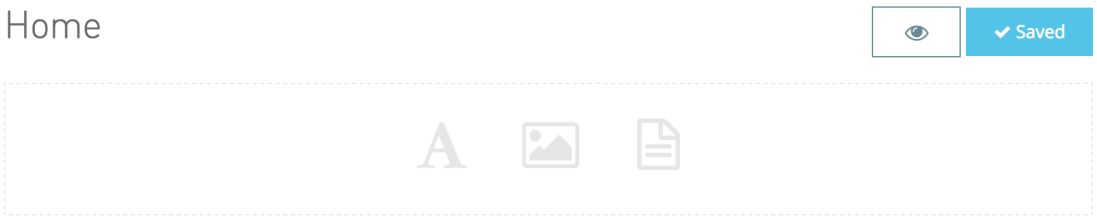
7. In the left-side navigation, click the API reference link.
8. Examine the API reference page.

   Note: The API reference page is only populated if the API definition has a base URI specified.



Use Markdown to add content to a page

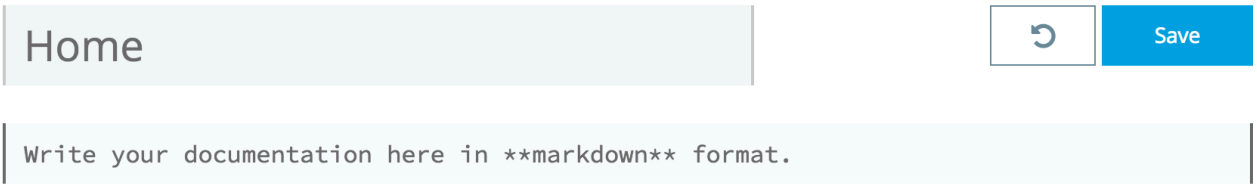
9. In the left-side navigation, click the Home page link.
10. In the Home page editor, click the letter A to begin adding content.



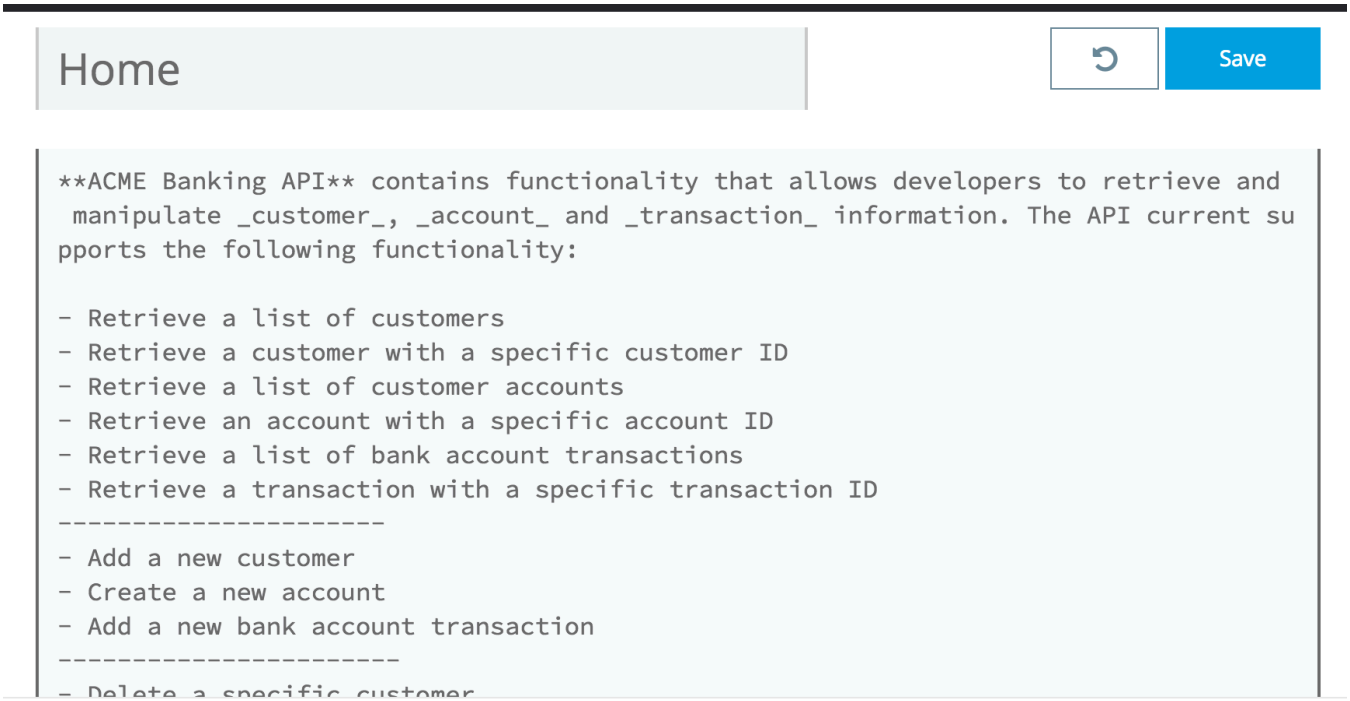
11. Return to the course snippets.txt file.
12. Locate the Module 8 section and copy the ACME Banking API Home content.

```
-------Module 8--------

ACME Banking API Home content -

**ACME Banking API** contains functionality that allows developers to retrieve and manipulate
_customer_, _account_ and _transaction_ information. The API currently supports the following
functionality:

- Retrieve a list of customers
- Retrieve a customer with a specific customer ID
- Retrieve a list of customer accounts
- Retrieve an account with a specific account ID
- Retrieve a list of bank account transactions
- Retrieve a transaction with a specific transaction ID
-----------------------
- Add a new customer
- Create a new account
- Add a new bank account transaction
-----------------------
- Delete a specific customer
- Delete a specific account
- Delete a specific transaction
-----------------------
- Update a specific customer profile information
- Update a specific bank account information

Check out the **API Reference** page to learn more about the resources and methods.

HTTP status codes URL:
```

13. Return to the API Portal Designer and click the box with the text Write your documentation here in **markdown** format.



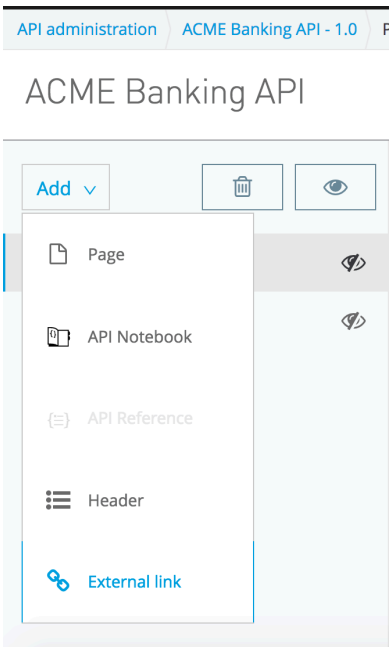
14. Paste the copied lines here.



15. Click the Save button.

Add external links

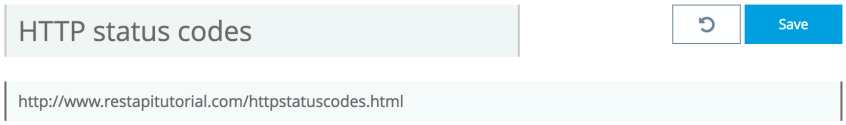
16. Click the Add button located above the left-side navigation and in the drop-down menu, select External link.



17. Click the title External Link in the editor and type HTTP status codes.

18. Change the URL from http://example.com to http://www.restapitutorial.com/httpstatuscodes.html.

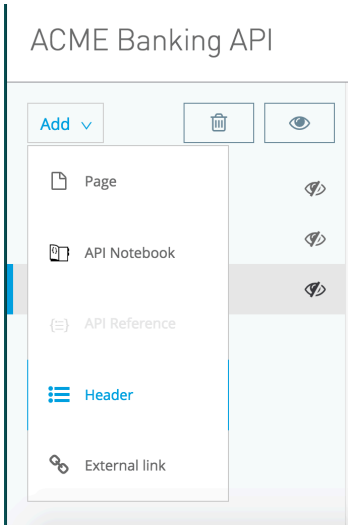
    Note: You can copy this URL from the course snippets.txt file.



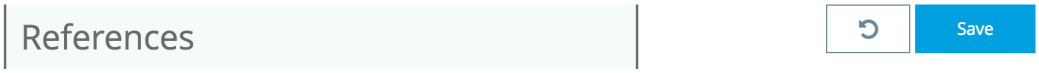
19. Click the Save button.

Add a header

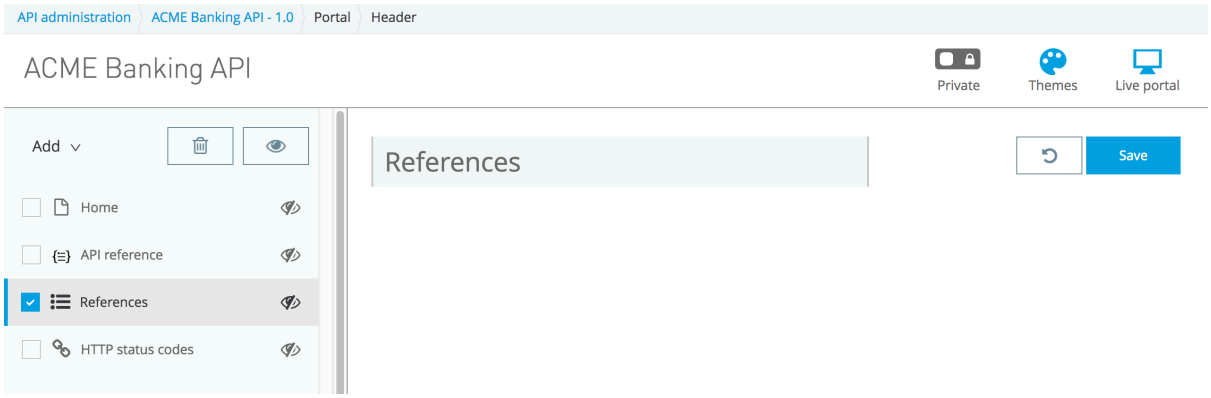
20. Click the Add button again and select Header in the drop-down menu.



21. Click the title Header in the editor and replace the text with References.

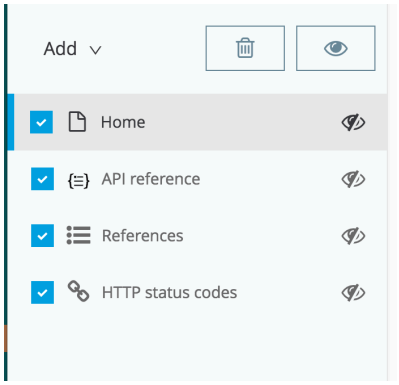


22. Click the Save button.
23. In the left-side navigation, click and drag the References header and drop it above the HTTP status codes external link.



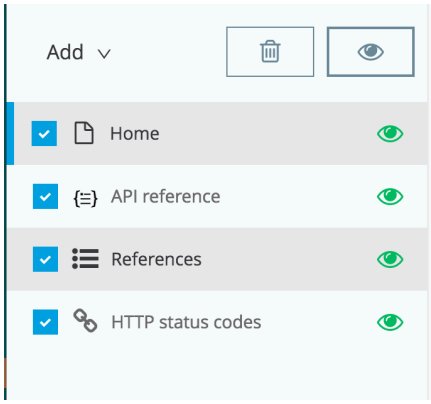
Publish the API Portal

24. In the left-side navigation, select the checkboxes for all the components.

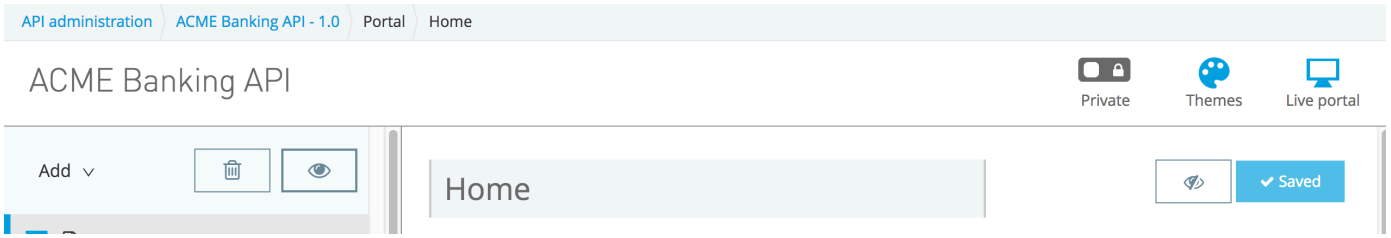


25. Click the eye button to publish the selected pages.

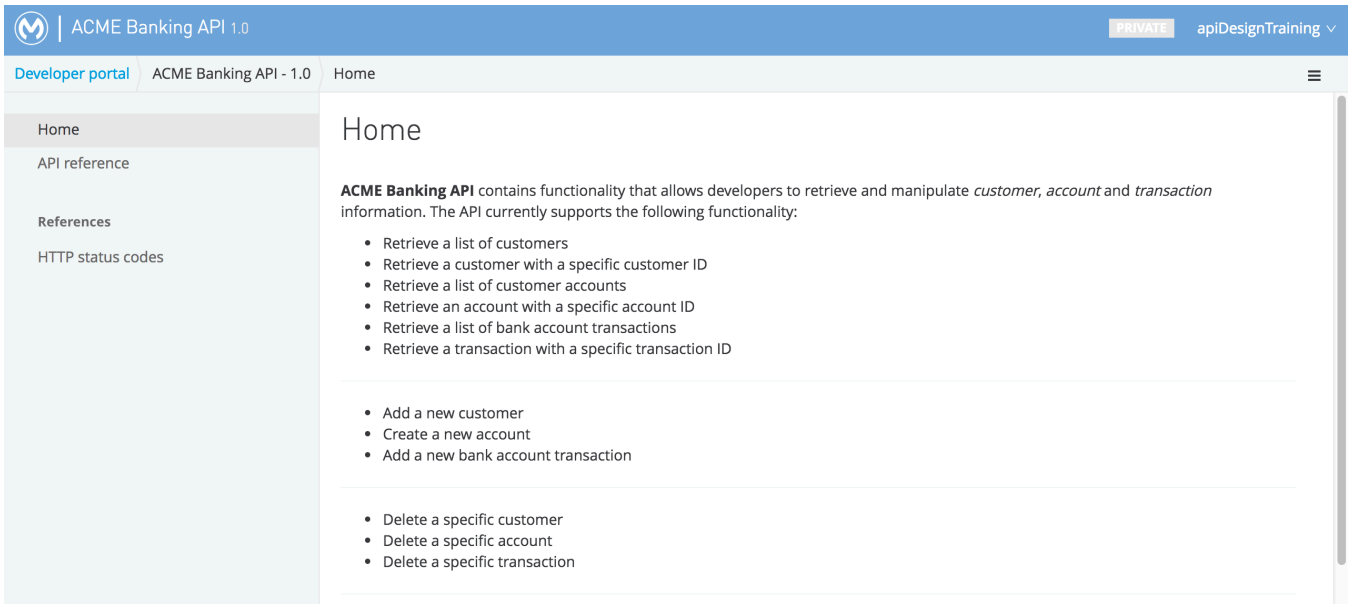
    Note: Anypoint Platform will not publish a page with zero content. Once you click the eye icon, it should turn green for all the selected pages.



26. Click the Live portal button in the upper-right corner of the API Portal Designer.



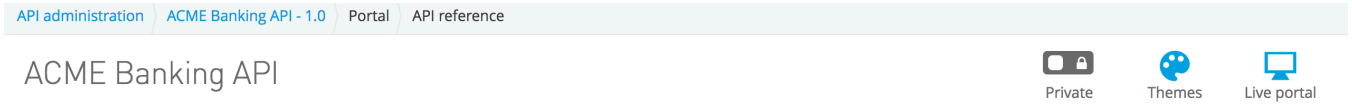
27. Examine the generated API Portal.



28. Close the tab with containing the Live Portal and return to the tab with the API Portal Designer.

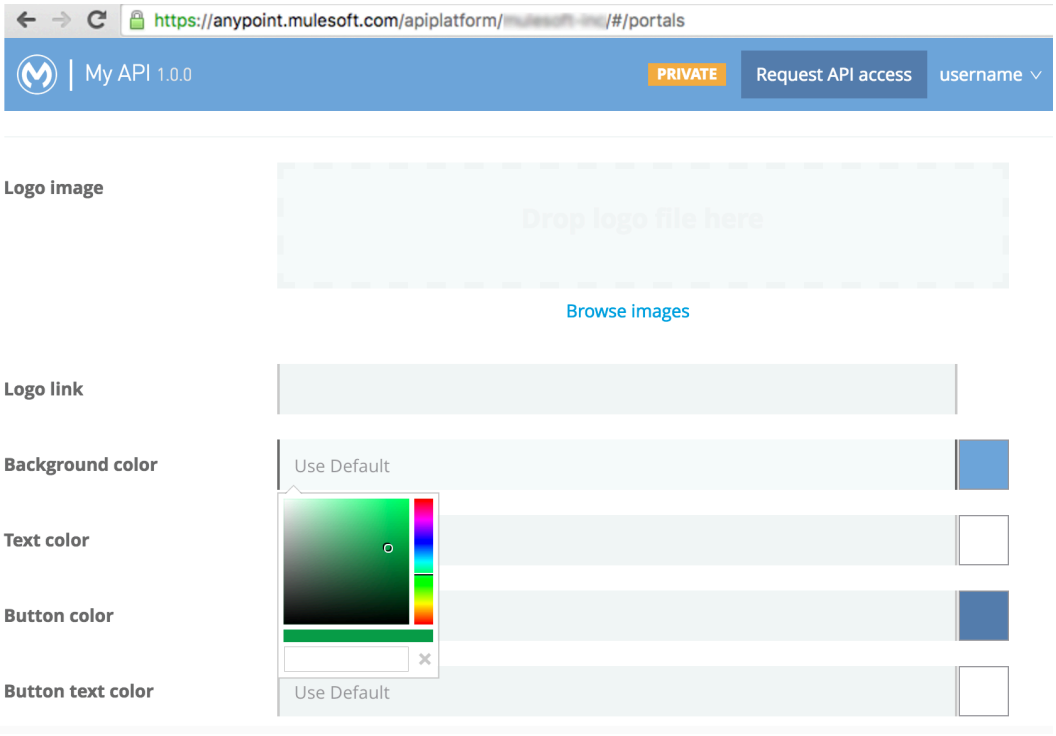
Define a color scheme for the API Portal

29. In the right corner of the API Portal Designer, click Themes.

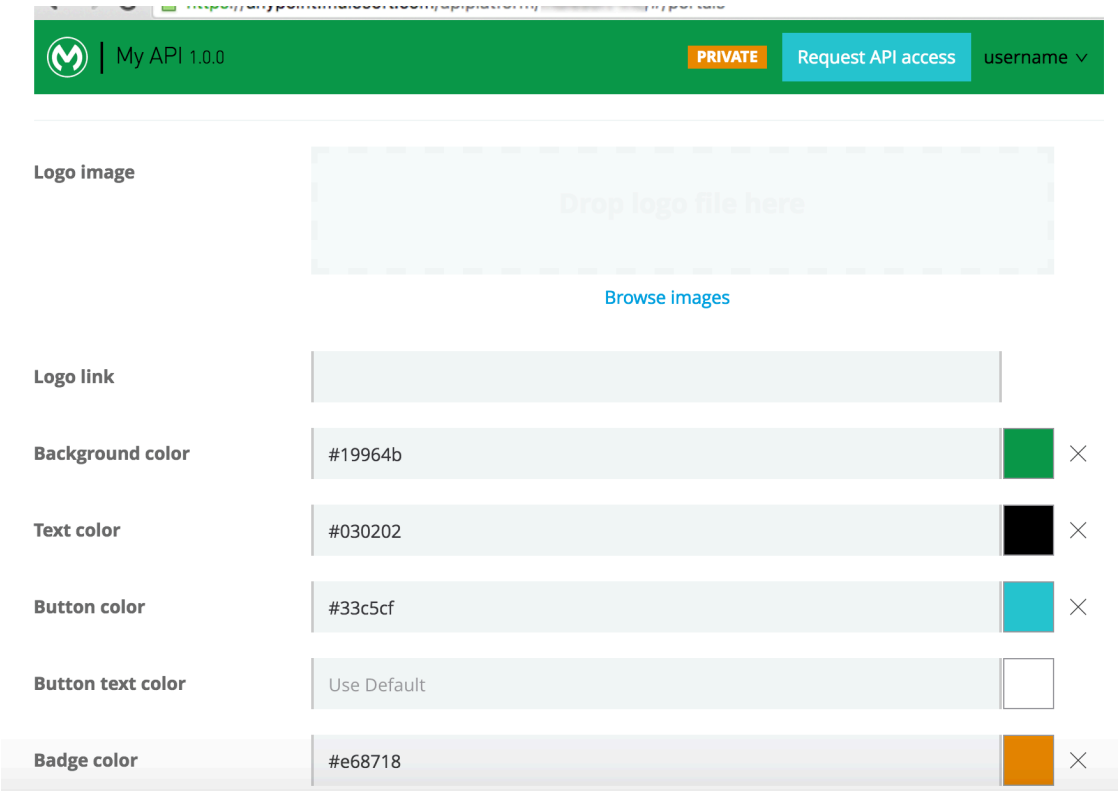


30. In the API portal theme settings dialog box, go to the Background color option and click the text area that mentions Use Default.

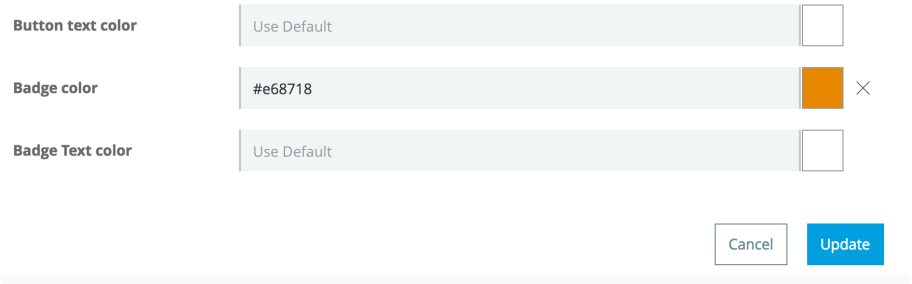
31. Select a color from the drop-down graph or type a color's hex value.



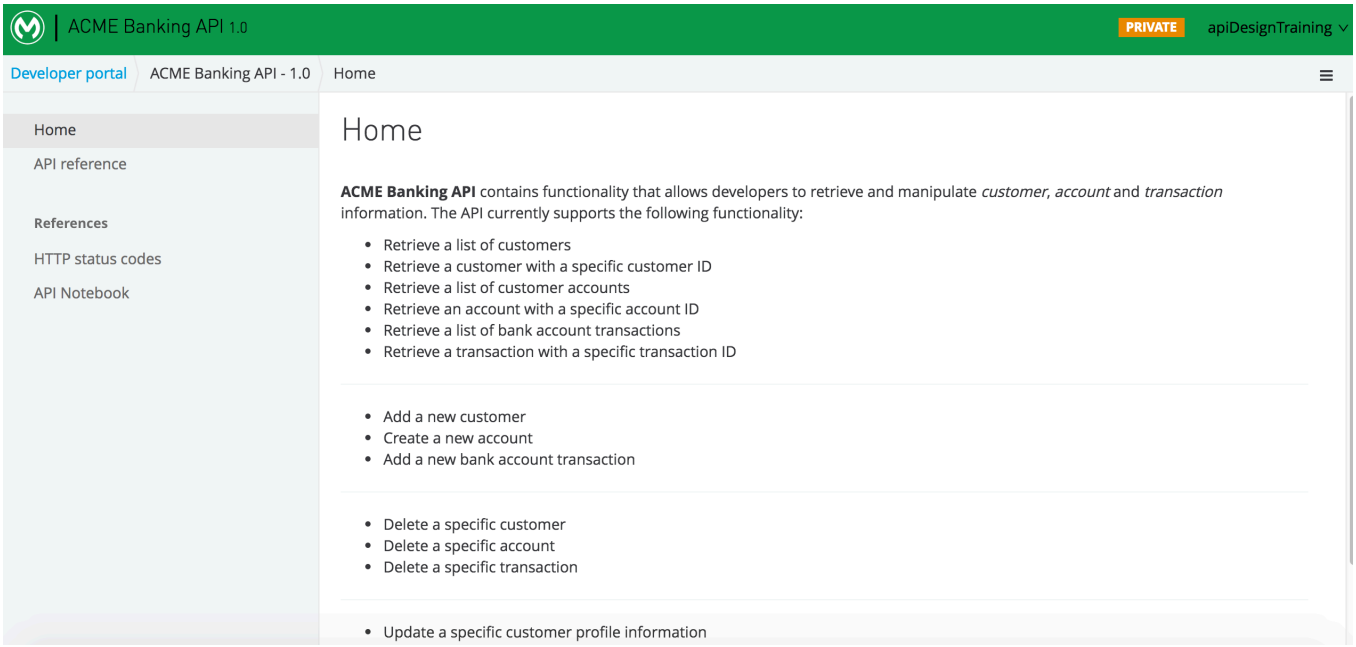
32. Change the text color, button color and button text color using the same approach of click over the Use Default text box and picking a desired color.



33. Scroll down and click the Update button.



34. Click the Live Portal button again; the live portal should open and you should see your changes.

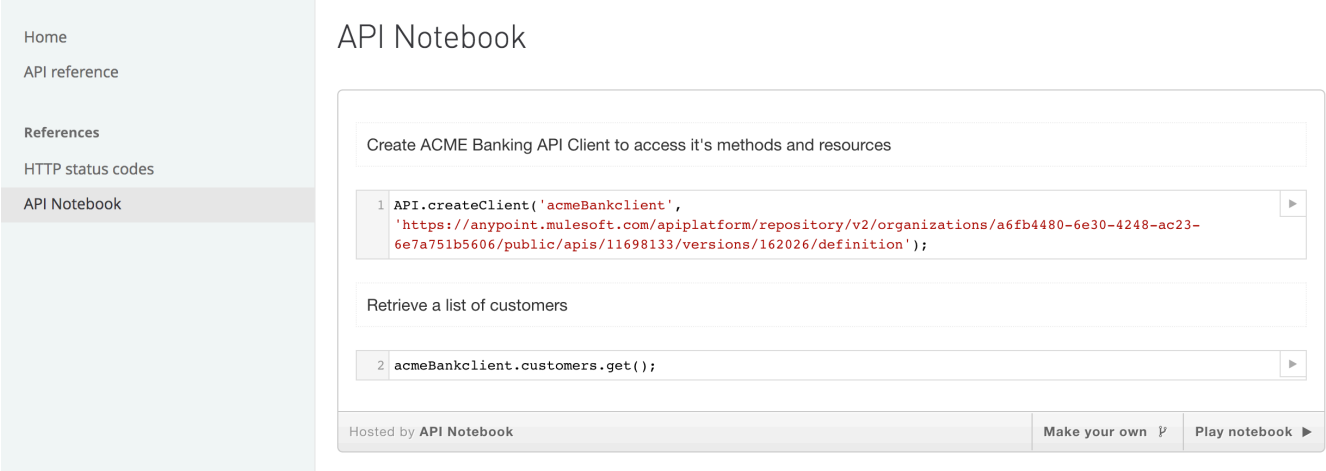


35. Close the Live Portal and return to the API Portal Designer.

Walkthrough 8-2: Create a sample use case with API Notebook inside the API Portal

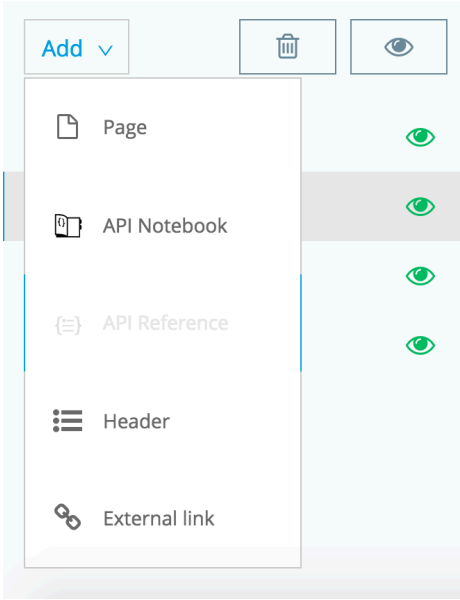
In this walkthrough, you create an API Notebook inside the API Portal. You will:

- Add an API Notebook in the API Portal Designer.
- Perform string manipulation on the datatypes.
- Create and use JavaScript functions inside API Notebook.
- View the API Notebook in the Live Portal.

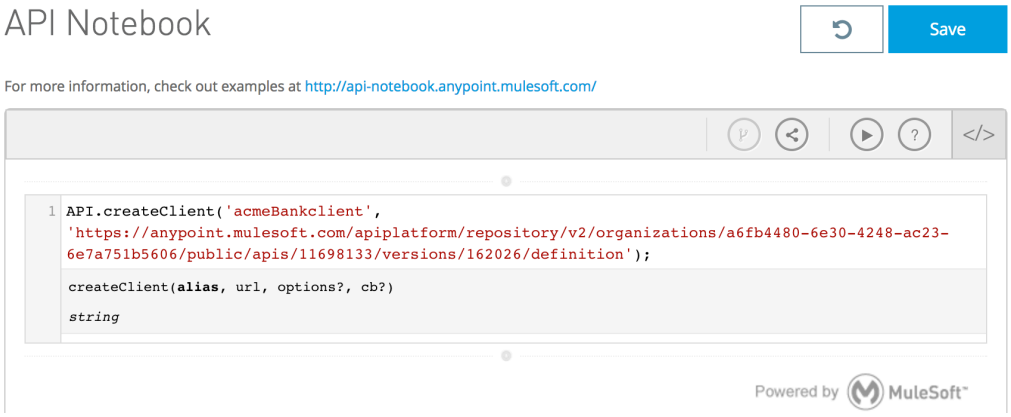


Add an API Notebook in the API Portal Designer

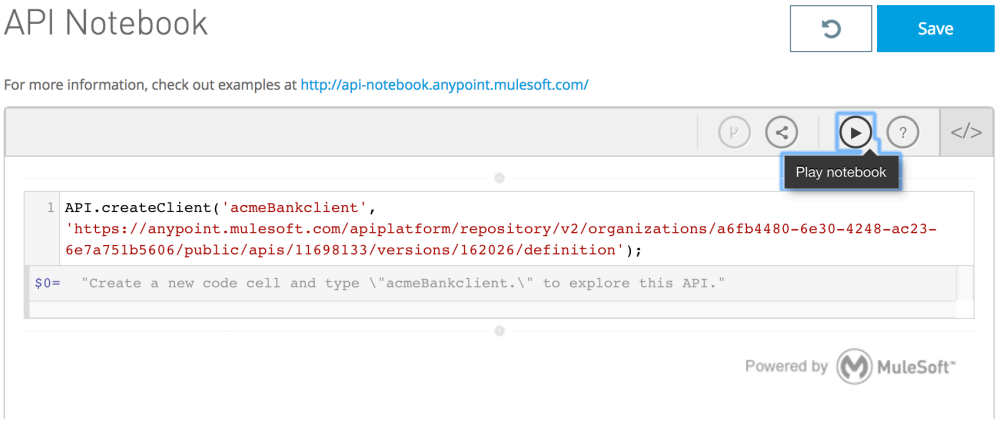
1. Return to the API Portal Designer.
2. In the left-side navigation, click Add and select API Notebook in the drop-down menu.



3. In the API Notebook editor, change the first parameter in the createClient function from ‘client’ to ‘acmeBankclient’.

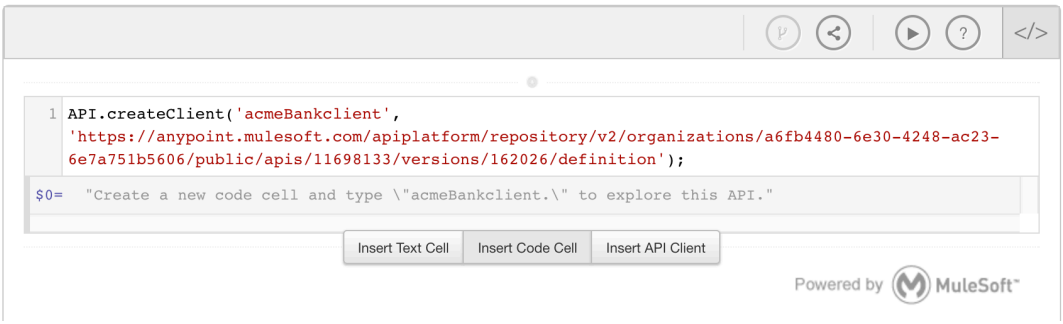


4. Click the Play notebook button to play the Notebook.
5. Verify you get this message below the code cell: Create a new node cell and type \ “acmeBankClient.\” to explore this API.

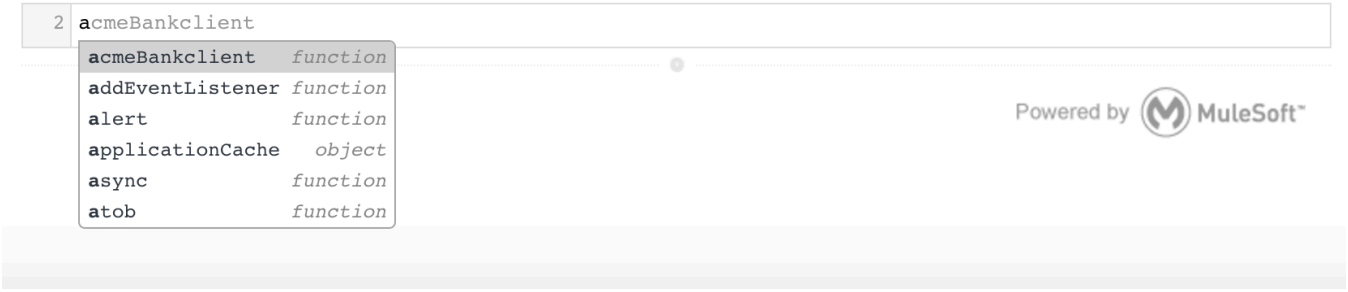


Insert a code cell to explore the ACME Banking API

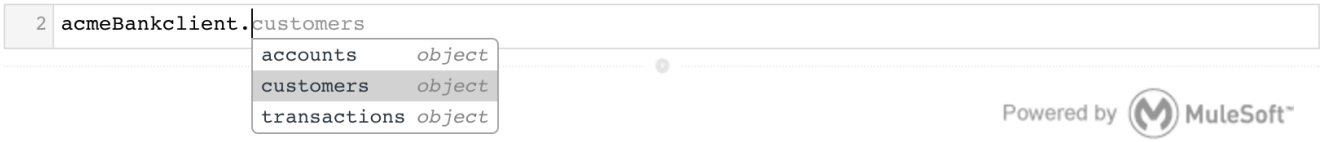
6. Hover over the small circle that is in the middle underneath the createClient code cell.



7. Click Insert Code Cell.
8. In the new code cell, type a and press the right arrow key.



9. Type . after acmeBankclient and select customers in the drop-down menu.



10. Type . after customers and click on get from the drop-down menu.
11. Add opening and closing parenthesis and a semicolon after the get method; the expression should look like this:

```
2 acmeBankclient.customers.get();
```

12. Press enter to execute the code cell.
13. Verify that you see the status code 200 returned in the result below the code cell.

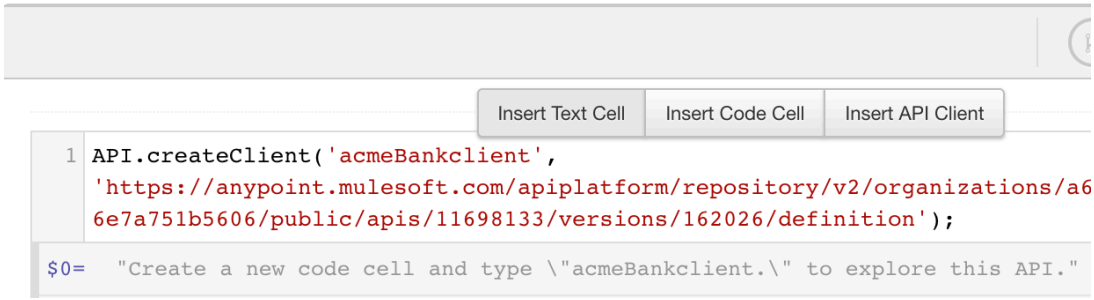
Note: When you expand the body, you can see the customer example returned.

14. Click the menu icon in the right corner of code cell 3 and click Delete from the drop-down menu.

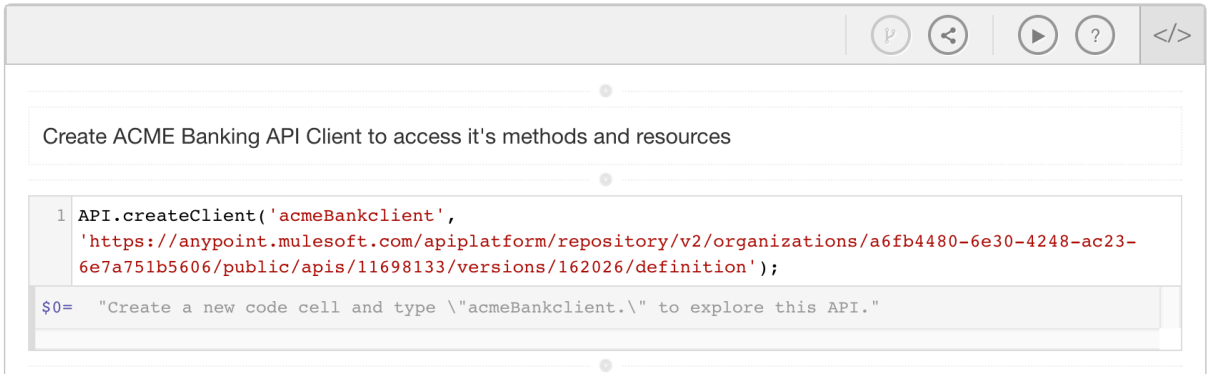
Insert text cells to make the API Notebook readable

15. Locate the circle in the middle above code cell 1 and hover over it.

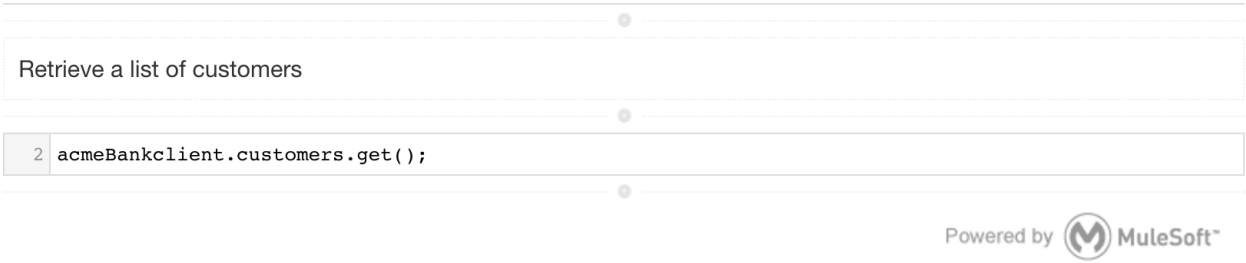
16. Click Insert Text cell.



17. Click inside the new text cell and type Create ACME Banking API client to access its methods and resources.



18. Locate the circle in the middle above code cell 2 and hover over it.
19. Click Insert Text Cell.
20. Click inside the new text cell and type retrieve a list of customers.



21. Click Save.

Access the customer datatype example and its attributes and nested datatypes attributes in the API Notebook

22. Type the following inside the text cell: Retrieve a specific customer information.

```
Retrieve a specific customer information
```

23. Return to the course snippet.txt file and copy the JavaScript code snippet for the Code cell value to retrieve a specific customer in Module 8.
24. Return to API Notebook and press enter and in the code cell, paste the Javascript code snippet.

Retrieve a specific customer information

```
3 acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
```

25. Press enter inside the code cell to execute the line of code.
26. In the response, click the arrow before the body Object.

```
3 acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
```


```
$2= ▼Object {"body": Object, "status": 200, "headers": Object}
    ▼body: Object
      ▶address: Object
       customerID: "8f19cb50-3f57-4d38"
       dateOfBirth: "1983-01-01"
       displayName: "John Doe"
       email: "johndoe@example.com"
       firstName: "John"
       lastName: "Doe"
       phone: "415-000-0000"
       ssn: "123-456-7890"
    ▶headers: Object
     status: 200
```

27. Click the arrow before the address object.

```
3 acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
```


```
$2= ▼Object {"body": Object, "status": 200, "headers": Object}
    ▼body: Object
     ▼address: Object
        addressLine1: "1234 Lane"
        addressLine2: "Apt.#620"
        city: "San Francisco"
        country: "United States"
        state: "California"
        zipCode: "94108"
      customerID: "8f19cb50-3f57-4d38"
      dateOfBirth: "1983-01-01"
      displayName: "John Doe"
      email: "johndoe@example.com"
      firstName: "John"
      lastName: "Doe"
      phone: "415-000-0000"
      ssn: "123-456-7890"
     ▶headers: Object
      status: 200
```


28. In the same code cell, place the cursor at the beginning of the code cell and type CustomerA=.

```
3 CustomerA=acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
```

29. Go to the end of the code and press enter to re-execute the line.
30. In the new code cell generated below, type CustomerA.body.displayName; and press enter.

```
4 CustomerA.body.displayName;
```

```
$3=  "John Doe"
```

31. In the new code cell generated below, type CustomerA.body.address.city; and press enter.

```
5 CustomerA.body.address.city;
```

```
$4=  "San Francisco"
```

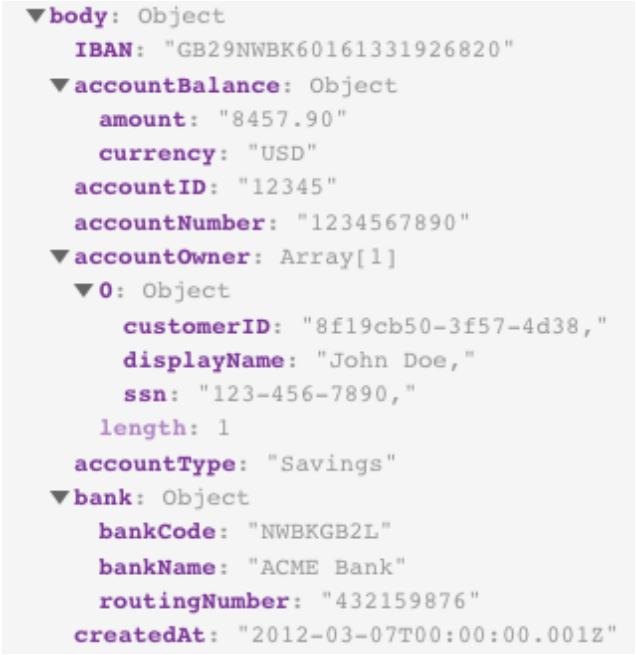
Perform string manipulation on the datatypes

32. Hover over the circle in the middle before the new code cell and click Insert New Text Cell.
33. Type Retrieve a specific account information.
34. Return to the course snippets.txt and copy the code from Module 8 with the heading Code cell value to retrieve a specific account information.

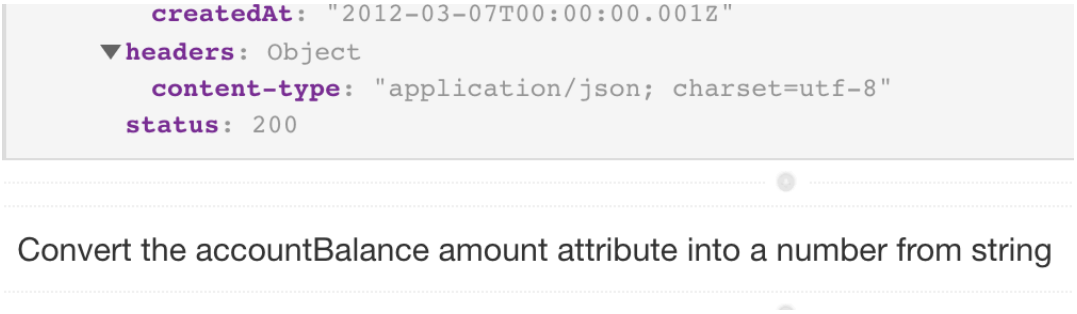
35. Return to API Notebook and paste it in the code cell and press enter.

```
6 accountA=acmeBankclient.accounts.account_id("12345").get();
```

36. In the response returned, click the arrow before body to observe the structure of the account information.
37. Expand the accountBalance, accountOwner, and the bank objects to see the underlying values.



38. Hover over the circle between the new code cell and previous code cell and click Insert Text Cell.
39. Type the value of the text cell as Convert the accountBalance amount into a number from string.



40. Return to the course snippets.txt and copy the JavaScript snippet from Module 8 with the heading Code cell value to convert accountBalance amount into number from string.

41. Return to API Notebook and in the new code cell paste the snippet and press enter.

```
Convert the accountBalance amount attribute into a number from string
```

```
7 accountBalanceNumber=Number(accountA.body.accountBalance.amount);
$6=  8457.9
```

42. Hover over the circle between the new code cell and previous code cell and click Insert Text Cell.
43. Type the value of the text cell as: Write a conditional statement to print the rewards awarded based on the account balance.

```
Write a conditional statement to print that 1.5x rewards are awarded when the balance in the account is greater than 5000 or 1x rewards are awarded
```

44. Return to the course snippets.txt and copy the JavaScript snippet from Module 8 with the heading Code cell value for a conditional statement.
45. Return to API Notebook and in the new code cell paste the snippet and press enter.

```
8  if ( accountBalanceNumber > 5000.00) {
9    text = "You will earn 1.5x times the reward for every dollar you spend"; }
10 else {
11   text = "You will earn 1x times the reward for every dollar you spend";
12 }
13 text;
$7=  "You will earn 1.5x times the reward for every dollar you spend"
```

Create and use JavaScript functions inside the API Notebook

46. Hover over the circle between the new code cell and previous code cell and click Insert Text Cell.
47. Type the value of the text cell as: Write a JavaScript function to concatenate the currency and amount attributes to display the amount in one string.

```
Write a JavaScript function to concatenate the currency and amount attributes in accountBalance to display the amount in one string
```

48. Return to the course snippets.txt file and copy the JavaScript snippet from Module 8 with the heading Code cell value for a function to concatenate and display currency and amount.
49. Return to API notebook and in the new code cell, paste the snippet and press enter.

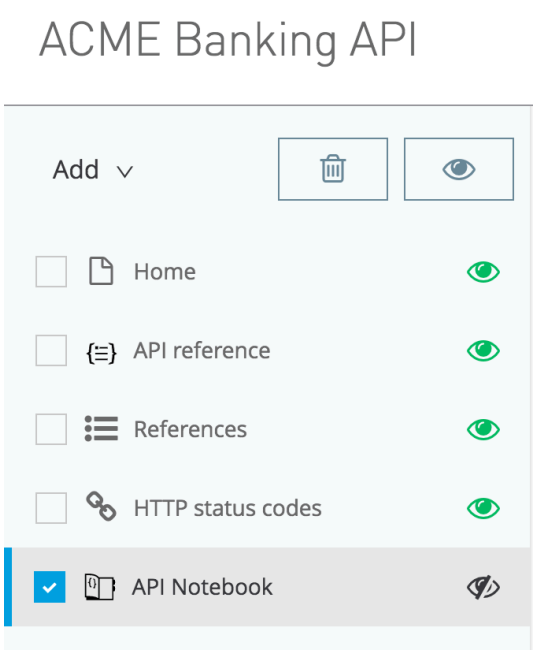
```
14 function concatAccountBalance(accountA) {
15   return "The accountBalance is "+ accountA.body.accountBalance.currency.concat(" "+
   accountA.body.accountBalance.amount);
16 }
17 concatAccountBalance(accountA);
```

```
$8=   "The accountBalance is USD 8457.90"
```

50. Scroll up in the API Notebook area and click Save.
51. Click the link ACME Banking API (1.0) – Settings to go back to the API version details page.
52. In the API version details page, click Edit API in API Designer inside the API Definition panel.

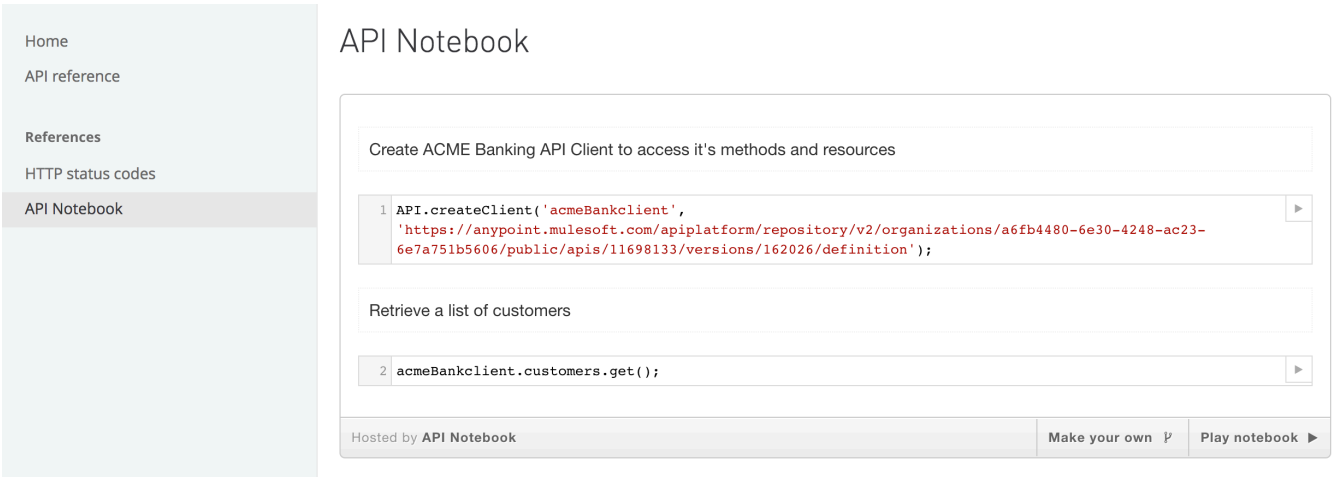
View the API Notebook in the Live Portal

53. In the left-side navigation, check the box to select the API Notebook and click the eye button at the top of the section.

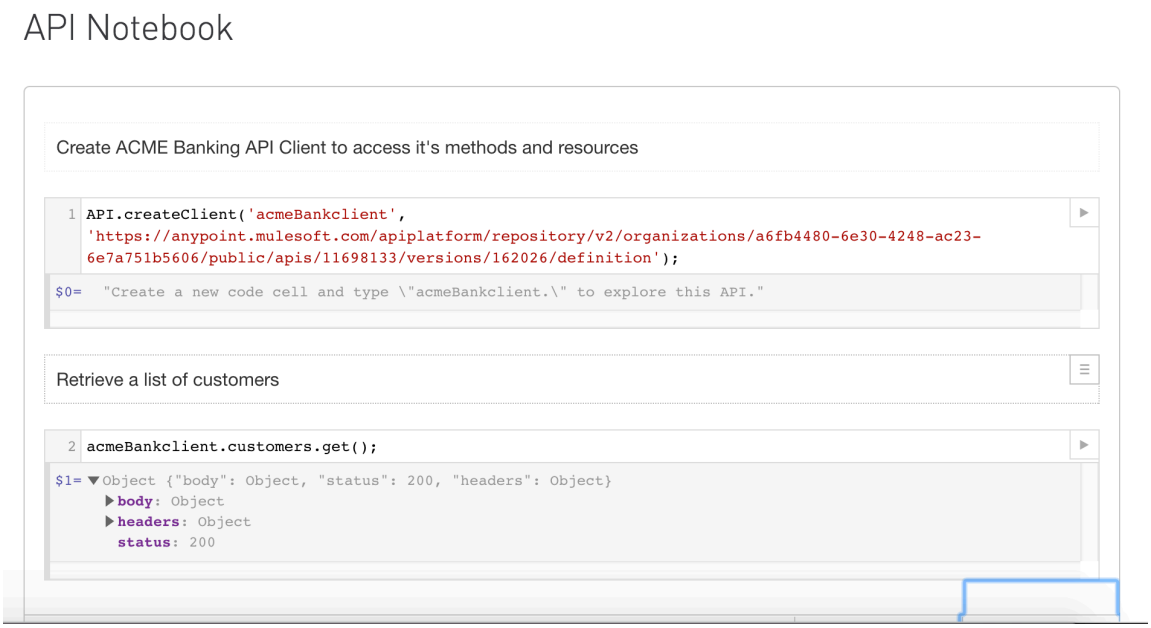


54. Click the Live Portal button in the top-right corner of API Portal Designer.

55. In the left -side navigation in the API Portal, select API Notebook.



56. Click the Play notebook icon at the bottom to play all the code cells in the API Notebook.
57. Verify that you see the results returned below each cell in the API Notebook.

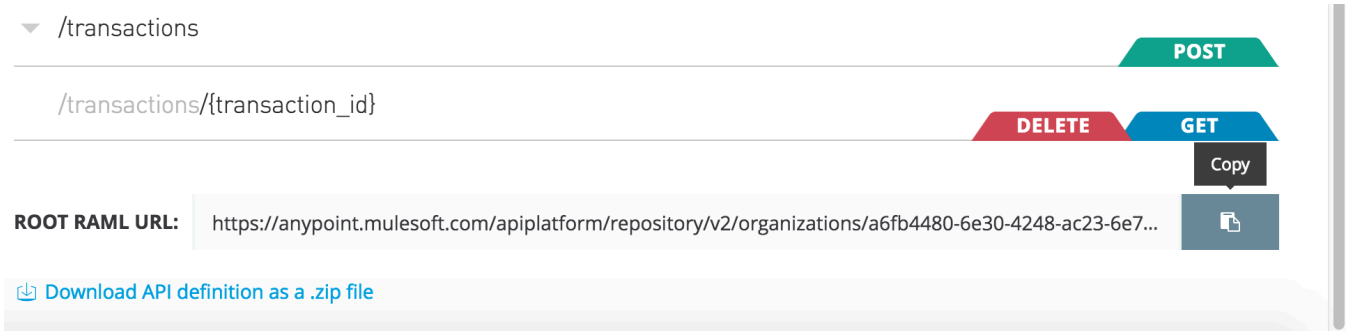


58. Go to the Home page in the API Portal and copy the Portal URL.
59. Return to the course snippets.text file.
60. Locate the Module 8 section and paste API Portal URL in the designated section.

```
API Portal URL: https://anypoint.mulesoft.com/apiplatform/mulesoft-1581/#/portals/organizations/a6fb4480-6e30-4248-ac23-6e7a751b5606/apis/11698133/versions/162026/pages/225830

RAML URL:
```

61. Return to the API Portal.
62. Select the API Reference page.

63. Scroll down to the bottom of the API Console and click the Copy button next to the Root RAML URL.

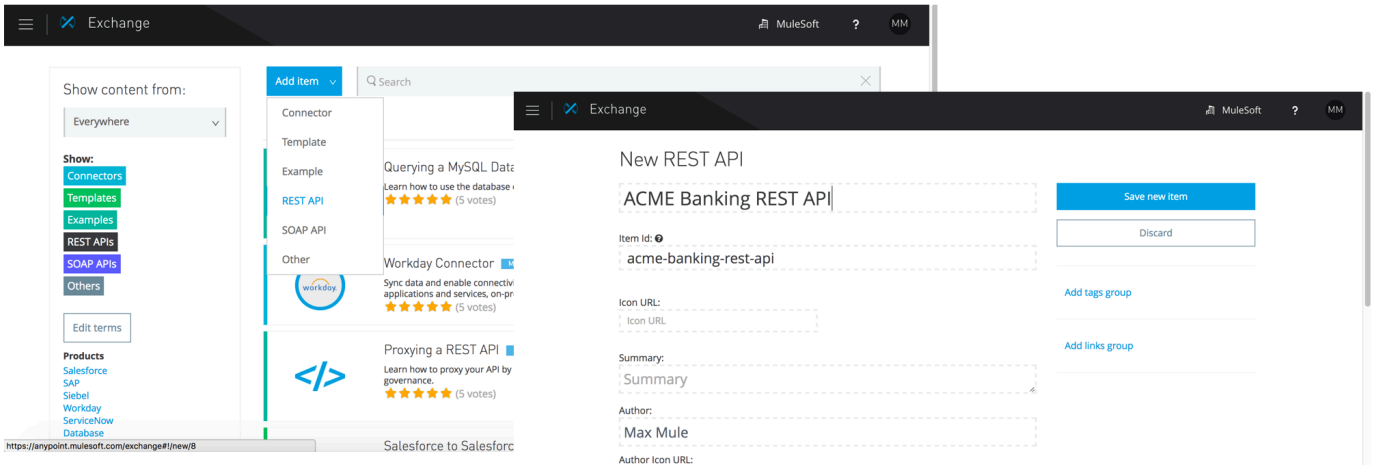


64. Return to the course snipepets.txt file.
65. Paste the RAML URL in the designated section.
66. Return to the API Portal and close it.

Walkthrough 8-3: Publish a RAML API definition to Anypoint Exchange

In this walkthrough, you enhance the discoverability of an API by adding it to the private Anypoint Exchange. You will:

- Give yourself permission to publish items on the Exchange.
- Add a new RAML API to the private Exchange.
- Submit an item for approval.
- Approve and publish an item to the private Exchange.



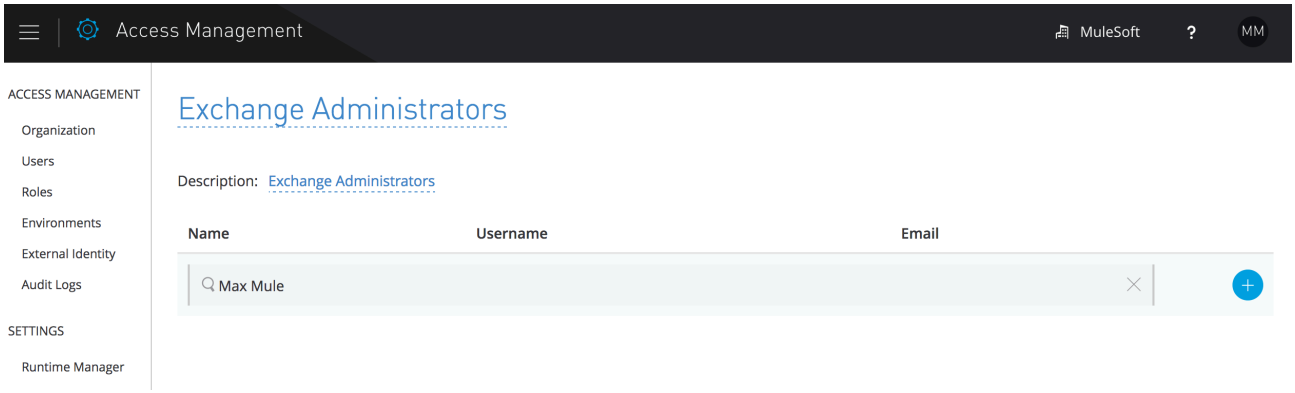
View Exchange roles

1. In the API Portal Designer, select Access Management from the main menu.
2. In the left-side navigation, click Roles.
3. View the Exchange roles.

○	Cloudhub Support (Production)	0	Cloudhub (Production) Support users
	Exchange Administrators	1	Exchange Administrators
	Exchange Contributors	0	Exchange Contributors
	Exchange Viewers	0	Exchange Viewers
	Organization Administrators	1	Organization Administrators
	Portals Viewer	0	Viewer of all portals in the organization

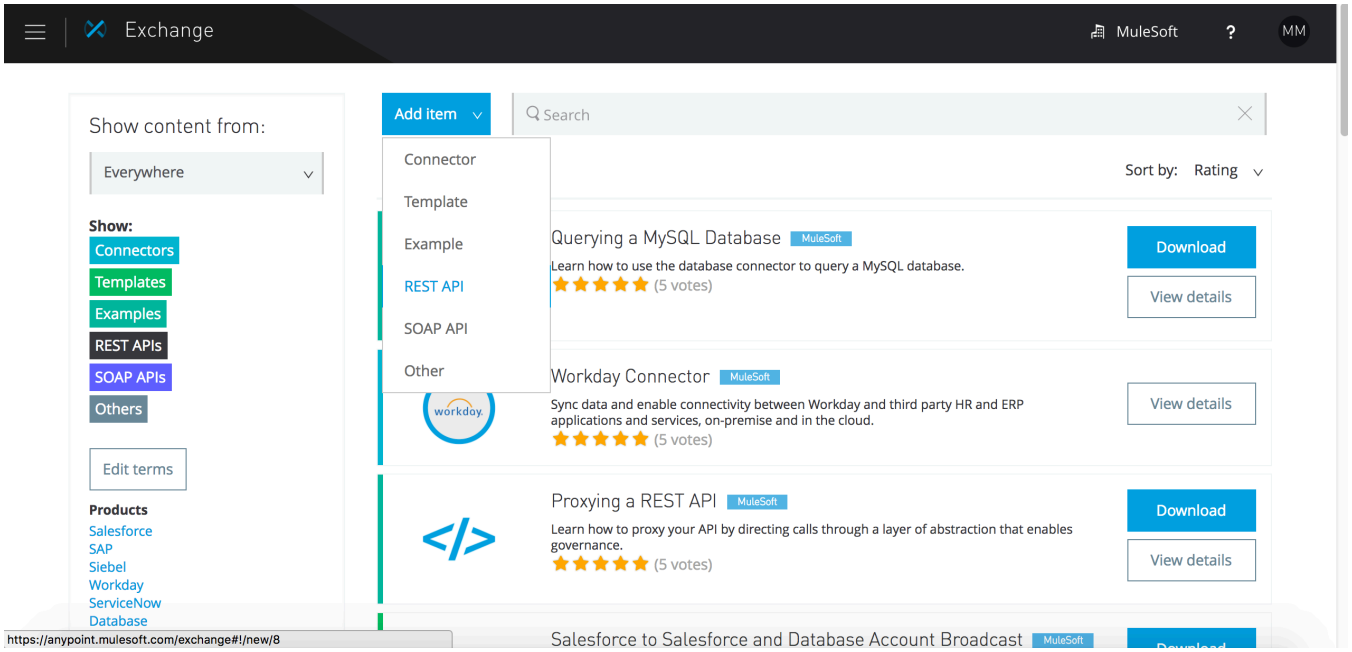
4. Click Exchange Administrators.

5. Click Exchange Administrators; you should see that you are already an Exchange Administrator.
6. Locate the add button that can be used to add additional people.

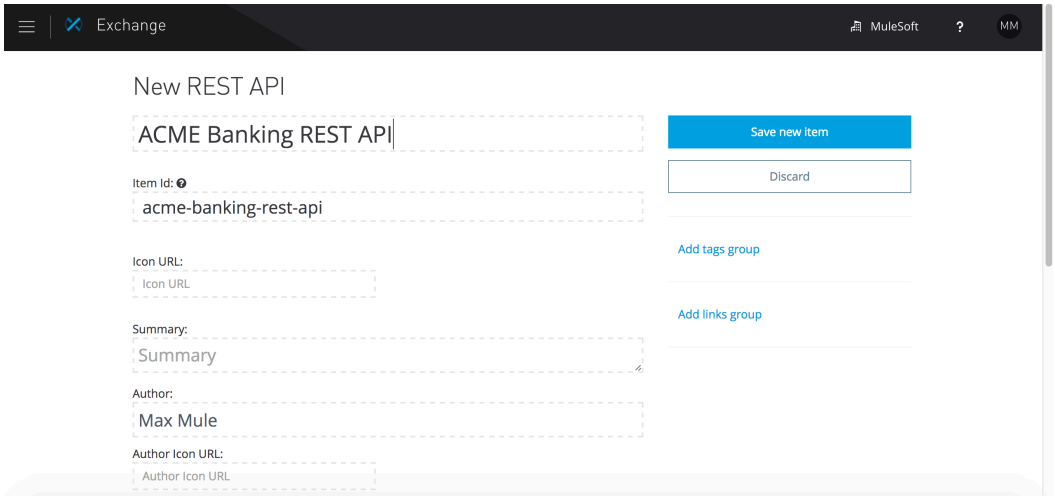


Create a new item to add to the Exchange

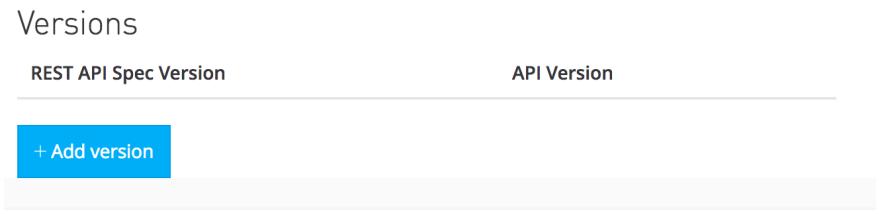
7. In the main menu, select Exchange.
8. In the Exchange, click the Add item button and select REST API.



9. Set the item name to ACME Banking REST API.

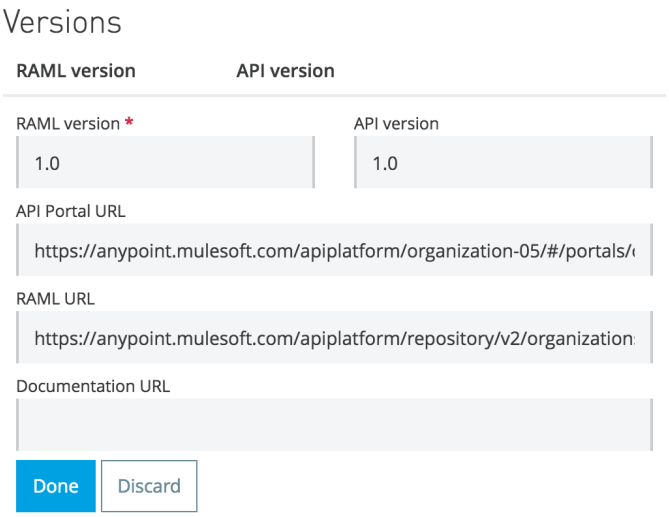


10. Scroll down to the versions section and click the Add version button.



11. Enter the following information:

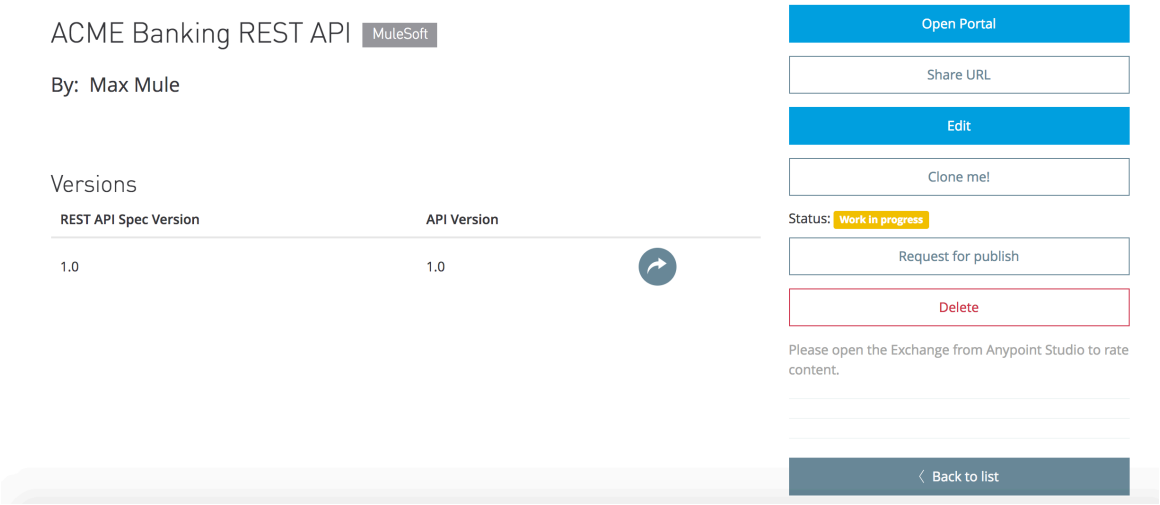
- REST API Spec version: 1.0
- API version: 1.0
- API Portal URL: *Use the URL for your portal that you saved in the course snippets.txt file*
- RAML URL: *Use the URL for your RAML that you saved in the course snippets.txt file*



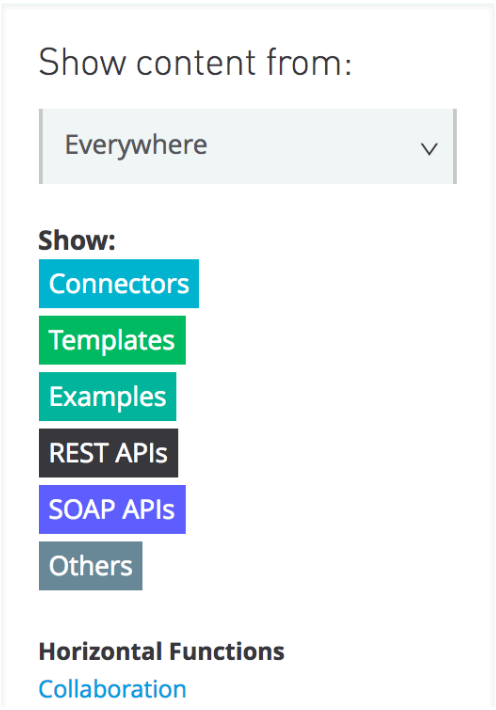
12. Click Done.
13. Scroll up and click the Save new item button.

Locate the item

14. Click the Back to list button.

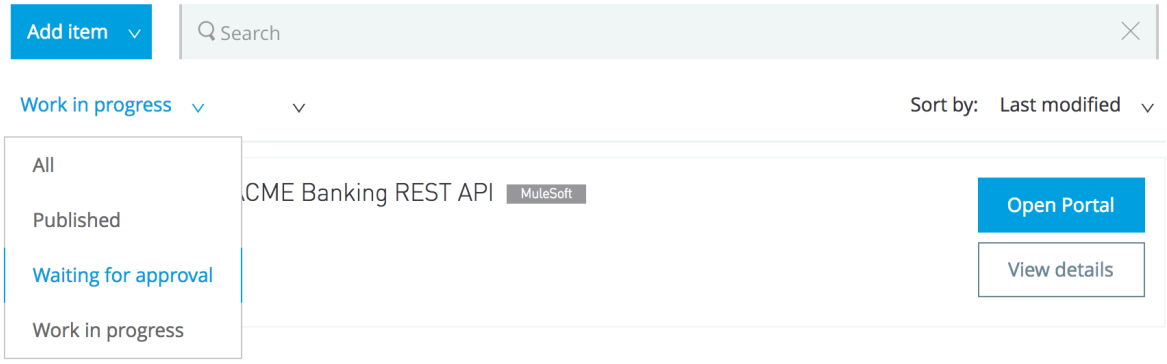


15. Click the REST APIs button in the upper-left corner of the Exchange.



16. In the Status menu beneath the Add item button, select Waiting for approval; you should see nothing listed.

17. Change the Status menu to Work in progress; you should the ACME Banking REST API.



Publish the item

18. Click the View details button for the ACME Banking REST API.
19. Locate the Work in progress status.
20. Click the drop-down button on the Publish button and select your organization.

    Note: Exchange contributors would have a Request to Publish instead of a Publish button and then an Exchange administrator would have to approve the item.

View the new item in the Exchange

21. In the Status menu beneath the Add item button, select Published; you should see the ACME Banking REST API listed.
22. Change the Status menu to All; you should see your RAML listed.

Module 9: Reusing Patterns

```
traits:
  cacheable: !include traits/cacheable.raml
  hasAcceptHeader: !include traits/hasAcceptHeader.raml

resourceTypes:
  collection: !include resourceTypes/collection.raml
  member: !include resourceTypes/member.raml

/accounts:
  type: { collection:
      { postRequestBodyDatatype: Account,
        postRequestBodyExample: !include examples/AccountExample.json,
        customErrorMessageType: CustomErrorMessage
      }
  }
  /{account_id}:
    type: { member:
      { uriParameterName: account_id,
        getResponseBodyDataType: Account,
        getResponseBodyExample: !include examples/AccountExample.json,
        customErrorMessageType: CustomErrorMessage
      }
    }
```

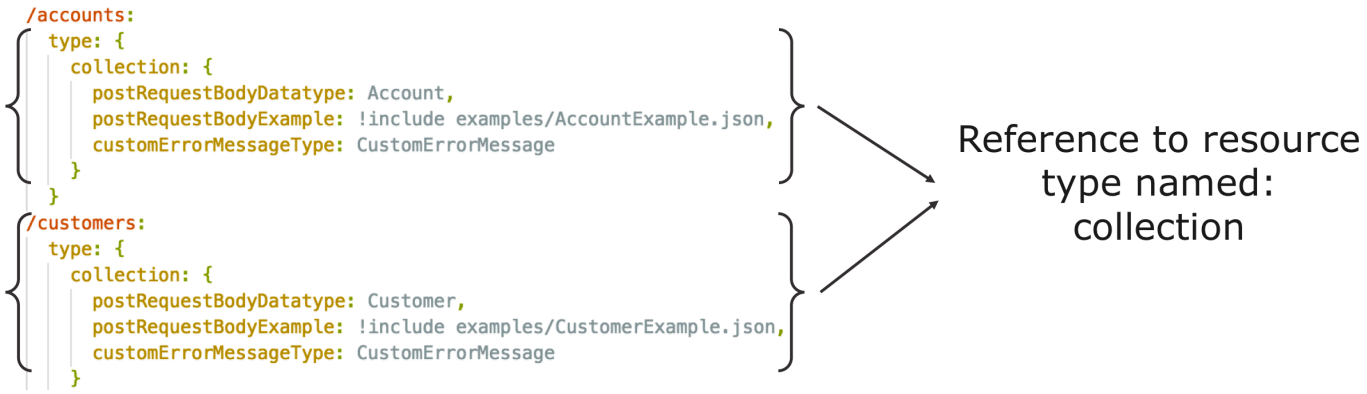
Objectives:

- Use resource types to reuse parts of method definitions.
- Use traits to enhance responses.
- Use the Concurrent Modification API design pattern when using shared resources.
- Use the Asynchronous API design pattern to handle and track concurrent requests.

Walkthrough 9-1: Define and use a resource type for resources that perform operations on a collection

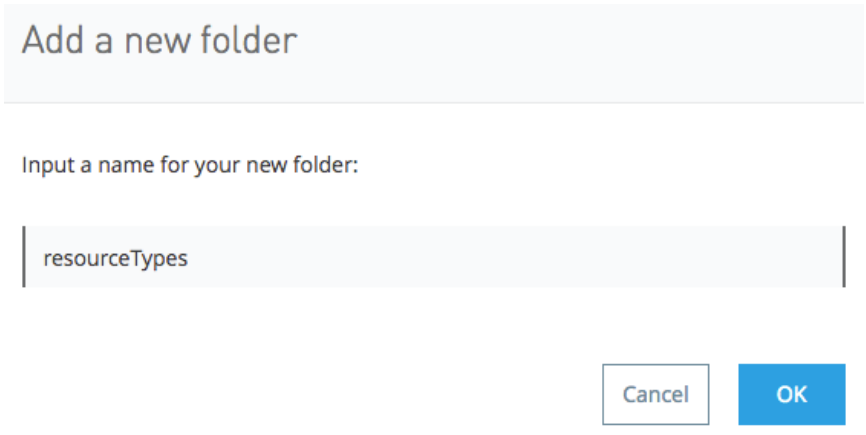
In this walkthrough, you define a collection resource type to reuse some parts of the method definition for resources that perform operations on a collection. You will:

- Define a collection resource type fragment.
- Use a mapping to pass parameter values to a resource type.
- Reference the resource type in the RAML API definition.



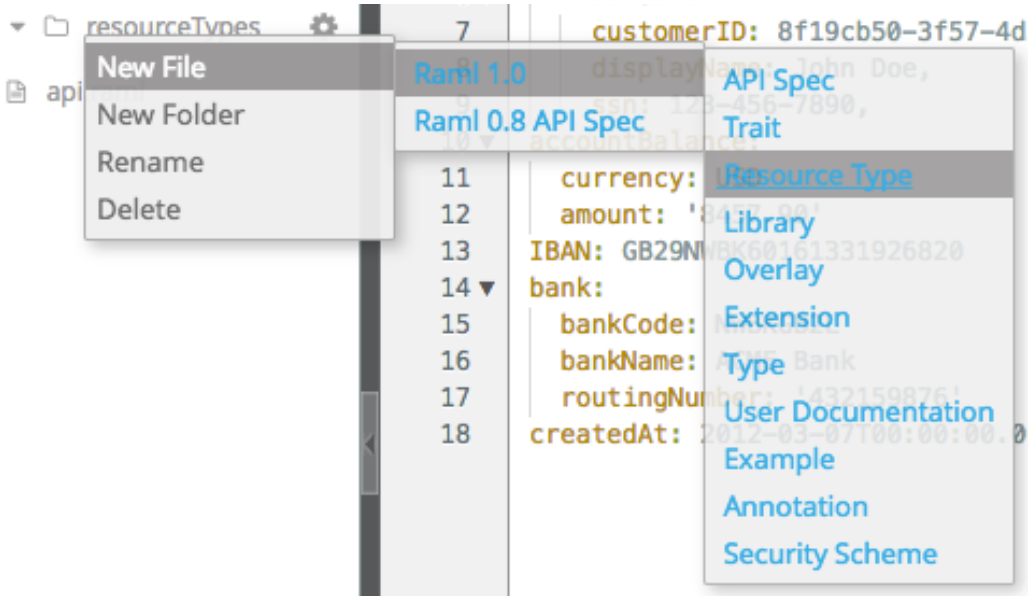
Define a collection resource type fragment

1. Return to API Designer.
2. Right-click inside the file browser section, and select New Folder.
3. In the Add a new folder dialog box, set the name of the folder to resourceTypes.

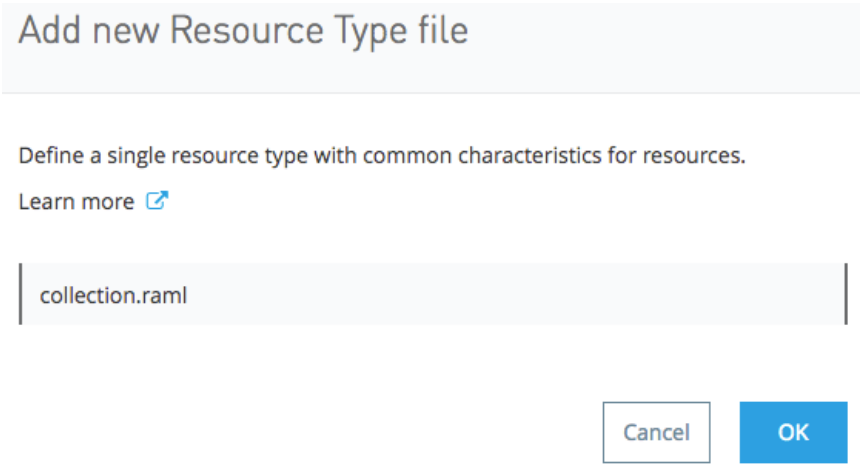


4. Click OK.

5. In the file browser, right-click the resourceTypes folder and select New File > RAML 1.0 > Resource Type.



6. In the Add new Resource Type file dialog box, rename the file as collection.raml.



7. Click OK.
8. In the file browser, select api.raml to open the RAML definition file.

9. In the /customers resource, select and copy the post method definition.

```
post:
  description: Add a new customer
  body:
    application/json:
      type: Customer
      example: !include examples/CustomerExample.raml
  responses:
    201:
      headers:
        Location:
          description: URL of the new customer information
      body:
        application/json:
    503:
      body:
        application/json:
          properties:
            statusCode: string
            message: string
```

10. In the file browser, expand the resourceTypes folder and click collection.raml.
11. Paste the lines after the first line in collection.raml.

```
#%RAML 1.0 ResourceType
post:
  description: Add a new customer
  body:
    application/json:
      type: Customer
      example: !include examples/CustomerExample.raml
  responses:
    201:
      headers:
        Location:
          description: URL of the new customer information
      body:
        application/json:
    503:
      body:
        application/json:
          properties:
            statusCode: string
            message: string
```

12. Select the pasted lines and press Shift + Tab to move a tab space to the left and properly indent the lines.

Use mappings to pass parameter values to resource types

13. In the collection.raml resource type file, go to the line that contains the description for the post method.

14. Replace the word customer with <<resourcePathName | !singularize>>

```
#%RAML 1.0 ResourceType
post:
  description: Add a new <<resourcePathName | !singularize>>
  body:
    application/json:
```

15. Similarly, in the description node for the Location header, replace the word customer with <<resourcePathName | !singularize>>.
16. Locate the line of code that sets the post request body type to Customer.
17. Delete the Customer datatype and set it to the following mapping:

```
<<resourcePathName | !singularize | !uppercamelcase>>
```

18. Go to the next line that defines the example for the post method's request body datatype and delete the example value and type <<postRequestBodyExample>>.

```
#%RAML 1.0 ResourceType
post:
  description: Add a new <<resourcePathName | !singularize>>
  body:
    application/json:
      type: <<postRequestBodyDatatype>>
      example: <<postRequestBodyExample>>
  responses:
    201:
```

19. Go to the line that contains the HTTP 503 response body type CustomErrorMessage and replace the type node value as <<customErrorMessageType>>.

```
    503:
      body:
        application/json:
          type: <<customErrorMessageType>>
```

20. Save the project.

Reference the resource type in the RAML API definition

21. In the file browser section, select api.raml.
22. In the /customers resource, go to the post method definition and delete it.

23. Add a new line before the /customers resource and add the resourceTypes node:

```
resourceTypes:
```

```
19
20  resourceTypes:
21  |
22▼ /customers:
```

24. Add a new line below and press tab.
25. Include the collection resourcetype by typing:

```
collection: !include resourceTypes/collection.raml
```

```
18    CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
19
20  resourceTypes:
21    collection: !include resourceTypes/collection.raml
22▼ /customers:
```

26. Press enter.
27. In the /customers resource, add a new line before the get method.

```
20▼ /customers:
21
22▼   get:
23      description: Retrieve a list of customers
24▼     headers:
```

28. Reference the type collection and add the mapping values by typing:

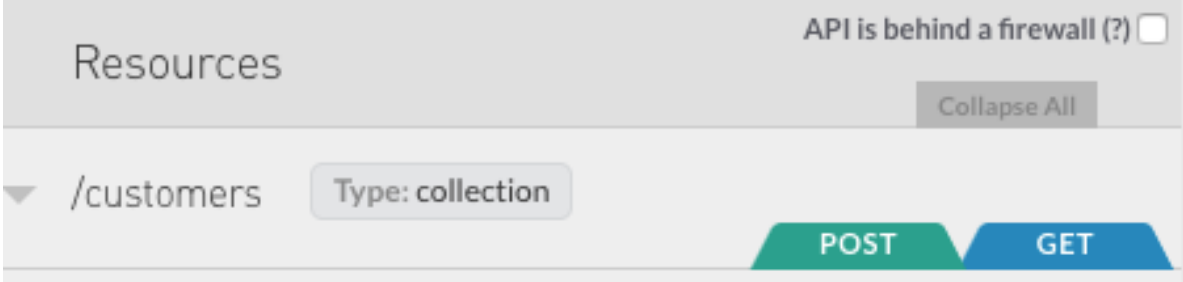
```
type: { collection: { postRequestBodyExample: !include
../examples/CustomerExample.raml, customErrorMessageType:
CustomErrorMessage } }
```

```
23▼ /customers:
24    type: { collection: { postRequestBodyExample: !include
      ../examples/CustomerExample.raml, customErrorMessageType:
      CustomErrorMessage } }
25▼   get:
```

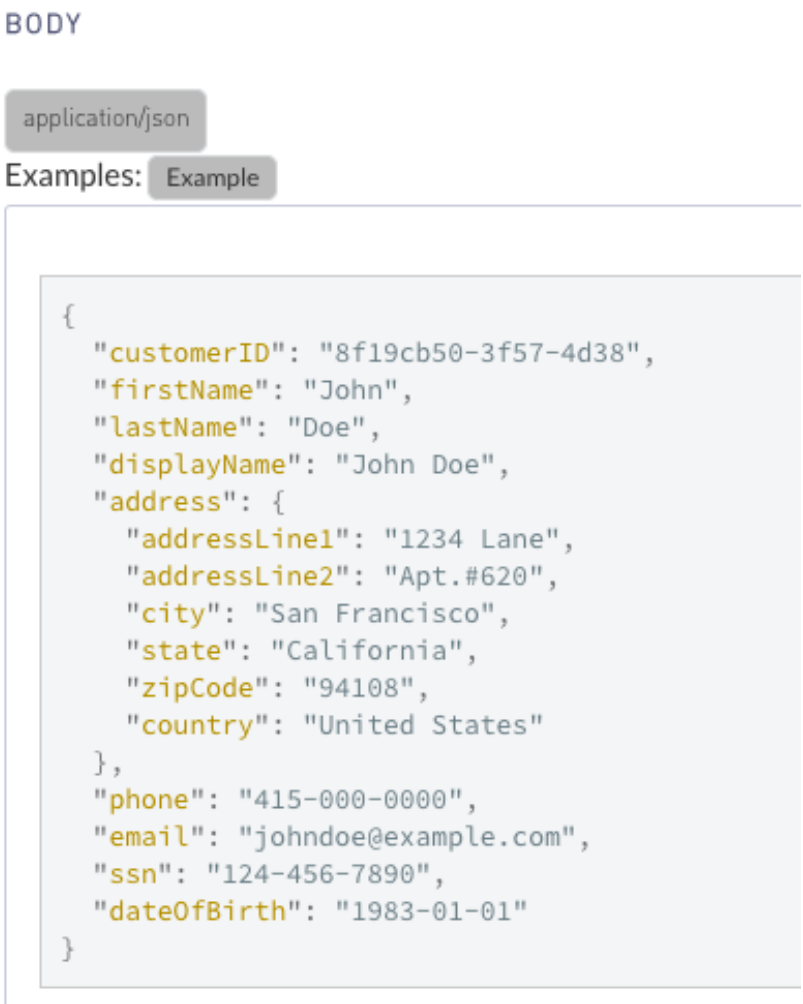
Note: Ignore the warning in the line that references the resource type collection. The warning is an issue in the parser caused because of including the root in the path for the example.

View the documentation

29. In the API Console, verify that you see the resource type listed across the /customers resource.



30. Click the POST method for the /customers resource and verify the example for the post request body is in JSON format.



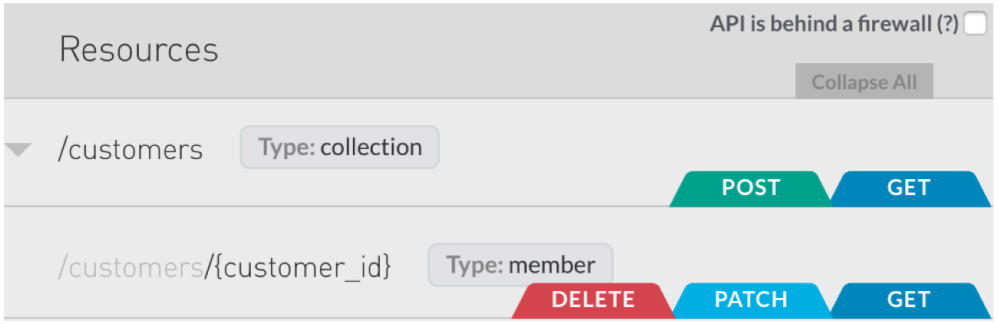
31. Click CLOSE X in the /customers resource.

Walkthrough 9-2: Define and use a resource type for resources that perform operations on a member referenced with an ID

In this walkthrough, you define a member resource type to reuse some parts of the method definition for resources that perform operations on a member. You will:

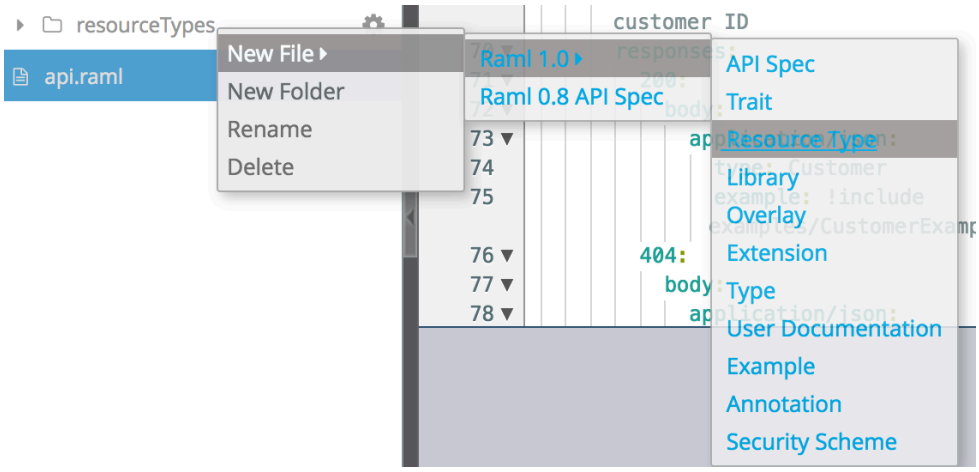
- Define a member resource type fragment.
- Use a mapping to pass parameter values to the resource type.
- Reference the resource type in the RAML API definition.

```
68          description: ID of the customer
69        type: { member: { getResponseBodyDataType: Customer,
          getResponseBodyExample: !include
          examples/CustomerExample.json,
          customErrorMessageType: CustomErrorMessage } }
70 ▼      patch:
71          description:
            information
72 ▼        responses:
```

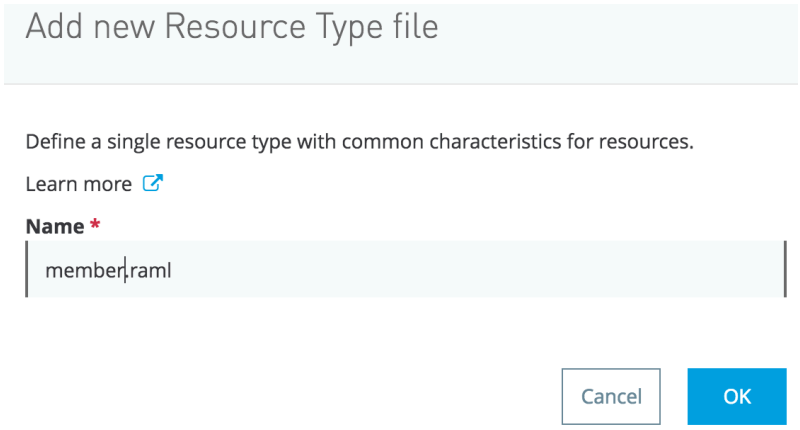


Define a member resource type fragment

1. Return to API Designer.
2. In the file browser, right-click the resourceTypes folder and select New File > RAML 1.0 > Resource Type.



3. In the Add new Resource Type file dialog box, set the name to member.raml and click OK.



4. In the file browser, select api.raml.
5. Locate the /{customer_id} resource and select the lines from the uriParameters to the end of the get method definition.

```
/{customer_id}:
  uriParameters:
    customer_id:
      description: ID of the customer
  get:
    description: Retrieve a customer with a specific customer ID
    headers:
      Accept?:
        description: Specify the media type of the response
        example: application/xml
    responses:
      200:
        body:
          application/json:
            type: Customer
            example: !include examples/CustomerExample.raml
          application/xml:
            type: Customer
            example: !include examples/CustomerExample.raml
      404:
        body:
          application/json:
            type: CustomErrorMessage
      406:
        body:
          application/json:
            type: CustomErrorMessage
```

6. In the file browser, click the resourceTypes/member.raml file.
7. Paste the copied lines after the first line in member.raml.

8. Select the pasted lines and press Shift+tab to indent the lines properly.

```
#%RAML 1.0 ResourceType
get:
  description: Retrieve a customer with a specific
  customer ID
  responses:
    200:
      body:
        application/json:
          type: Customer
          example: !include examples/CustomerExample.raml
    404:
      body:
        application/json:
          properties:
            statusCode: string
            message: string
```

Use mappings to pass parameter values to resource types

9. In the member.raml file, go to the line that contains the uriParameter customer_id and replace the variable with <<uriParameterName>>.

```
uriParameters:
  <<uriParameterName>>:
    description: ID of the customer
get:
```

10. In the next line that contains the description, replace customer with <<resourcePathName | !singularize>>.

```
#%RAML 1.0 ResourceType
uriParameters:
  <<uriParameterName>>:
    description: ID of the <<resourcePathName | !singularize>>
get:
```

11. Go to the line that contains the get method description and replace the word customer with <<resourcePathName | !singularize>>.

```
get:
  description: Retrieve a <<resourcePathName |
  !singularize>> with a specific <<resourcePathName |
  !singularize>> ID
  responses:
```

12. Go to the line that contains the HTTP 200 response body type Customer and delete the type value Customer, and replace it with <<resourcePathName | !singularize | !uppercamelcase>>.
13. Go to the next line and replace the value of the example node with <<getResponseBodyExample>>.

```
  responses:
    200:
      body:
        application/json:
          type: <<getResponseBodyDataType>>
          example: <<getResponseBodyExample>>
```

14. Go to the HTTP 404 response body and replace the CustomErrorMessage datatype with <<customErrorMessageType>>.
15. Similarly, in the HTTP 406 response body change the CustomErrorMessage datatype with the mapping

```
    404:
      body:
        application/json:
          type: <<customErrorMessageType>>
delete:
```

16. Save the project.

Reference the member resource type fragment in the RAML API definition

17. In the file browser, select api.raml.
18. Go the /{customer_id} resource and delete the lines from the uriParameters to the get method definition.

```
/{customer_id}:
  uriParameters:
    customer_id:
     description: ID of the customer
  patch:
    description: Update a specific customer
    information
    responses:
      204:
      501:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
```

19. Go to the resource type definition node above the /customers resource and add a new line below the collection resource type.

20. In the new line type:

    member: !include resourceTypes/member.raml

```
20 ▼ resourceTypes:
21     collection: !include resourceTypes/collection.raml
22     member: !include resourceTypes/member.raml
```

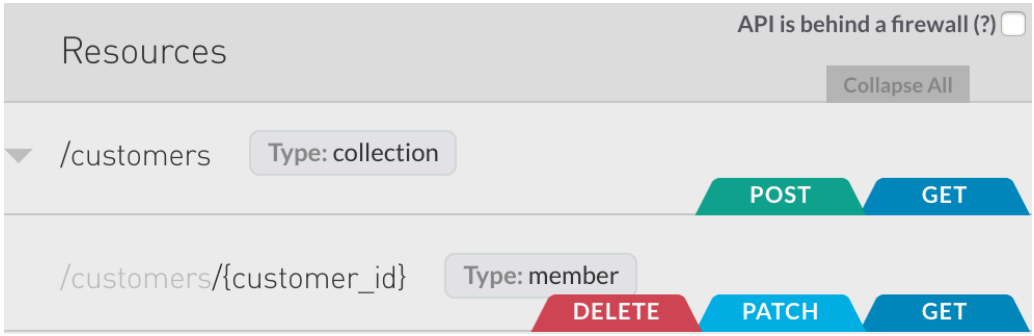
    Note: Make sure the indentation aligns with the previous collection resource type reference.

21. In the /{customer_id} resource go to the end of the line that contains the description for the URI parameter and press enter to add a new line below it.
22. Press backspace or delete two time to align the indentation with the next line and type the following (or copy from the snippets.txt):

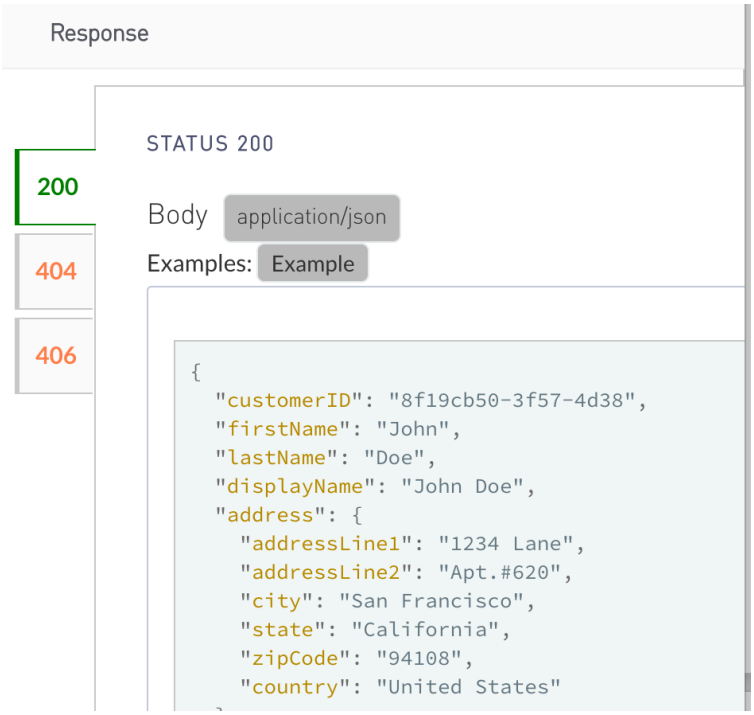
```
type: { member: { getResponseBodyExample: !include
../examples/CustomerExample.raml, customErrorMessageType:
CustomErrorMessage } }
```

```
61 ▼ /{customer_id}:
62     type: { member: { uriParameterName: customer_id,
       getResponseBodyExample: !include ../examples/CustomerExample.raml,
       customErrorMessageType: CustomErrorMessage } }
63 ▼   patch:
64       description: Update a specific customer information
```

23. Save the project.
24. In API Console, verify that the /customers/{customer_id} resource contains a type: member tag.



25. Click GET and scroll down and verify that you see the HTTP 200 response message example.

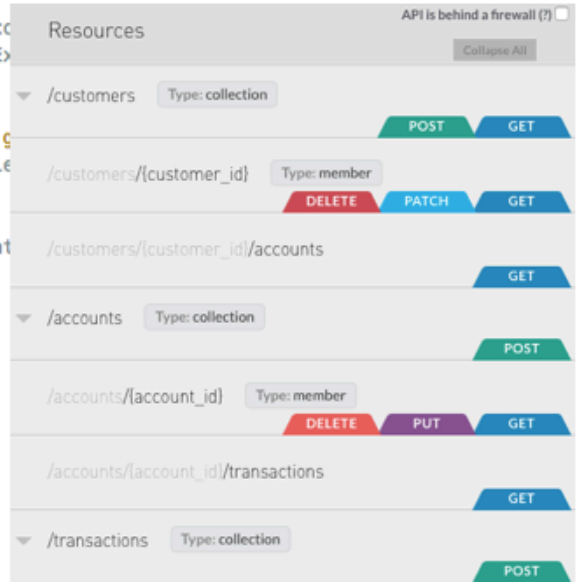


Walkthrough 9-3: Refactor resources to use resource types

In this walkthrough, you refactor the ACME Banking API. You will:

- Refactor the resource methods by referencing the already created resource types.

```
94 ▼ /accounts:
95     type: { collection: { postRequestBodyDatatype: Acc
       postRequestBodyExample: !include examples/AccountEx
       customErrorMessageType: CustomErrorMessage } }
96 ▼   /{account_id}:
97       type: { member: { uriParameterName: account_id, g
         Account, getResponseBodyExample: !include example
         customErrorMessageType: CustomErrorMessage } }
98 ▼     put:
99         description: Update a specific account informat
100 ▼      body:
101 ▼        application/json:
102            type: Account
```



Refactor /accounts resource

1. Return to API Designer.
2. In api.raml, go to the /accounts main resource and delete the post method definition.
3. Return to the course snippets.text file and copy the lines for the Collection resource type for /accounts resource.

```
108 ▼ /accounts:
109     type: { collection: { postRequestBodyExample: !include
        ../examples/AccountExample.raml, customErrorMessageType:
        CustomErrorMessage } }
110 ▼   /{account_id}:
```

Refactor /accounts/{account_id} resource

4. In the /{account_id} nested resource, delete the lines starting from the uriParameters to the end of the get method definition.

```
/{account_id}:
  uriParameters:
    account_id:
      description: ID of the account
  get:
    description: Retrieve an account information with a specific account ID
    responses:
      200:
        body:
          application/json:
            type: Account
            example: !include examples/AccountExample.raml
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
  delete:
    description: Delete an account with a specific account ID
    responses:
      204:
      200:
        body:
          application/json:
  put:
```

5. Copy the lines from the course snippets.txt file for the Member resource type reference code for /{account_id} resource.

```
/{account_id}:
  type: { member: { uriParameterName: account_id,
  getResponseBodyExample: !include ../examples/AccountExample.raml,
  customErrorMessageType: CustomErrorMessage } }
  delete:
```

Refactor /transactions resource

6. Go to the /transactions main resource and delete the post method definition.
7. Copy the lines from the course snippets.txt for the Collection resource type reference code for /transactions resource.

```
/transactions:
  type: { collection: { postRequestBodyExample: !include
  ../examples/TransactionExample.raml, customErrorMessageType:
  CustomErrorMessage } }
  /{transaction_id}:
```

Refactor /transactions/{transaction_id} resource

8. In the /{transaction _id} nested resource, delete the lines starting from the uriParameters to the end of the get method definition.

```
/{transaction_id}:
  uriParameters:
    transaction_id:
      description: ID of the transaction
  get:
    description: Retrieve a transaction with a specific transaction ID
    responses:
      200:
        body:
          application/json:
            type: Transaction
            example: !include examples/TransactionExample.raml
      404:
        body:
          application/json:
            properties:
              statusCode: string
              message: string
  delete:
    description: Delete a specific transaction
    responses:
      204:
      200:
        body:
          application/json:
```

9. Copy the lines from the course snippets.txt file for the Member resource type reference code for /{transaction_id} resource.

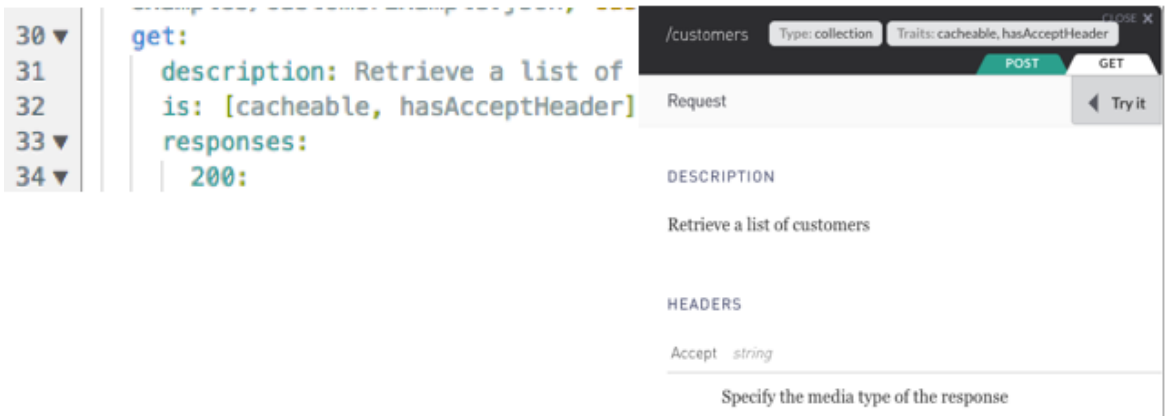
```
/{transaction_id}:
  type: { member: { uriParameterName: transaction_id,
  getResponseBodyExample: !include
  ../examples/TransactionExample.raml, customErrorMessageType:
  CustomErrorMessage } }
```

10. Save the project.

Walkthrough 9-4: Define and use various traits for resources and methods

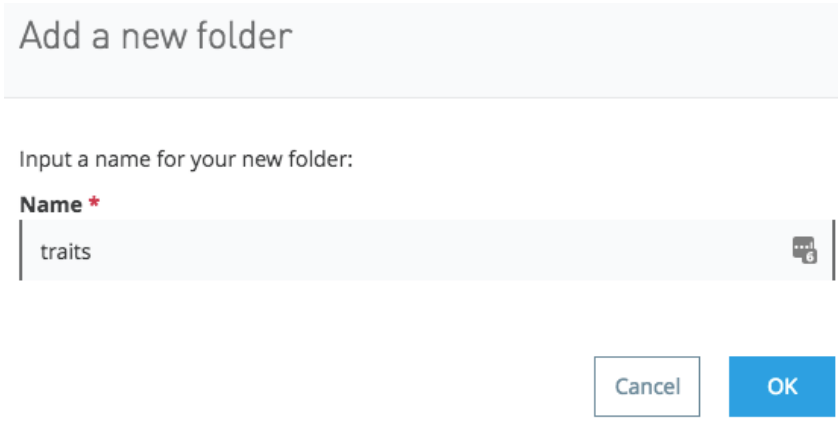
In this walkthrough, you refactor the API definition by creating reusable traits for resources and methods. You will:

- Define a cacheable trait to be applied to the /customers resource get method.
- Define a flexible content type trait to be applied to resource methods with an Accept header in the request.
- Refactor the resource methods to use the trait.



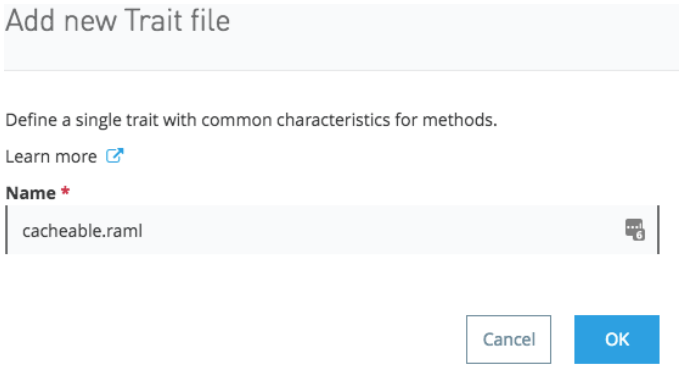
Define a cacheable trait to be applied to the /customers resource get method

1. Return to API Designer.
2. From the Project menu, select New Folder.
3. In the Add new folder dialog box, set the name to traits and click OK.



4. In the file browser, right-click the traits folder and select New File > RAML 1.0 > Trait.

5. In the Add new Trait file dialog box, set the name to cacheable.raml and click OK.



6. In the file browser, select api.raml.
7. Go to the /customers resource get method HTTP responses line.
8. Select the lines from responses to the Expires header format example line and copy them.

```
32 ▼   responses:
33 ▼     200:
34 ▼       headers:
35 ▼         Cache-Control:
36 ▼           description: |
37               Activates caching and defines cache behavior through cache response directives.
38               Usually defines public or private (cacheable by proxy or not) and max-age for resource.
39               See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
40           example: private, max-age=31536000
41 ▼         Expires:
42 ▼           description: |
43               Sets a date in RFC 1123 format from which the cached resource should no longer be considered
                 valid.
44               If both the Expires header and max-age in the Cache-Control header are set, max-age will take
                 precedence.
45           type: datetime
46           example: Tue, 18 Apr 2017 09:30:41 GMT
47           format: rfc2616
```

9. In the file browser, select traits/cacheable.raml.
10. Paste the copied lines, in the second line and correct the indentation for the lines.

```
1  #%RAML 1.0 Trait
2 ▼ responses:
3 ▼   200:
4 ▼     headers:
5 ▼       Cache-Control:
6 ▼         description: |
7             Activates caching and defines cache behavior through cache response directives.
8             Usually defines public or private (cacheable by proxy or not) and max-age for resource.
9             See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
10        example: private, max-age=31536000
11 ▼      Expires:
12 ▼        description: |
13            Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid.
14            If both the Expires header and max-age in the Cache-Control header are set, max-age will take
              precedence.
15        type: datetime
16        example: Tue, 18 Apr 2017 09:30:41 GMT
17        format: rfc2616
```

11. Go to the beginning of the responses line and add a press enter to add a new line above it.
12. Go to the new line and In the shelf, click usage.

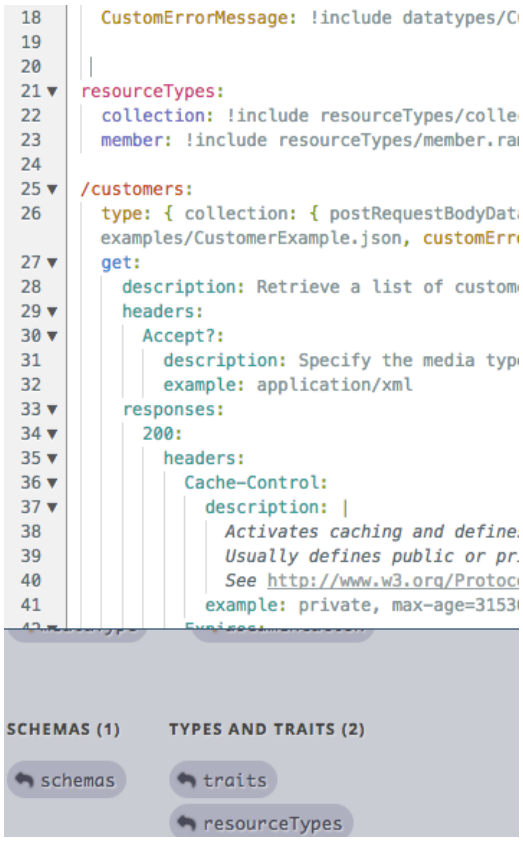
13. Type the value for the usage node as: Apply this trait to any GET method that supports caching control.

```
#%RAML 1.0 Trait
usage: Apply this trait to any GET method that supports caching control.
responses:
  200:
    headers:
      Cache-Control:
        description: |
          Activates caching and defines cache behavior through cache response directives.
          Usually defines public or private (cacheable by proxy or not) and max-age for resource.
          See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
        example: private, max-age=31536000
      Expires:
        description: |
          Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid.
          If both the Expires header and max-age in the Cache-Control header are set, max-age will take
          precedence.
        type: datetime
        example: Tue, 18 Apr 2017 09:30:41 GMT
        format: rfc2616
```

14. Save the project.

Refactor the /customers get method to apply the cacheable trait

15. In the file browser, select api.raml.
16. In the RAML editor, add a new line above the resourceTypes declaration.
17. In the shelf, click traits.



18. In the new line type the following to add the cacheable trait to the definition:

```
cacheable: !include traits/cacheable.raml
```

```
20  traits:
21    cacheable: !include traits/cacheable.raml
22  resourceTypes:
23    collection: !include resourceTypes/collection.raml
24    member: !include resourceTypes/member.raml
```

19. Press enter.
20. In the /customers resource get method HTTP 200, select lines from headers to the line before the body definition and delete them.

```
34      responses:
35        200:
36          headers:
37            Cache-Control:
38              description: |
39                Activates caching and defines cache behavior through cache response directives.
40                Usually defines public or private (cacheable by proxy or not) and max-age for resource.
41                See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
42              example: private, max-age=31536000
43            Expires:
44              description: |
45                Sets a date in RFC 1123 format from which the cached resource should no longer be considered
                  valid.
46                If both the Expires header and max-age in the Cache-Control header are set, max-age will take
                  precedence.
47              type: datetime
48              example: Tue, 18 Apr 2017 09:30:41 GMT
49              format: rfc2616
50          body:
```

21. Add a new line below the line that contains the description for the get method.
22. In the shelf below, click is.

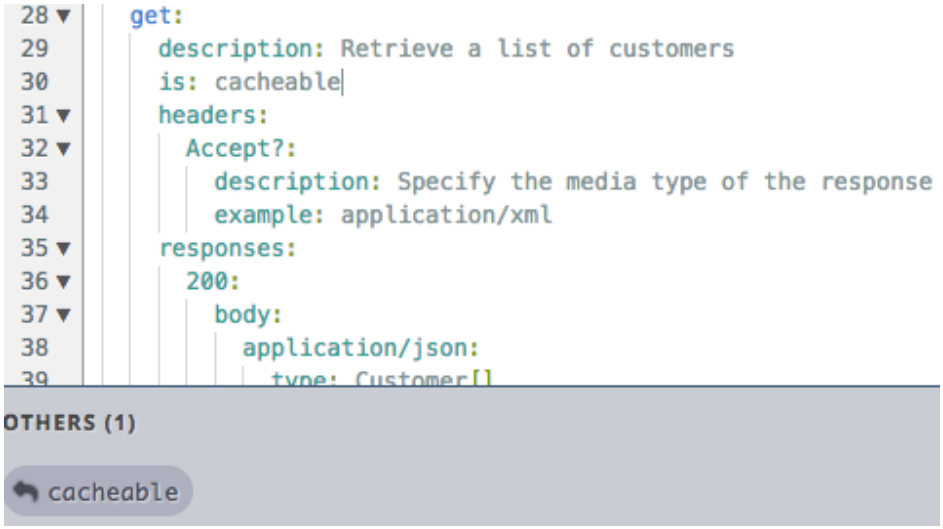
```
26  /customers:
27    type: { collection: { postRequestBodyDatatype: Customer, postRequestBodyExample: !include
      examples/CustomerExample.json, customErrorMessageType: CustomErrorMessage } }
28    get:
29      description: Retrieve a list of customers
30      |
31      headers:
32        Accept?:
33          description: Specify the media type of the response
34          example: application/xml
35      responses:
36        200:
37          body:
38            application/json:
39              type: Customer[]
```

protocols displayName queryParameters responses securedBy
headers

BODY (1) TYPES AND TRAITS (1)

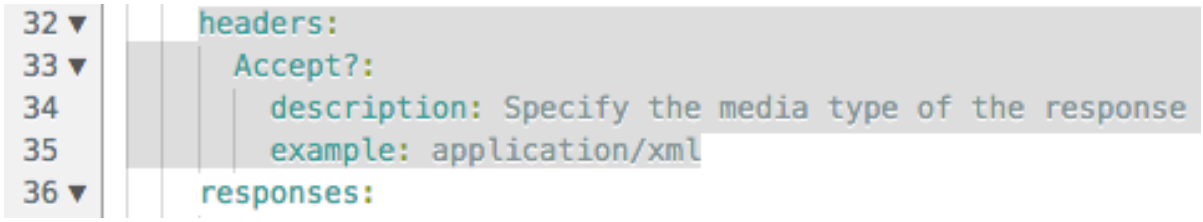
body is

23. In the shelf, click cacheable.

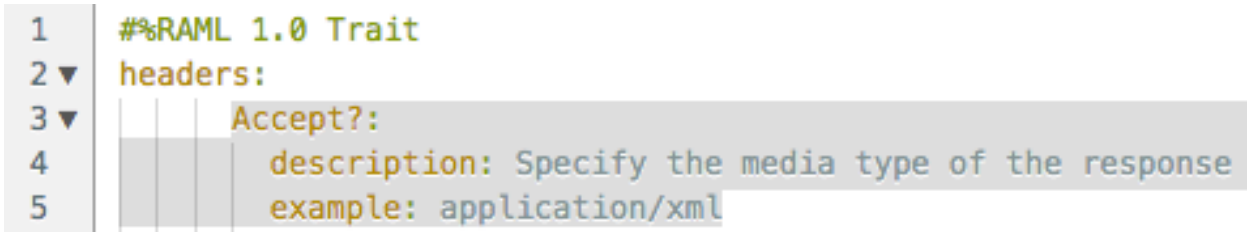


Define a flexible content type trait to be applied to resource methods with an Accept header in the request

24. In the file browser, right-click the traits folder and select New File > RAML 1.0 > Trait.
25. In the Add new trait file dialog box, set the name to hasAcceptHeader.raml and click OK.
26. In the file browser, select api.raml.
27. Go to the /customers resource get method.
28. Select the lines from headers to the example line for the Accept header and copy them.



29. In the file browser, select hasAcceptHeader.raml and paste the lines in the second line.



30. Fix the indentation and save the file.
31. In the file browser, select api.raml
32. Add a new line after the line that includes the cacheable trait.

33. Include the hasAcceptHeader trait.

```
traits:
  cacheable: !include traits/cacheable.raml
  hasAcceptHeader: !include traits/hasAcceptHeader.raml

resourceTypes:
  collection: !include resourceTypes/collection.raml
  member: !include resourceTypes/member.raml
```

34. Go to the /customers resource get method and delete the lines that define the Accept header for the request.

```
/customers:
  type: { collection: { postRequestBodyDa
  examples/CustomerExample.json, customEr
  get:
    description: Retrieve a list of custo
    is: cacheable
    responses:
      200:
        body:
```

35. In the /customers resource get method, go to the line that contains the cacheable trait applied and change the line to:

```
is: [cacheable, hasAcceptHeader]
```

```
  get:
    description: Retrieve a list of customers
    is: [cacheable, hasAcceptHeader]
    responses:
      200:
```

36. Similarly, in the /accounts nested resource get method, delete the Accept header definition lines and add the trait.

```
  /accounts:
    get:
      description: Retrieve a list of accounts for a specific customer
      is: hasAcceptHeader
      responses:
```

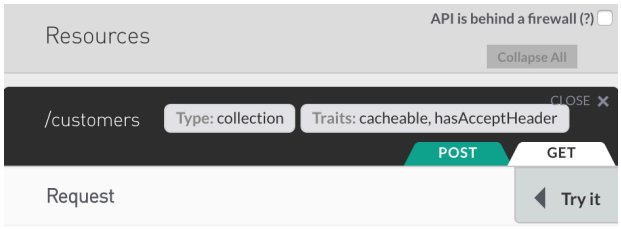
37. Similarly, in the /transactions nested resource get method, delete the Accept header definition lines and add the trait.

```
123 ▼    /transactions:
124 ▼      get:
125          description: Retrieve a list of
             transactions for a specific account
126          is: hasAcceptHeader
127 ▼        responses:
```

38. Save the project.

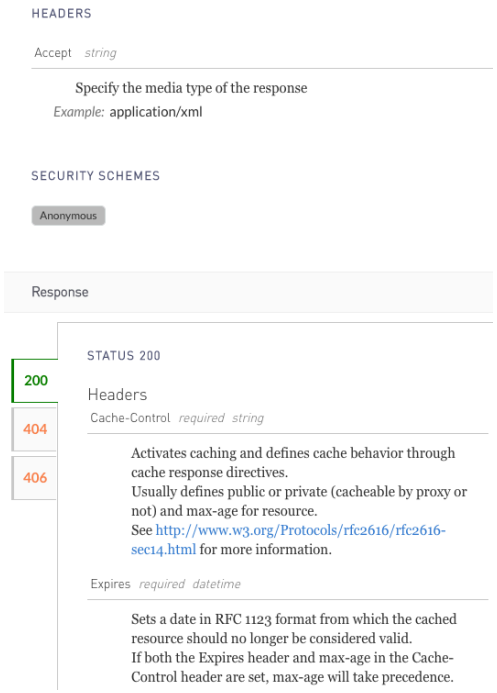
View the documentation

39. In the API Console, click GET for the /customers resource.



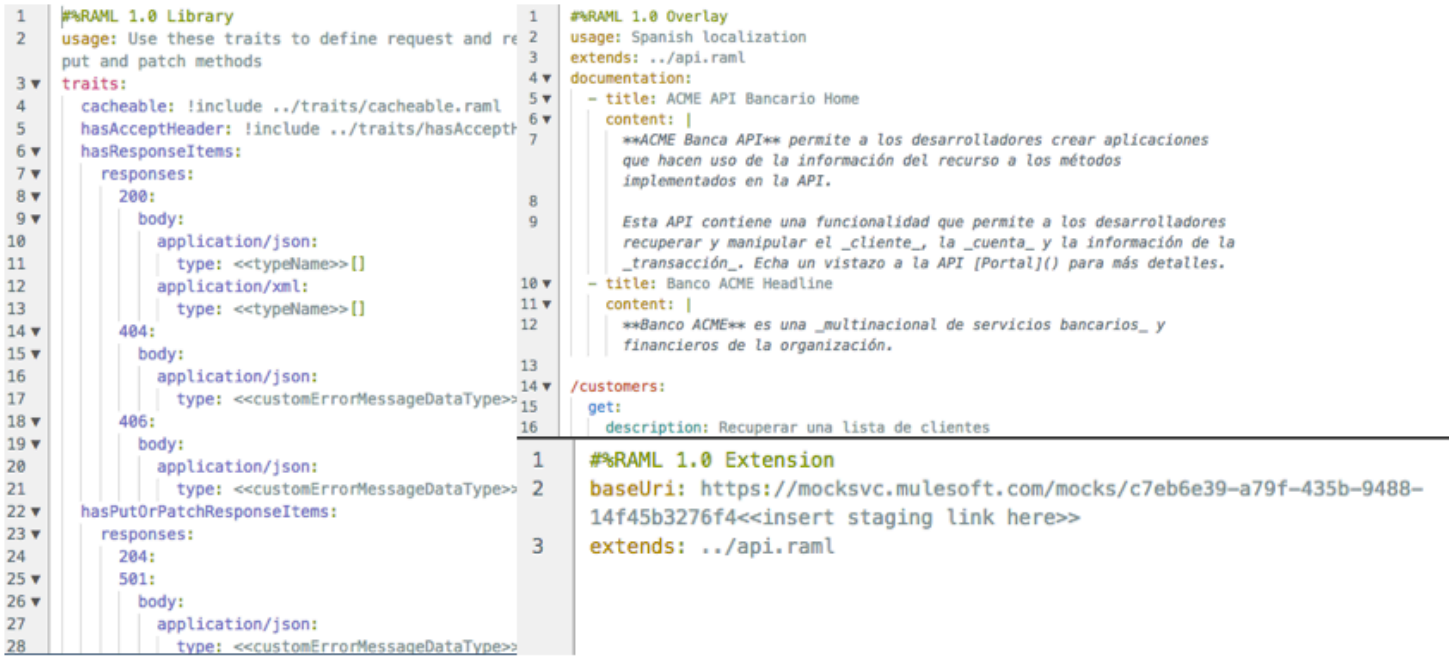
DESCRIPTION

40. Verify that you can view the Accept header in the request and the cache-control headers that were added using traits in the HTTP 200 response.



41. Click CLOSE X in the /customers resource.

Module 10: Modularizing APIs



```
#%RAML 1.0 Library
usage: Use these traits to define request and re
put and patch methods
traits:
  cacheable: !include ../traits/cacheable.raml
  hasAcceptHeader: !include ../traits/hasAcceptH
  hasResponseItems:
    responses:
      200:
        body:
          application/json:
            type: <<typeName>>[]
          application/xml:
            type: <<typeName>>[]
      404:
        body:
          application/json:
            type: <<customErrorMessageDataType>>
      406:
        body:
          application/json:
            type: <<customErrorMessageDataType>>
  hasPutOrPatchResponseItems:
    responses:
      204:
      501:
        body:
          application/json:
            type: <<customErrorMessageDataType>>
```

```
#%RAML 1.0 Overlay
usage: Spanish localization
extends: ../api.raml
documentation:
  - title: ACME API Bancario Home
    content: |
      **ACME Banca API** permite a los desarrolladores crear aplicaciones
      que hacen uso de la información del recurso a los métodos
      implementados en la API.

      Esta API contiene una funcionalidad que permite a los desarrolladores
      recuperar y manipular el _cliente_, la _cuenta_ y la información de la
      _transacción_. Echa un vistazo a la API [Portal]() para más detalles.
  - title: Banco ACME Headline
    content: |
      **Banco ACME** es una _multinacional de servicios bancarios_ y
      financieros de la organización.

/customers:
  get:
    description: Recuperar una lista de clientes
```

```
#%RAML 1.0 Extension
baseUri: https://mocksvc.mulesoft.com/mocks/c7eb6e39-a79f-435b-9488-
14f45b3276f4<<insert staging link here>>
extends: ../api.raml
```

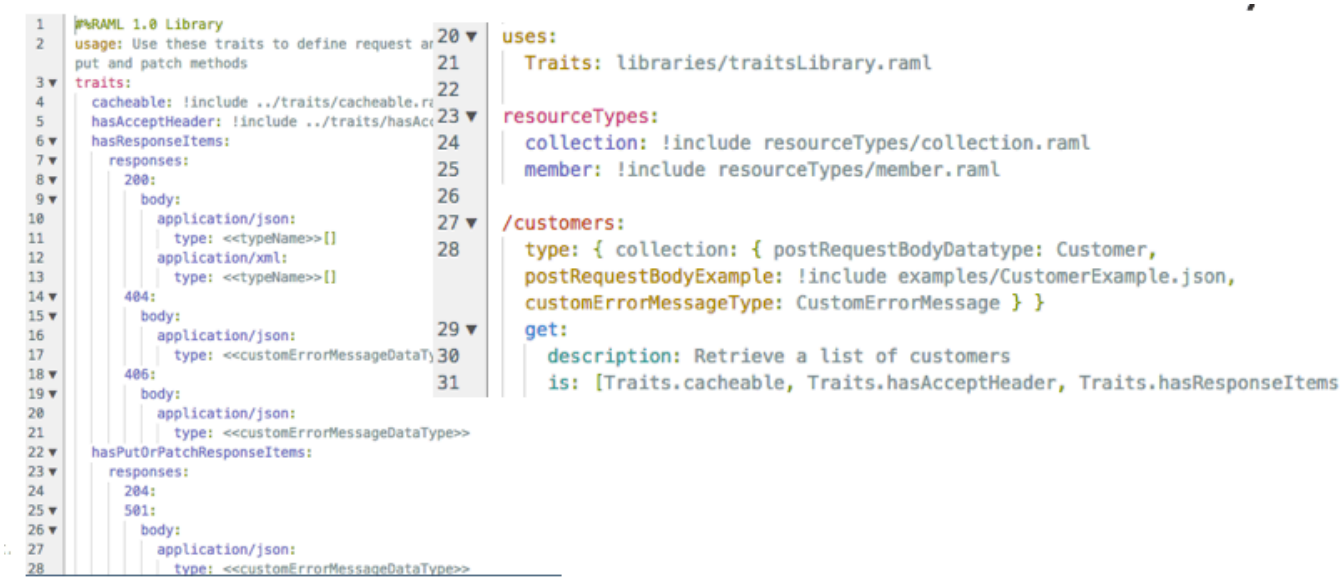
Objectives:

- Use libraries for greater API composability.
- Override resource information using overlays.
- Use overlays to internationalize resources.
- Use extensions to enhance resources.

Walkthrough 10-1: Create and use a library of traits

In this walkthrough, you will create a library file with several traits listed in them. You will:

- Create a library fragment file.
- Refactor and move existing traits inside the library file.
- Create and define new resource method request and response traits.
- Refactor the API definition to reuse the traits from the library.

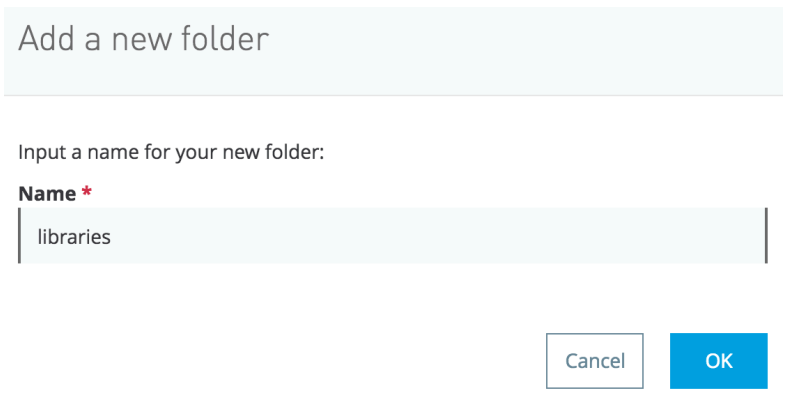


```
#%RAML 1.0 Library
usage: Use these traits to define request and
put and patch methods
traits:
  cacheable: !include ../traits/cacheable.raml
  hasAcceptHeader: !include ../traits/hasAcceptHeader.raml
  hasResponseItems:
    responses:
      200:
        body:
          application/json:
            type: <<typeName>>[]
          application/xml:
            type: <<typeName>>[]
      404:
        body:
          application/json:
            type: <<customErrorMessageDataType>>
      406:
        body:
          application/json:
            type: <<customErrorMessageDataType>>
  hasPutOrPatchResponseItems:
    responses:
      204:
      501:
        body:
          application/json:
            type: <<customErrorMessageDataType>>
```

```
uses:
  Traits: libraries/traitsLibrary.raml

resourceTypes:
  collection: !include resourceTypes/collection.raml
  member: !include resourceTypes/member.raml

/customers:
  type: { collection: { postRequestBodyDatatype: Customer,
  postRequestBodyExample: !include examples/CustomerExample.json,
  customErrorMessageType: CustomErrorMessage } }
  get:
    description: Retrieve a list of customers
    is: [Traits.cacheable, Traits.hasAcceptHeader, Traits.hasResponseItems
```

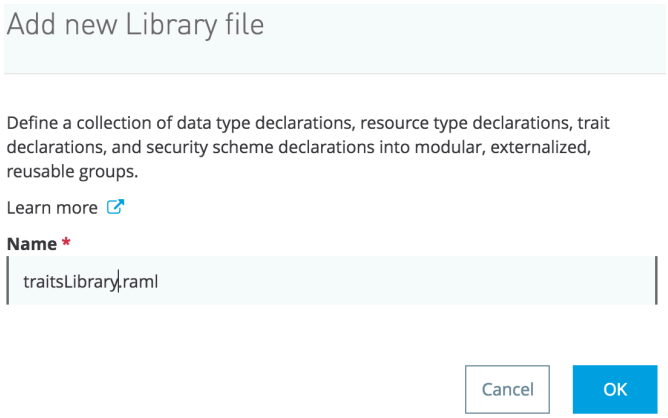
Create a library file

1. Return to API Designer.
2. From the Project menu, select New Folder.
3. In the Add a new folder dialog box, set the name of the folder to libraries and click OK.



4. In the file browser, right-click the libraries folder and select New File > RAML 1.0 > Library.

5. In the Add new Library file dialog box, set the name to traitsLibrary.raml and click OK.



6. In the RAML editor, go to the usage node and type the value as: Use these traits to define response structures for get and delete methods.

```
1  #%RAML 1.0 Library
2  usage: Use these traits to define request and response structures for get and delete methods
```

Define traits for response and request structures in the library file

7. Add a new line and in the shelf, click traits.
8. In the new line type hasResponseItems: and press enter.

```
1  #%RAML 1.0 Library
2  usage: Use these traits to define request and response structures
   for get, put and patch methods
3  traits:
4    hasResponseItems:
5
```

9. In the file browser, select api.raml.

10. Go to the /customers get method, and copy the lines from the responses to the HTTP 406 response application/json type CustomErrorMessage.

```
get:
  description: Retrieve a list of customers
  is: [cacheable, hasAcceptHeader]
  responses:
    200:
      body:
        application/json:
          type: Customer[]
        application/xml:
          type: Customer[]
    404:
      body:
        application/json:
          type: CustomErrorMessage
    406:
      body:
        application/json:
          type: CustomErrorMessage
/{customer_id}:
```

11. In file browser, select traitsLibrary.raml and paste the lines in the new line added to the end of the file.

```
usage: Use these traits to define request and response structures
for get, put and patch methods
traits:
  hasResponseItems:
    responses:
      200:
        body:
          application/json:
            type: Customer[]
          application/xml:
            type: Customer[]
      404:
        body:
          application/json:
            type: CustomErrorMessage
      406:
        body:
          application/json:
            type: CustomErrorMessage
```

Note: If necessary, correct the indentation in the pasted lines.

12. Go to the first line with the red error that contains the Customer array type.

13. Delete the value Customer[] and type <<resourcePathName | !singularize | !uppercamelcase>>[].
14. Replace the type Customer[] in the line under application/xml with << resourcePathName | !singularize | !uppercamelcase>>[].

```
hasResponseItems:
  responses:
    200:
      body:
        application/json:
          type: <<resourcePathName | !singularize | !uppercamelcase>>[]
        application/xml:
          type: <<resourcePathName | !singularize | !uppercamelcase>>[]
```

15. In the HTTP status 404 and 406 body type, delete the reference to CustomErrorMessage datatype, and type <<customErrorMessageDataType>>.

```
404:
  body:
    application/json:
      type: <<customErrorMessageDataType>>
406:
  body:
    application/json:
      type: <<customErrorMessageDataType>>
```

16. Add a new line below the HTTP 406 body definition, to define a new trait.
17. In the new line, type:

```
hasDeleteResponseItems:
```

```
18 ▼        406:
19 ▼          body:
20              application/json:
21                type: <<customErrorMessageDataType>>
22 ▼    hasDeleteResponseItems:
```

18. In the file browser, select api.raml.
19. Locate the /{customer_id} nested resource delete method definition and copy the responses section.

20. In the file browser, select traitsLibrary.raml and paste the lines below the hasdeleteResponsesItems trait.

```
22 ▼   hasDeleteResponseItems:
23 ▼     responses:
24 ▼       200:
25           body:
26             application/json:
27 ▼       404:
28 ▼         body:
29             application/json:
30             type: customErrorMessage
```

21. Replace the last line containing the CustomErrorMessage datatype with <<customErrorMessageDataType>>.

Apply the traits in the main RAML API definition

22. In the file browser, select api.raml.
23. Copy the cacheable and hasAcceptHeader reference lines and delete them from api.raml.
24. In the file browser, select traitsLibrary.raml and add the copied lines before the hasResponseItems trait definition.

```
 3 ▼ traits:
 4     cacheable: !include traits/cacheable.raml
 5     hasAcceptHeader: !include traits/hasAcceptHeader.raml
 6 ▼   hasResponseItems:
```

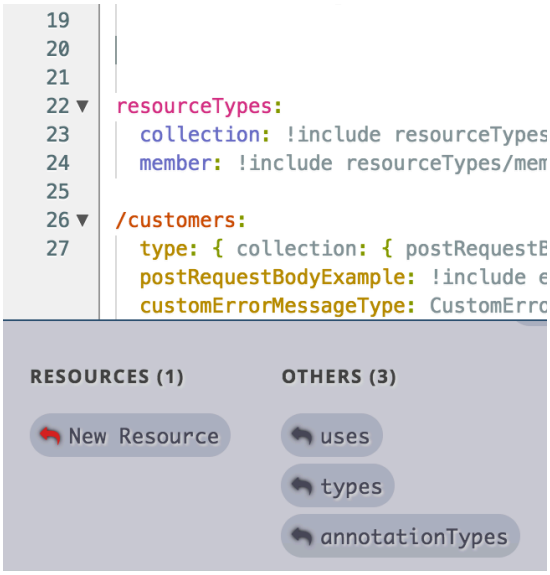
Note: There can only be one traits node in the api.raml file. Because the library file requires a traits node, we should move the other traits references over here.

25. To get rid of the error in lines that include the traits fragment, add ../ before the trait's path in the include statement.

```
3 ▼ traits:
4     cacheable: !include ../traits/cacheable.raml
5     hasAcceptHeader: !include ../traits/hasAcceptHeader.raml
6 ▼   hasResponseItems:
7 ▼     responses:
```

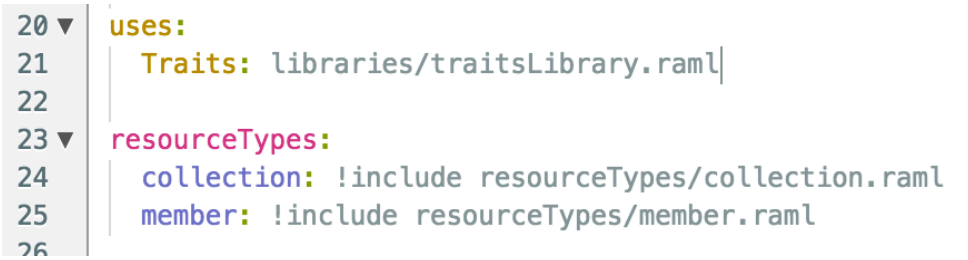
26. Save the project.
27. In the file browser, select api.raml.

28. Go to the line before the resourceTypes definition and in the shelf, click uses.



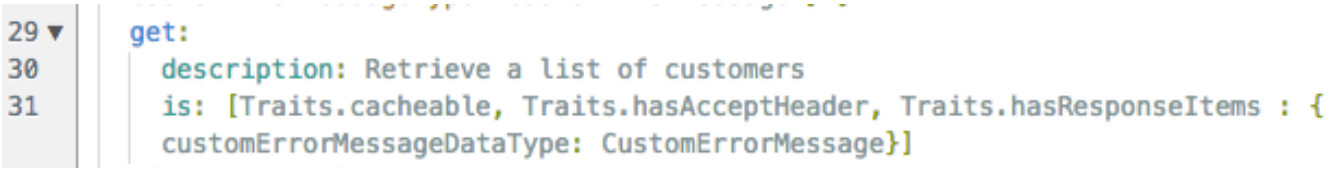
29. In the new line type:

```
Traits: libraries/traitsLibrary.raml.
```



30. In the /customers resource get method, go to the line that applies the traits cacheable and hasAcceptHeader and change the value to:

```
is: [Traits.cacheable, Traits.hasAcceptHeader,
Traits.hasResponseItems: { customErrorMessageDataType:
CustomErrorMessage} ]
```



31. Delete the responses structure in the get method.

```
/customers:
  type: { collection: { postRequestBodyExample: !include examples/CustomerExample.json,
  customErrorMessageType: CustomErrorMessage } }
  get:
    description: Retrieve a list of customers
    is: [Traits.cacheable, Traits.hasAcceptHeader, Traits.hasResponseItems : {
    customErrorMessageDataType: CustomErrorMessage}]
  /{customer_id}:
    type: { member: { uriParameterName: customer_id, getResponseBodyExample: !include
    examples/CustomerExample.json, customErrorMessageType: CustomErrorMessage } }
    patch:
```

32. Similarly, in the /accounts and /transactions resources, delete the get method responses, and change the traits application line to:

```
is: [Traits.hasAcceptHeader, Traits.hasResponseItems: {
customErrorMessageDataType: CustomErrorMessage} ]
```

```
  /accounts:
    get:
      description: Retrieve a list of accounts for a specific customer
      is: [Traits.hasAcceptHeader, Traits.hasResponseItems: { customErrorMessageDataType:
      CustomErrorMessage} ]

/accounts:
```

```
    /transactions:
      get:
        description: Retrieve a list of transactions for a specific account
        is: [Traits.hasAcceptHeader, Traits.hasResponseItems: { customErrorMessageDataType:
        CustomErrorMessage } ]
```

33. In the /{customer_id} resource delete method, delete the lines that define the method responses.
34. In the new line below the description node, type the following:

```
is: [Traits.hasDeleteResponseItems : {customErrorMessageDataType:
CustomErrorMessage}]
```

```
    delete:
      description: Delete a specific customer
      is: [Traits.hasDeleteResponseItems : {customErrorMessageDataType: CustomErrorMessage}]
    /accounts:
      get:
```

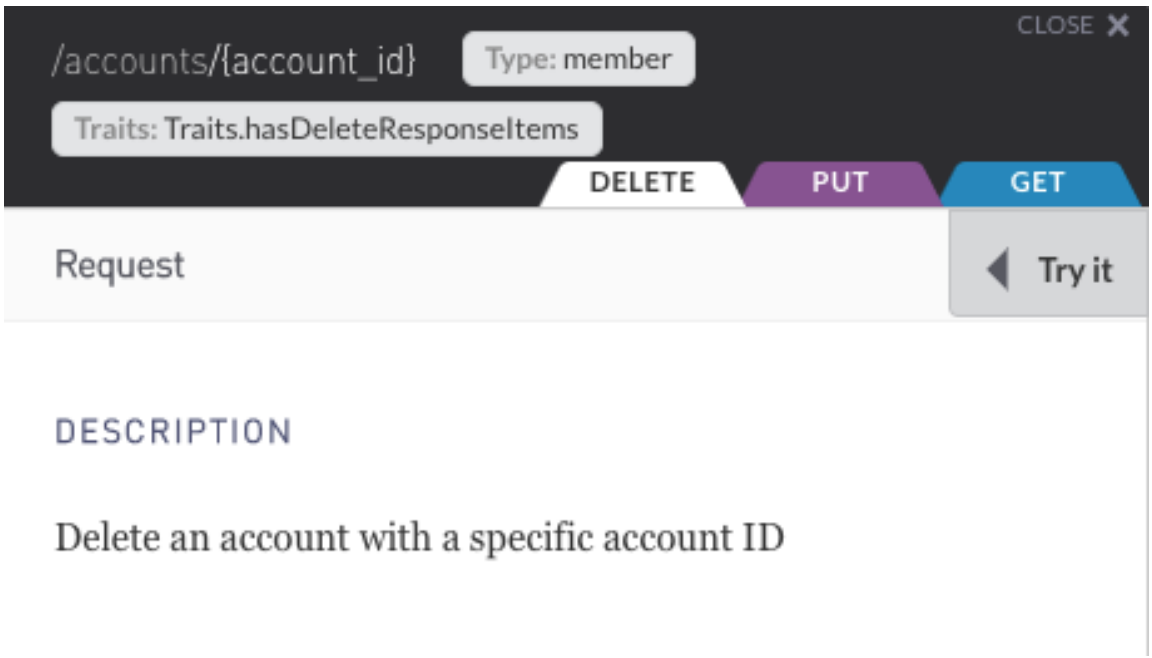
35. Similarly, in the /{account_id} resource delete method, delete the lines that define the responses and apply the hasDeleteResponseItems trait.

```
52 ▼  /{account_id}:
53      type: { member: { uriParameterName: account_id, getResponseBodyExample: !include
        examples/AccountExample.json, customErrorMessageType: CustomErrorMessage } }
54 ▼    delete:
55        description: Delete an account with a specific account ID
56        is: [Traits.hasDeleteResponseItems : {customErrorMessageDataType: CustomErrorMessage}]
```

36. Save the project.

View the documentation

37. In API Console, scroll down to the /accounts/{account_id} and click DELETE.
38. Verify that the hasDeleteResponseItems trait has been applied to the resource.

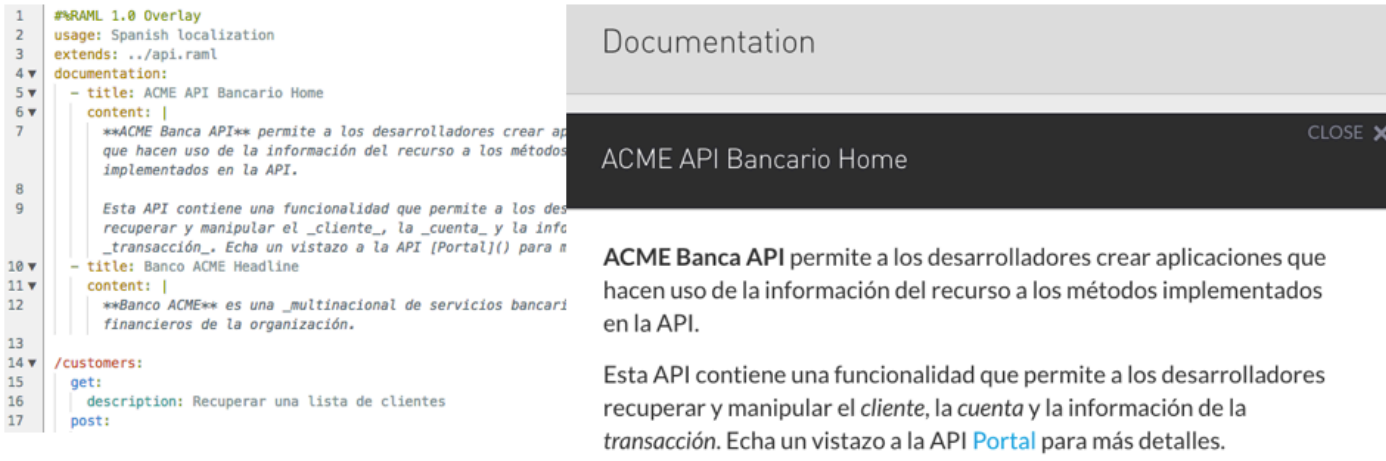


39. Click CLOSE X.

Walkthrough 10-2: Internationalize documentation and resource description using an overlay

In this walkthrough, you internationalize the ACME Banking API using overlays. You will:

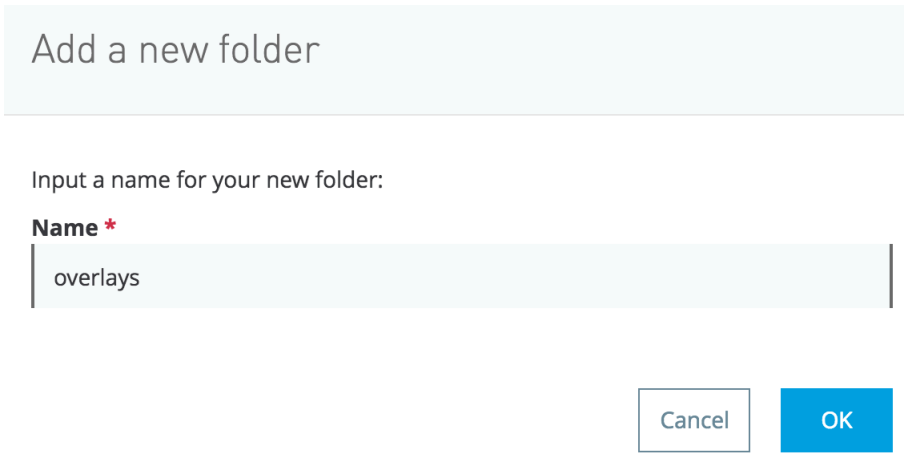
- Create a RAML overlay fragment file.
- Add documentation and resource method description nodes in the overlay file to support Spanish localization.



```
#%RAML 1.0 Overlay
usage: Spanish localization
extends: ../api.raml
documentation:
  - title: ACME API Bancario Home
    content: |
      **ACME Banca API** permite a los desarrolladores crear ap
      que hacen uso de la información del recurso a los métodos
      implementados en la API.

      Esta API contiene una funcionalidad que permite a los des
      recuperar y manipular el _cliente_, la _cuenta_ y la info
      _transacción_. Echa un vistazo a la API [Portal]() para m
  - title: Banco ACME Headline
    content: |
      **Banco ACME** es una _multinacional de servicios bancari
      financieros de la organización.

/customers:
  get:
    description: Recuperar una lista de clientes
  post:
```


Create a RAML overlay file

1. Return to API Designer.
2. From the Project menu, select New Folder.
3. In the Add a new folder dialog box, set the name to overlays and click OK.



4. In the file browser, right-click the overlays folder and select New File > RAML 1.0 > Overlay.
5. In the Add a new overlay file dialog box, set the name to internationalization.raml and click OK.

Add documentation and description nodes to support Spanish localization

6. In the second line that contains the extends node, type the value as

   `../api.raml`

```
1  #%RAML 1.0 Overlay
2  extends: ../api.raml
```

7. Enter a new line before the extends node and type:

   `usage: Spanish localization.`

8. Return to the course snippets.txt file.
9. Copy the text from the Module 10, Spanish documentation snippet and paste it in internationalization.raml.

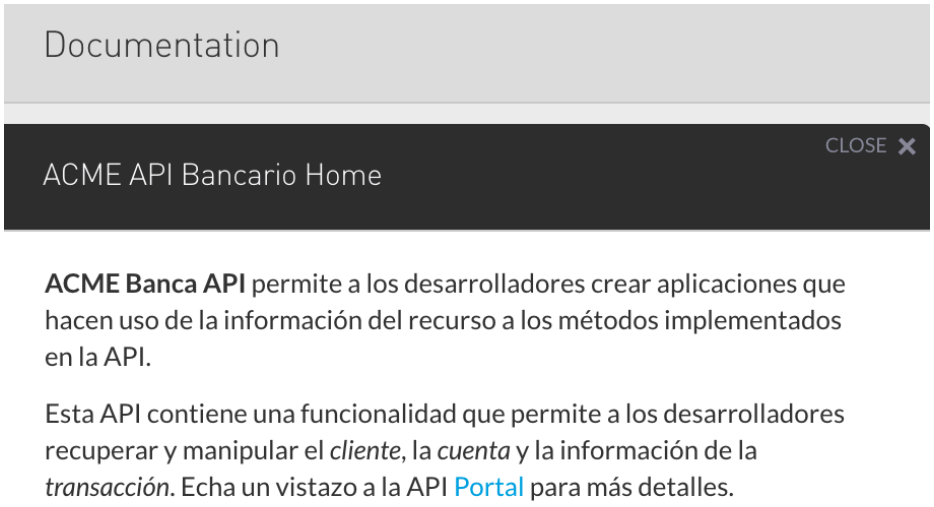
```
1  #%RAML 1.0 Overlay
2  usage: Spanish localization
3  extends: ../api.raml
4  documentation:
5    - title: ACME API Bancario Home
6      content: |
7        **ACME Banca API** permite a los desarrolladores crear
         aplicaciones que hacen uso de la información del recurso a
         los métodos implementados en la API.
8
9        Esta API contiene una funcionalidad que permite a los
         desarrolladores recuperar y manipular el _cliente_, la
         _cuenta_ y la información de la _transacción_. Echa un
         vistazo a la API [Portal]() para más detalles.
10   - title: Banco ACME Headline
11     content: |
12       **Banco ACME** es una _multinacional de servicios bancarios_
         y financieros de la organización.
```

10. From the course snippets.txt file, copy the lines under the Spanish localization for method description heading.
11. Return to API Designer and enter a new line below the documentation node and place the cursor at the beginning of the line.
12. Add a new line and paste the copied text.

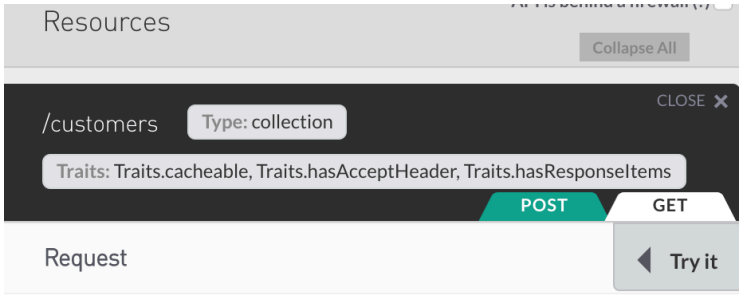
```
11     content: |
12       **Banco ACME** es una _multinacional de servicios bancarios_
         y financieros de la organización.
13
14 /customers:
15   get:
16     description: Recuperar una lista de clientes
17   post:
18     description: Agregue un nuevo cliente
19   /{customer_id}:
20     get:
21       description: Recuperar un cliente con una ID de cliente
         específico
22     delete:
23       description: Eliminar un cliente específico
```

View the documentation

13. In API Console, verify that the documentation, resources, and types show up like the api.raml API Console.
14. Go to the Documentation section and expand the documentation title to view the Spanish localization of the document.



15. Click CLOSE X.
16. Go to the /customers resource and click GET.
17. Verify you can view the GET method description in Spanish.



18. Click CLOSE X.
19. Save the project.

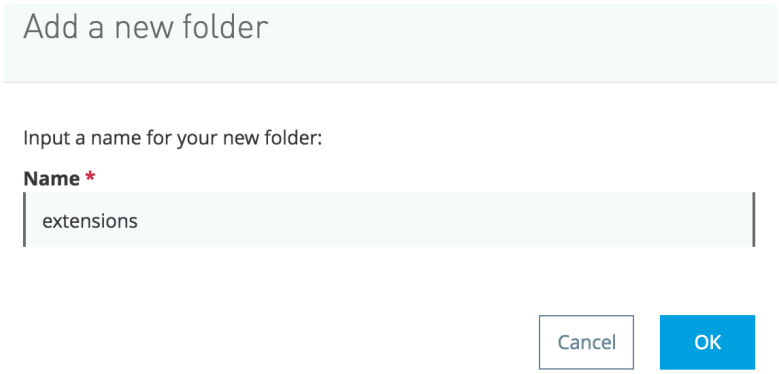
Walkthrough 10-3: Define and use API extensions to promote portability to test in multiple environments

In this walkthrough, you will define an extension. You will:

- Create API extension files.
- Add a baseUri parameter to the extension files and enter a placeholder for environment specific implementation URL.

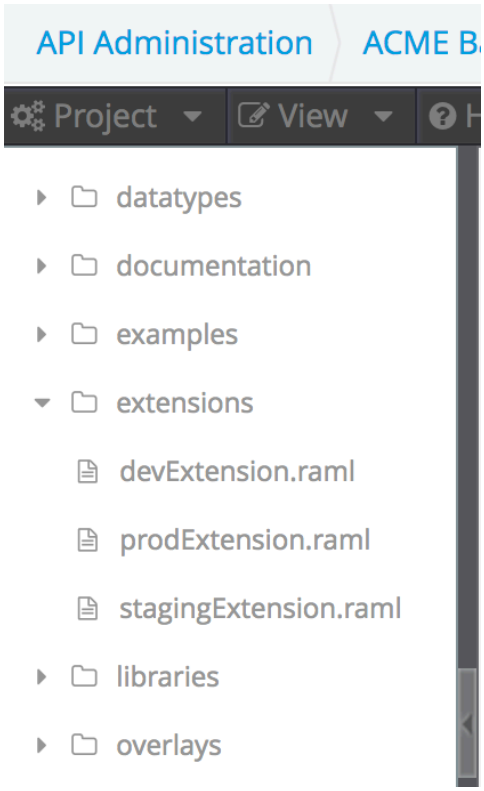
Create API extension files

1. Return to API Designer.
2. In the file browser, create a new folder.
3. In the Add a new folder dialog box, set the name to extensions and click OK.



4. Right-click the extensions folder and select New File > RAML 1.0 > Extension.
5. In the Add new Extension file dialog box, set the name to stagingExtension.raml and click OK.

6. Create two more extensions in the extensions folder called devExtension.raml and prodExtension.raml.



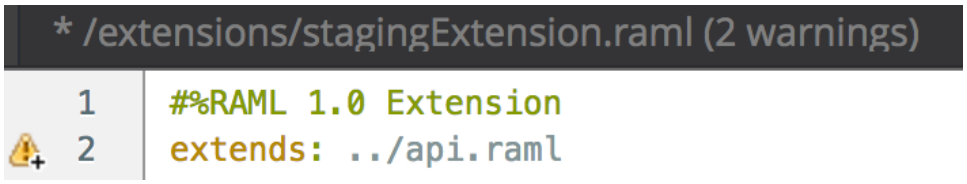
Add the baseUri node to the extension files and enter a placeholder for environment specific implementation URL

7. In the file browser, select api.raml.
8. Go to the line that defines the baseUri at the top and delete it.

   Note: The baseUri parameter cannot be overridden if declared in the root RAML file. We will be declaring it in the individual extension files.

9. Save the project.
10. In the file browser, select stagingExtension.raml.
11. Go to the line that contains the extends node and type the value as:

```
../api.raml
```



12. In API Console, turn on the mocking service at the top right corner.
13. Verify that you see the baseUri added to the extension file.

Note: You can replace the Mock service URL with a staging environment specific implementation URL.

14. Save the project.
15. In API Console, click GET for the /accounts/{account_id} resource.
16. Click Try It.
17. Enter the account_id value as 12345 and click GET.
18. Verify that you see the response structure of the get method.

content-type:
application/json; charset=utf-8

date:
Wed, 19 Apr 2017 18:35:58 GMT

server:
nginx

vary:
Accept-Encoding

Body


```
{
  "accountID": "12345",
  "accountType": "Savings",
  "accountNumber": "1234567890",
  "accountOwner": [
    {
      "customerID": "8f19cb50-3f57-4d38",
      "displayName": "John Doe",
      "ssn": "123-456-7890"
    }
  ],
  "accountBalance": {
    "currency": "USD",
    "amount": "8457.90"
```


19. Similarly, go to the devExtension.raml file and include the api.raml file in the extends node.

20. Add a new line before the extends node, and type the following:

```
baseUri: <<insert dev environment specific implementation URL>>
```

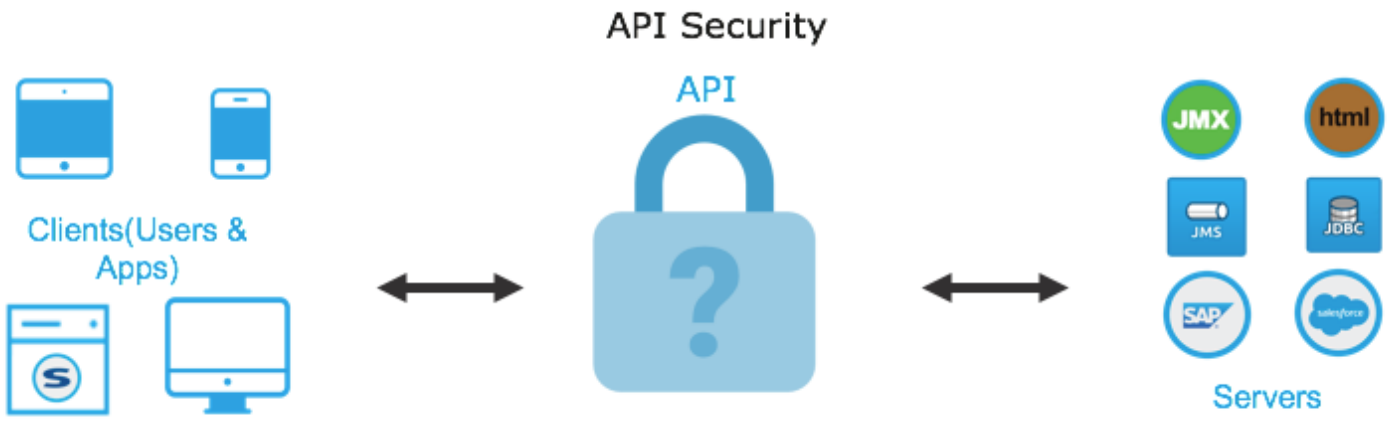
```
* /extensions/devExtension.raml (2 warnings)
1  #%RAML 1.0 Extension
2  baseUri: <<insert dev environment specific implementation URL>>
3  extends: ../api.raml
```

21. Save the project.
22. Similarly, in the prodExtension.raml file, add in a placeholder for the baseUri and reference api.raml in the extends node.

```
* /extensions/prodExtension.raml (2 warnings)
1  #%RAML 1.0 Extension
2  baseUri: <<insert prod environment specific implementation URL>>
3  extends: ../api.raml
```

23. Save the project.

Module 11: Securing APIs



Objectives:

- Define API security requirements.
- Implement basic authentication for a resource URL using traits.
- Use RAML to define an API-to-OAuth2.0 policy.

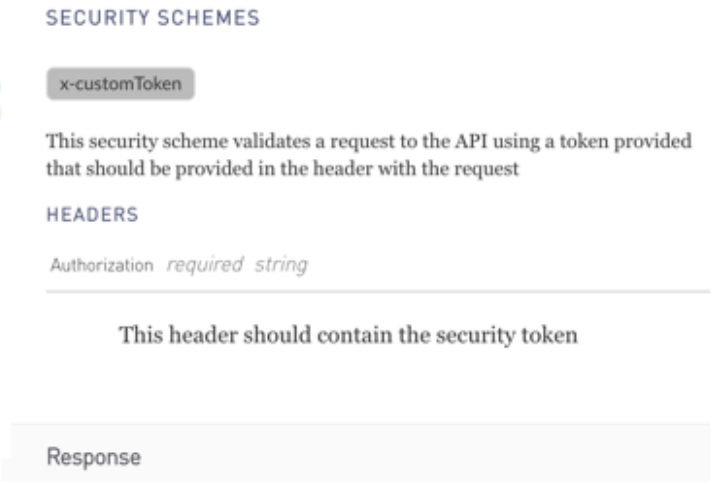
Walkthrough 11-1: Define a custom security scheme for an API

In this walkthrough, you will define a custom security scheme for your API. You will:

- Create a custom security scheme file.
- Reference the custom security scheme in the main RAML API definition.
- Apply the security scheme to certain resource methods.

```
1  #%RAML 1.0 SecurityScheme
2  type: x-customToken
3  description: This security scheme validates a requ
   token provided that should be provided in the head
4  describedBy:
5    headers:
6      Authorization:
7        description: This header should contain the
8        type: string
```

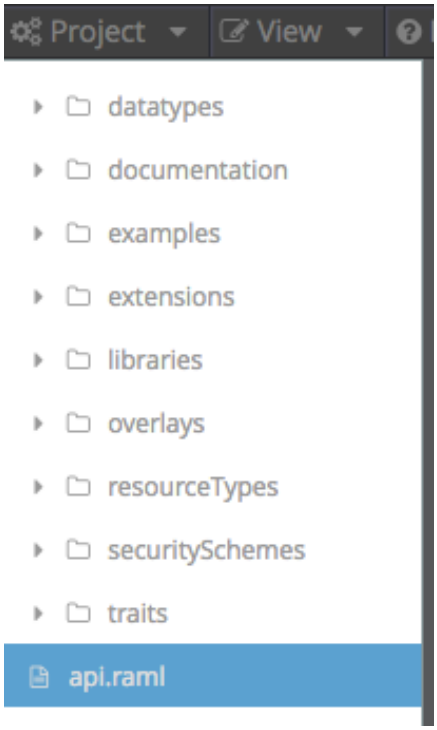
```
23 securitySchemes:
24   customTokenSecurity: !include
     securitySchemes/customTokenSecurity.raml
25
26 resourceTypes:
```



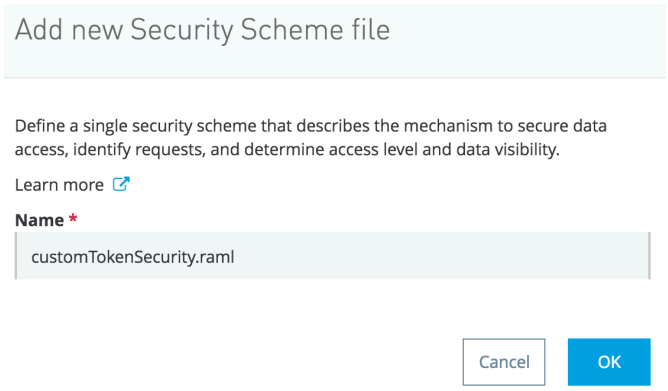
Create a security scheme file

1. Return to API Designer.

2. In the file browser, create a new folder called securitySchemes.



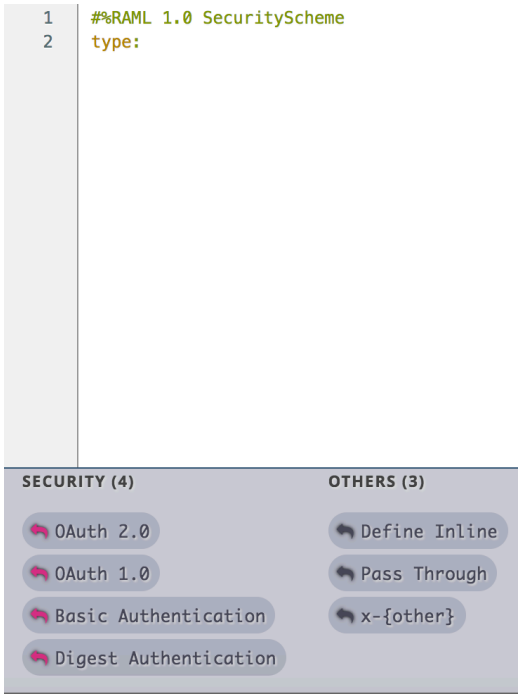
3. Right-click the securitySchemes folder and select New File > RAML 1.0 > Security Scheme.
4. In the Add new Security Scheme file dialog box, set the name to customTokenSecurity.raml.



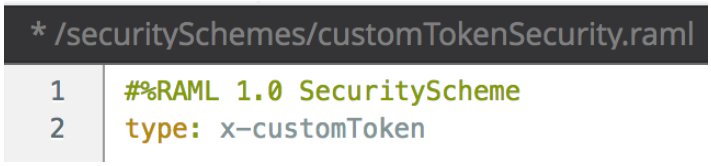
Define a custom security scheme

5. In the RAML editor, go to the line that contains type node and press space bar after the colon.

6. In the shelf below the editor, click x-{other}.



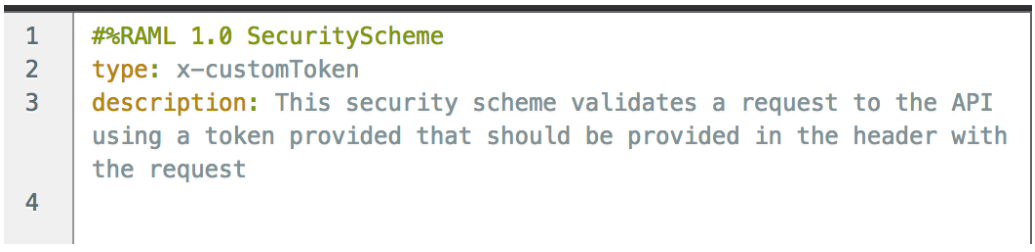
7. Replace {other} with customToken.



8. Add a new line below the type node.
9. In the shelf, click description.
10. Type the value of the description node as:

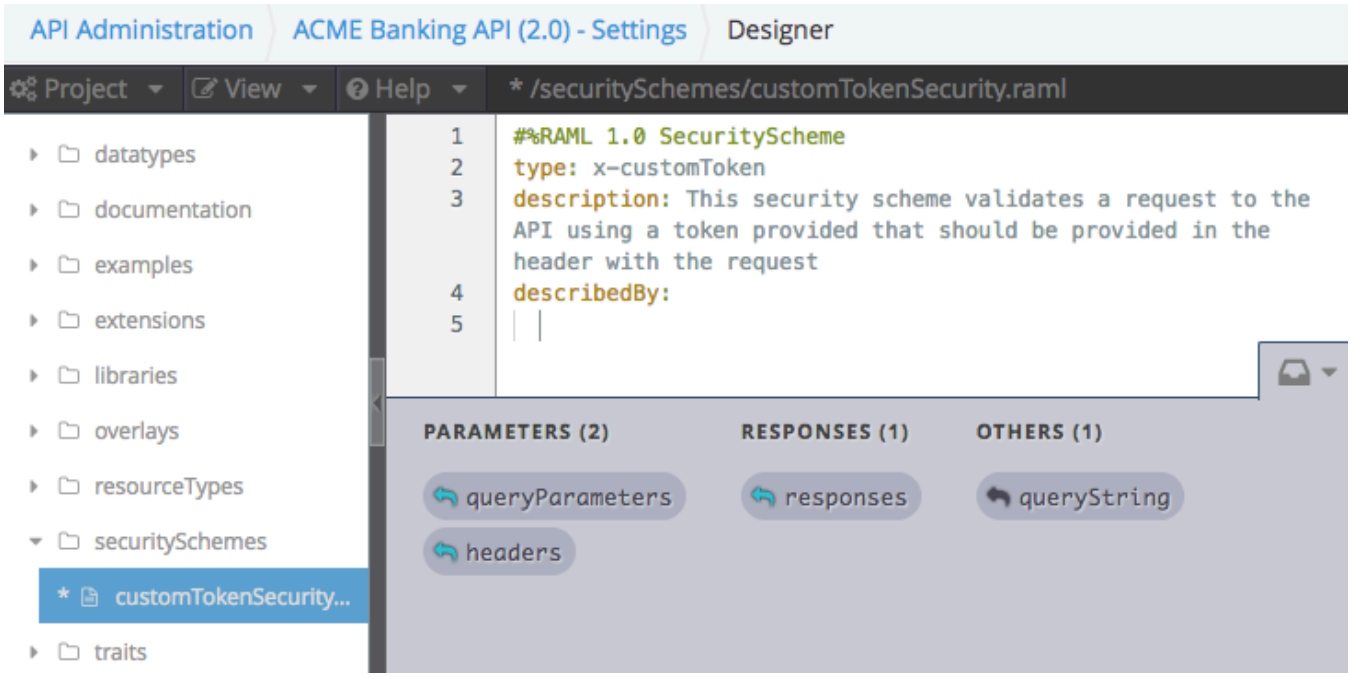
    This security scheme validates the request to the API using a token provided in the header with the request.

11. Press enter to add a new line.

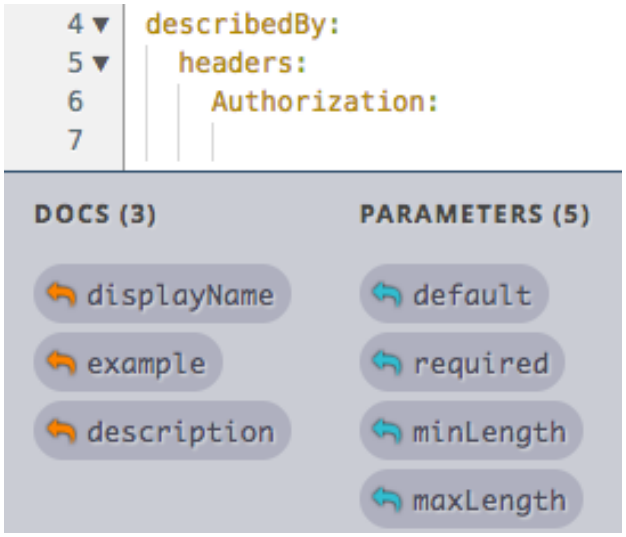


12. In the shelf, click describedBy.

13. In the shelf, click headers.



14. In the shelf, click Authorization.
15. In the shelf, click description.

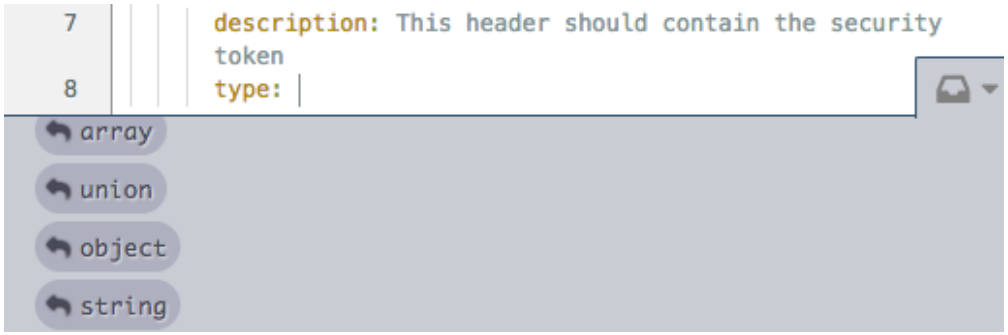


16. Type the value of the description node as: This header should contain the security token.

```
4 describedBy:
5   headers:
6     Authorization:
7       description: This header should contain the security token
```

17. Press enter to add a new line.

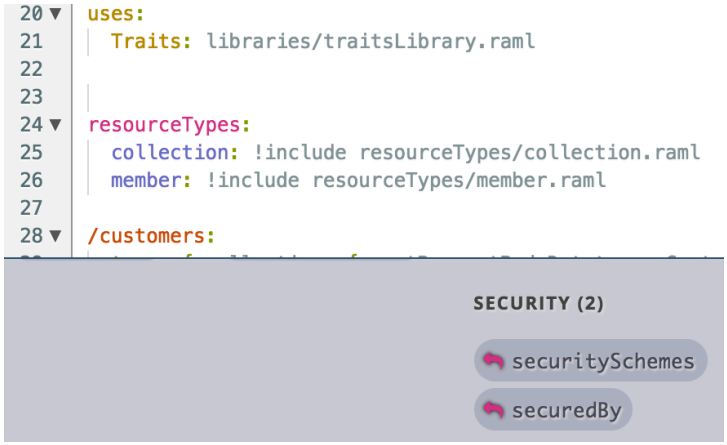
18. In the shelf, click type.
19. In the shelf, click string.



20. Save the project.

Reference the custom security scheme file in the main API definition

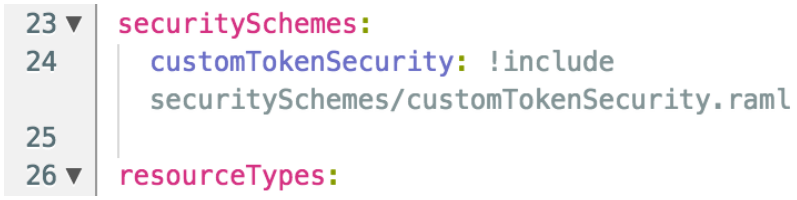
21. In the file browser, select api.raml.
22. Go to the empty line after the line that references the traitsLibrary file.
23. Add a new line below it.
24. In the shelf, click securitySchemes.



25. In the new line type:

```
customTokenSecurity: !include securitySchemes/customTokenSecurity.raml
```

26. Press enter to add a new line.



Apply the custom security scheme to all the resource methods in the API

27. Go to the empty line before the /customers resource and press enter.
28. In the shelf, click securedBy.

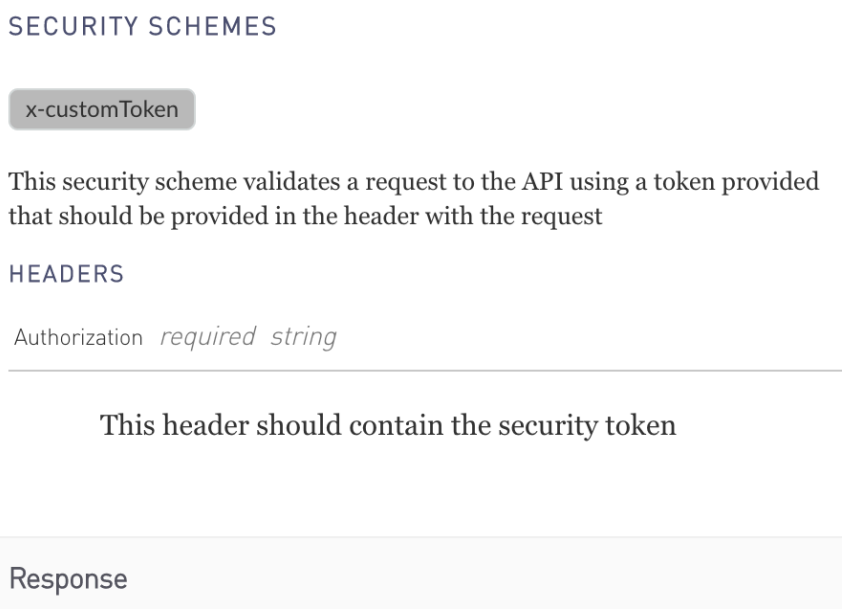
29. In the shelf, click customTokenSecurity.
30. Press enter to add a new line.

31. Save the project.

View the documentation

32. In API Console, click GET for the /customers resource.

33. Scroll down to the Headers section of the Request and verify that you can see the security scheme information below.



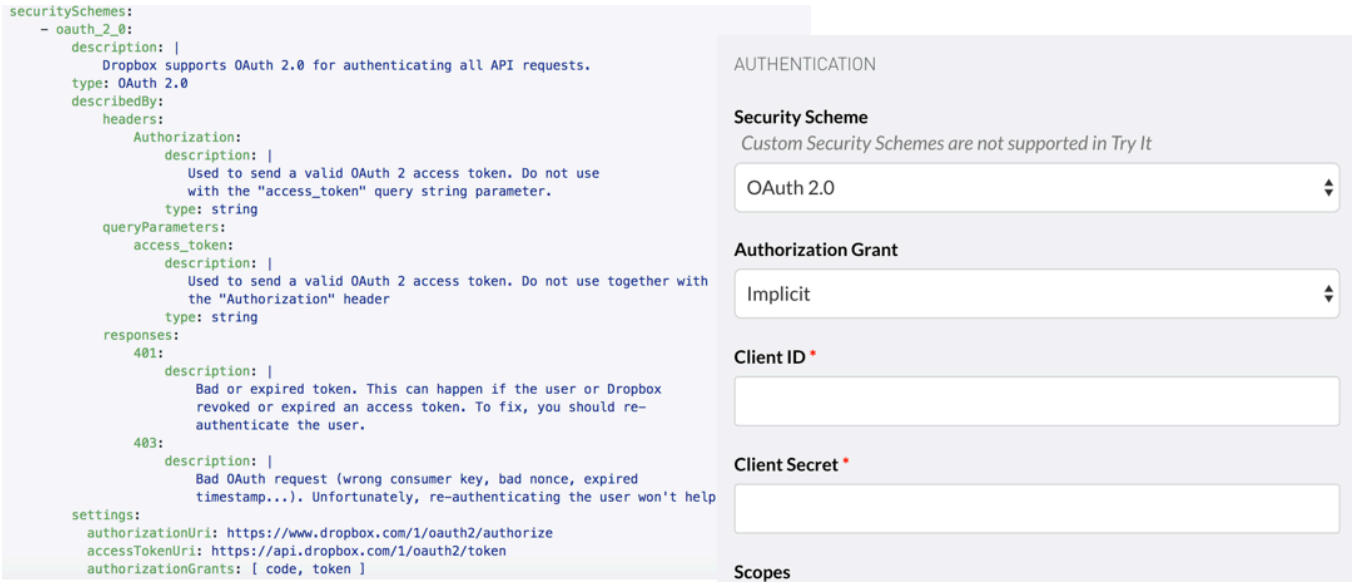
Note: Go to any other resource method and notice the security scheme header added to all the resource methods by adding to the root of the RAML definition. Custom security schemes like the customTokenSecurity is not supported for testing using the Try It option.

34. Click CLOSE X.

Walkthrough 11-2: Define a security scheme for an API and secure API resources

In this walkthrough, you define a security scheme to secure API resources using OAuth2.0. You will:

- Create a security scheme fragment file.
- View the OAuth2.0 protocol security scheme snippet from the policy details.
- Define the OAuth2.0 protocol in the security scheme fragment.
- Reference the OAuth2.0 security scheme in the RAML API definition.
- Apply the security scheme in the API resource methods.

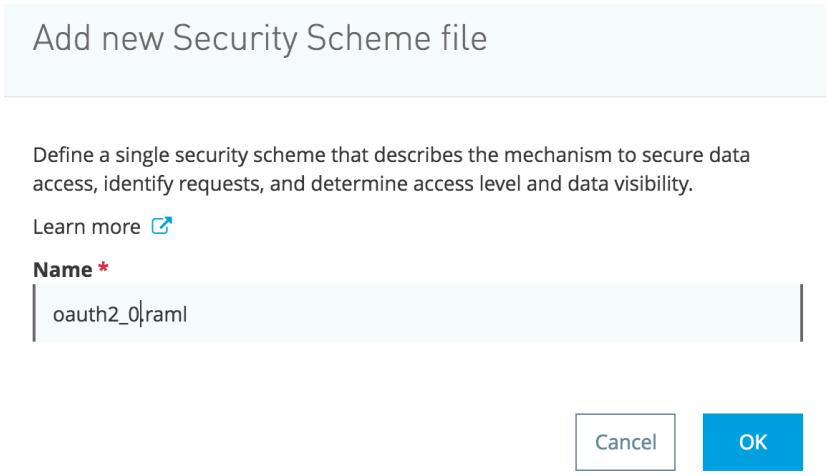


```
securitySchemes:
    - oauth_2_0:
        description: |
            Dropbox supports OAuth 2.0 for authenticating all API requests.
        type: OAuth 2.0
        describedBy:
            headers:
                Authorization:
                    description: |
                       Used to send a valid OAuth 2 access token. Do not use
                       with the "access_token" query string parameter.
                    type: string
            queryParameters:
                access_token:
                    description: |
                       Used to send a valid OAuth 2 access token. Do not use together with
                       the "Authorization" header
                    type: string
            responses:
                401:
                    description: |
                        Bad or expired token. This can happen if the user or Dropbox
                        revoked or expired an access token. To fix, you should re-
                        authenticate the user.
                403:
                    description: |
                        Bad OAuth request (wrong consumer key, bad nonce, expired
                        timestamp...). Unfortunately, re-authenticating the user won't help
        settings:
          authorizationUri: https://www.dropbox.com/1/oauth2/authorize
          accessTokenUri: https://api.dropbox.com/1/oauth2/token
          authorizationGrants: [ code, token ]
```


Create a security scheme fragment file

1. In the file browser, right-click the securitySchemes folder and select New File > RAML 1.0 > Security Scheme.

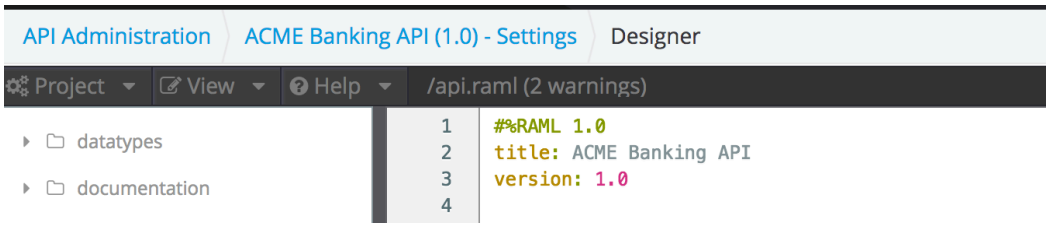
2. In the Add new Security Scheme file dialog box, set the name to oauth2_0.raml.



3. Save the project.

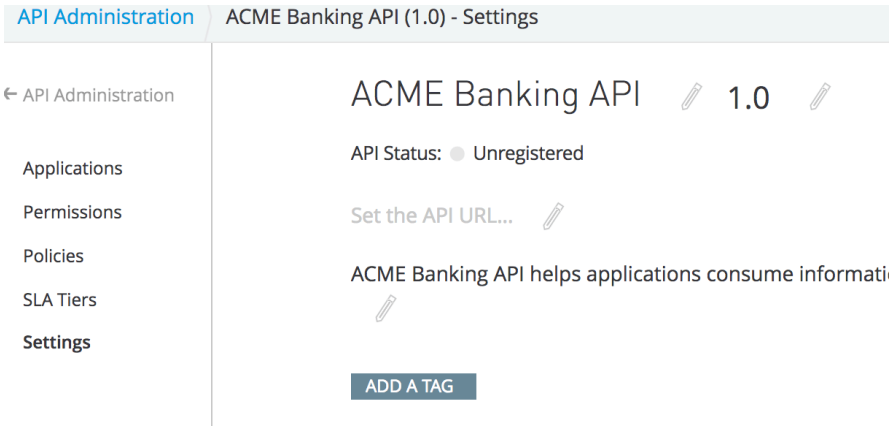
View the security scheme snippet from the OAuth2.0 policy details

4. Click the ACME Banking API (1.0) – Settings link above the API Designer menu.



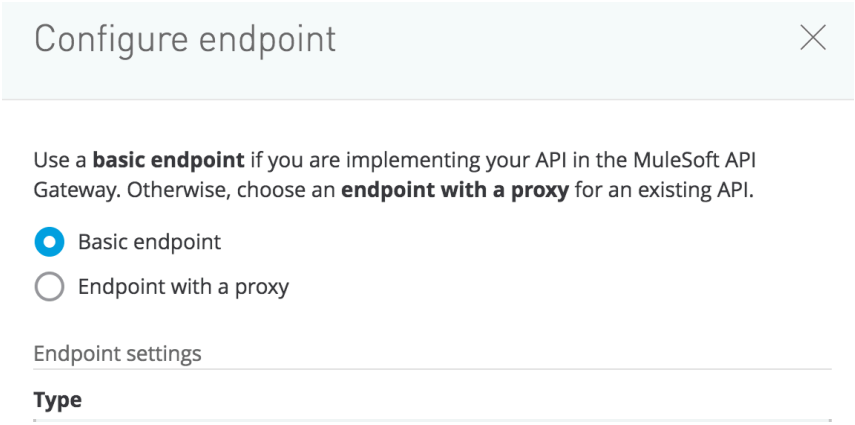
5. On the API version details page, select Policies in left-side navigation; you should not see a list of policies.

   Note: To see the list of policies, you need to set the API URL.

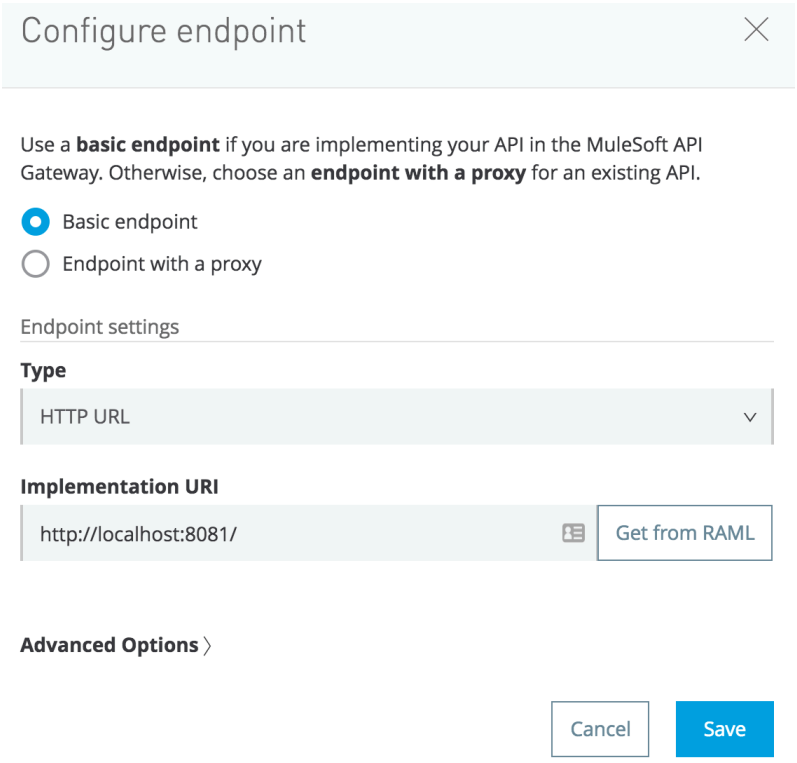


6. in the left-side navigation, select Settings.

7. On the API version details page, click Configure endpoint in the API Status section.
8. In the Configure endpoint dialog box, select the Basic endpoint radio button.



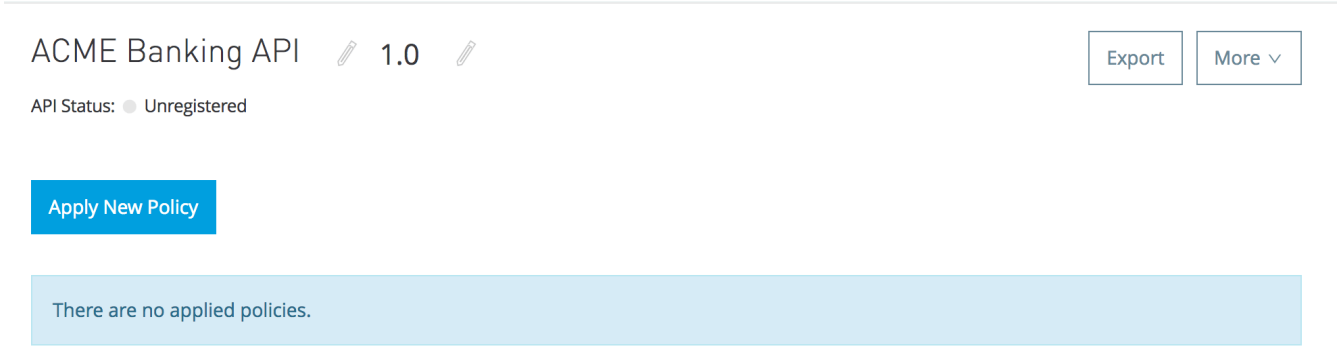
9. In the Endpoint settings, select the Type as HTTP URL and type the Implementation URI as http://localhost:8081/.



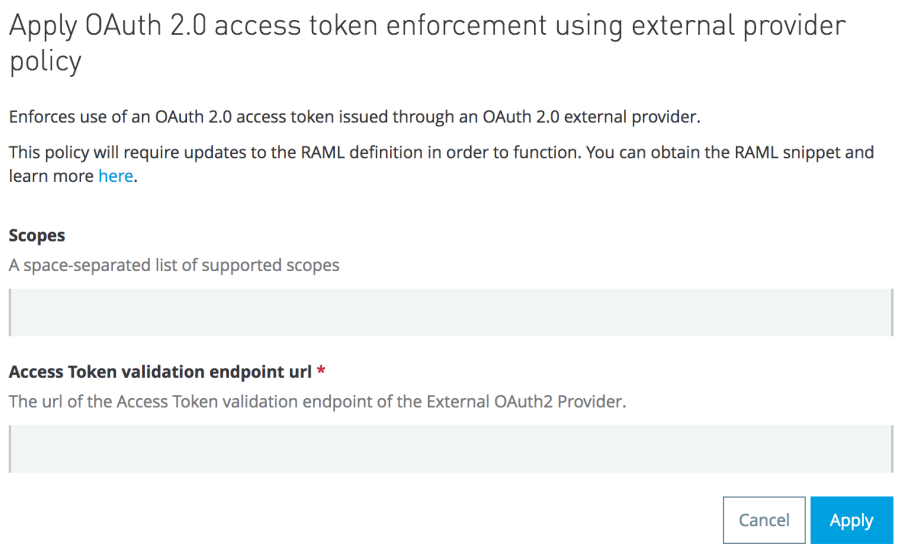
Note: This step is carried out to set the API URL to view the list of policies. We are not deploying any applications with these values. To understand more about implementing your APIs or proxies, please take the Anypoint Platform Operations: API Management course.

10. Click Save.
11. In the left-side navigation, select Policies.

12. Click Apply New Policy.



13. In the Select Policy dialog box, click the All Categories drop-down bar on top and click Security.
14. View the list of security policies.
15. Scroll down and select the radio button for OAuth 2.0 access token enforcement using external provider policy and click Configure Policy.
16. In the Apply OAuth 2.0 access token enforcement using external provider policy, click the link text ‘here’ to obtain the RAML snippet.



17. Review the code snippet with the OAuth 2.0 security scheme definition.

```
securitySchemes:
    - oauth_2_0:
        description: |
            Dropbox supports OAuth 2.0 for authenticating all API requests.
        type: OAuth 2.0
        describedBy:
            headers:
                Authorization:
                    description: |
                       Used to send a valid OAuth 2 access token. Do not use
                       with the "access_token" query string parameter.
                    type: string
            queryParameters:
                access_token:
                    description: |
                       Used to send a valid OAuth 2 access token. Do not use together with
                       the "Authorization" header
                    type: string
            responses:
                401:
                    description: |
                        Bad or expired token. This can happen if the user or Dropbox
                        revoked or expired an access token. To fix, you should re-
                        authenticate the user.
                403:
                    description: |
                        Bad OAuth request (wrong consumer key, bad nonce, expired
                        timestamp...). Unfortunately, re-authenticating the user won't help here.
        settings:
          authorizationUri: https://www.dropbox.com/1/oauth2/authorize
          accessTokenUri: https://api.dropbox.com/1/oauth2/token
          authorizationGrants: [ code, token ]
```

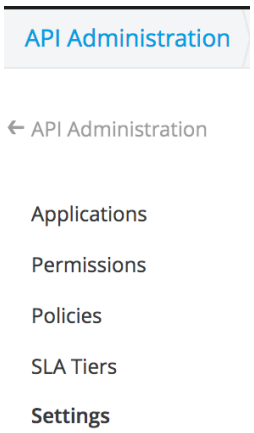
Add the OAuth2.0 security scheme definition

18. Return to the course snippets.txt file.
19. Copy the lines in Module 11 under the OAuth2.0 security scheme definition.

```
-------Module 11--------

OAuth 2.0 security scheme definition:

type: OAuth 2.0
description: ACME Banking API supports OAuth 2.0 security policy for methods that involve updating existing data
describedBy:
  headers:
      Authorization:
        description: |
          Used to send a valid OAuth 2 access token. Do not use with the "access_token" query string parameter.
        type: string
  queryParameters:
    access_token:
      description: |
        Used to send a valid OAuth 2 access token. Do not use together with the "Authorization" header
      type: string
  responses:
    401:
      description: |
        Bad or expired token. This can happen if the API consumer uses a revoked or expired access token. To fix, you should re-authenticate the user.
    403:
      description: |
        Bad OAuth request (wrong consumer key, bad nonce, expired timestamp...). Unfortunately, re-authenticating the user won't help here.
settings:
  authorizationUri: https://placeholder.com/oauth2/authorize
  accessTokenUri: https://placeholder.com/oauth2/access_token
  authorizationGrants: implicit
```

20. Return to Anypoint Platform.

21. Click Cancel to cancel applying the OAuth2.0 policy.
22. In the left-side navigation, select Settings to return to the API version details page.



23. In the API Definition section, click Edit in API designer.
24. In the file browser, expand the securitySchemes folder and select the oauth2_0.raml file.
25. Delete the line that contains the type node.
26. Paste the copied lines.

```
#%RAML 1.0 SecurityScheme
type: OAuth 2.0
description: ACME Banking API supports OAuth 2.0 security policy
for methods that involve updating existing data
describedBy:
  headers:
    Authorization:
      description: |
        Used to send a valid OAuth 2 access token. Do not use
        with the "access_token" query string parameter.
      type: string
  queryParameters:
    access_token:
      description: |
        Used to send a valid OAuth 2 access token. Do not use
        together with the "Authorization" header
      type: string
  responses:
    401:
```

Note: The authorizationUri, accessTokenUri and the authorizationGrants value should be the URI in the external OAuth2.0 provider that you configure. For now, we use placeholder values.

27. Save the project.

Reference the OAuth2.0 security scheme in the main API definition

28. In the left-side navigation, select api.raml.
29. Go to the end of the line that contains the customTokenSecurity scheme and add a new line below it.

```
securitySchemes:
  customTokenSecurity: !include
  securitySchemes/customTokenSecurity.raml


resourceTypes:
```

30. In the new line type:

`oauth2_0: !include securitySchemes/oauth2_0.raml.`

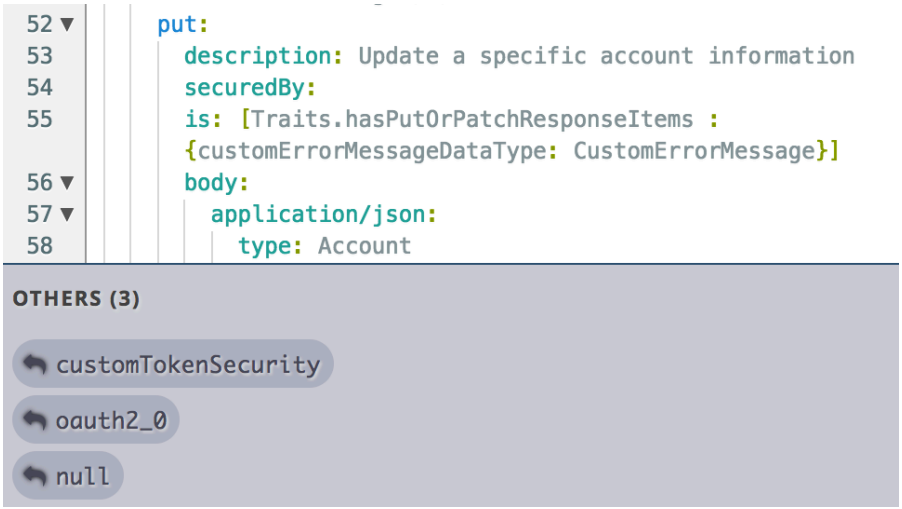
```
securitySchemes:
  customTokenSecurity: !include
  securitySchemes/customTokenSecurity.raml
  oauth2_0: !include securitySchemes/oauth2_0.raml

resourceTypes:
```

Secure the resource methods that update bank customer and account information with the OAuth2.0 security scheme

31. In the /{customer_id} nested resource patch method, add a new line below the line that contains the description node.
32. In the shelf, click securedBy.

33. In the shelf, click oauth2_0.

34. In the /{account_id} nested resource put method, add a new line below the line that contains the description node.
35. In the shelf, click securedBy.
36. In the shelf, click oauth2_0.

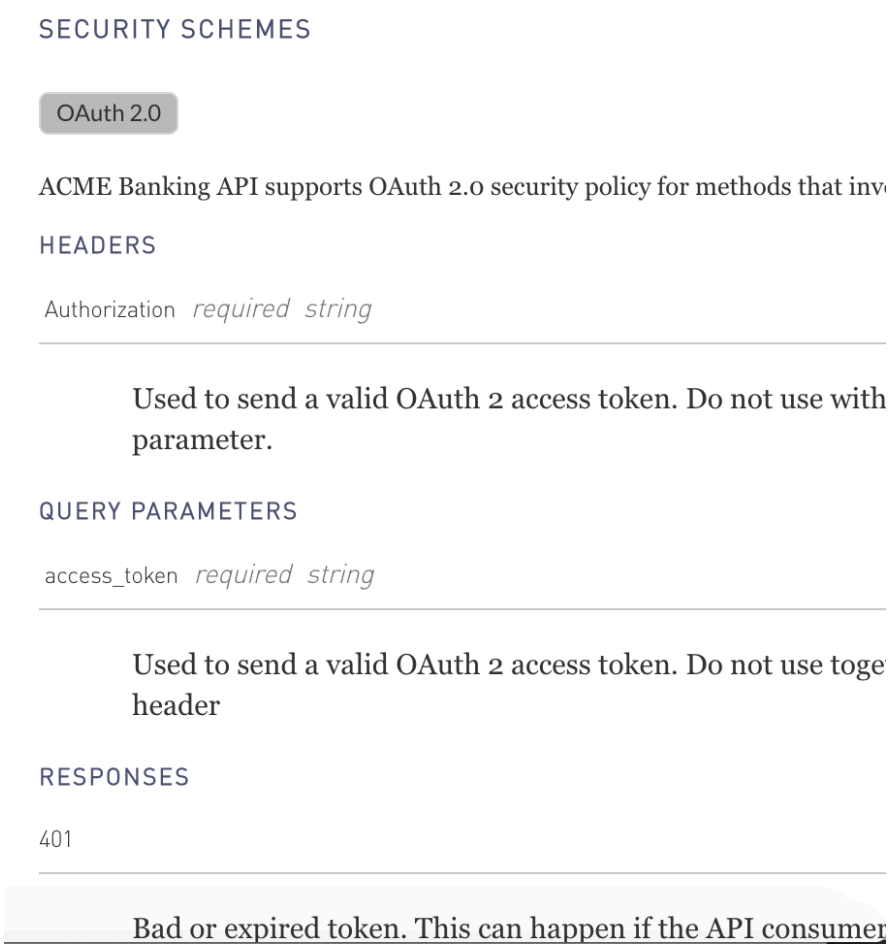


37. Save the project.
38. In the file browser, expand the extensions folder and click stagingExtension.raml file.

    Note: Because the mocking service is turned on for the extension RAML file, the baseUri gets added in the RAML editor multiple times. This can indicate an error in the API Console. To avoid the error, delete duplicate baseUri nodes that get created.

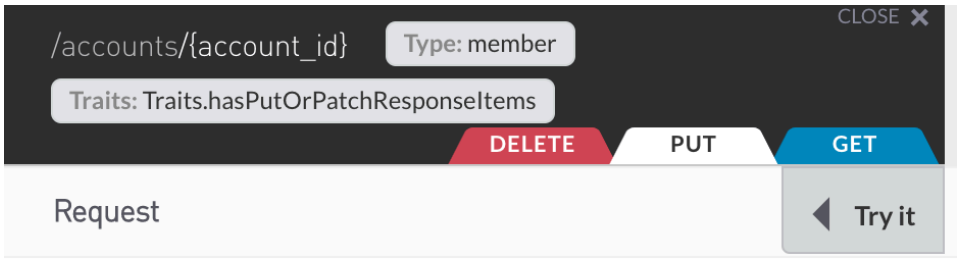
39. In API Console, click the PUT tab for the /accounts/{account_id} resource.
40. Scroll to the Security Schemes header under the Request section.

41. Review the details for the OAuth2.0 security scheme.

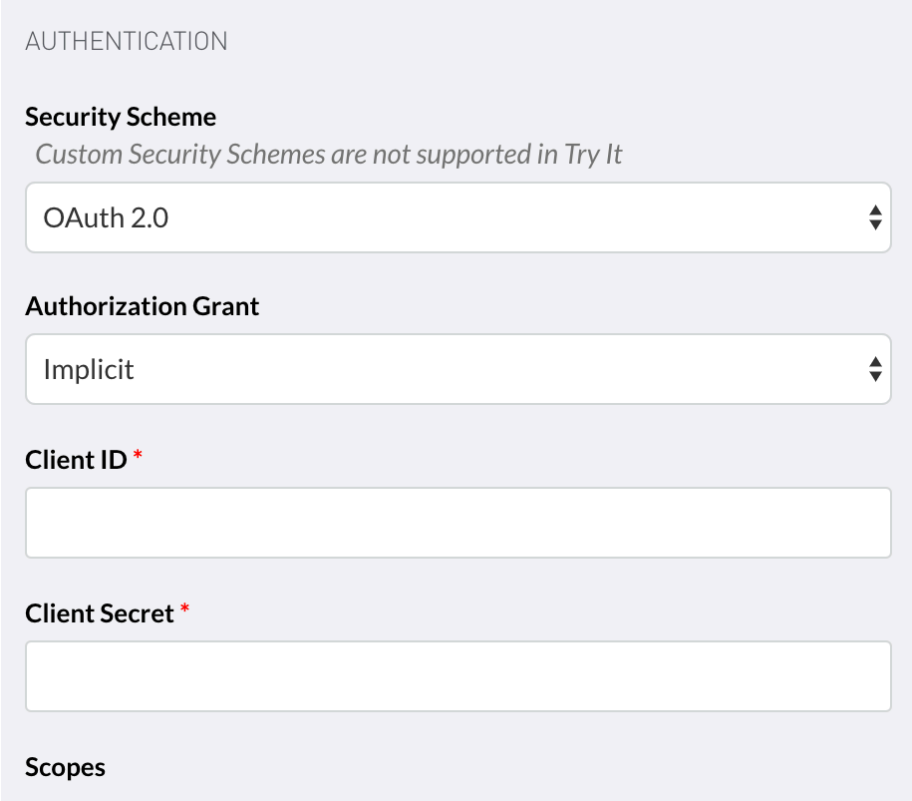


Note: Even though we applied the customToken security scheme at the root level, observe that resource method level security takes precedence when both are specified.

42. Scroll back up and click Try it across the Request section header.

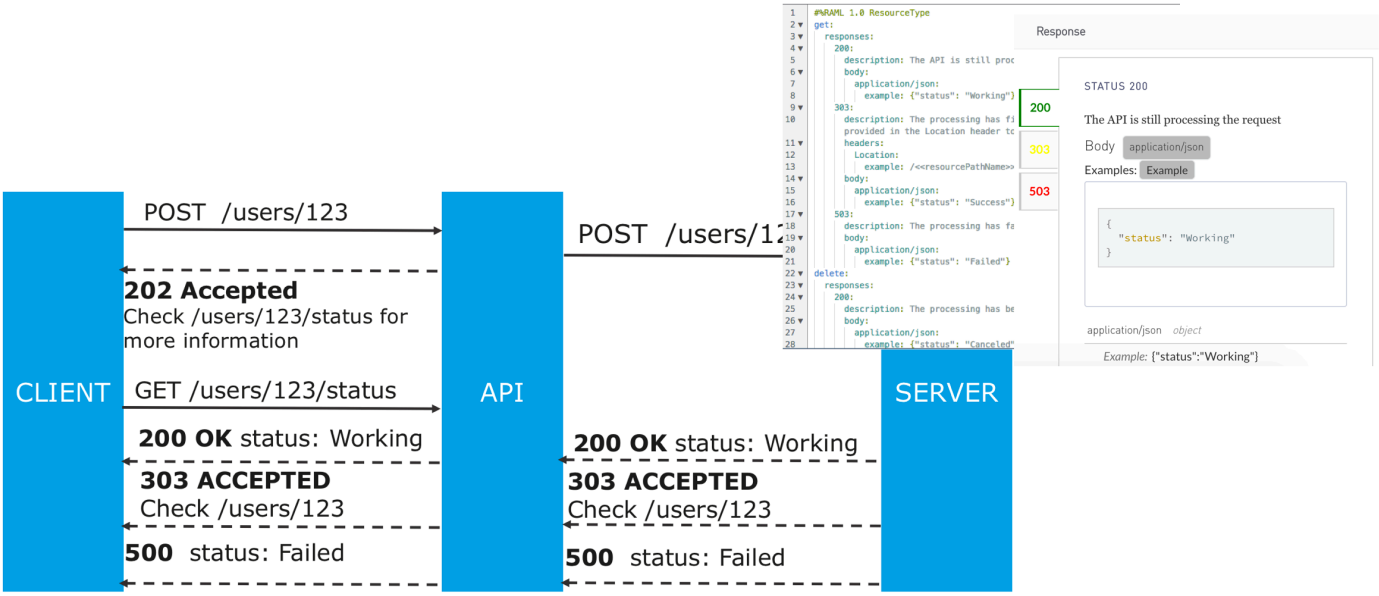


43. Locate the required input Client ID and Client Secret parameters for the OAuth2.0 Authentication.



Note: We carried out this step using the extension file because a baseUri is required to use the Try It option in API Console.

Module 12: Enhancing API Responses using Hypermedia



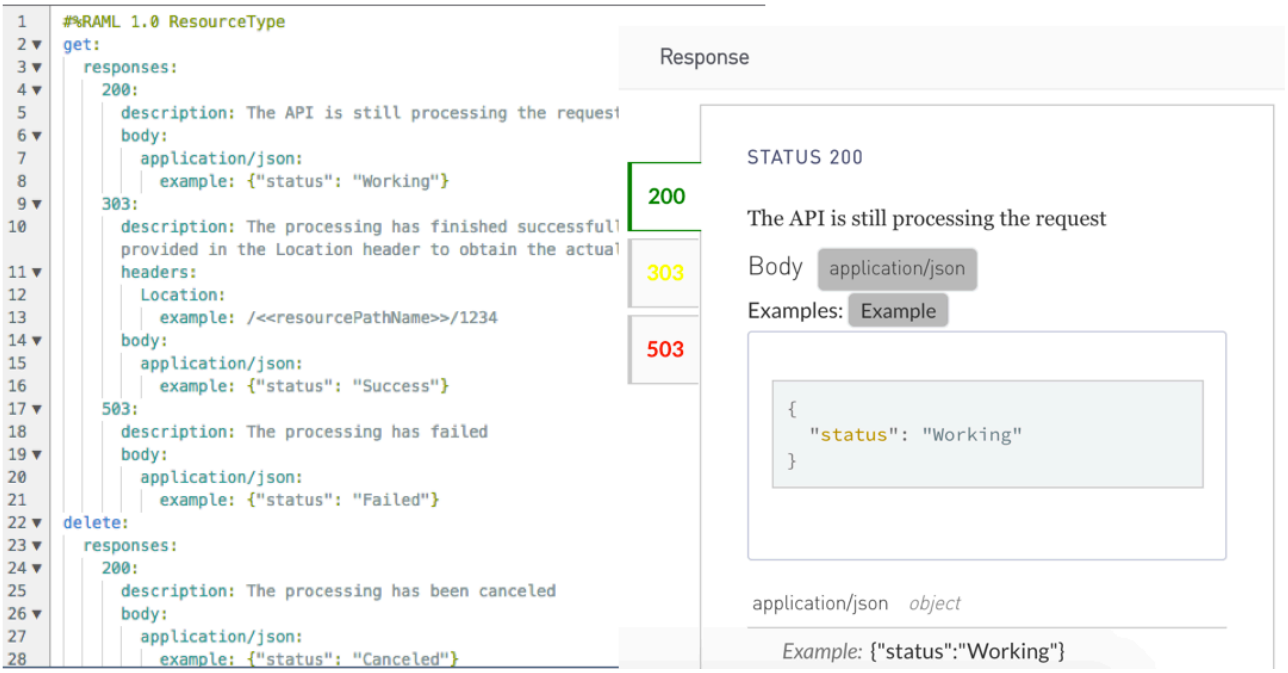
Objectives:

- Describe hypermedia.
- Simplify API discoverability using hypermedia.
- Use hypermedia to enhance API responses.
- Modify an API definition to generate state-specific client responses.

Walkthrough 12-1: Modify an API definition to generate state-specific client responses

In this walkthrough, you add state specific responses for methods that add data into the API implementation and go through multiple stages of processing information. You will:

- Create an HTTP 202 post body response with necessary headers.
- Add a new resource to track the state of post requests that are under processing.
- Add necessary HTTP methods with responses to give meaningful information back to the API consumers.



Create an HTTP 202 post body response with necessary headers

1. Return to API Designer.
2. In the file browser, expand the resourceTypes folder and select collection.raml.
3. Go to the line that contains the HTTP response status 503 and add a new line above it.

```
            Location:
              description: URL of the new <<resourcePathName |
              !singularize>> information
        body:
          application/json:

      503:
        body:
          application/json:
```

4. Align the cursor with the start of the next line and type

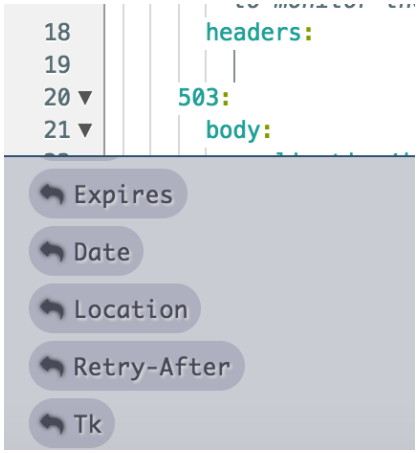
   `202:`

5. Add a new line below the response status line.
6. In the shelf, click description.
7. Type | and add a new line.
8. In the new line, type the following sentence:

   The request has been accepted for processing. Use the URI provided in the Location header in the response to monitor the status.

```
15 ▼    202:
16        description: |
17          The request has been accepted for processing. Use the URI
            provided in the Location header as a part of the response
            to monitor the status of the processing.
18 ▼    503:
```

9. Press enter and press backspace to go one tab space behind.
10. In the shelf, click headers.
11. In the shelf, scroll down and click Location.



12. In the shelf click example.
13. Type the value of the example node as:

    `/<<resourcePathName>>/1234/status`

```
15 ▼    202:
16        description: |
17          The request has been accepted for processing. Use the URI
            provided in the Location header as a part of the response
            to monitor the status of the processing.
18 ▼    headers:
19        Location:
20          example: /<<resourcePathName>>/1234/status
```

14. Save the project.

Add a new resource to track the state of post requests that are under processing

15. Go to the empty line before the /accounts main resource and press enter.
16. Press tab to align the cursor with the /accounts nested resource beginning and type:

```
/status:
```

```
47          is: [Traits.hasAcceptHeader, Traits.hasRe
            {typeName: Account, customErrorMessageDat
            CustomErrorMessage} ]
48      /status:
49
50 ▼ /accounts:
```

17. Save the project.

Add necessary methods and HTTP response statuses to give meaningful information back to the API consumers

18. In the file browser, right-click the resourceTypes folder and select New File > RAML 1.0 > Resource Type.
19. In the Add new Resource Type file dialog box, set the name to stateSpecificResourceType.raml and click OK.
20. In stateSpecificResourceType.raml, add a new line below the first line.
21. In the shelf, click get.

```
1  #%RAML 1.0 ResourceType
2
```

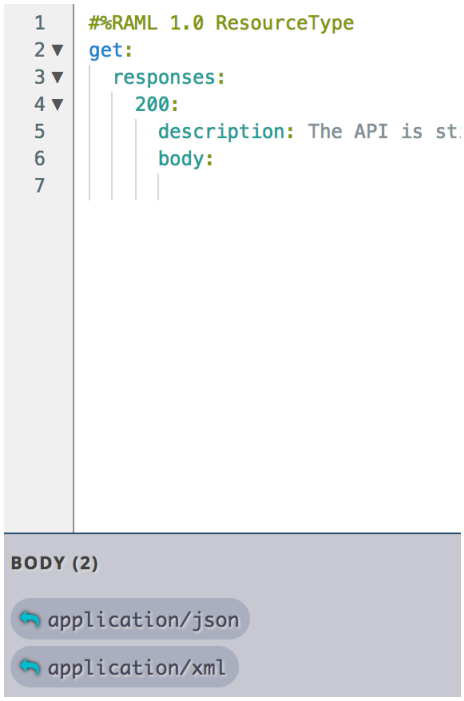
DOCS (3): description, displayName, usage

METHODS (7): get, put, post, delete, options, head, patch

22. In the shelf, click responses.
23. In the shelf, click 200.
24. In the shelf, click description.
25. Type the value of the description node as: The API is still processing the request.

```
#%RAML 1.0 ResourceType
get:
  responses:
    200:
      description: The API is still processing the request
```

26. Add a new line and in the shelf, click body.
27. In the shelf, click application/json.



28. In the shelf, click example, and type the value as:

```
{ “status”: “Working”}
```

```
#%RAML 1.0 ResourceType
get:
  responses:
    200:
      description: The API is still processing the request
      body:
        application/json:
          example: {"status": "Working"}
```

29. Press enter to add a new line.
30. Copy the lines containing the details for the 200 response paste it in the new empty line.

31. Press enter and paste the lines again.

```
8          example: {"status": "Working"}
9          200:
10       description: The API is still processing the request
11 ▼     body:
12 ▼       application/json:
13           example: {"status": "Working"}
14         200:
15       description: The API is still processing the request
16 ▼     body:
17         application/json:
18           example: {"status": "Working"}
```

32. In the first set of pasted lines, replace 200 with 303 and correct the indentation.

```
6 ▼    body:
7        application/json:
8          example: {"status": "Working"}
9 ▼    303:
10       description: The API is still processing the
11 ▼     body:
12 ▼       application/json:
```

33. Edit the description node value to:

    The processing has finished successfully. Use the URI provided in the Location header to obtain the actual result.

```
303:
  description: The processing has finished successfully. Use
  the URI provided in the Location header to obtain the actual
  result
  body:
    application/json:
      example: {"status": "Working"}
```

34. Press enter after the description to add a new line.
35. In the shelf, click headers.
36. In the shelf, click Location.
37. In the shelf, click example, type the value as:

    `/<<resourcePathName>>/1234`

```
9 ▼    303:
10       description: The processing has finished successfully. Use
         the URI provided in the Location header to obtain the actual
         result
11 ▼     headers:
12         Location:
13           example: /<<resourcePathName>>/1234
14 ▼     body:
```

Note: Always add a space between the node and the value

38. In the response JSON example, change the status value from Working to Success.
39. In the second set of pasted lines, change the status code from 200 to 503 and correct the indentation.

```
303:
  description: The processing has finished successfully. Use
  the URI provided in the Location header to obtain the actu
  result
  headers:
    Location:
      example: /<<resourcePathName>>/1234
  body:
    application/json:
      example: {"status": "Success"}
503:
  description: The API is still processing the request
```

40. In the description node below the 503 status code line, change the value of the node to: The processing has failed.
41. In the response JSON example, change the status value from Working to Failed.

```
      example: {"status": "Success"}
503:
  description: The processing has failed
  body:
    application/json:
      example: {"status": "Failed"}
```

42. Add a new line at the end and place the cursor at the beginning of the line.
43. In the shelf, click delete.
44. In the shelf, click responses.
45. In the shelf, click 200.
46. In the shelf, click description.

```
delete:
  responses:
    200:
      description:
```

47. Type the value of the description node as: The processing has been canceled.
48. Press enter and copy the lines for the HTTP response body and paste it in the HTTP 200 response body.

49. In the response JSON example, change the status value from Failed to Canceled.

```
delete:
  responses:
    200:
      description: The processing has been canceled
      body:
        application/json:
          example: {"status": "Canceled"}
```

50. Save the project.
51. In the file browser, select api.raml.
52. Go to the resourceType references at the root and add a new line below the member resourceType.
53. Indent the line and type:

```
stateSpecificResourceType: !include resourceTypes/stateSpecificResourceType.raml
```

```
resourceTypes:
  collection: !include resourceTypes/collection.raml
  member: !include resourceTypes/member.raml
  stateSpecificResourceType: !include
  resourceTypes/stateSpecificResourceType.raml
```

54. Add a new line below the line that contains /status nested method and press enter.
55. Press tab and type:

```
type: stateSpecificResourceType
```

```
    /status:
      type: stateSpecificResourceType
```

56. Copy the lines that define the status resource.

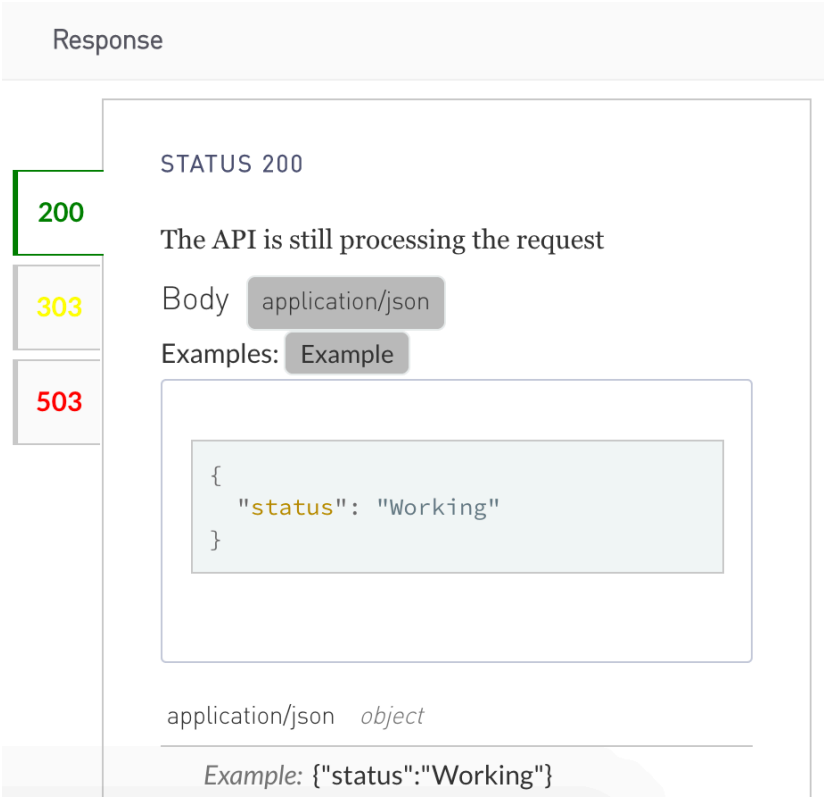
```
    /status:
      type: stateSpecificResourceType

/transactions:
```

57. Add the /status resource as a nested resource for /accounts and /transactions main resource.

```
  /{transaction_id}:
    type: { member: { uriParameterName: transaction_i
    getResponseBodyDataType: Transaction, getResponseE
    !include examples/TransactionExample.json,
    customErrorMessageType: CustomErrorMessage } }
  /status:
    type: stateSpecificResourceType
```

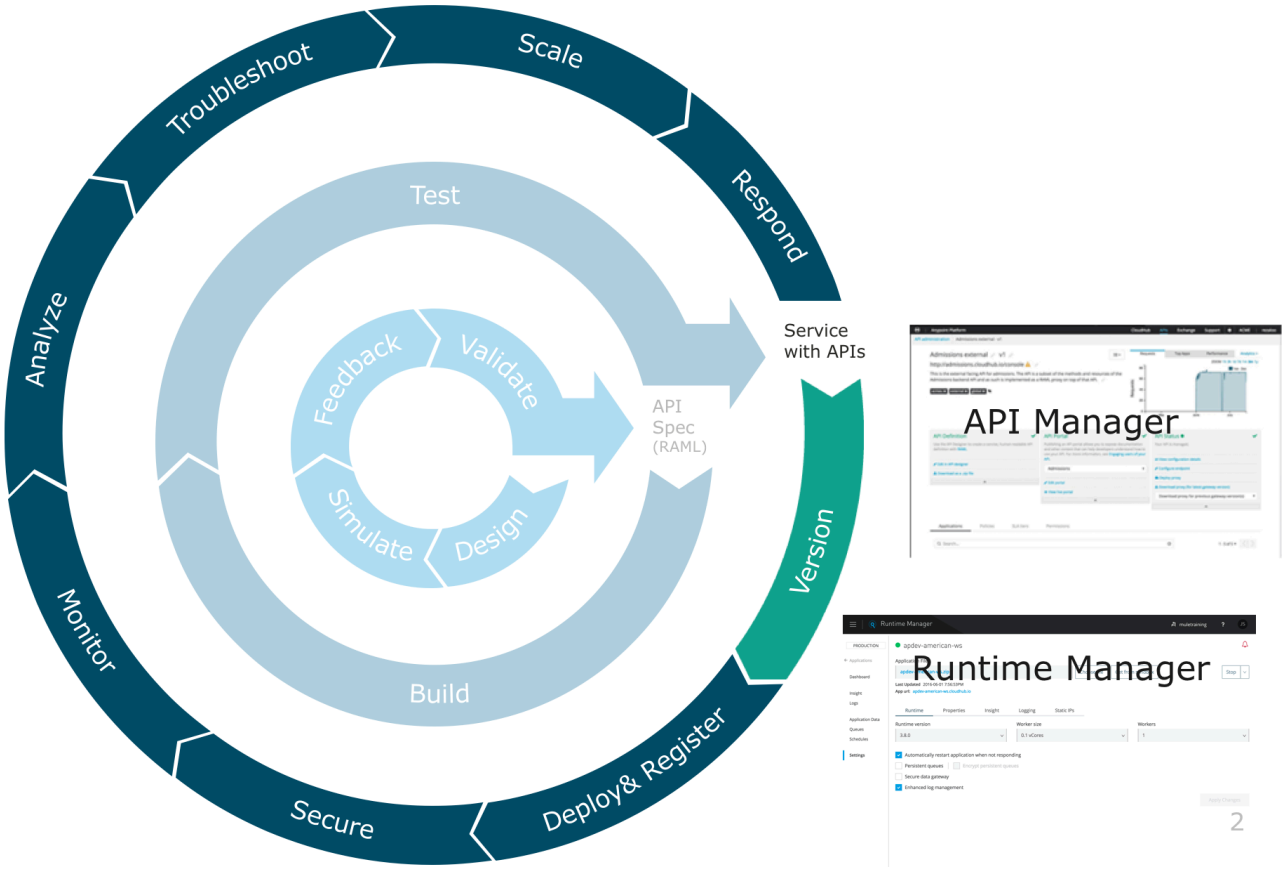
58. Save the project.
59. In API Console, click the GET tab for the /customers/{customer_id}/status resource.

60. Scroll down to the response and verify that you can see the various status codes for the responses.



61. Click CLOSE X.

Module 13: Versioning APIs



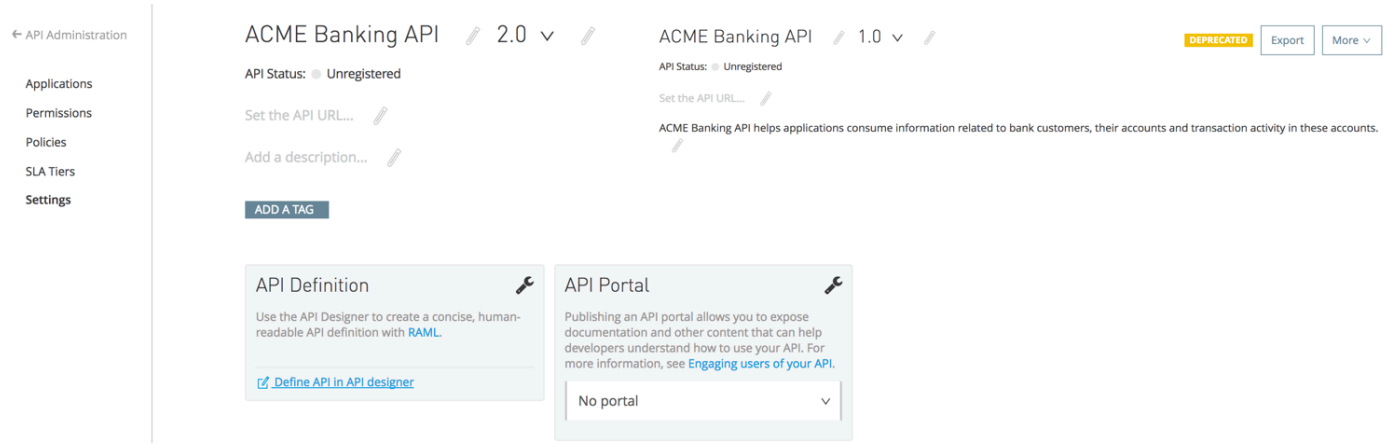
Objectives:

- Explain when and when not to version APIs.
- Describe the methods for versioning APIs.
- Document changes in new API versions using shared API Portals.
- Deprecate older versions of APIs.

Walkthrough 13-1: Add a new API version

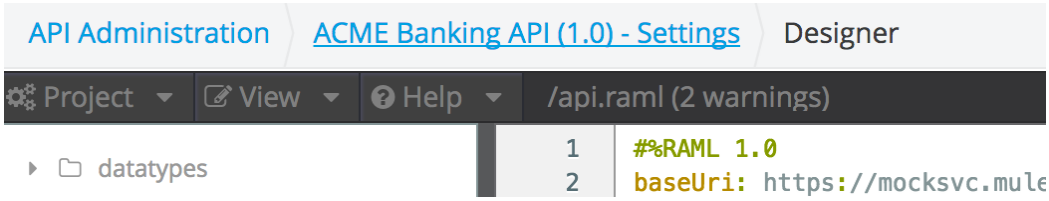
In this walkthrough, you create a new version of the ACME Banking API. You will:

- Create a new API version in API Manager.
- Share the older version API Portal for the new version of the API.
- Compare the API Portal items in the two versions.
- Deprecate the old version of the API.



Add a new version of API definition in the API Designer

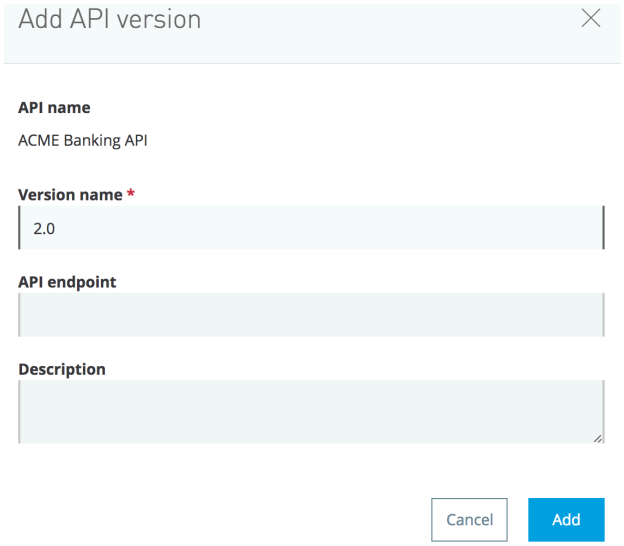
1. Return to API Designer.
2. Click the API Administration link above the API Designer menu.



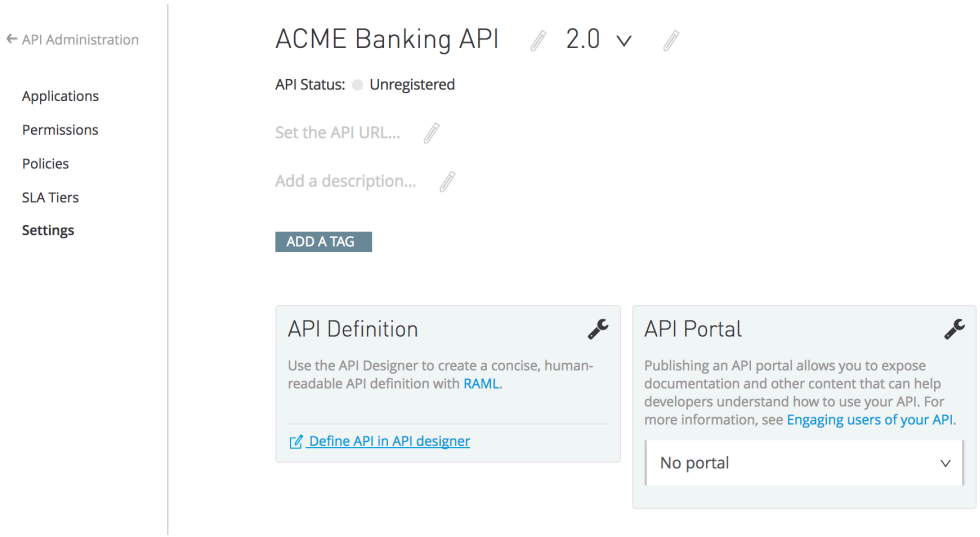
3. On the API administration page, click the Add Version button to the right of the ACME Banking API.

4. In the Add version drop-down menu, select New Version.

5. In the Add API version dialog box, set the Version name to 2.0.



6. Click Add.
7. In the API Definition section, click Define API in API Designer.



8. In the RAML editor, place the cursor at the end of the line that defines the version and press enter two times.
9. Type /{version}: and press enter.

```
1  #%RAML 0.8
2  title: ACME Banking API
3  version: 2.0
4
5  /{version}:
```

10. Press tab and type /customers:.

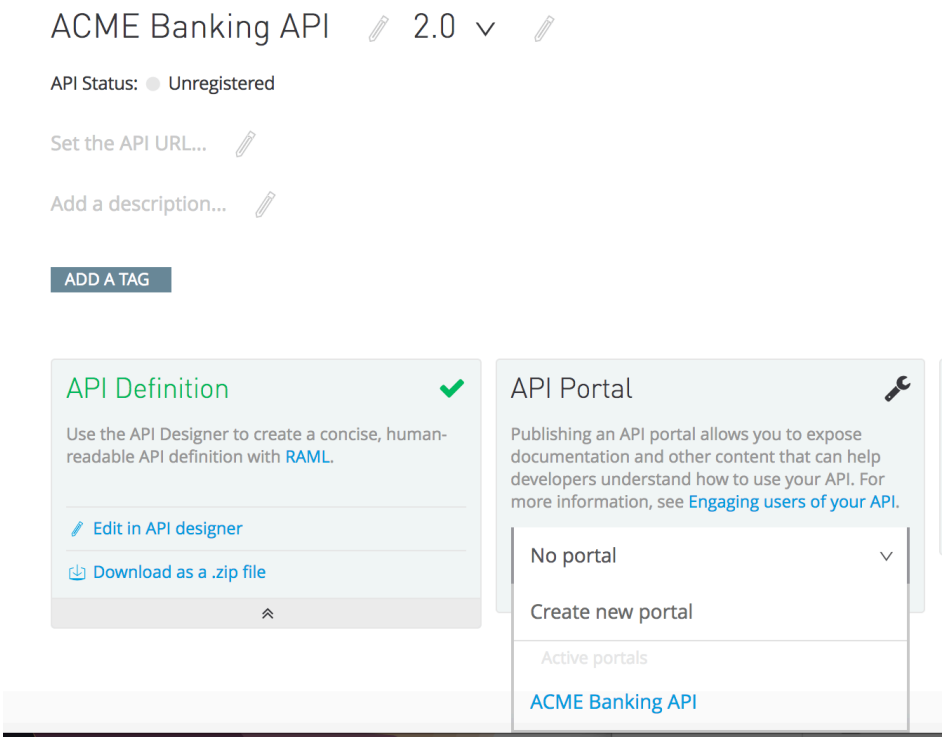
```
#%RAML 0.8
title: ACME Banking API
version: 2.0

/{version}:
  /customers:
```

11. Save the project.

    Note: Versioning is done only when APIs are live and when the previous version is deployed and being used by API consumers. We do not fill in the RAML definition for v2.0 since versioning is not necessary in this scenario.

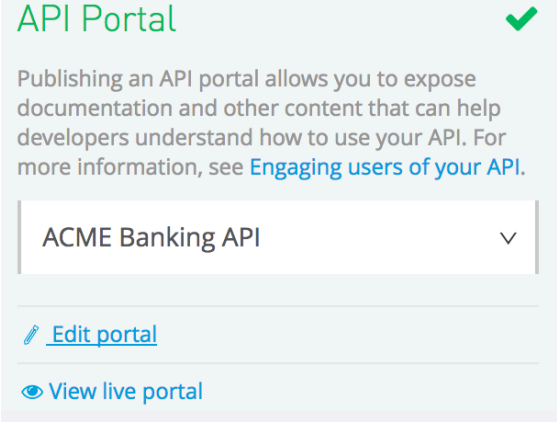
Link multiple versions of the API in the shared portal

12. In API Designer, click ACME Banking API (2.0) - Settings to go to the API version details page.
13. In the API Portal section, click the drop-down arrow next to the No portal option.



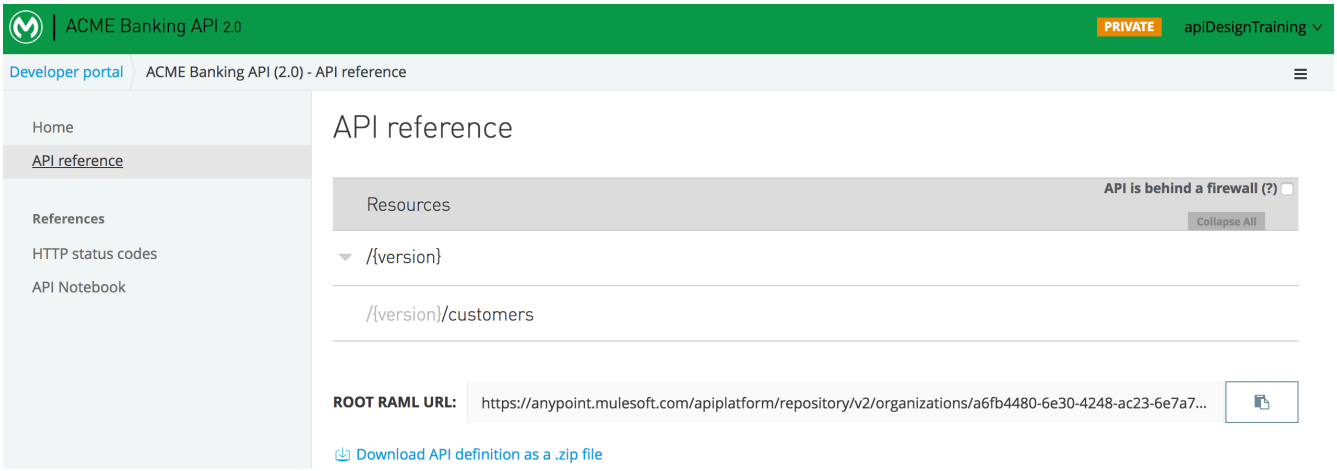
14. Select the ACME Banking API active portal from the list.

15. Click View live portal.



Compare the non-identical portal items

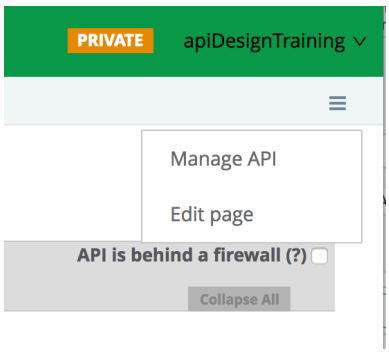
16. In the left-side navigation, select the API reference page.
17. Verify that you see the version 2.0 RAML API Definition resources in API Console.



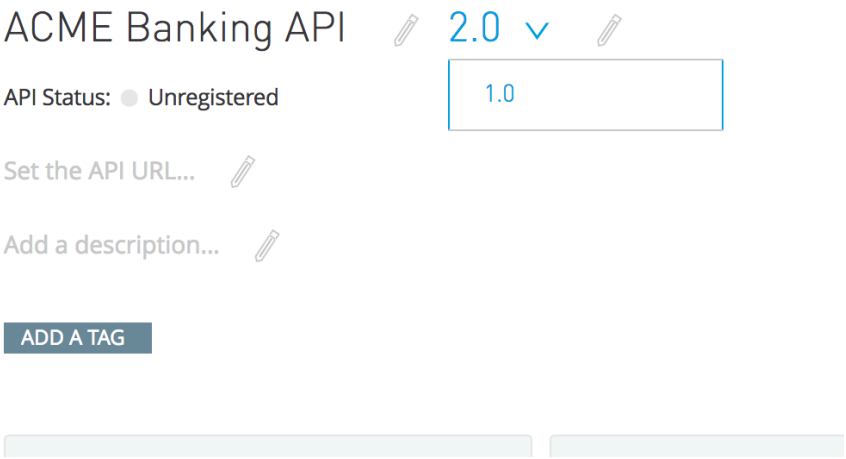
18. Copy the link of the API portal URL for ACME Banking API 2.0 and paste it in the course snippets.txt file in Module 13 under the Version 2.0 API Portal URL heading.

```
-------Module 13--------

Version 2.0 API Portal URL:

https://anypoint.mulesoft.com/apiplatform/mulesoft-1581/#/portals/organizations/a6fb4480-6e30-4248-ac23-6e7a751b5606/apis/11698133/versions/178909/pages/225831

Version 1.0 API Portal URL:
```

19. Click the menu icon in the right top corner and select Manage API.



20. Click the version 2.0 drop-down in the top center and select 1.0.



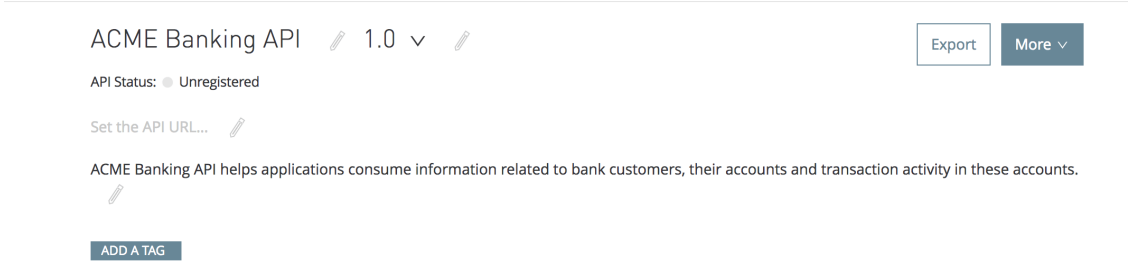
21. In the API Portal section, click View live portal.
22. Copy the link to the API portal URL for ACME Banking API 1.0 and paste it in the course snippets.txt file.
23. Verify that the version ids in the URL are different for the different versions.

```
Version 2.0 API Portal URL:

https://anypoint.mulesoft.com/
apis/11698133/versions/178909/

Version 1.0 API Portal URL:

https://anypoint.mulesoft.com/
apis/11698133/versions/162026
```

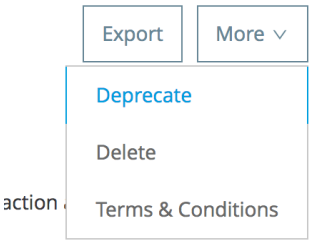
Deprecate the older version of the API

24. Click the menu icon in the top-right corner of the API Portal and select Manage API.

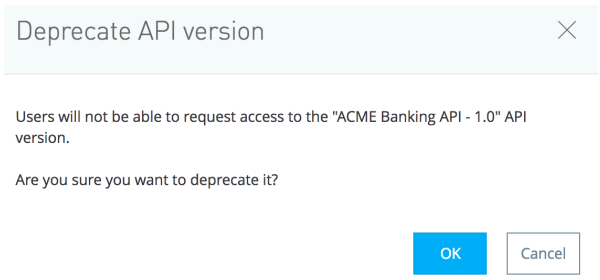
25. Click the More button to the top right corner in the API version details page.



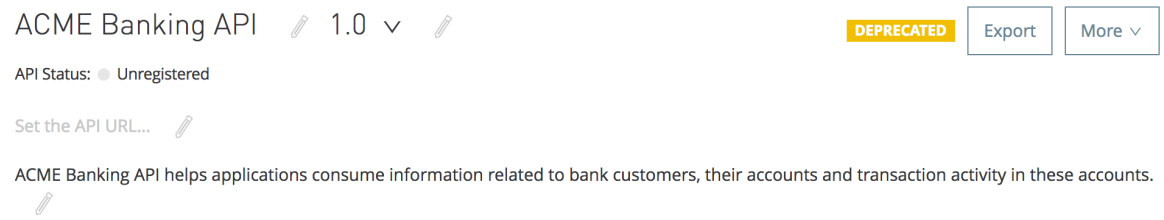
26. Select Deprecate.



27. In the dialog box that appears, click OK.



28. Verify that you see a yellow Deprecated tag next to the API name and version.



Note: It is not a best practice to deprecate APIs that are not being consumed. This is just a walkthrough to demonstrate the process of versioning and deprecation.