Module 3: Designing APIs

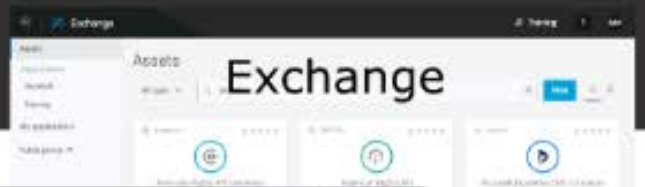
Spec driven development

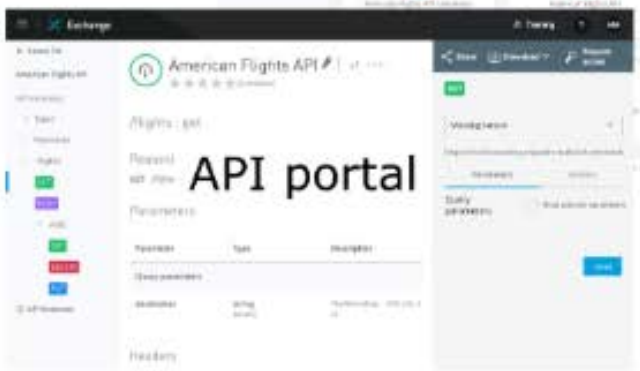
- We discussed in the last modules about the benefits of designing an API first before actually building it

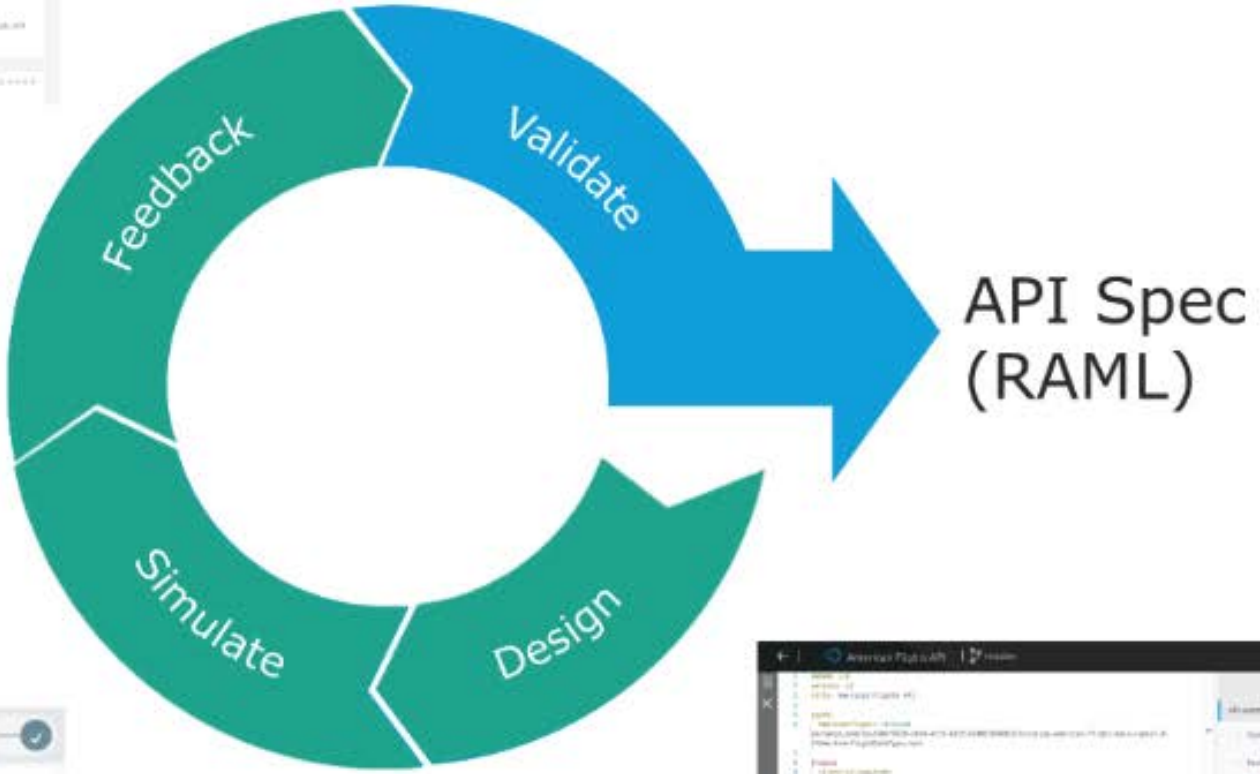
- This is often referred to as **spec driven development**
  - A development process where your application is built in two distinct phases
    - The creation of a spec (the design phase)
    - Development of code to match the spec (the development phase)

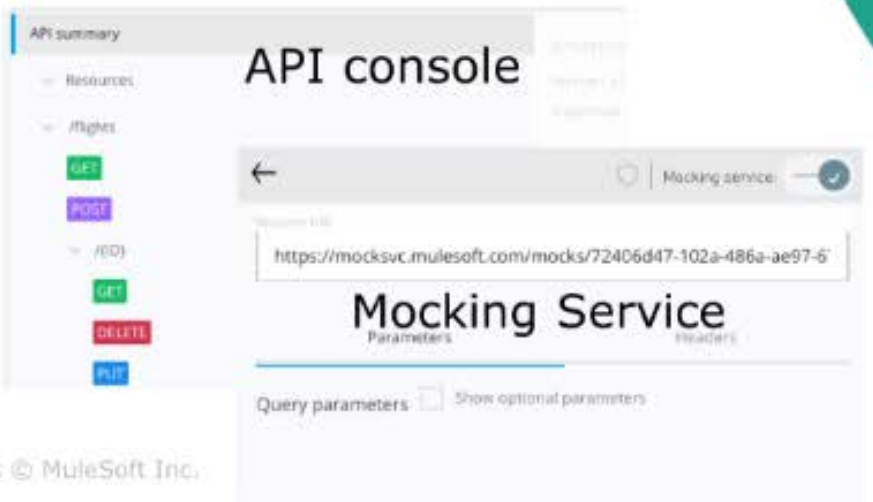
- In this module, we'll
  - Create this API specification using a standardized API description language (RAML)
  - Then learn to test it with users without writing any code

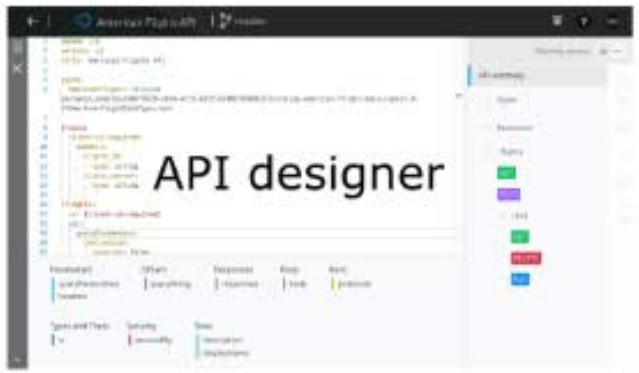
Goal











At the end of this module, you should be able to

- Define APIs with RAML, the Restful API Modeling Language
- Mock APIs to test their design before they are built
- Make APIs discoverable by adding them to the private Anypoint Exchange
- Create public API portals for external developers

Reviewing the options for defining APIs

Approaches to API design

MuleSoft

Hand coding

API Blueprint

apiary

OpenAPI Spec

OPEN
{API}
INITIATIVE

RAML

RAML

Introducing RAML

RAML: RESTful API Modeling Language

MuleSoft

- **A simple, structured, and succinct way of describing RESTful APIs**

RAML

- A non-proprietary, vendor-neutral open spec

http://raml.org

- Developed to help out the current API ecosystem
  - Encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices

- RAML files can be used to auto-generate documentation, mocked endpoints, interfaces for API implementations, and more!

RAML syntax

- RAML is based on broadly-used standards such as YAML and JSON
- Uses a human-readable data serialization format where **data structure hierarchy is specified by indentation**
  - Not additional markup characters

```
1   #%RAML 1.0
2   version: v1
3   title: American Flights API
4
5   /flights:
6     get:
7     post:
8
9     /{ID}:
10      get:
11      delete:
12      put:
13        responses:
14          200:
15            body:
16              application/json:
```

Notice the indentation used to specify to what each line applies

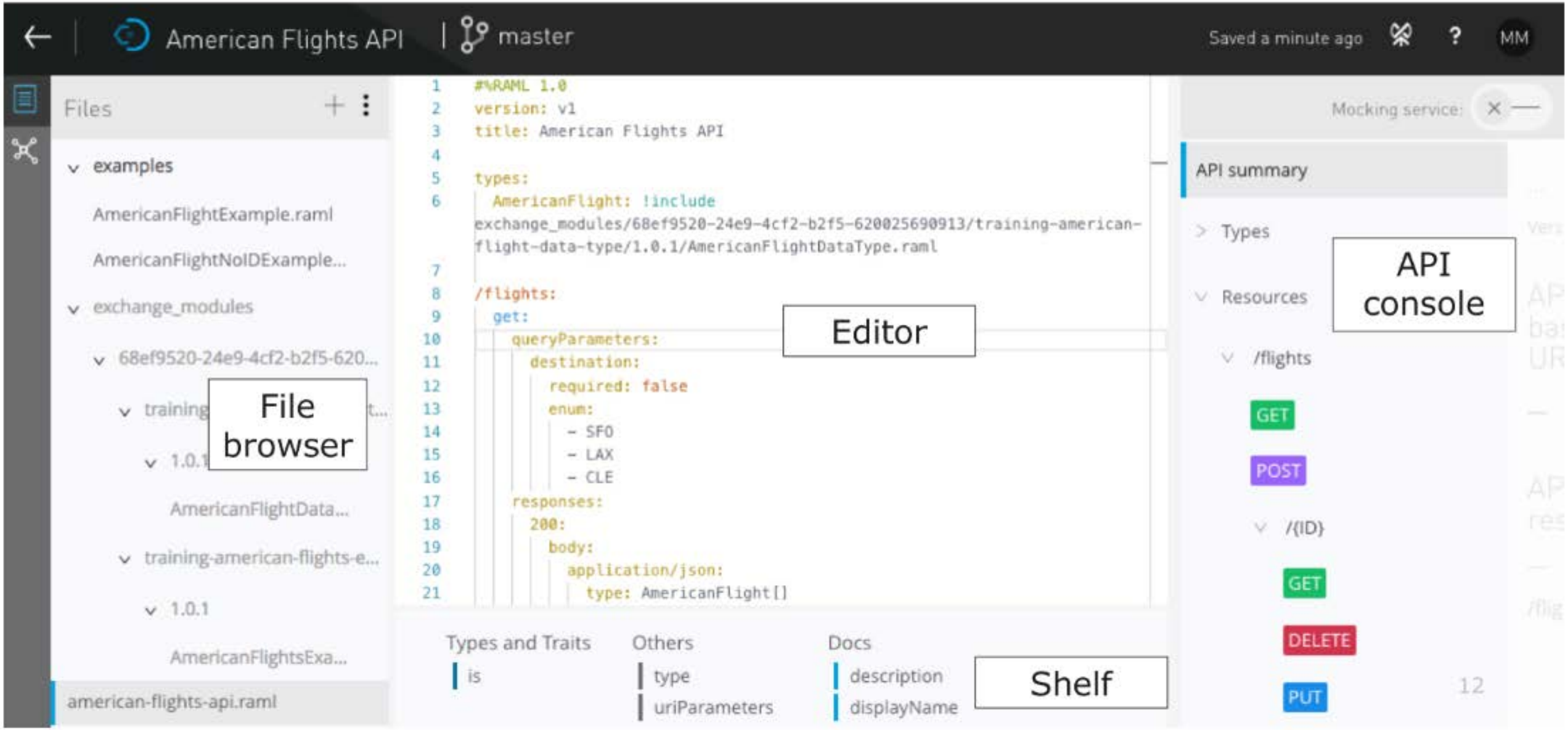
Defining resources and methods

- Resources are the objects identified by the web service URL that you want to act upon using the HTTP method used for the request
- All resources begin with a slash
- Any methods and parameters nested under a resource belong to and act upon that resource
- Nested resources are used for a subset of a resource to narrow it
  - URI parameter are enclosed in {}

```
1   #%RAML 1.0
2   version: v1
3   title: American Flights API
4
5   /flights:
6     get:
7     post:
8
9     /{ID}:
10      get:
11      delete:
12      put:
13        responses:
14          200:
15            body:
16              application/json:
```

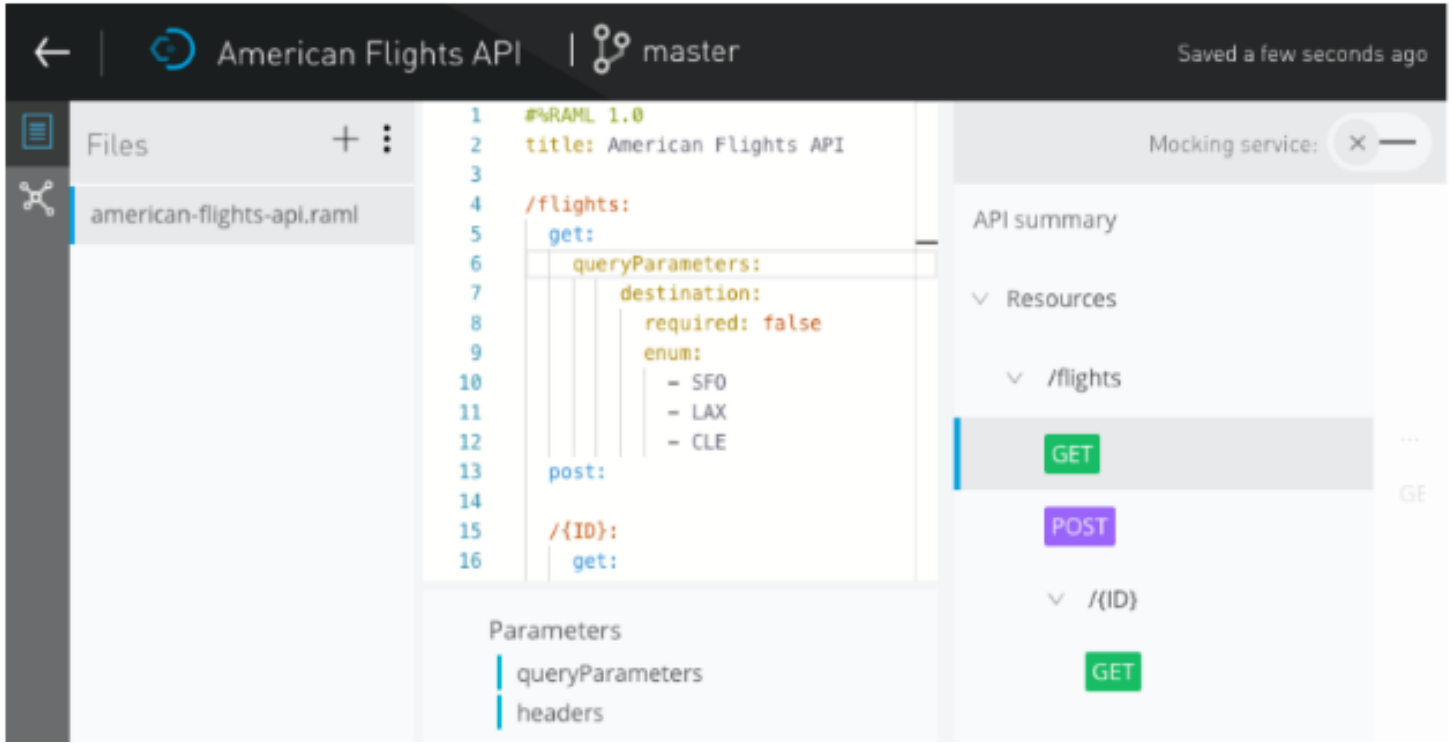
Using API designer to define APIs with RAML

API designer



Walkthrough 3-1: Use API designer to define an API with RAML

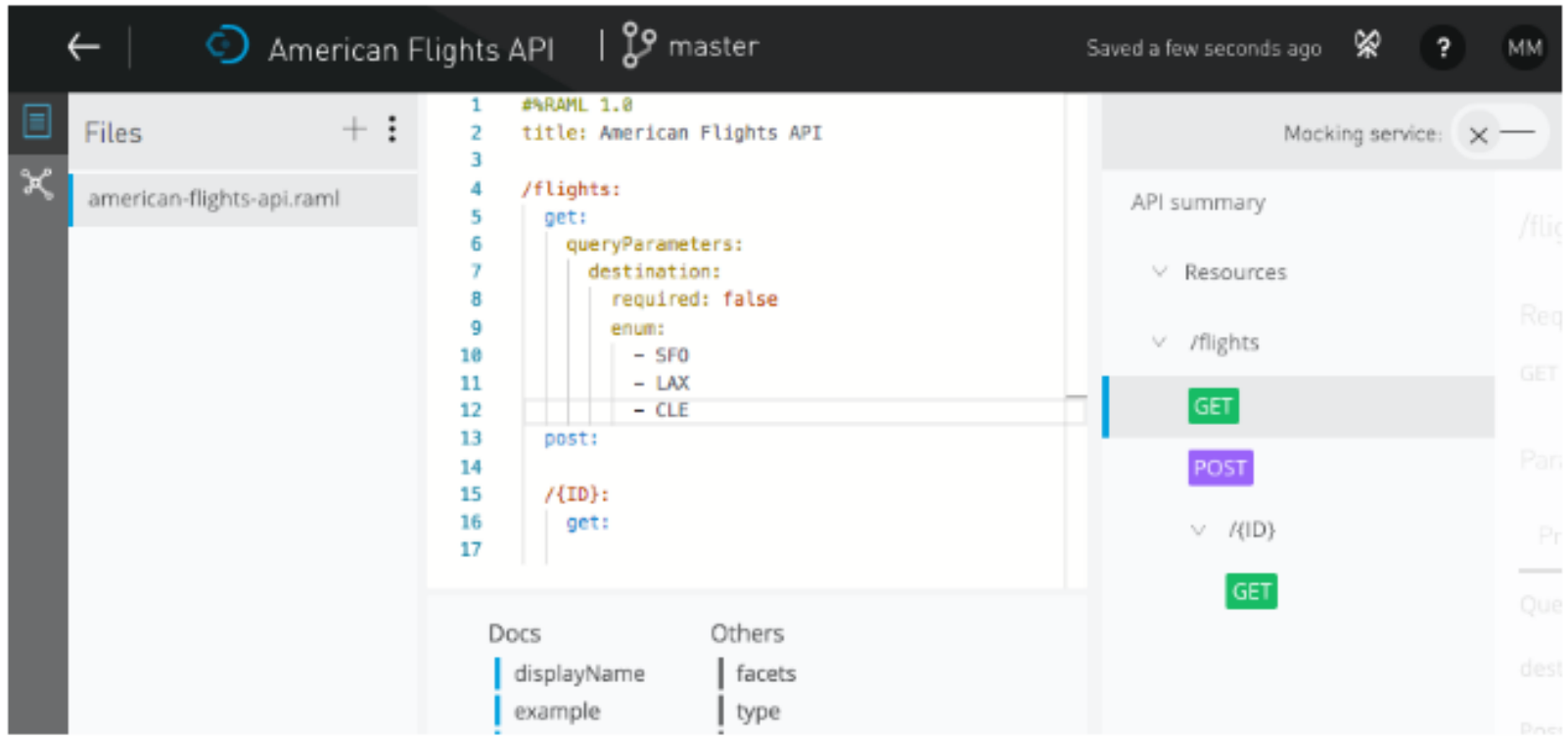
- Define resources and nested resources
- Define get and post methods
- Specify query parameters
- Interact with an API using the API console



Walkthrough 3-1: Use API designer to define an API with RAML

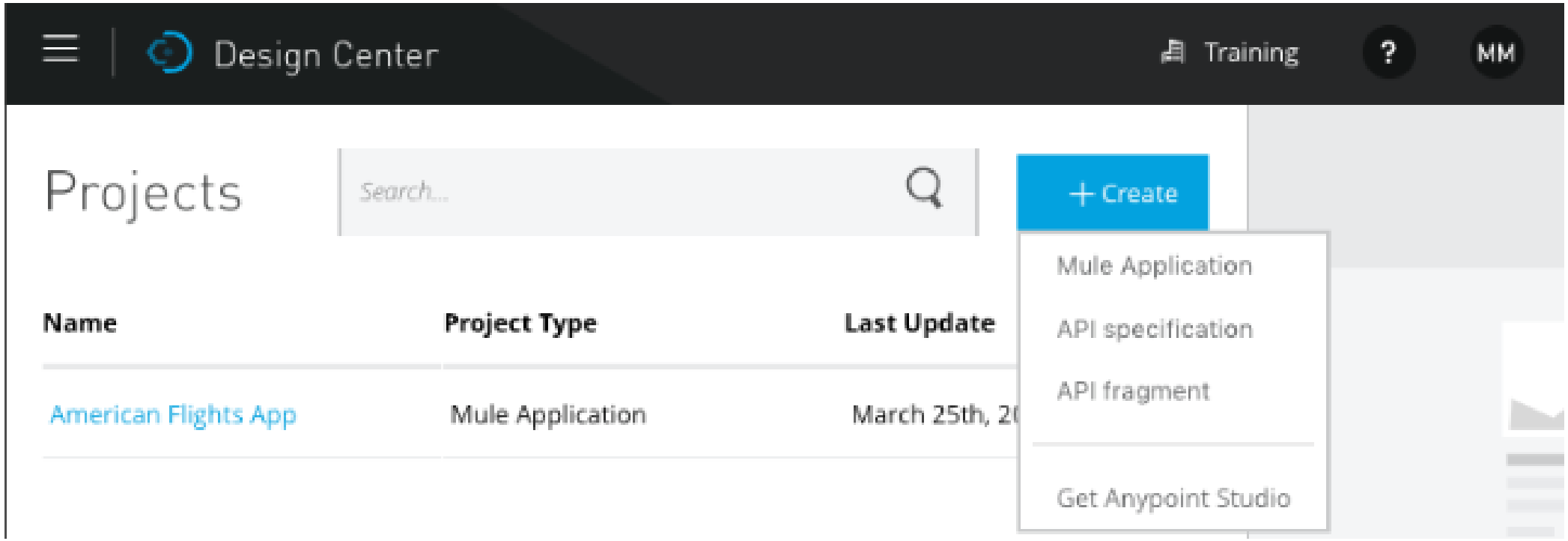
In this walkthrough, you create an API definition with RAML using API designer. You will:

- Define resources and nested resources.
- Define get and post methods.
- Specify query parameters.
- Interact with an API using the API console.



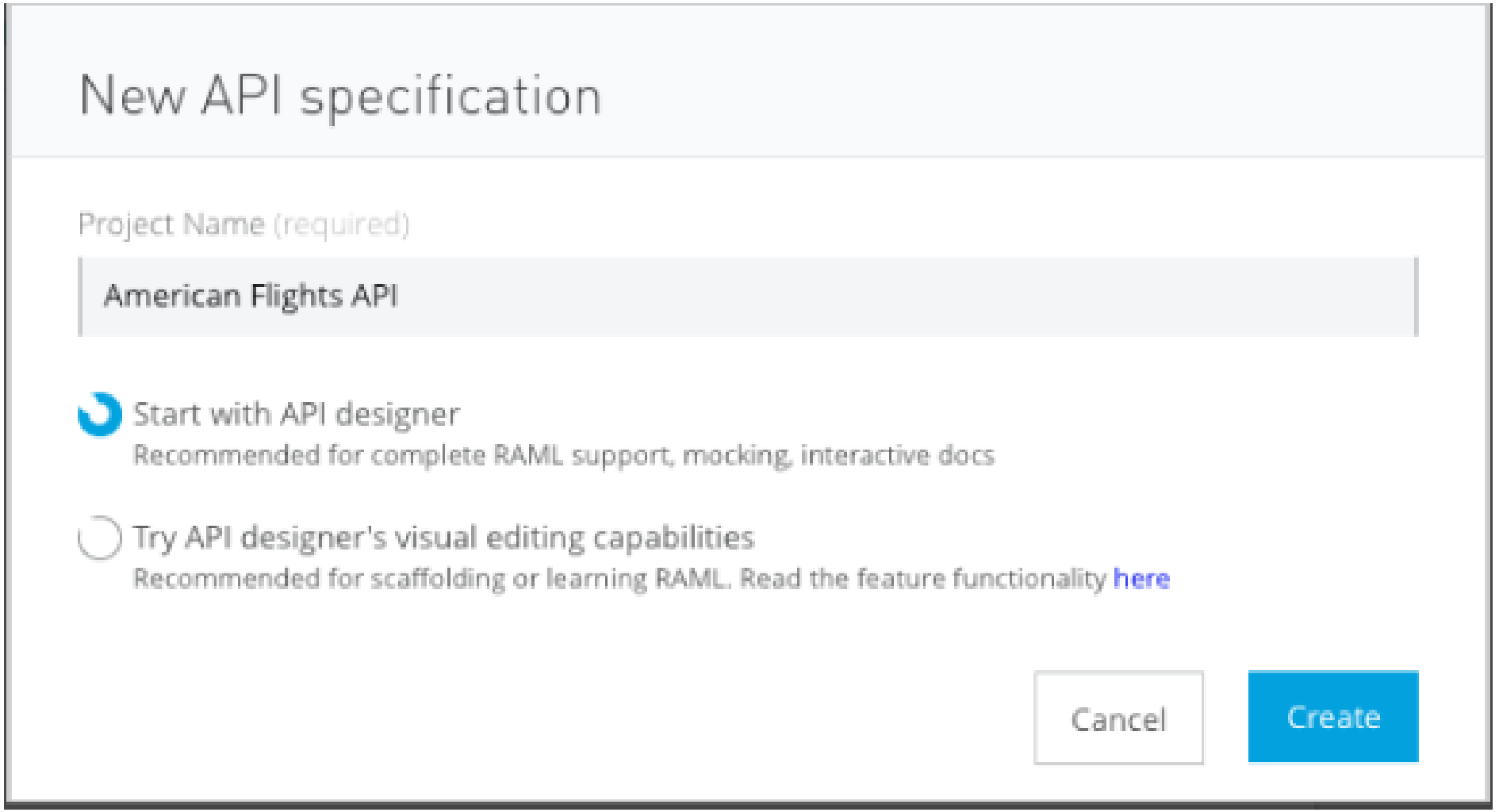
Create a new Design Center project

1. Return to Design Center.
2. Click the Create button and select API specification.



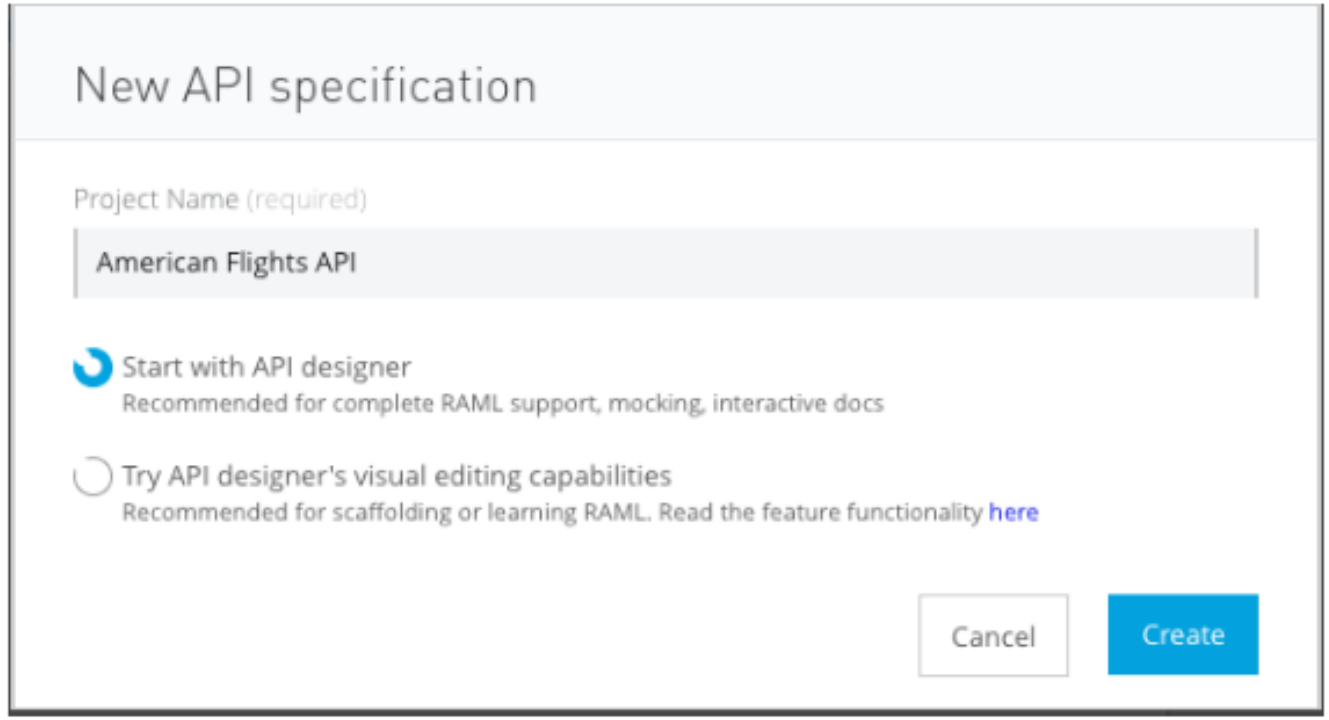
3. In the New API specification dialog box, set the project name to American Flights API.

4. Select Start with API designer and click Create; API designer should open.

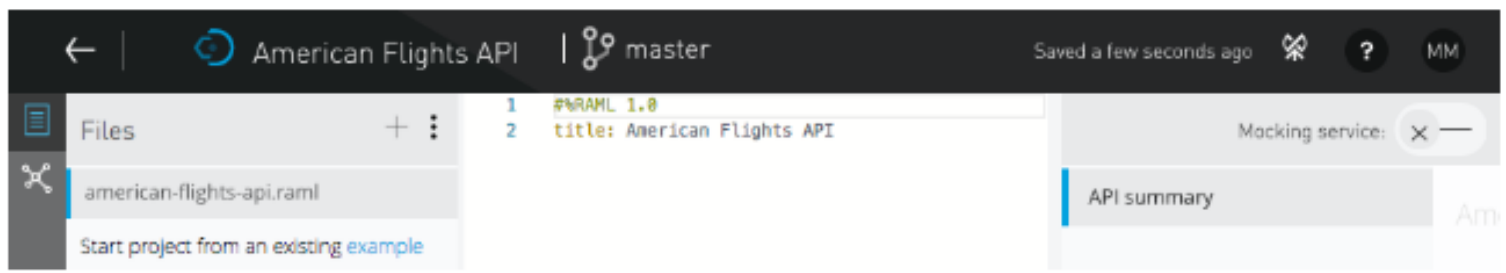


5. In API designer, click the Hide these tips link in the popup window.

4. Select Start with API designer and click Create; API designer should open.



5. In API designer, click the Hide these tips link in the popup window.
6. Review the three sections of API designer: the file browser, the editor, and the API console.



Add a RAML resource

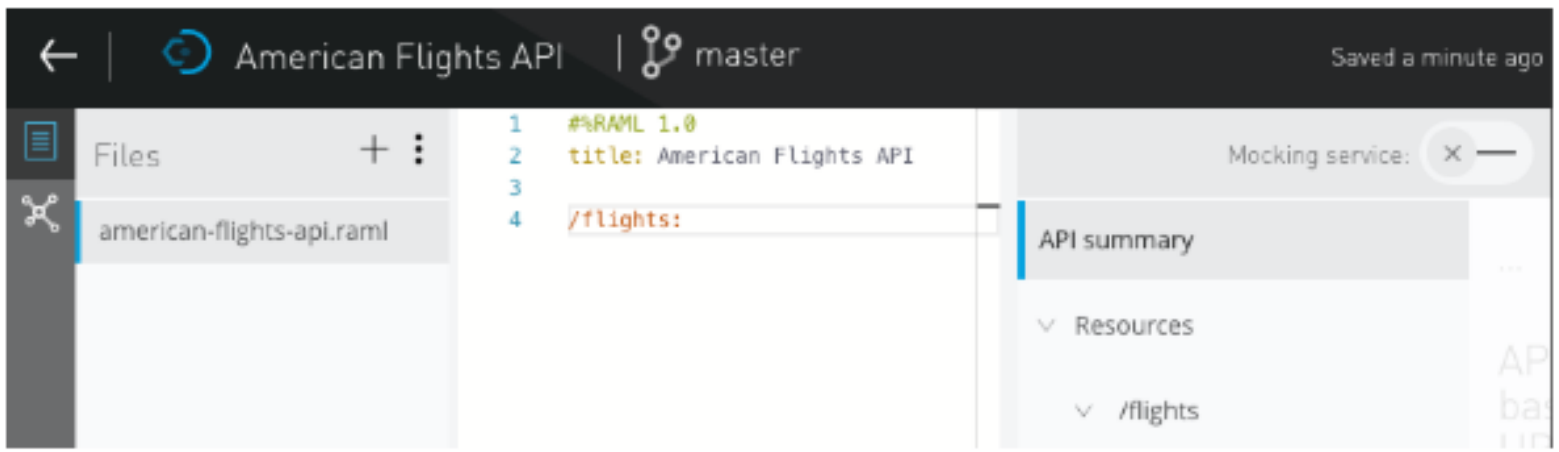
7. In the editor, place the cursor on a new line of code at the end of the file.
8. Add a resource called flights.

```
1    #%RAML 1.0
2    title: American Flights API
3
4    /flights:
```

View the API console

9. Look at the API console on the right side of the window; you should see summary information for the API.

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



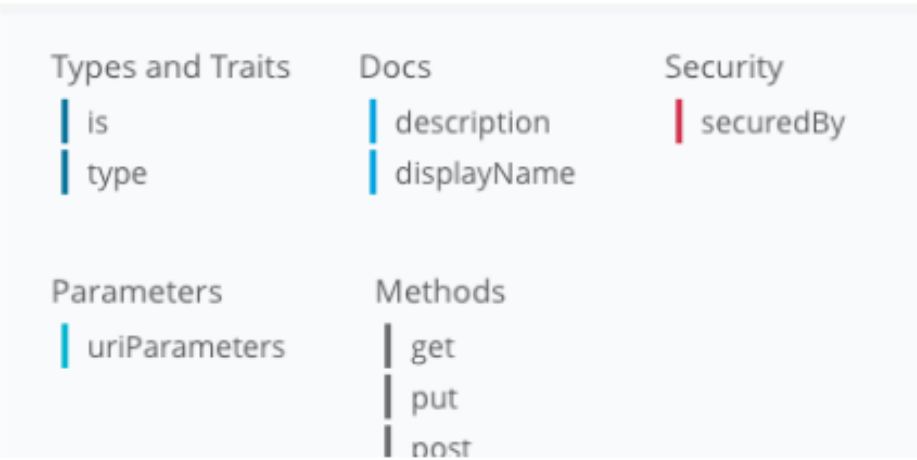
Add RAML methods

10. In the editor, go to a new line of code and look at the contents of the API designer shelf.

    Note: If you don't see the API designer shelf, it is either minimized or there is an error in your code. To check if it is minimized, go to the bottom of the web browser window and look for an arrow. If you see the arrow, click it to display the shelf.

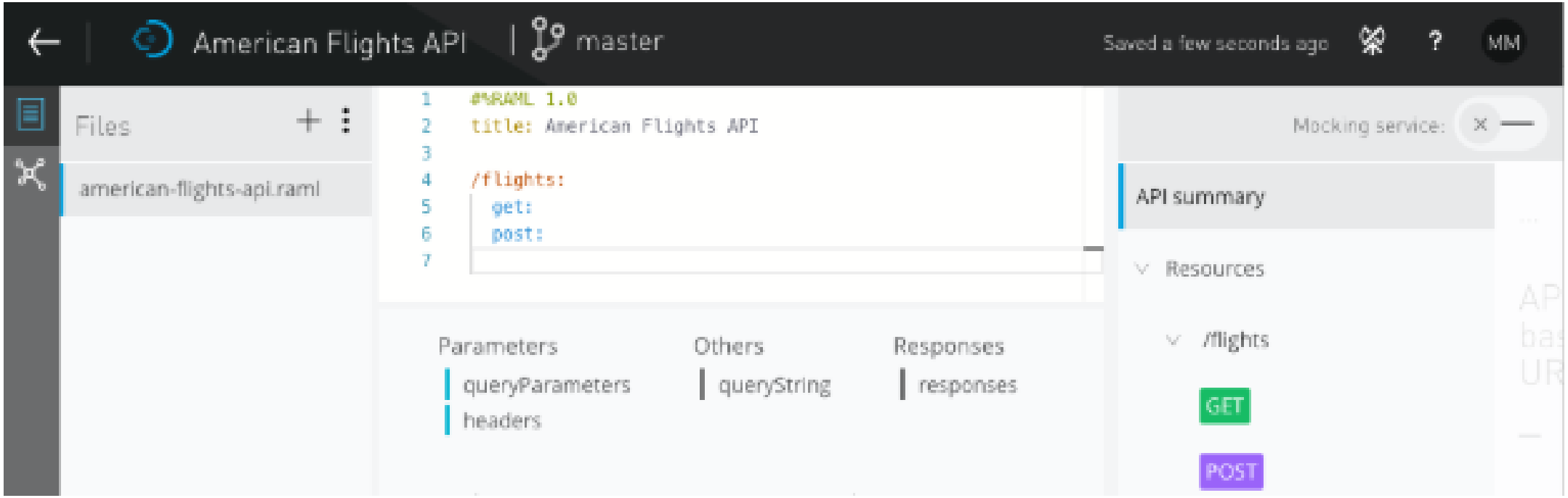
11. Indent by pressing the Tab key; the contents in the API designer shelf should change.

```
1    #%RAML 1.0
2    title: American Flights API
3
4    /flights:
5
```



12. Click the get method in the shelf.
13. Look at the API console; you should see a GET method for the flights resource.
14. In the editor, backspace so you are indented the same amount as the get method.
15. Click the post method in the shelf.

16. Look at the API console; you should see GET and POST methods for the flights resource.

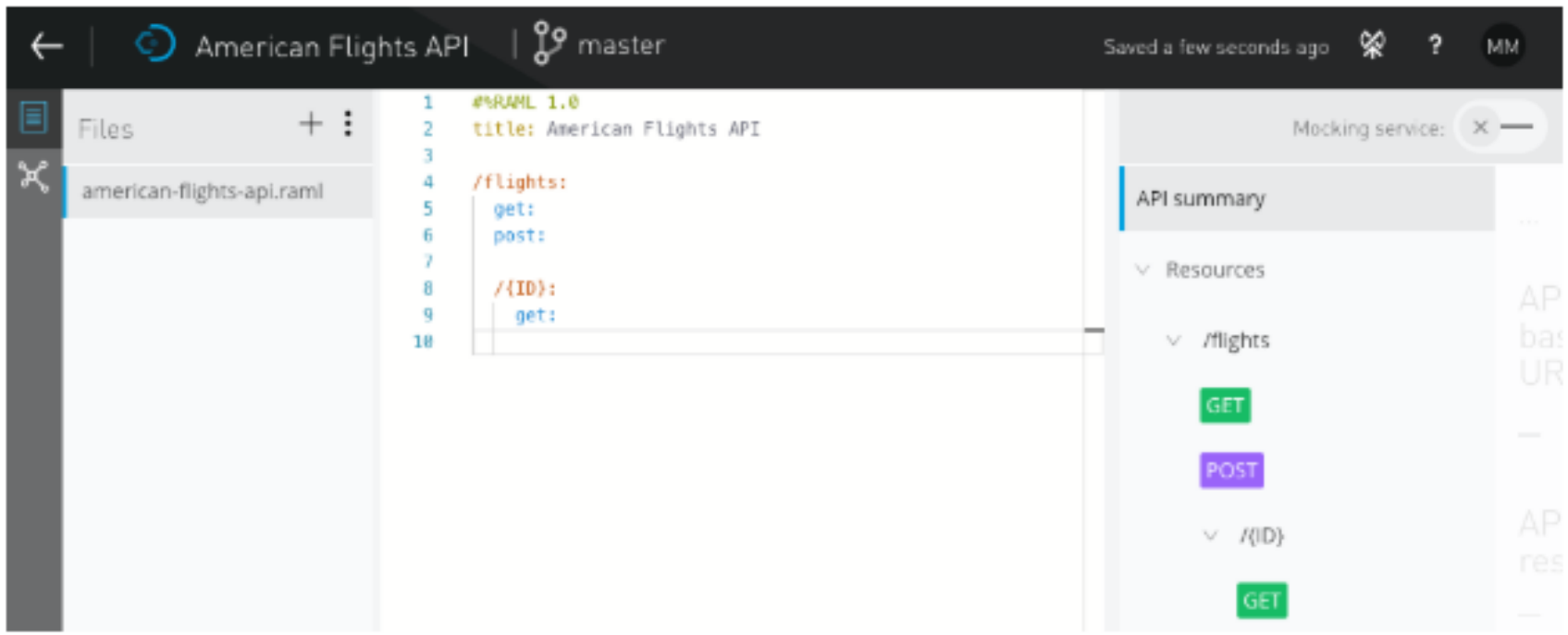


Add a nested RAML resource

17. In the editor, backspace and then go to a new line.
18. Make sure you are still under the flights resource (at the same indentation as the methods).
19. Add a nested resource for a flight with a particular ID.

    `/{ID}:`

20. Add a get method to this resource.
21. Look at the API console and expand the /{ID} resource; you should see the nested resource with a GET method.



Add an optional query parameter

22. In the editor, indent under the /flights get method (not the /flights/{ID} get method).
23. In the shelf, click the queryParameters parameter.
24. Add a key named destination.

```
1    #%RAML 1.0
2    title: American Flights API
3
4    /flights:
5      get:
6        queryParameters:
7          destination:
8      post:
9
10     /{ID}:
```

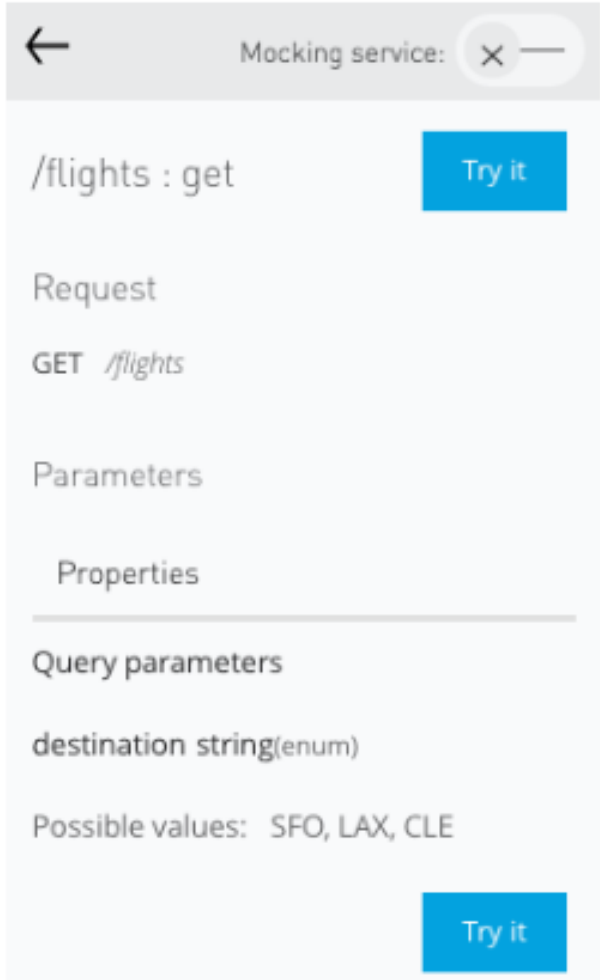
25. Indent under the destination query parameter and look at the possible parameters in the shelf.
26. In the shelf, click the required parameter.
27. In the shelf, click false.
28. Go to a new line of code; you should be at the same indent level as required.
29. In the shelf, click the enum parameter.

30. Set enum to a set of values including SFO, LAX, and CLE.

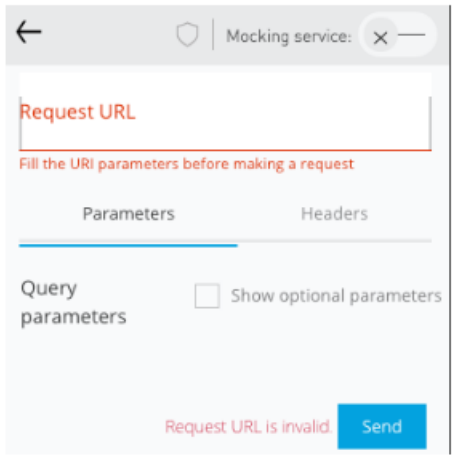
```yaml
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
```

Try to call an API method using the API console

31. In the API console, click the GET method for the /flights resource.
32. Review the information.



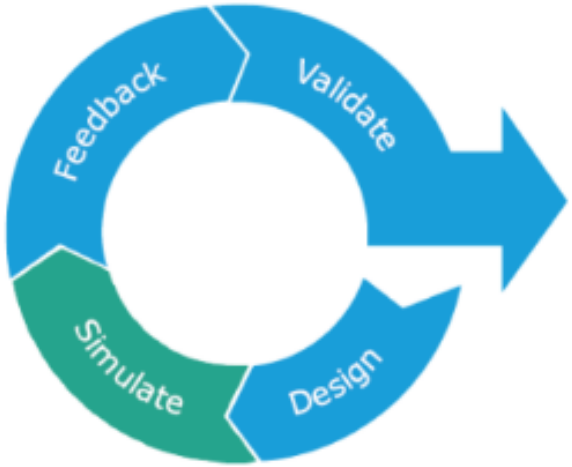
33. Click the Try it button; you should get a message that the Request URL is invalid; a URL to call to try the API needs to be added to the RAML definition.



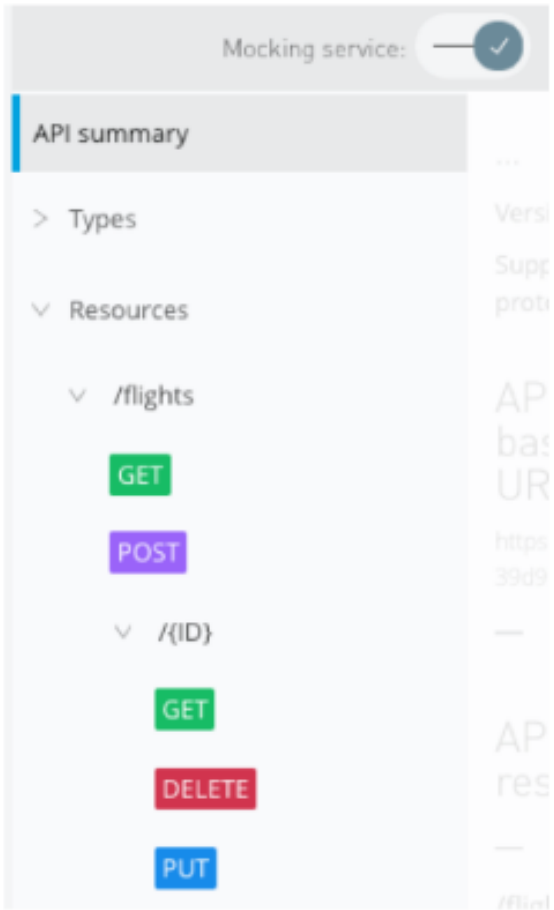
Testing API design without writing code

Simulating an API

- You can mock an API to test it before it is implemented
  - Useful to get early feedback from developers



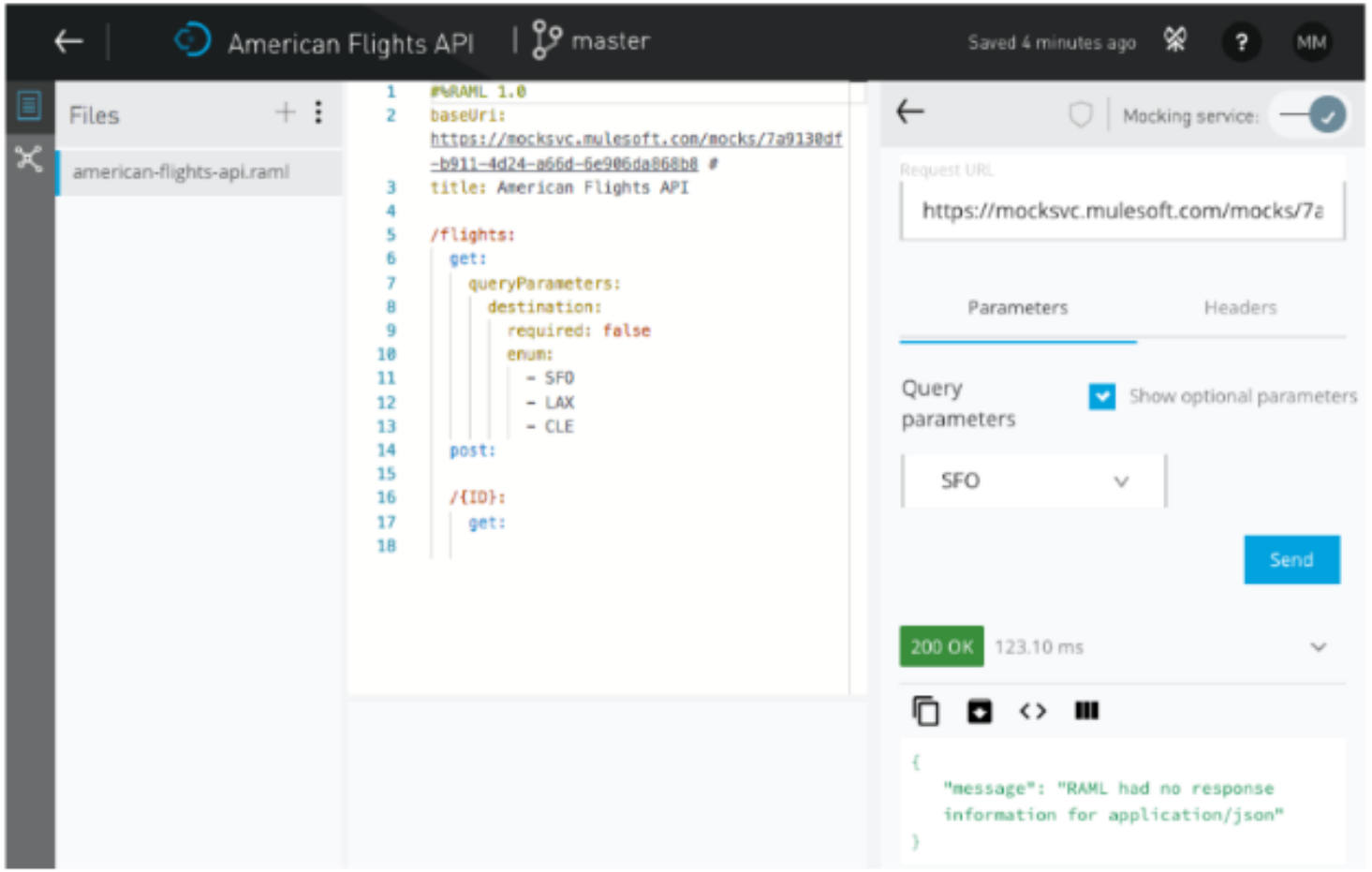
- Use the **API console** and the **mocking service** to run a live simulation
  - Returns sample API responses defined in the API definition



- The API console is available in
  - **API designer** – so the API designer can test it
  - **API portals in Exchange** – so users/developers can test it

Walkthrough 3-2: Use the mocking service to test an API

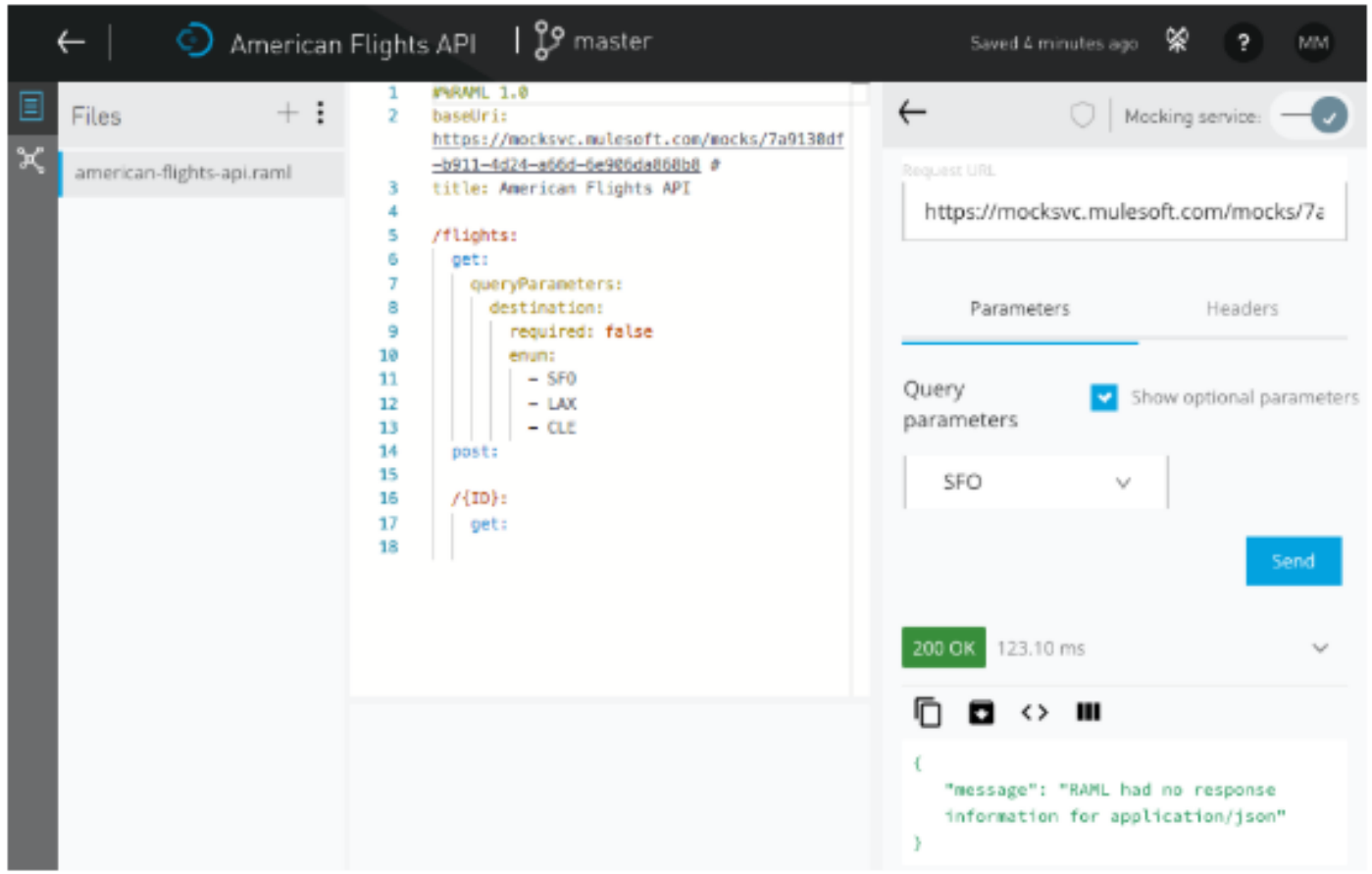
- Turn on the mocking service
- Use the API console to make calls to a mocked API



Walkthrough 3-2: Use the mocking service to test an API

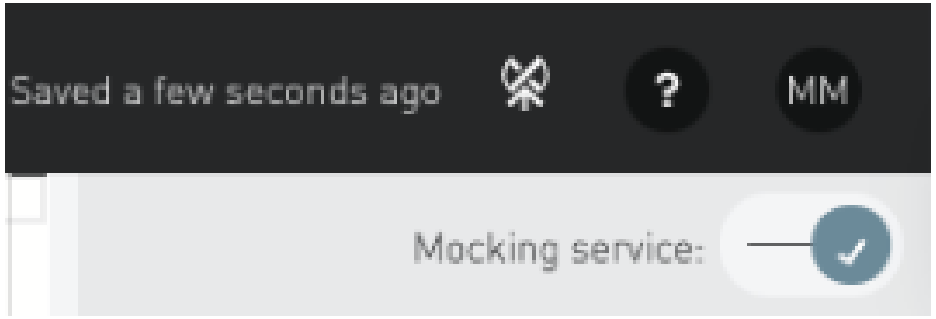
In this walkthrough, you test the API using the Anypoint Platform mocking service. You will:

- Turn on the mocking service.
- Use the API console to make calls to a mocked API.



Turn on the mocking service

1. Return to API designer.
2. Locate the Mocking Service slider in the menu bar at the top of the API console.
3. Click the right side of the slider to turn it on.

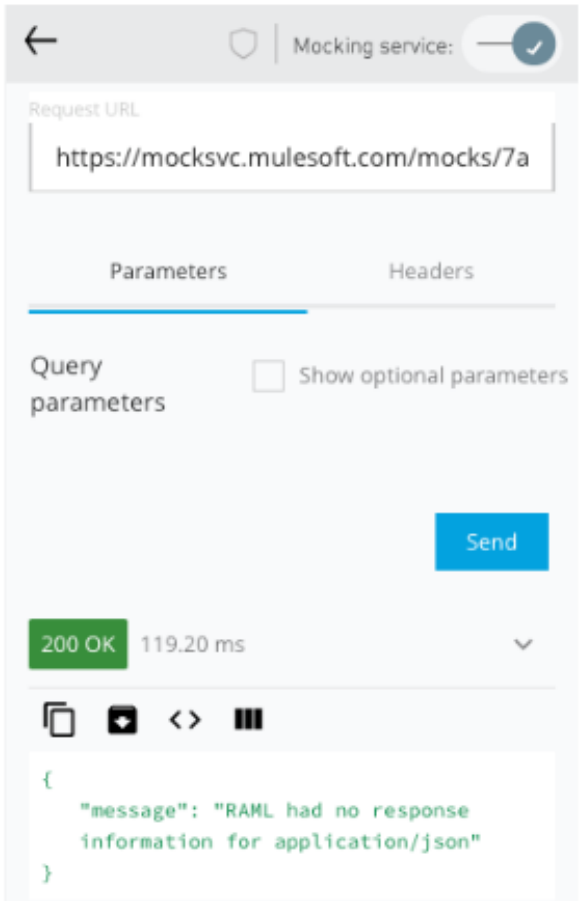


4. Look at the baseUri added to the RAML definition in the editor.

```
1   #%RAML 1.0
2   baseUri: https://mocksvc.mulesoft.com/mocks/f02c935c-66b1-4a68-9bf2-d7beb3affe53 #
3   title: American Flights API
```

Test the /flights:get resource

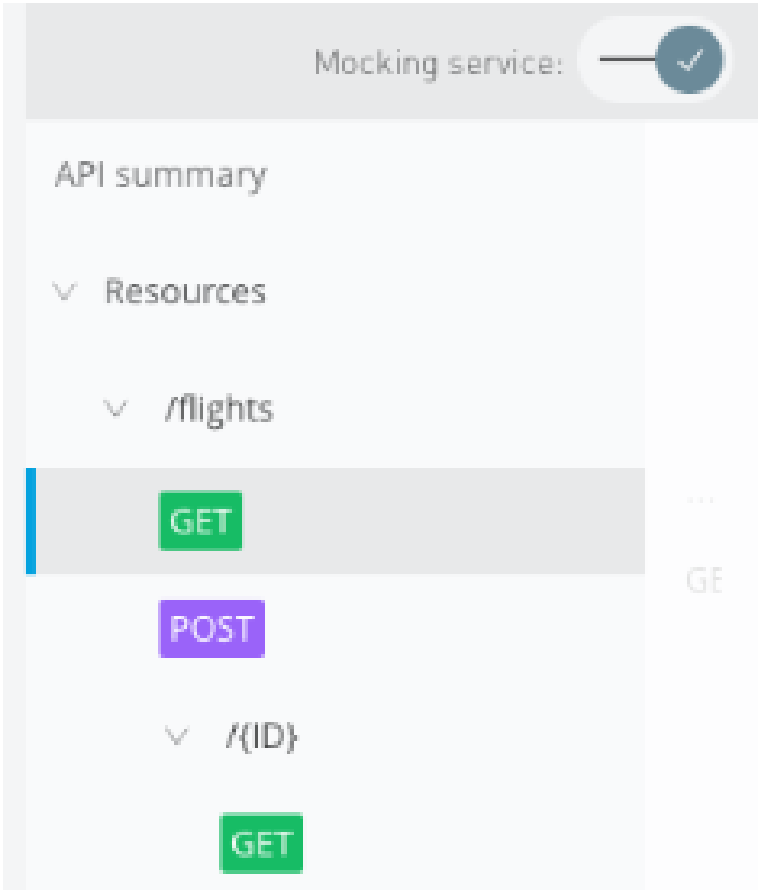
5. In API console, click the Send button for the flights GET method; you should get a 200 status code, a content-type of application/json, and a general RAML message placeholder.



6. Select the Show optional parameters checkbox.
7. In the destination text field, select SFO and click Send; you should get the same response.

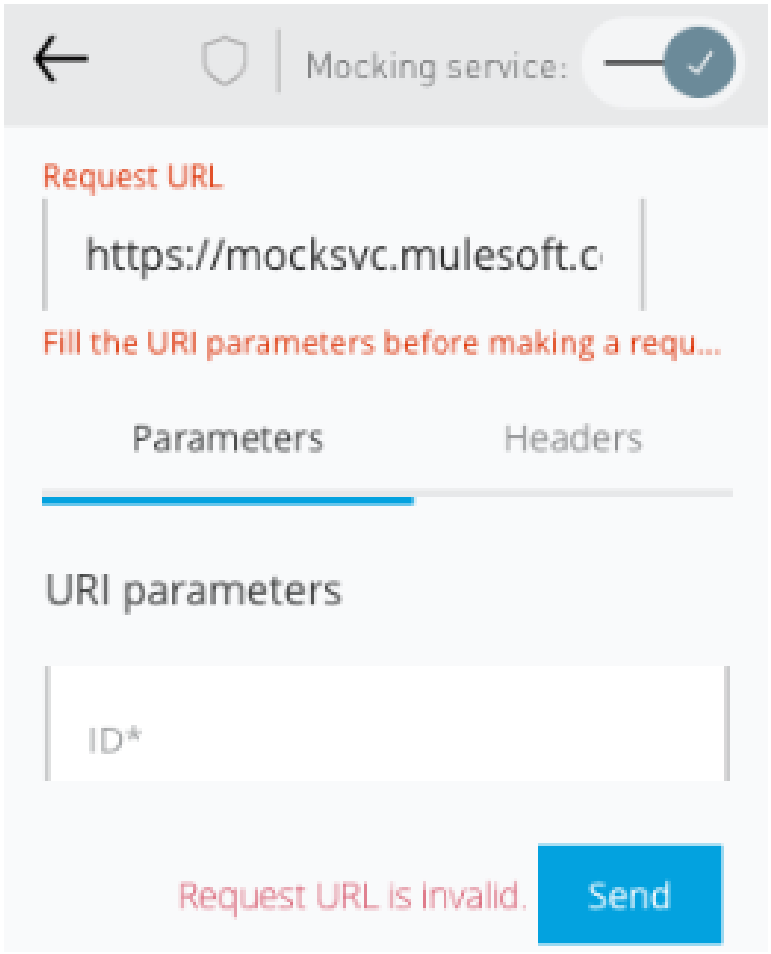
Test the /flights/{ID} resource

8. Click the back arrow at the top of API console twice to return to the resource list.



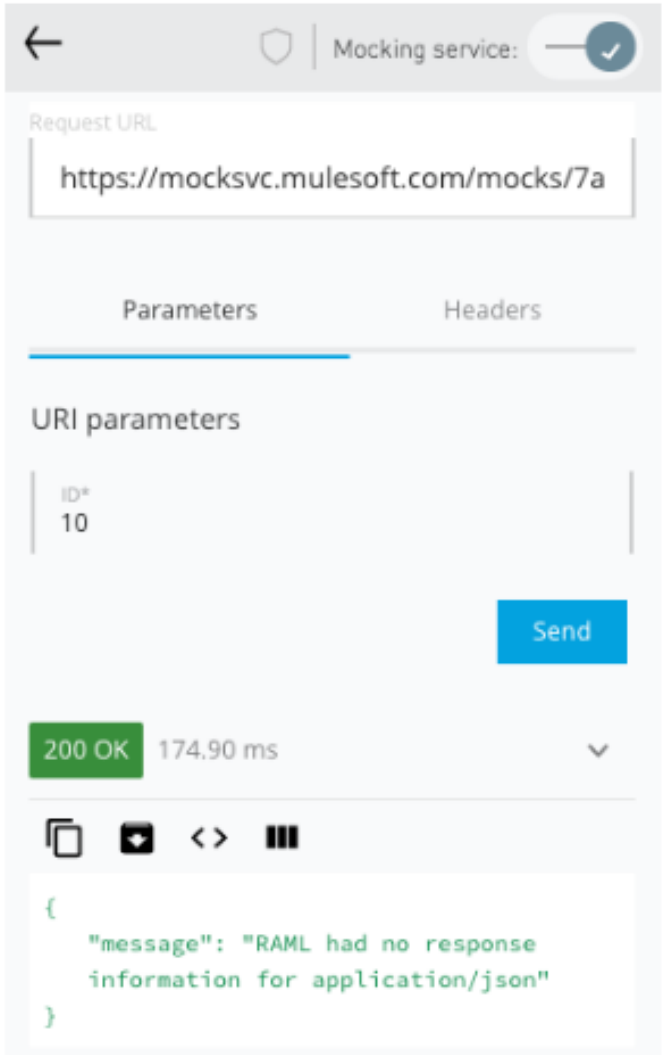
9. Click the GET method for the /{ID} nested resource.

10. Click Try it; you should see a message next to the Send button that the Request URL is invalid.



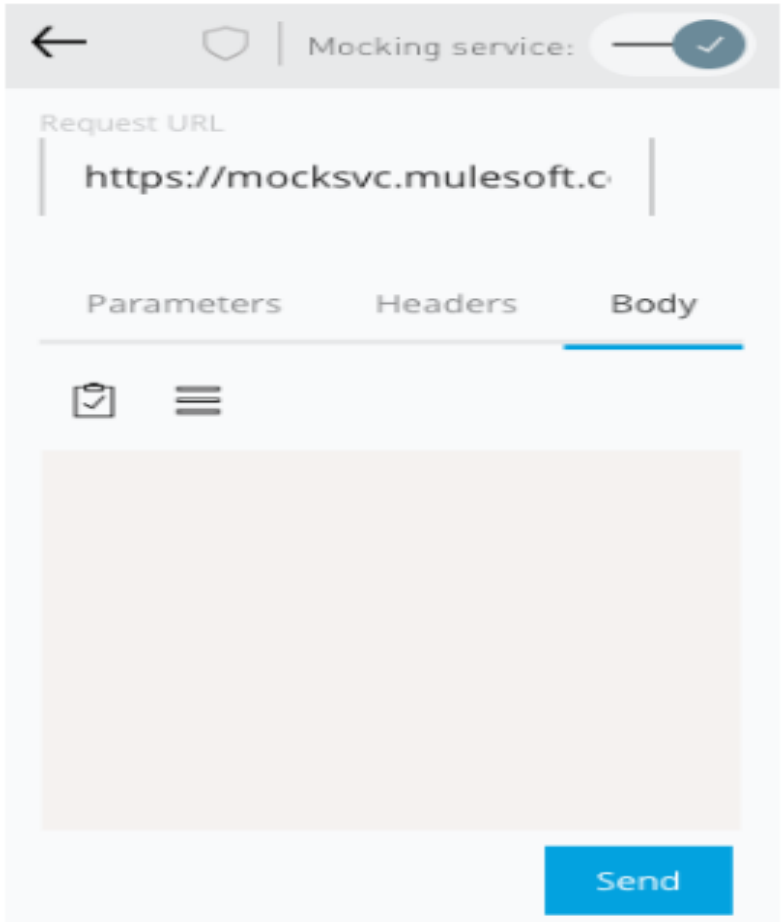
11. In the ID text box, enter a value of 10.
12. Click the Send button.

13. Look at the response; you should get the same default response with a 200 status code, a content-type of application/json, and the general RAML mess placeholder.

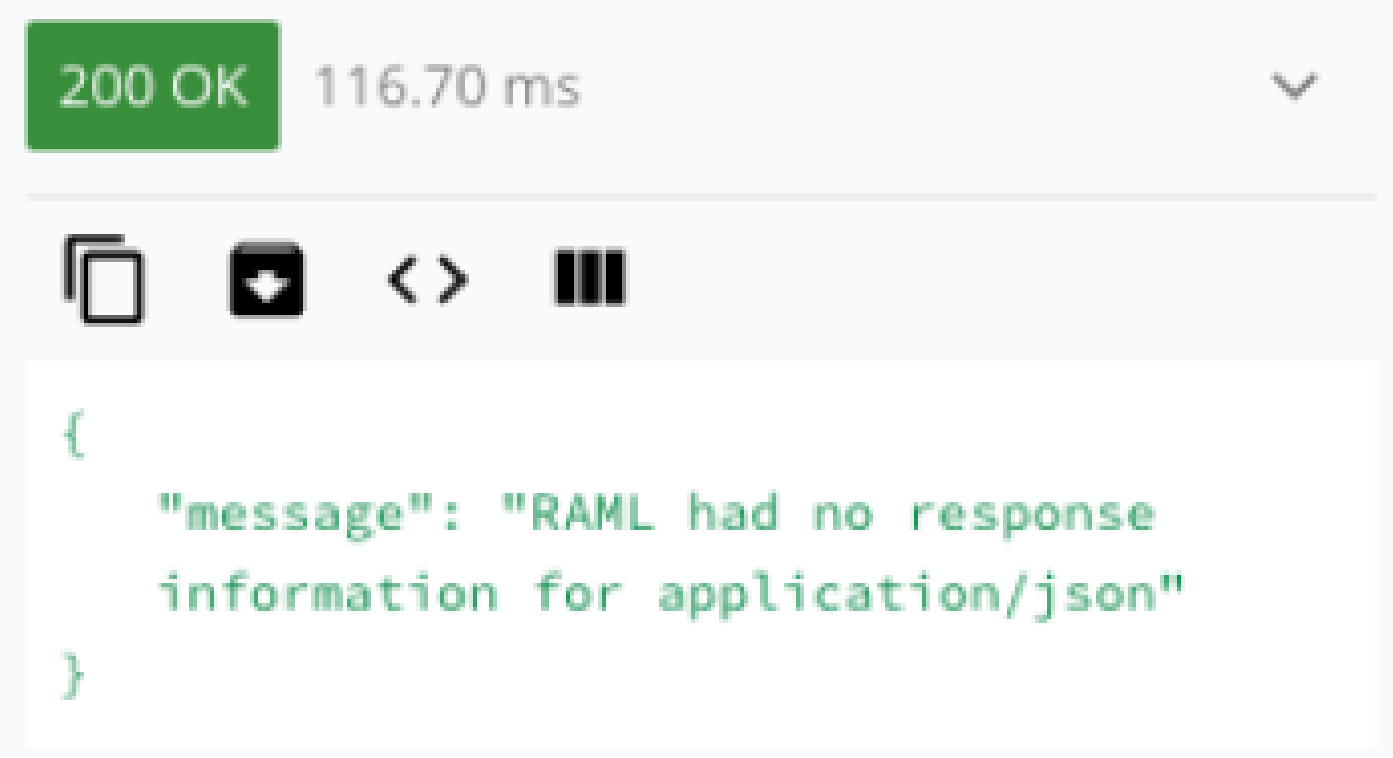


Test the /flights:post resource

14. Click the back arrow at the top of API console twice to return to the resource list.
15. Click the POST method.
16. Click Try it.
17. Select the Body tab; it should not have any content.



18. Click the Send button.
19. Look at the response; you should get the same generic 200 status code response.



Using RAML to define specifications for requests and responses

Defining method response details with RAML

- Responses must be a map of one or more HTTP status codes
- For each response, specify possible return data types along with descriptions and examples

```
/flights:
  get:
    responses:
      200:
        body:
          application/json:
            example:
                ID: 1
                code: GQ574
                price: 399
                departureDate: 2016/12/20
                origin: ORD
                destination: SFO
                emptySeats: 200
                plane:
                  type: Boeing 747
                  totalSeats: 400
```

Defining method request details with RAML

MuleSoft

- For a request, similarly specify the possible request data types along with data types, descriptions, and examples

```
6   /flights:
7     get: ⋯
23    post:
24      displayName: Add a flight
25      body:
26        application/json:
27          example:
28              code: GQ574
29              price: 399
30              departureDate: 2016/12/20
31              origin: ORD
32              destination: SFO
33              emptySeats: 200
34              plane:
35                type: Boeing 747
36                totalSeats: 400
```

Modularizing APIs

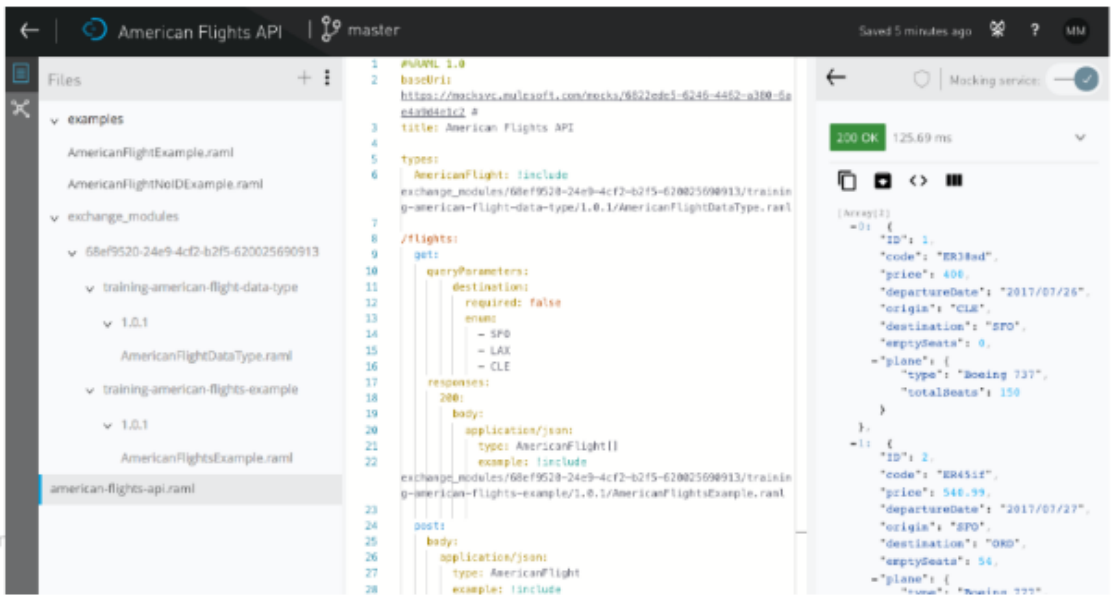
MuleSoft

- Instead of including all code in one RAML file, you can modularize it and compose it of reusable fragments
  - **Data types**, **examples**, traits, resource types, overlays, extensions, security schemes, documentation, annotations, and libraries

- Fragments can be stored
  - In different files and folders within a project
  - In a separate API fragment project in Design Center
  - In a separate RAML fragment in Exchange

Walkthrough 3-3: Add request and response details

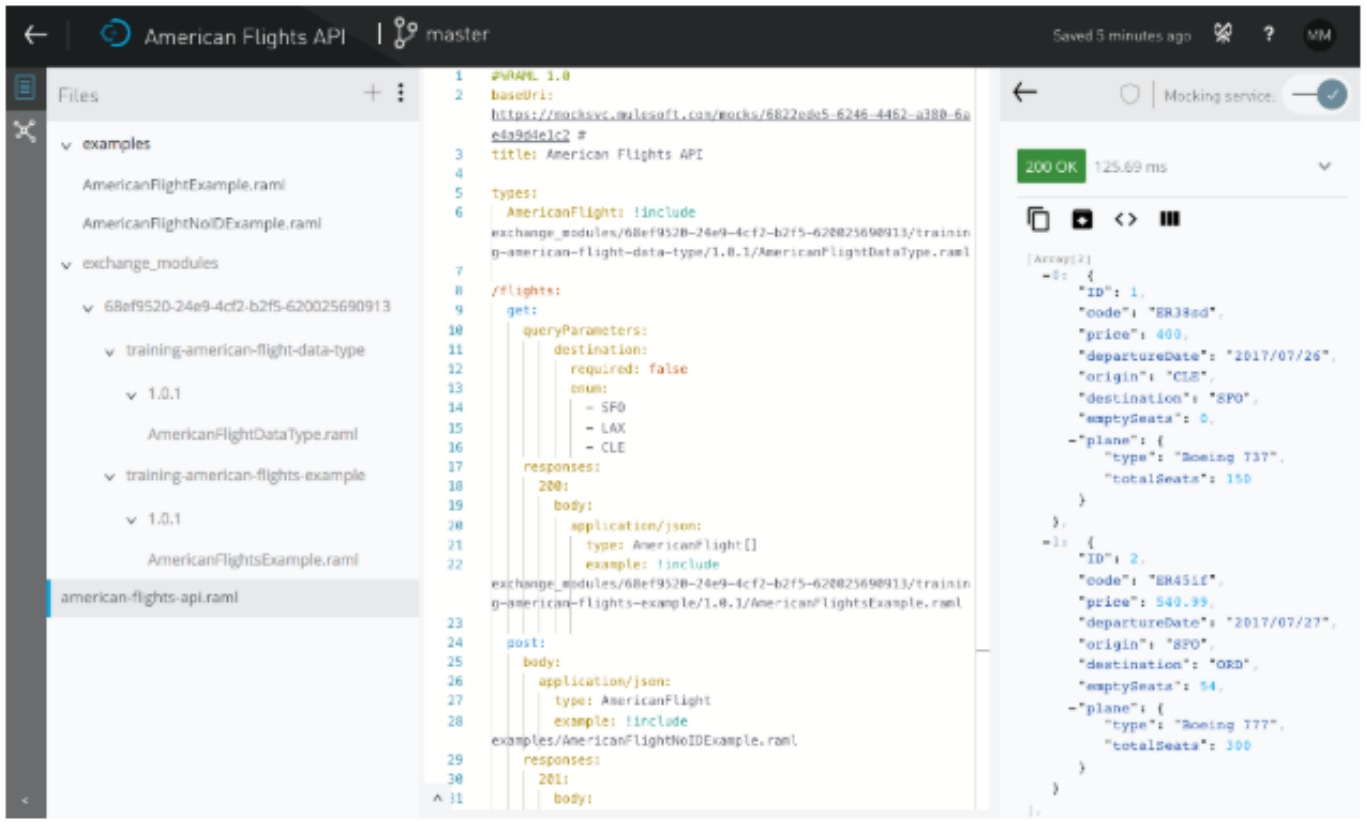
- Use API fragments from Exchange
- Add a data type and use it to define method requests and responses
- Add example JSON requests and responses
- Test an API and get example responses



Walkthrough 3-3: Add request and response details

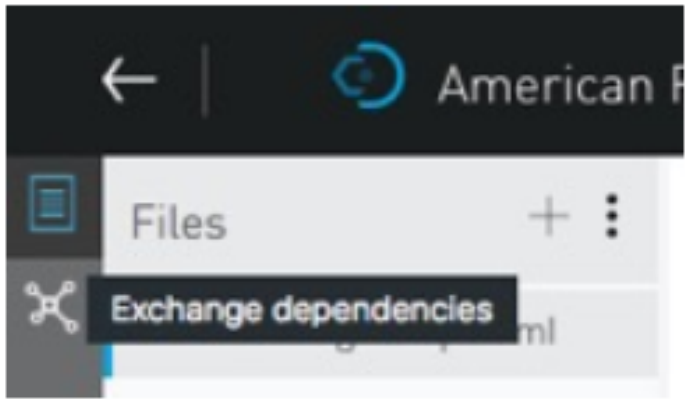
In this walkthrough, you add information about each of the methods to the API specification. You will:

- Use API fragments from Exchange.
- Add a data type and use it to define method requests and responses.
- Add example JSON requests and responses.
- Test an API and get example responses.

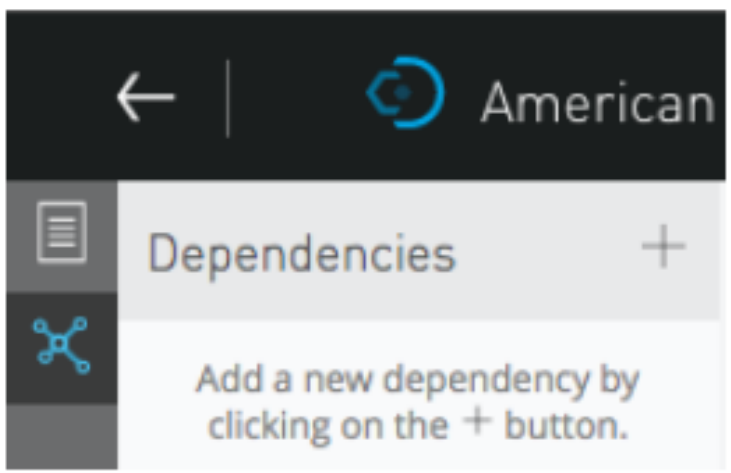


Add data type and example fragments from Exchange

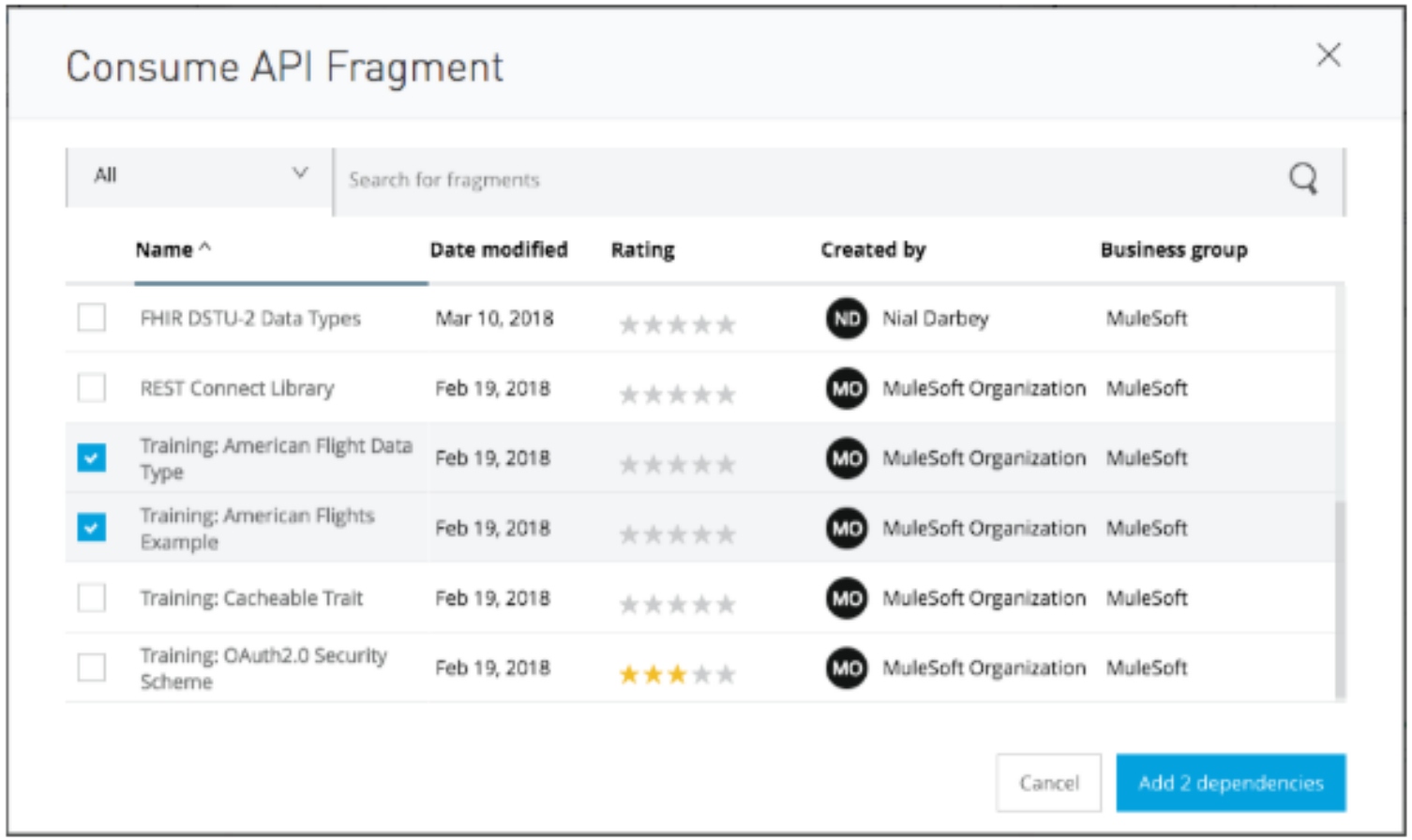
1. Return to API designer.
2. In the file browser, click the Exchange dependencies button.



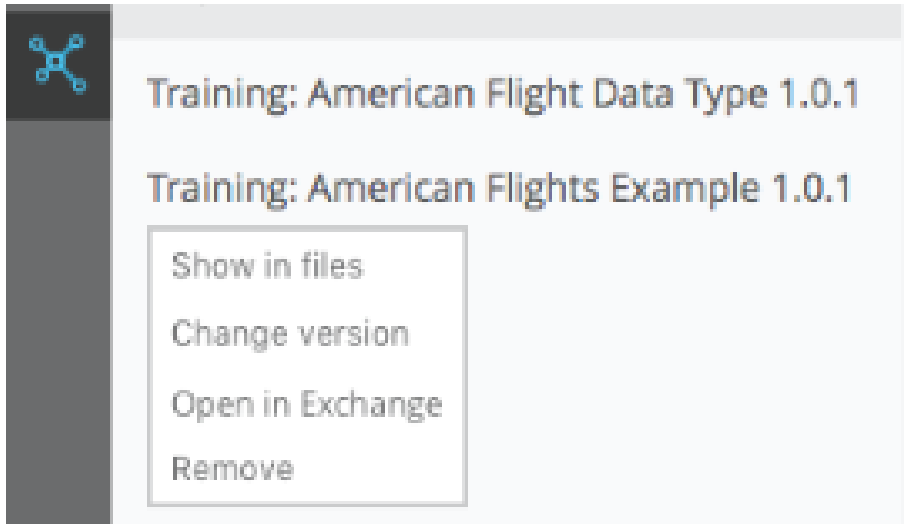
3. Click the Add button.



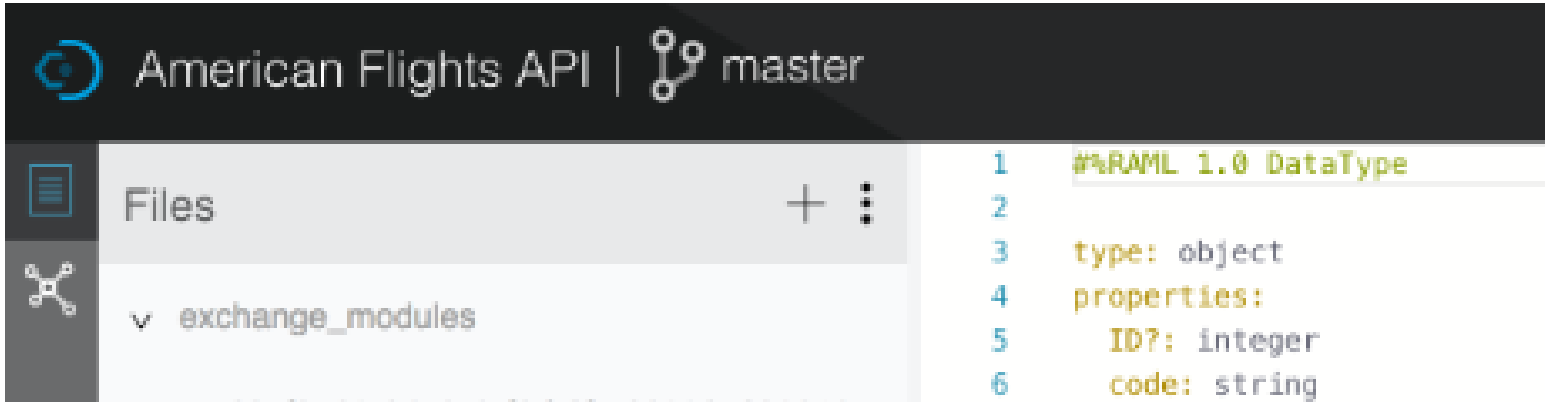
4. In the Consume API Fragment dialog box, select the Training: American Flight Data Type and the Training: American Flights Example.



5. Click the Add 2 dependencies button.



7. Click the Files button (above the Exchange dependencies button).
8. Expand the exchange_modules section until you see AmericanFlightDataType.raml.
9. Click AmericanFlightDataType.raml and review the code.



10. In the file browser, click the options menu button next to AmericanFlightsDataType.raml and select Copy path to clipboard.

Define an AmericanFlight data type for the API

11. Return to american-flights-api.raml.
12. Near the top of the code above the /flights resource, add a types element.
13. Indent under types and add a type called AmericanFlight.
14. Add the !include keyword and then paste the path you copied.

Note: You can also add the path by navigating through the exchange_modules folder in the shelf.

```
1   #%RAML 1.0
2   baseUri: https://mocksvc.mulesoft.com/mocks/3220238a-17c6-4e31-b5fa-413e8129e12b #
3   title: American Flights API
4
5   types:
6     AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/tr
7
8   /flights:
9     get:
```

Specify the /flights:get method to return an array of AmericanFlight objects

15. Go to a new line of code at the end of the /flights get method and indent to the same level as queryParameters.
16. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

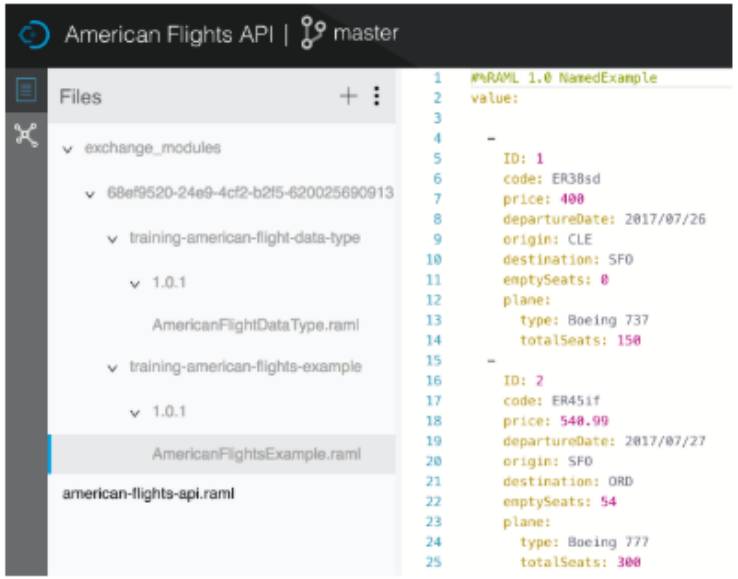
```
8   /flights:
9     get:
10      queryParameters:
11        destination:
12          required: false
13          enum:
14            - SFO
15            - LAX
16            - CLE
17      responses:
18        200:
19          body:
20            application/json:
21              type: AmericanFlight
```

17. Set the type to be an array of AmericanFlight objects: AmericanFlight[].

```
17      responses:
18        200:
19          body:
20            application/json:
21              type: AmericanFlight[]
```

Add an example response for the /flights:get method

18. In the file browser, locate AmericanFlightsExample.raml in exchange_modules and review the code.



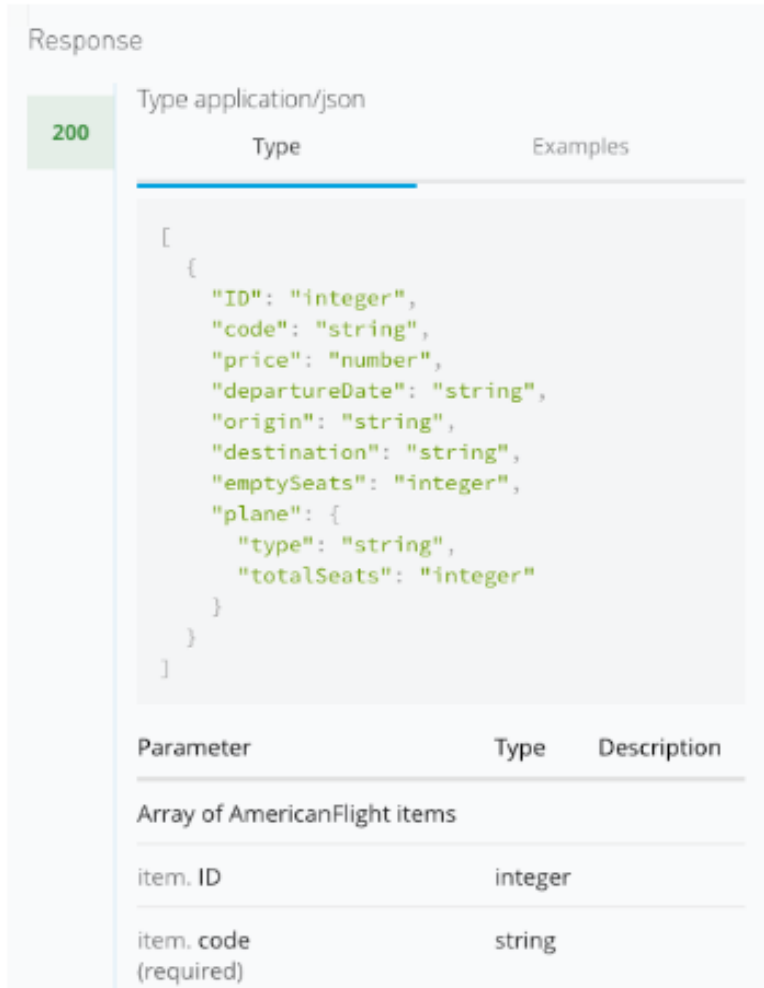
19. In the file browser, click the options menu next to AmericanFlightsExample.raml and select Copy path to clipboard.
20. Return to american-flights-api.raml.
21. In the editor, go to a new line after the type declaration in the /flights:get 200 response (at the same indentation as type).
22. In the shelf, click example.
23. Add the !include keyword and then paste the path you copied.

    Note: You can also add the path by navigating through the exchange_modules folder in the shelf.

```
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight[]
            example: !include exchange_modules/86f74f12-decb-452c
  post:
```

Review and test the /flights:get resource in API console

24. In API console, click the /flights:get method.
25. Look at the type information in the response information.



26. Click the Examples tab; you should see the example array of AmericanFlight objects.

27. Click the Try it button and click Send; you should now see the example response with two flights.

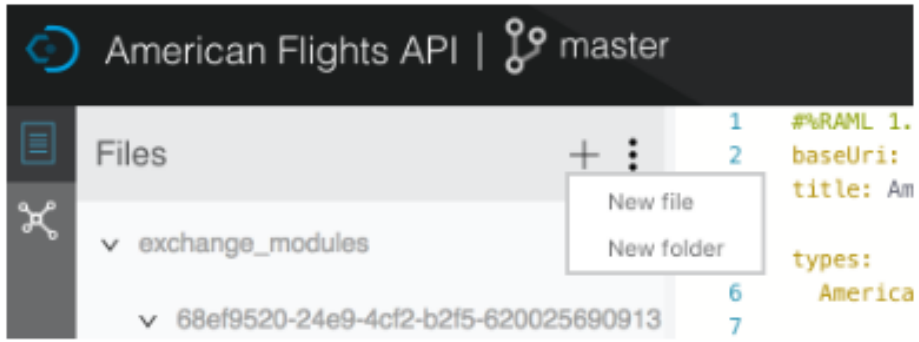
Specify the /{ID}:get method to return an AmericanFlight object

28. In the editor, indent under the /{ID} resource get method.
29. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

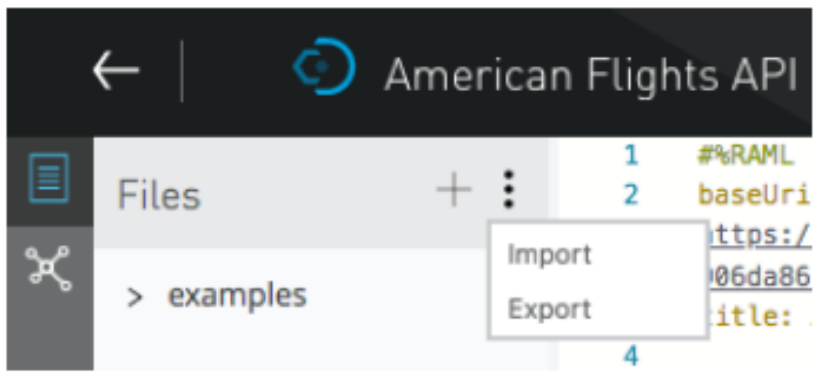
```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
```

Define an example response for the /flights:get method in a new folder

30. In the file browser, click the add button and select New folder.

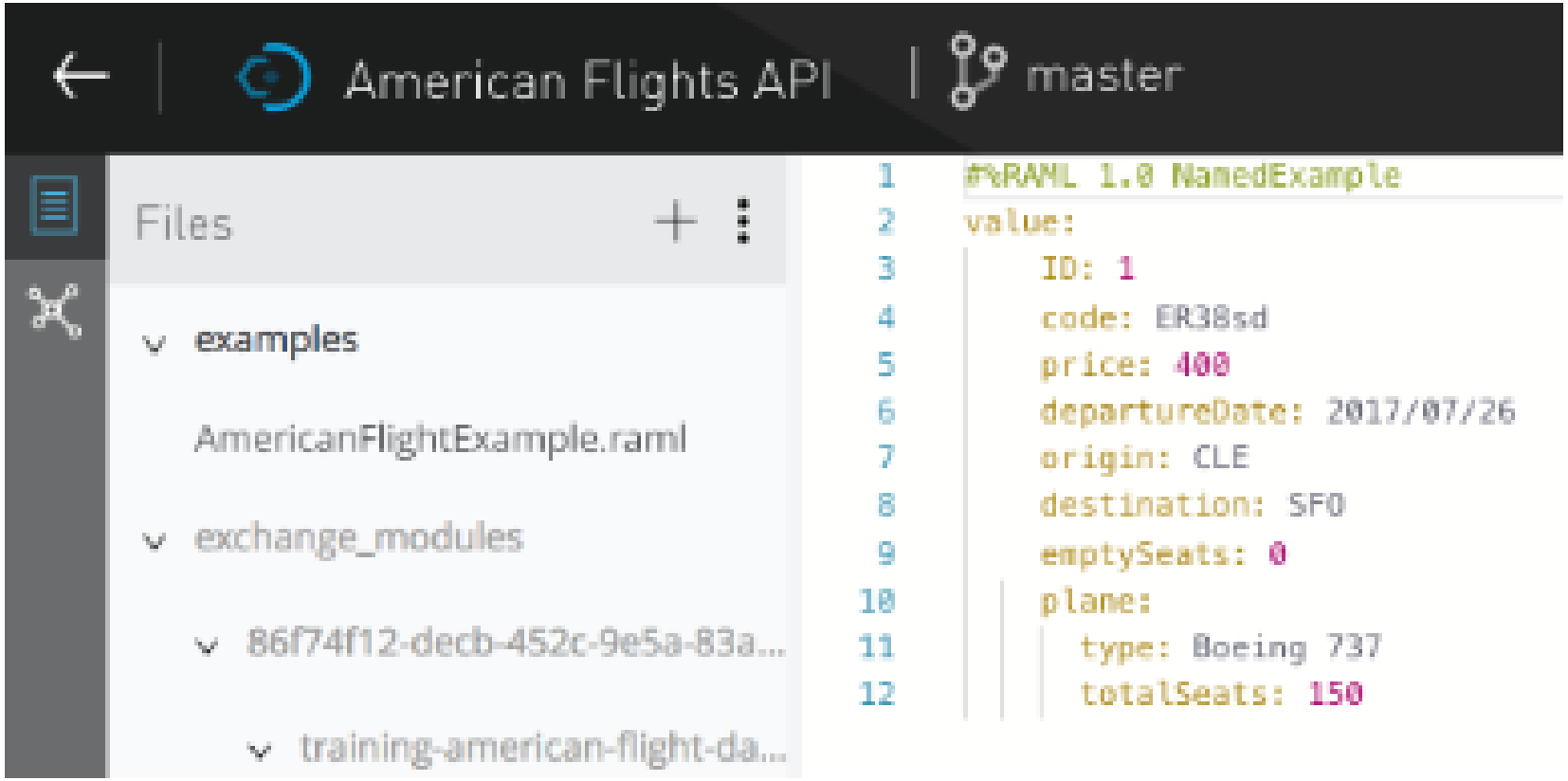


31. In the Add new folder dialog box, set the name to examples and click Create.
32. In the file browser, click the menu button and select Import.



33. In the Import dialog box, leave File or ZIP selected and click the Choose file button.
34. Navigate to your student files and select the AmericanFlightExample.raml file in the examples folder.
35. In the Import dialog box, click Import; you should see the new file in API designer.
36. Review the code.

37. In the file browser, drag AmericanFlightExample.raml into the examples folder.



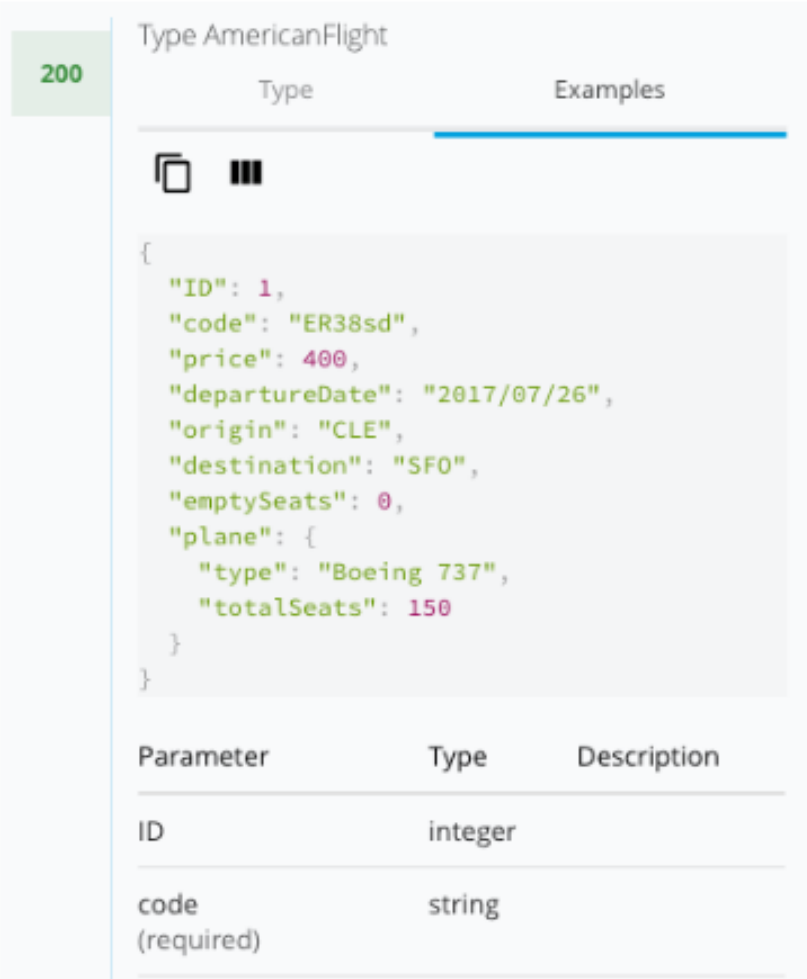
Add an example response for the /{ID}:get method

38. Return to american-flights-api.raml.
39. In the editor, go to a new line after the type declaration in {ID}:/get (at the same indentation).
40. In the shelf, click example.
41. Add an include statement and include the example in examples/AmericanFlightExample.raml.

```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
            example: !include examples/AmericanFlightExample.raml
```

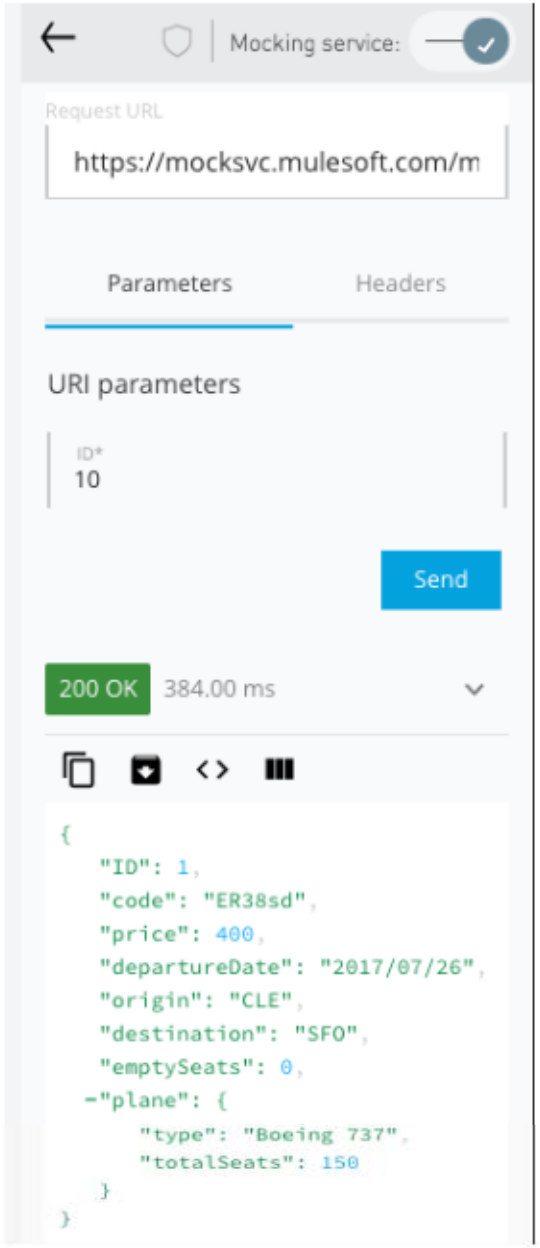
Review and test the /{ID}:get resource in API console

42. In API console, return to the /{ID}:get method; you should now see the response will be of type AmericanFlight.



43. Select the Examples tab; you should see the example AmericanFlightExample data.

44. Click the Try it button, enter an ID, and click Send; you should now see the example flight data returned.



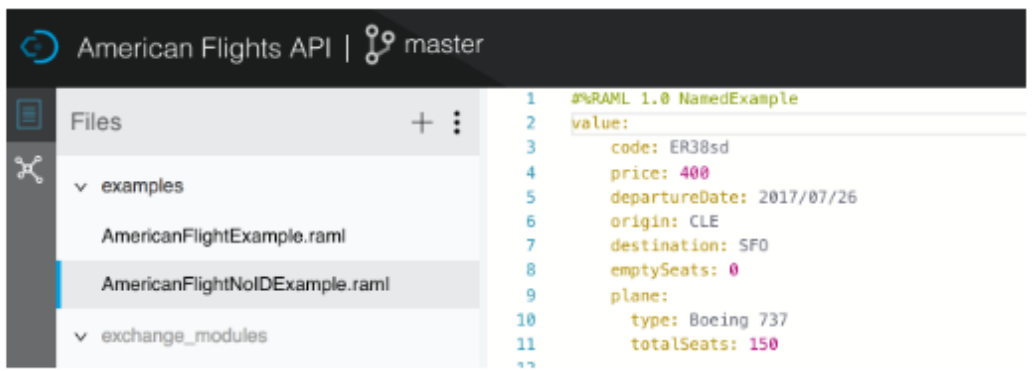
Specify the /flights:post method request to require an AmericanFlight object

45. In the editor, indent under the /flights post method.
46. In the shelf, click body > application/json > type > AmericanFlight.

```
24      post:
25        body:
26          application/json:
27            type: AmericanFlight
```

Define an example request body for the /flights:post method

47. Return to AmericanFlightExample.raml and copy all the code.
48. In the file browser, click the add button next to the examples folder and select New file.
49. In the Add new file dialog box, set the following values:
    - Version: RAML 1.0
    - Type: Example
    - File name: AmericanFlightNoIDExample.raml
50. Click Create.
51. Delete any code in the new file and then paste the code you copied.
52. Delete the line of code containing the ID.



53. Return to american-flights-api.raml.
54. In the post method, go to a new line under type and add an example element.
55. Use an include statement to set the example to examples/AmericanFlightNoIDExample.raml.

```
post:
  body:
    application/json:
      type: AmericanFlight
      example: !include examples/AmericanFlightNoIDExample.raml
```

Specify an example response for the /flights:post method

56. Go to a new line of code at the end of the /flights:post method and indent to the same level as body.
57. Add a 201 response of type application/json.

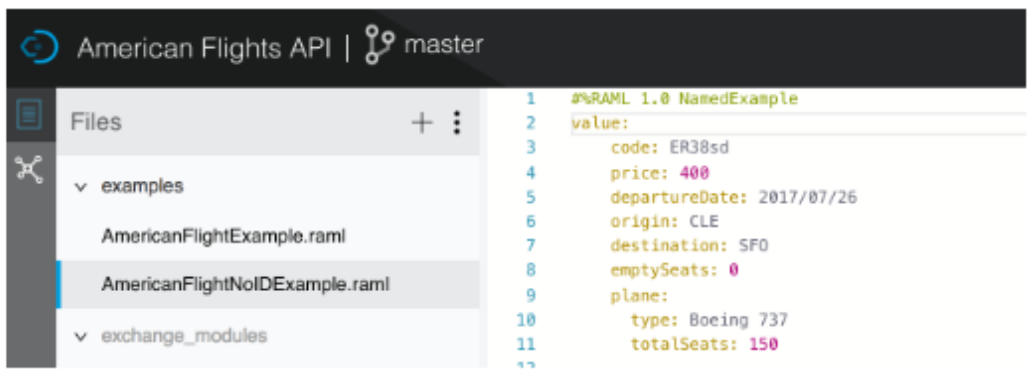
```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
29    responses:
30      201:
31        body:
32          application/json:
33
```

58. In the shelf, click example.
59. Indent under example and add a message property equal to the string: Flight added (but not really).

```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
29    responses:
30      201:
31        body:
32          application/json:
33            example:
34              message: Flight added (but not really)
```

Define an example request body for the /flights:post method

47. Return to AmericanFlightExample.raml and copy all the code.
48. In the file browser, click the add button next to the examples folder and select New file.
49. In the Add new file dialog box, set the following values:
    - Version: RAML 1.0
    - Type: Example
    - File name: AmericanFlightNoIDExample.raml
50. Click Create.
51. Delete any code in the new file and then paste the code you copied.
52. Delete the line of code containing the ID.



53. Return to american-flights-api.raml.
54. In the post method, go to a new line under type and add an example element.
55. Use an include statement to set the example to examples/AmericanFlightNoIDExample.raml.

```
post:
  body:
    application/json:
      type: AmericanFlight
      example: !include examples/AmericanFlightNoIDExample.raml
```

Specify an example response for the /flights:post method

56. Go to a new line of code at the end of the /flights:post method and indent to the same level as body.
57. Add a 201 response of type application/json.

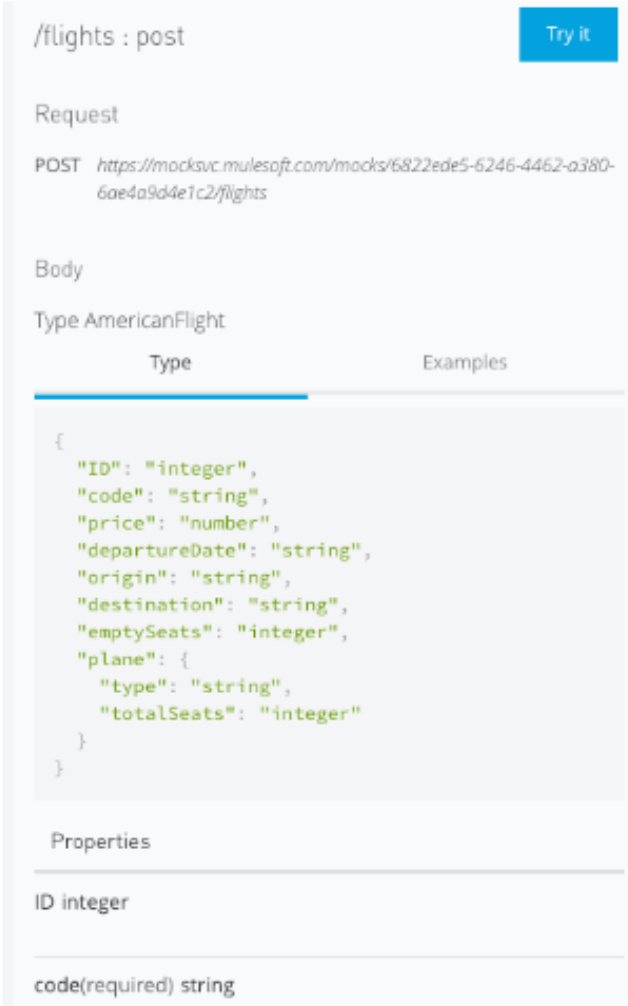
```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
29    responses:
30      201:
31        body:
32          application/json:
33
```

58. In the shelf, click example.
59. Indent under example and add a message property equal to the string: Flight added (but not really).

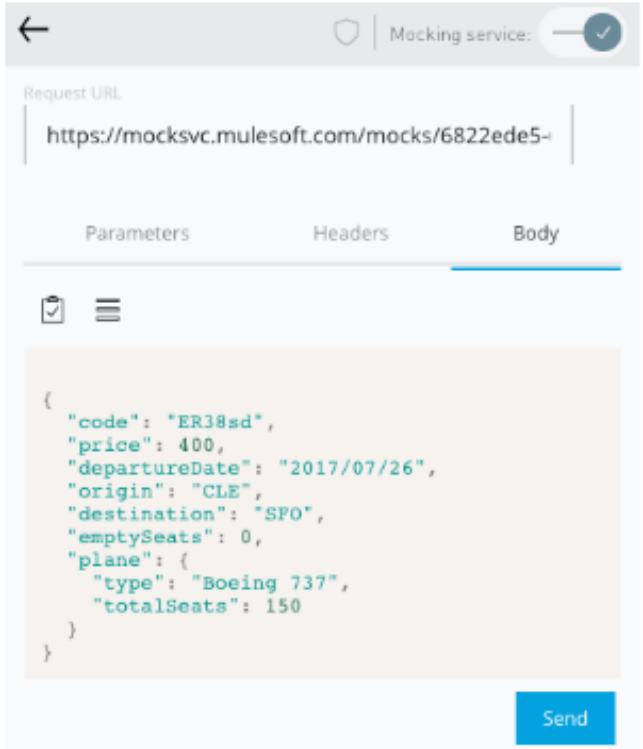
```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
29    responses:
30      201:
31        body:
32          application/json:
33            example:
34              message: Flight added (but not really)
```

Review and test the /flights:post resource in API console

60. In API console, return to the /flights:post method.
61. Look at the request information; you should now see information about the body - that it is type AmericanFlight and it has an example.

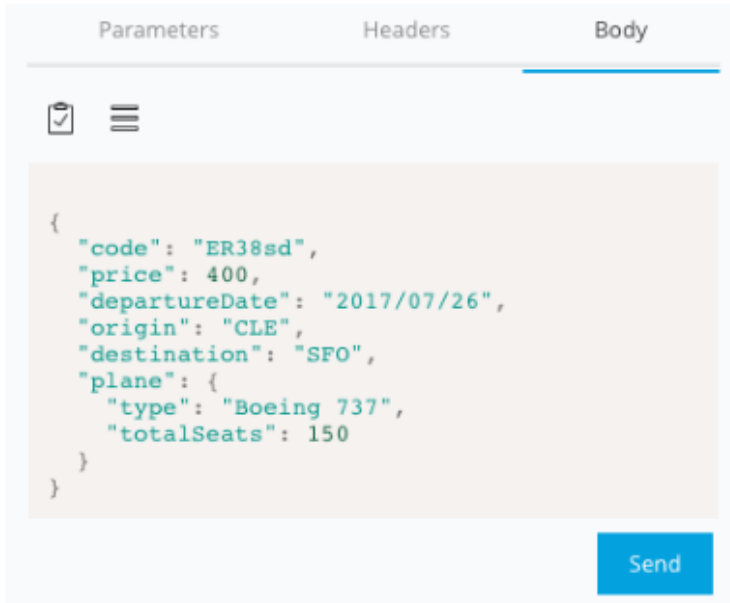


62. Click the Try it button and select the Body tab again; you should now see the example request body.

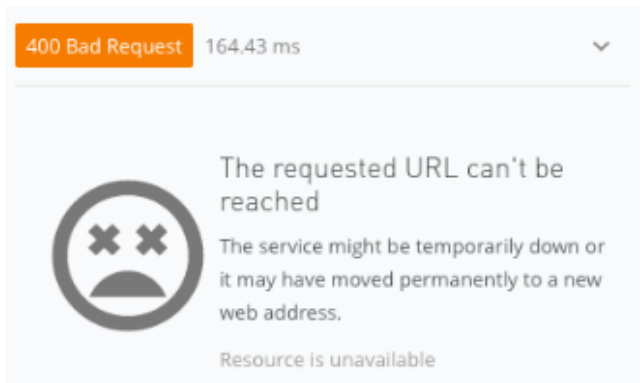


63. Click the Send button; you should now get a 201 response with the example message.

64. In the body, remove the emptySeats properties.



65. Click Send again; you should get a 400 Bad Request response.

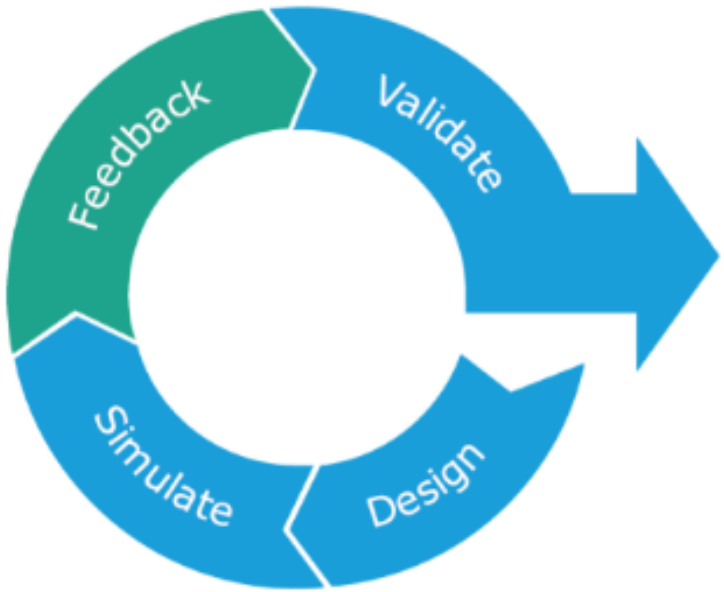


66. Add the emptySeats property back to the body.

Engaging with users

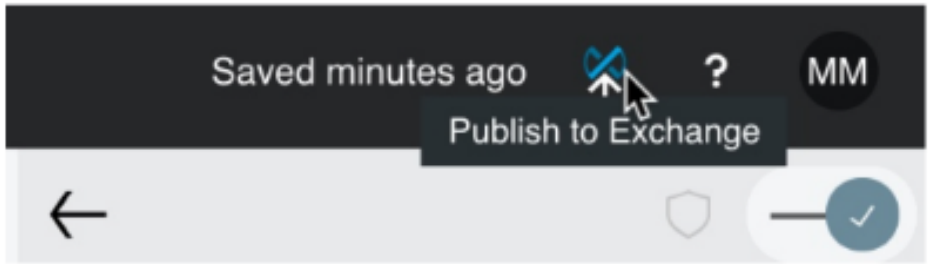
Engaging users during the API design phase

- To build a successful API, you should define it iteratively
  - Get feedback from developers on usability and functionality along the way
- To do this, you need to provide ways for developers to discover and play with the API
- Anypoint Platform makes this easy with **API portals in Exchange**
  - In **private Exchange** for internal developers
  - In a **public portal** for external developers



Publishing RAML APIs to Anypoint Exchange

- You publish RAML API Specifications and RAML fragments to the Exchange **from API designer**
  - Not from Exchange itself

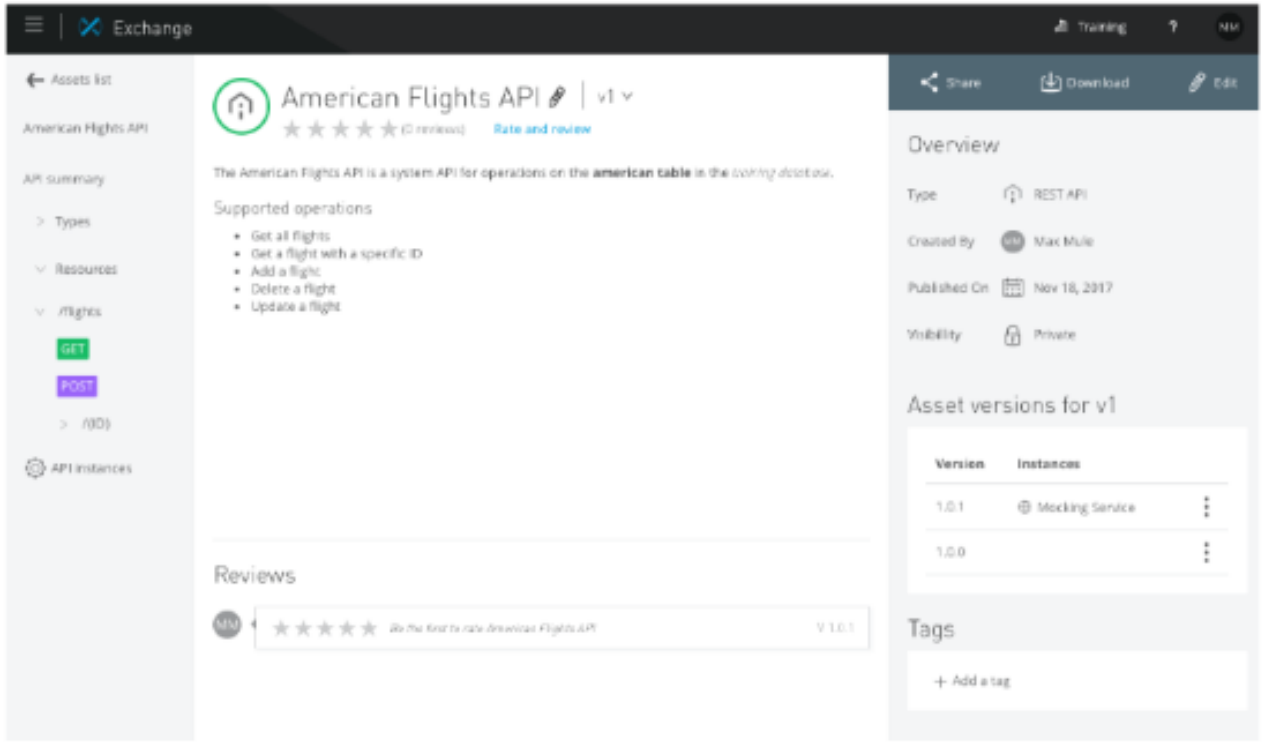


- **API portals** are automatically created for REST APIs added to Exchange
  - An **API console** for consuming and testing APIs
  - An **automatically generated API endpoint** that uses a **mocking service** to allow the API to be tested without having to implement it

- API portals can be shared with both internal and external users

Walkthrough 3-4: Add an API to Anypoint Exchange

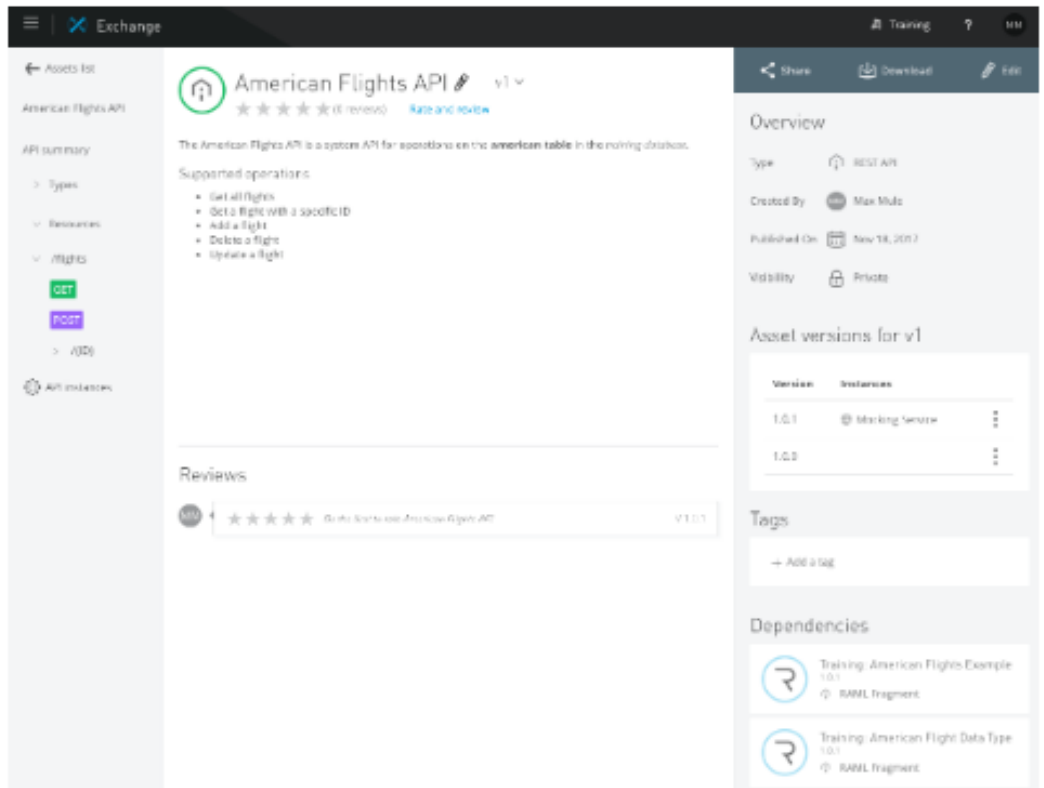
- Publish an API to Exchange from API designer
- Review an auto-generated API portal in Exchange and test the API
- Add information about an API to its API portal
- Create and publish a new API version to Exchange



Walkthrough 3-4: Add an API to Anypoint Exchange

In this walkthrough, you make an API discoverable by adding it to Anypoint Exchange. You will:

- Publish an API to Exchange from API designer.
- Review an auto-generated API portal in Exchange and test the API.
- Add information about an API to its API portal.
- Create and publish a new API version to Exchange.



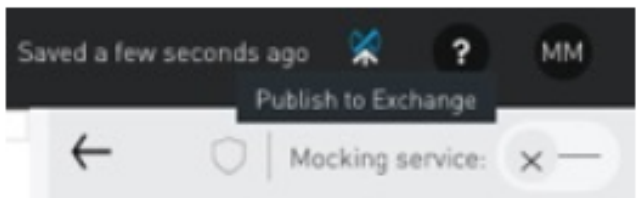
Remove the baseUri by turning off the mocking service

1. Return to API designer.
2. Click the slider to turn off the mocking service; the RAML code should no longer have a baseUri.

```
1    #%RAML 1.0
2    title: American Flights API
3
```

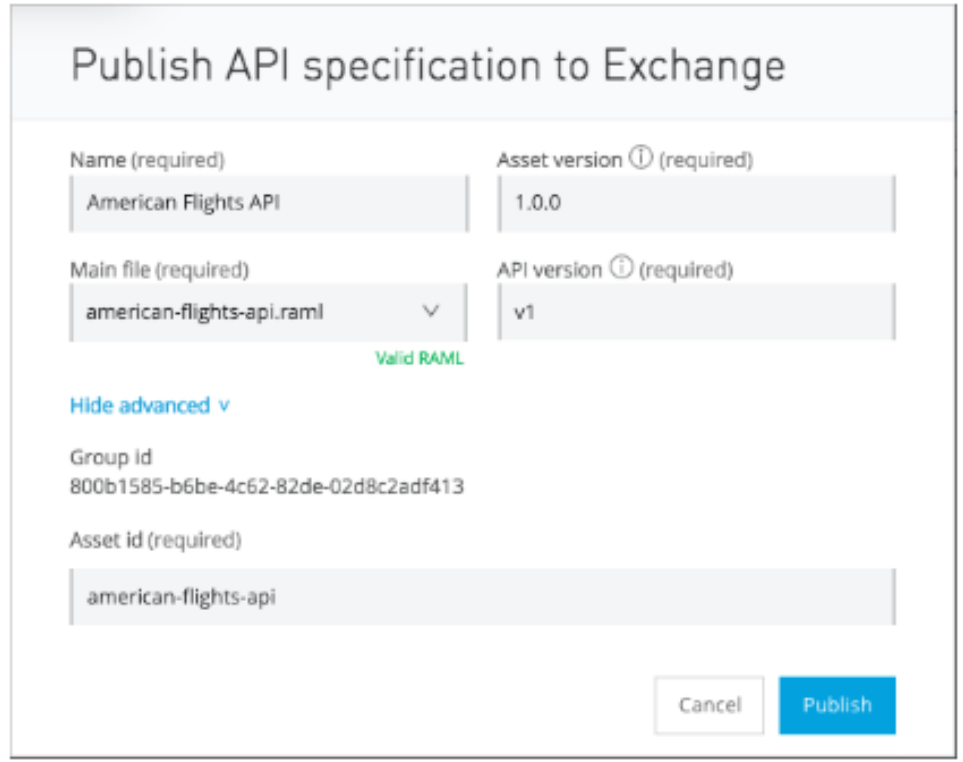
Publish the API to Anypoint Exchange from API designer

3. Click the Publish to Exchange button.



4. In the Publish API specification to Exchange dialog box, leave the default values:
    - Name: American Flights API
    - Main file: american-flights-api.raml
    - Asset version: 1.0.0
    - API version: v1
5. Click the Show advanced link.
6. Look at the ID values.

7. Click Publish.

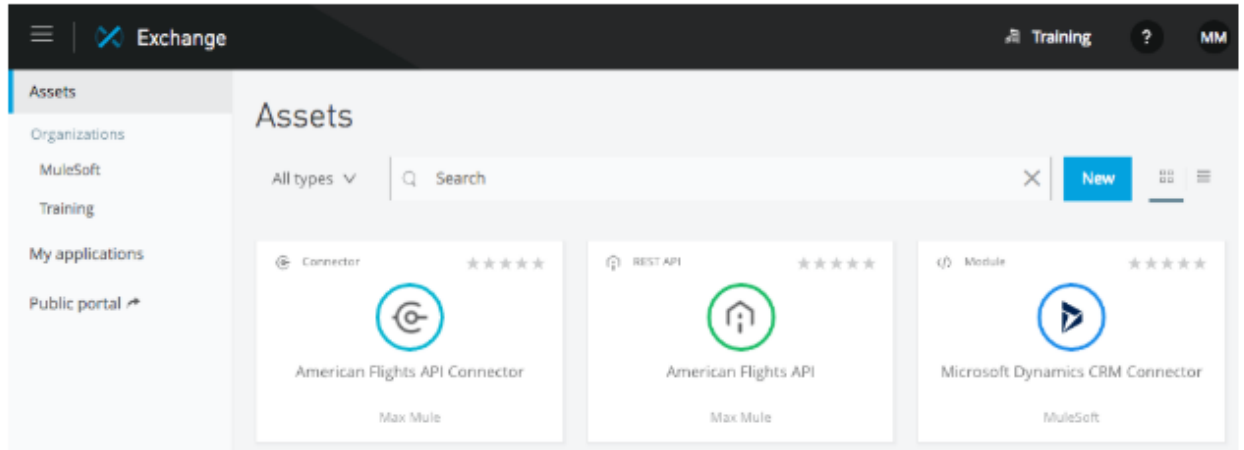


8. Click the Publish button.
9. In the Publish API specification to Exchange dialog box, click Done.
10. Look at the RAML file; you should see a version has been added.

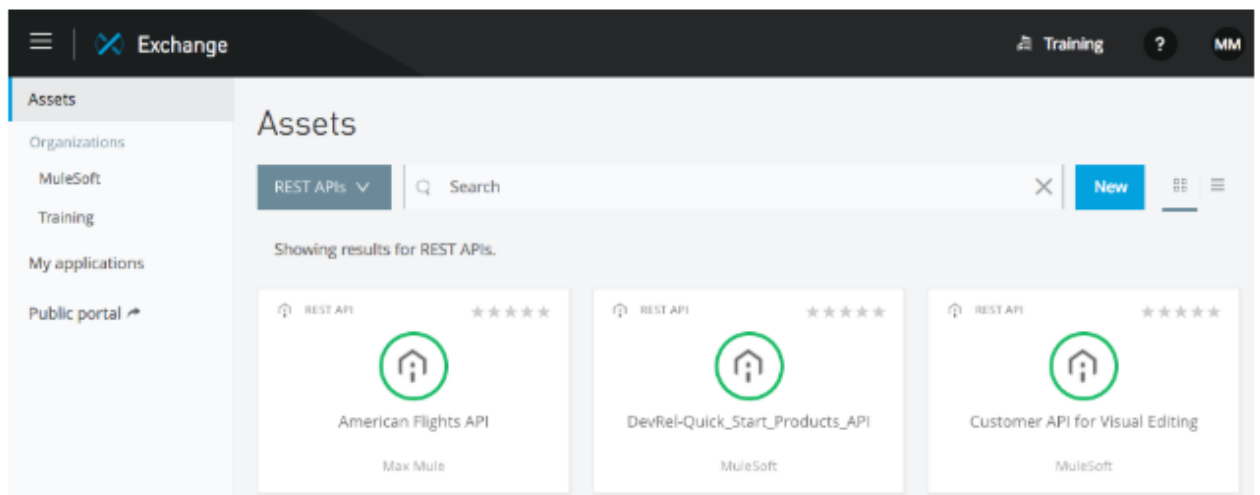
```
1   #%RAML 1.0
2   version: v1
3   title: American Flights API
```

Locate your API in Anypoint Exchange

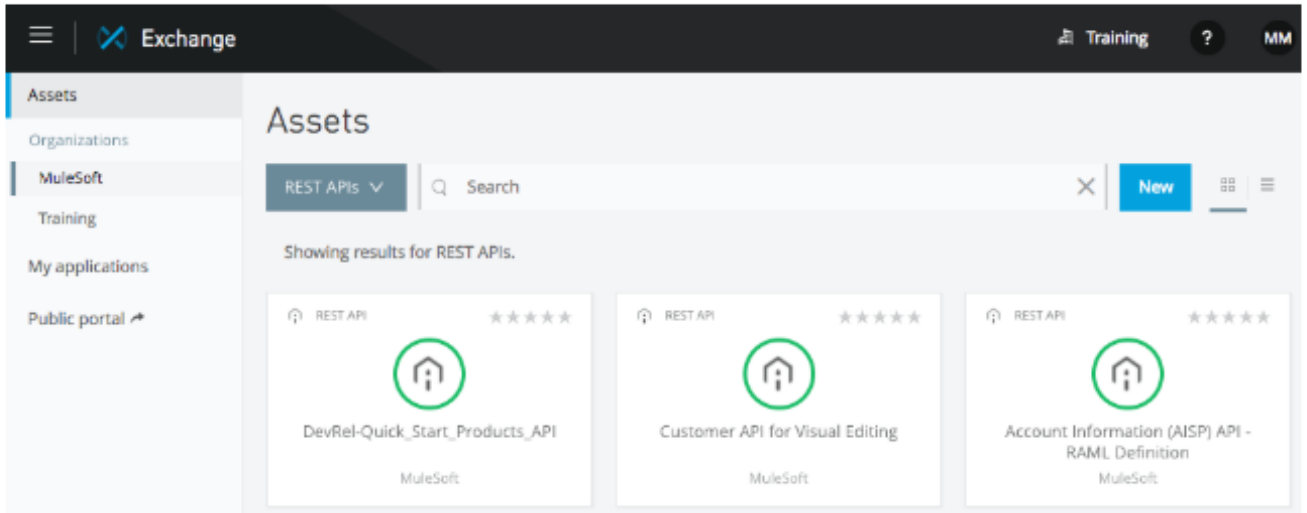
11. Return to Design Center.
12. In the main menu, select Exchange; you should see your American Flights API.



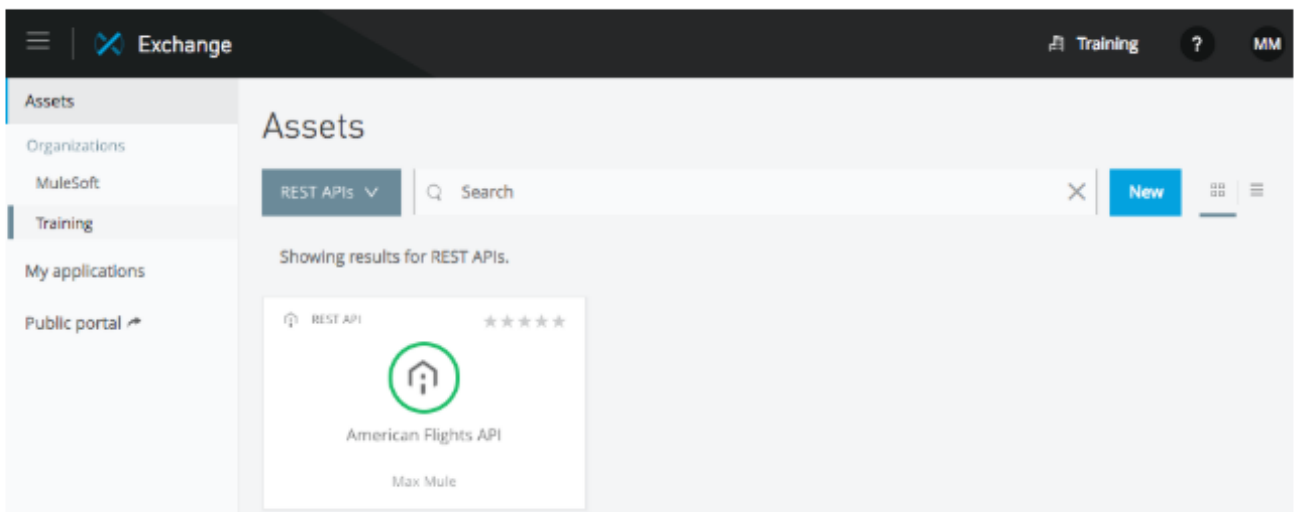
13. In the types drop-down menu, select REST APIs; you should still see your API.



14. In the left-side navigation, select MuleSoft; you should not see your American Flights API in the public Exchange (just the Training: American Flights API).

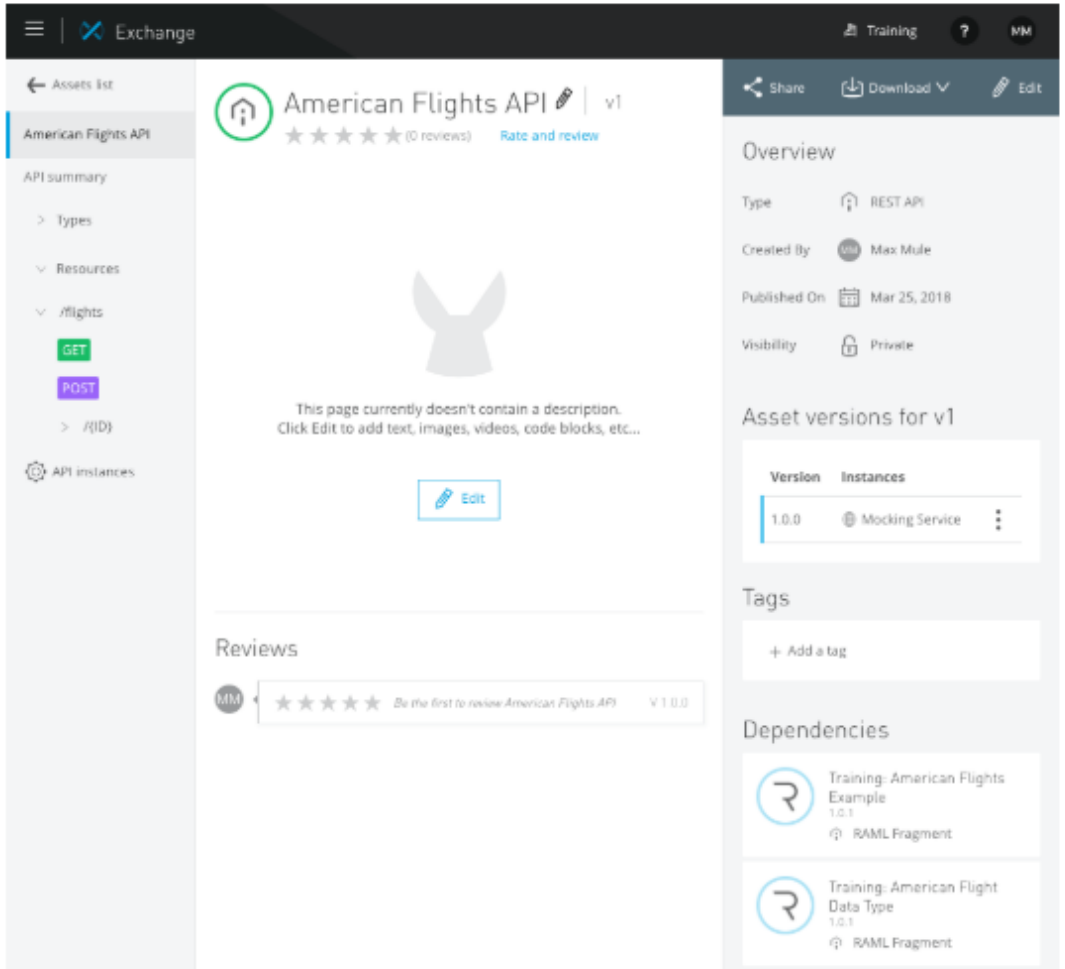


15. In the left-side navigation, select the name of your organization (Training in the screenshots); you should see your American Flights API in your private Exchange.



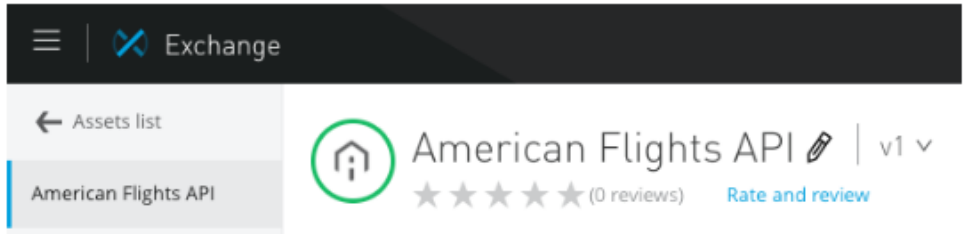
Review the auto-generated API portal

16. Click the American Flights API.
17. Review the page; you should see that as the creator of this API, you can edit, review, share, download, and add tags to this version.

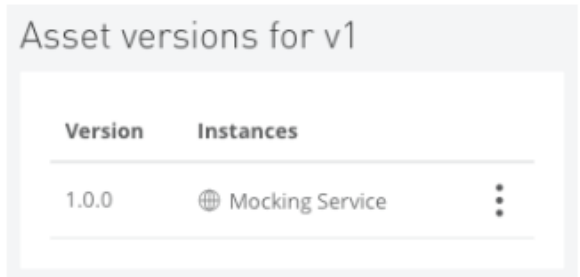


18. Locate the API dependencies in the lower-right corner.

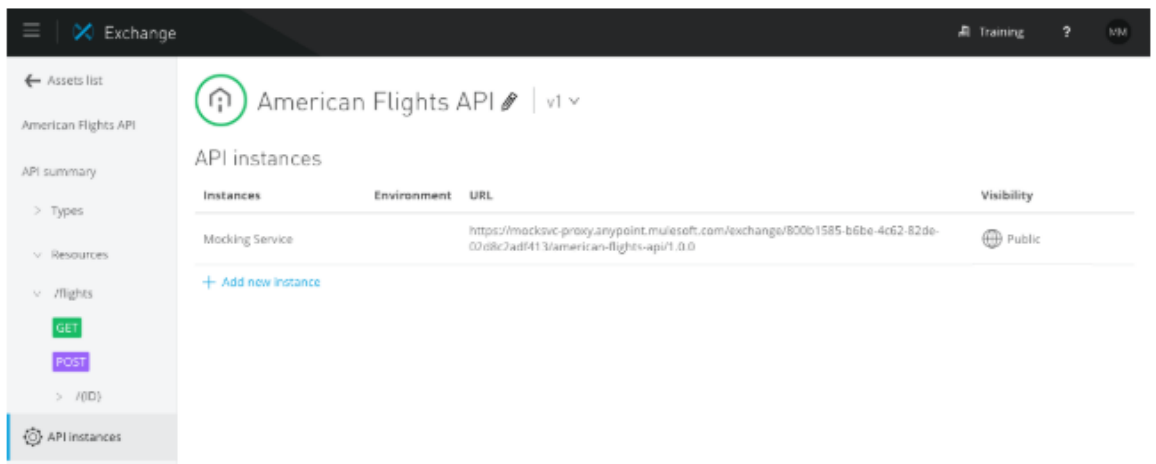
19. Locate the API version (v1) next to the name of the API at the top of the page.



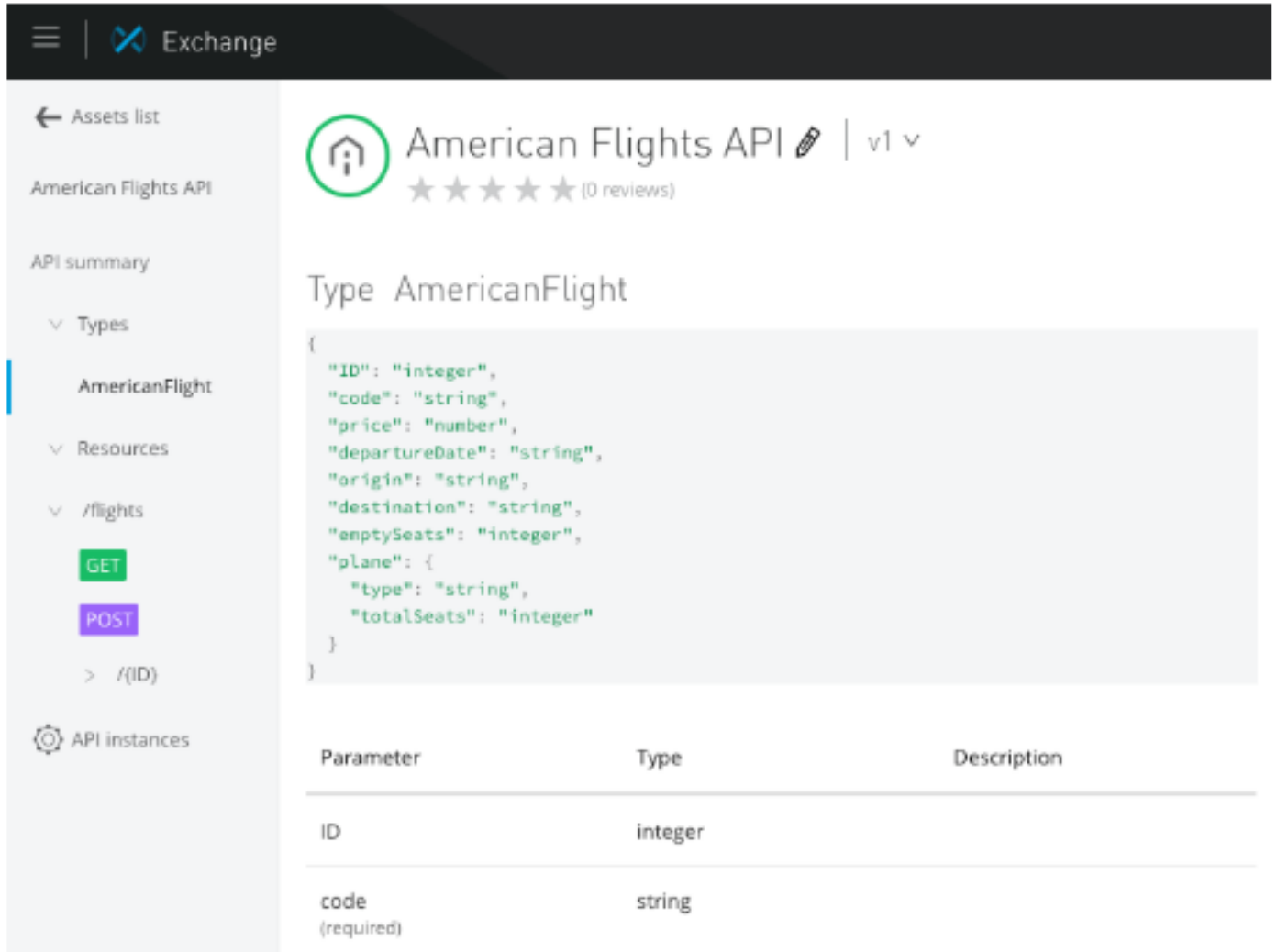
20. Locate the asset versions for v1; you should see one version (1.0.0) of this API specification (v1) has been published and there is one API instance for it and that uses the mocking service.



21. In the left-side navigation, select API instances; you should see information for the API instance generated from the API specification using the mocking service.

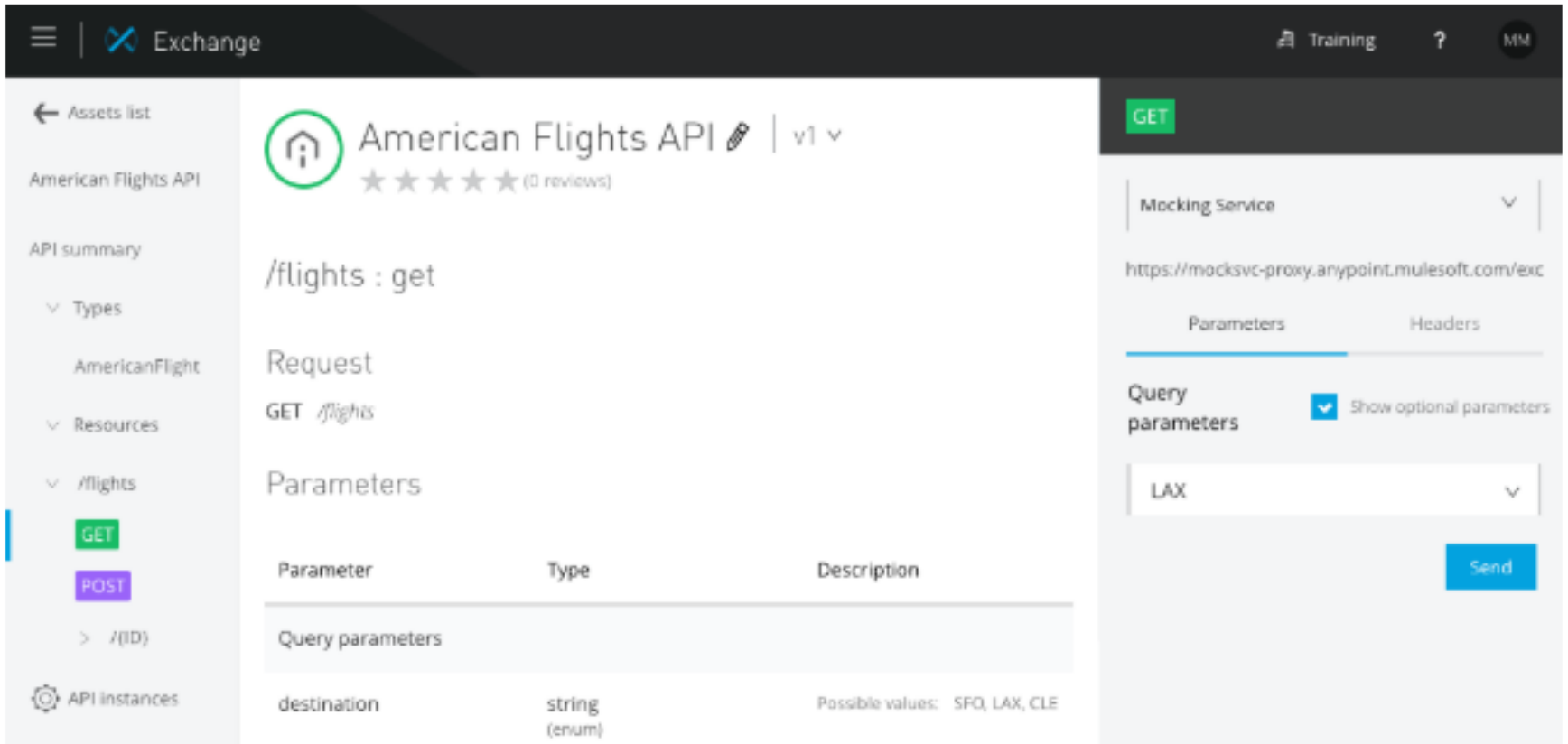


22. In the left-side navigation, expand Types.
23. Select AmericanFlight and review the information.

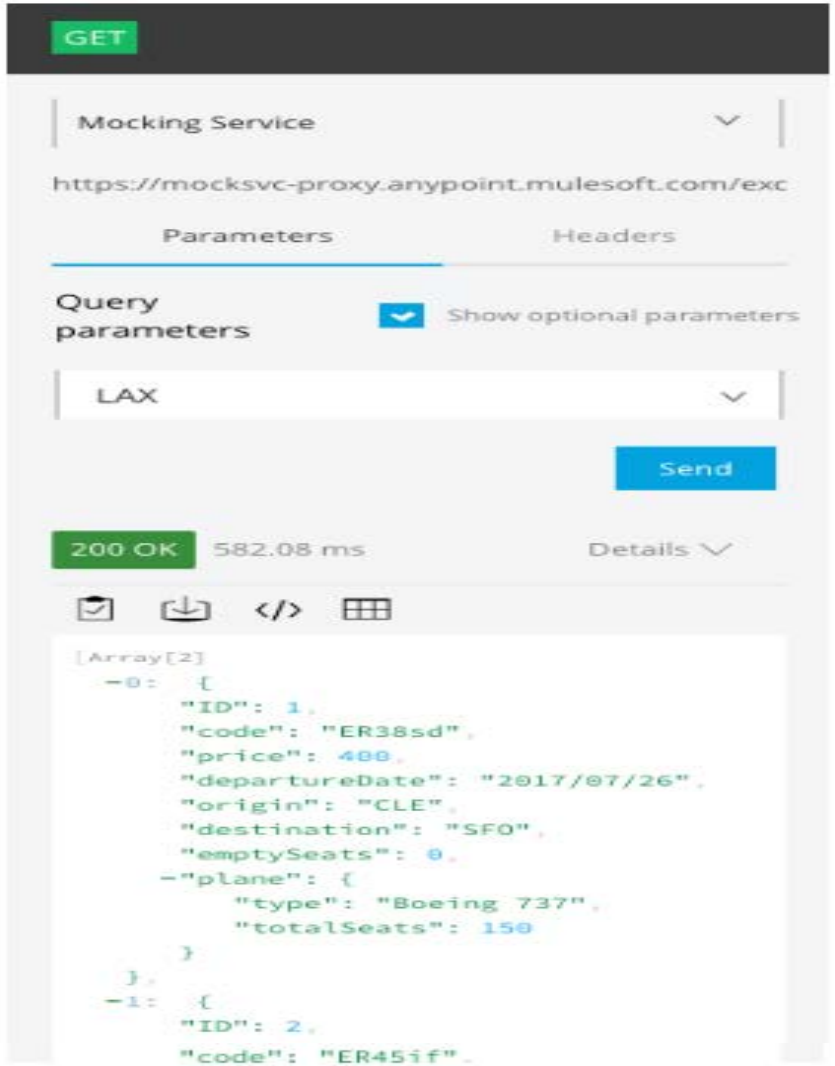


Test the API in its API portal in Exchange

24. In the left-side navigation, select the /flights GET method; you should now see the API console on the right side of the page.
25. In the API console, select to show optional query parameters.
26. Click the destination drop-down and select a value.

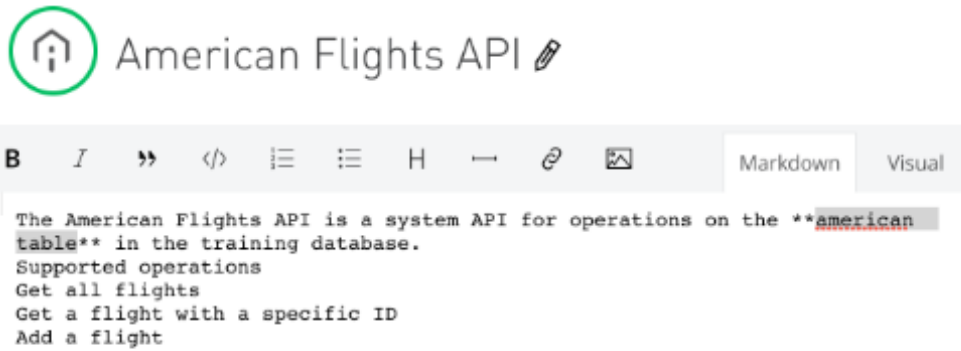


27. Click Send; you should get a 200 response and the example data displayed – just as you did in API console in API designer.

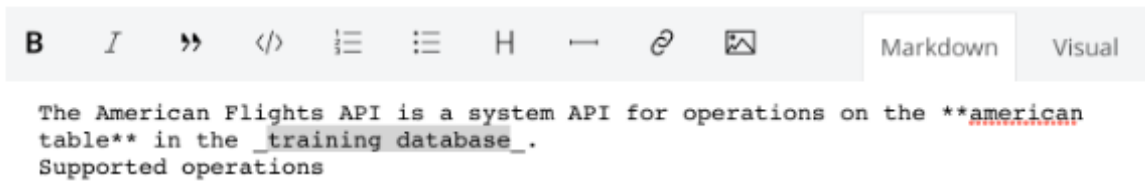


Add information about the API

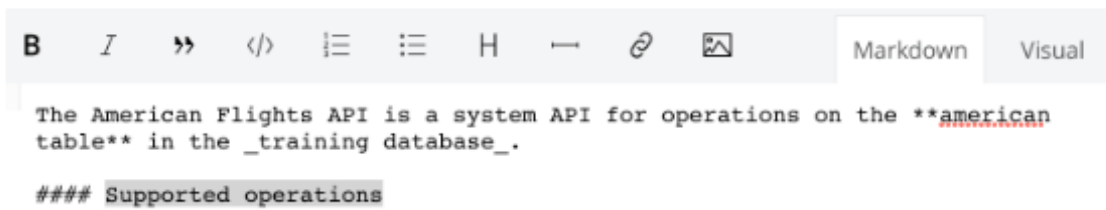
28. In the left-side navigation, select the name of the API: American Flights API.
29. Click one of the Edit buttons for the API.
30. Return to the course snippets.txt file and copy the text for the American Flights API description text.
31. Return to the editor in Anypoint Exchange and paste the content.
32. Select the words american table and click the strong button (the B).



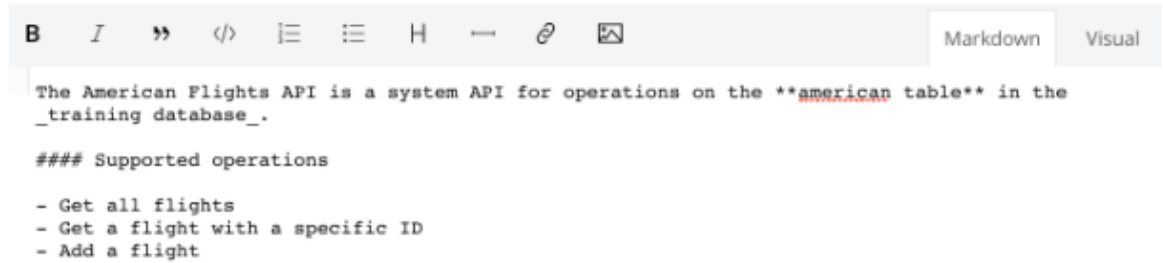
33. Select the words training database and click the emphasis button (the I).



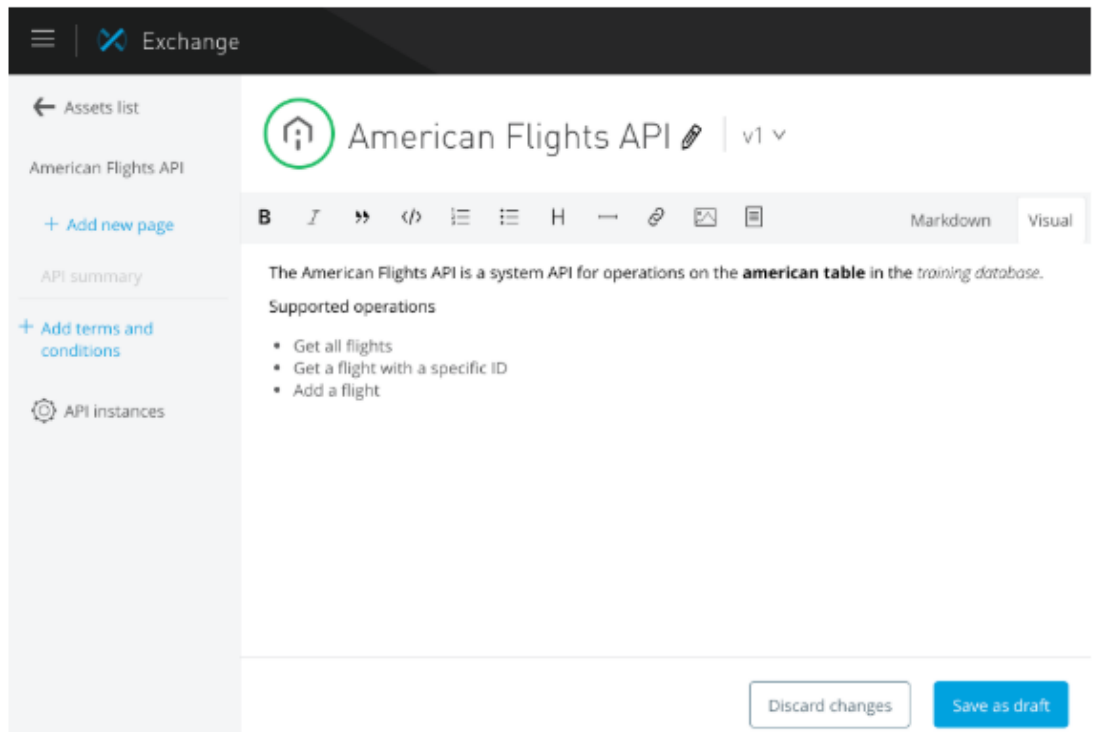
34. Select the words Supported operations and click the heading button four times (the H).



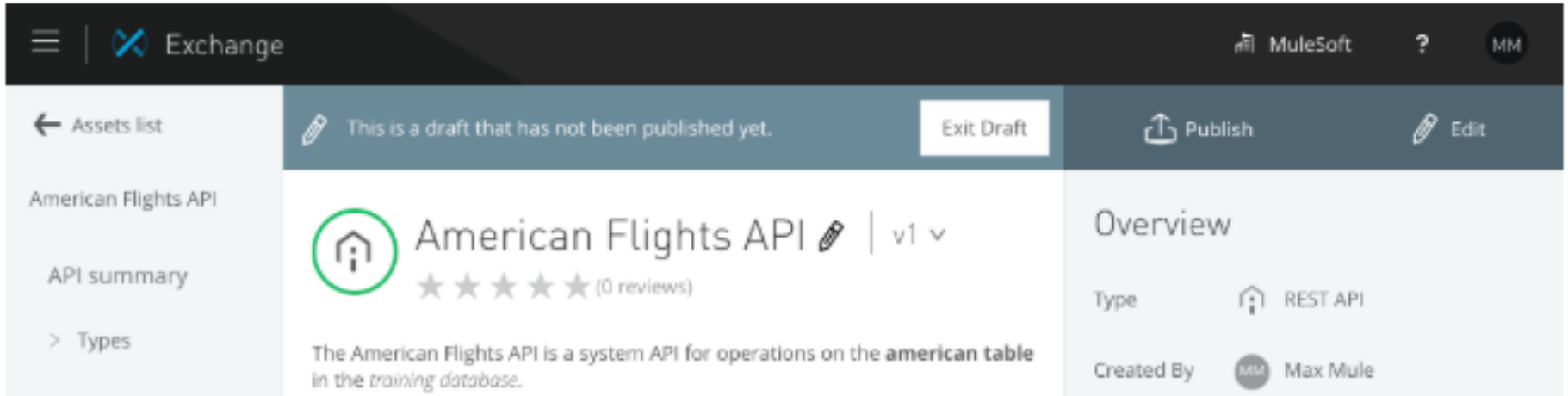
35. Select Get all flights and click the bulleted list button.
36. Repeat for the other operations.



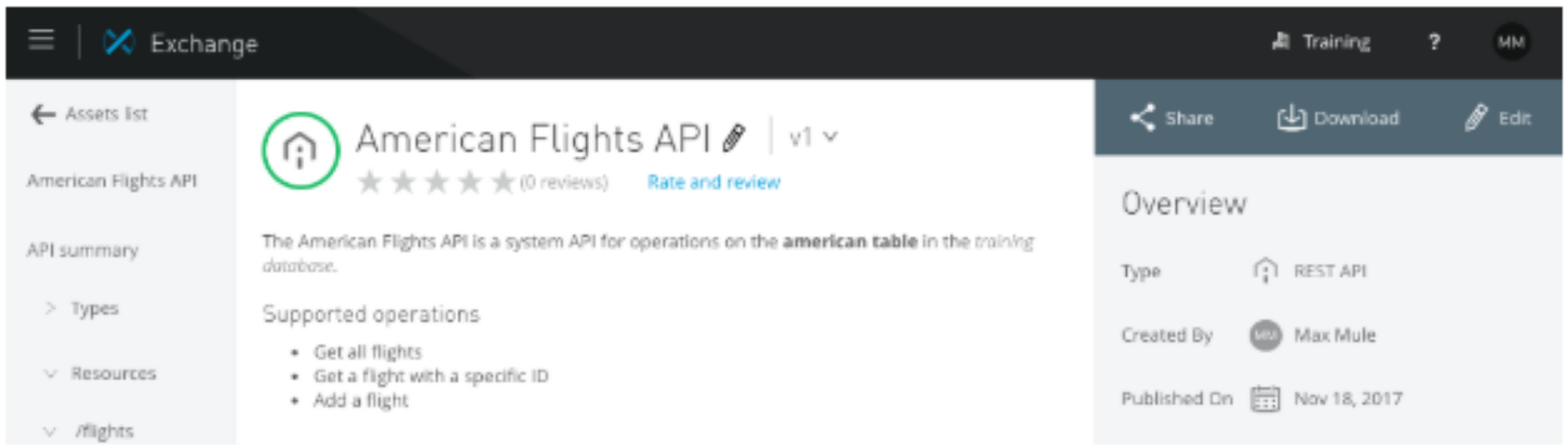
37. Select the Visual tab in the editor toolbar and view the rendered markdown.



38. Click the Save as draft button; you should now see buttons to exit the draft or publish it.



39. Click the Publish button; you should now see the new information about the API in the API portal.



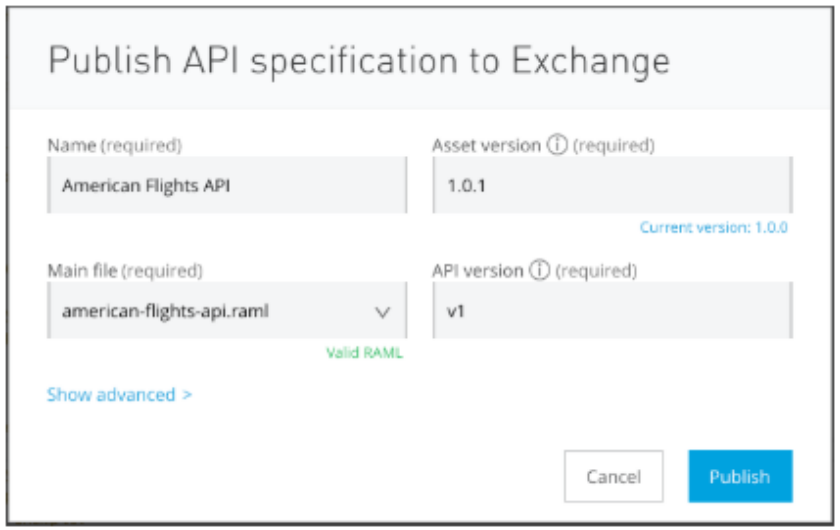
Modify the API and publish the new version to Exchange

40. Return to your American Flights API in API designer.
41. Return to the course snippets.txt file and copy the American Flights API - /{ID} DELETE and PUT methods.
42. Return to american-flights-api.raml and paste the code after the {ID}/get method.
43. Fix the indentation.
44. Review the code.

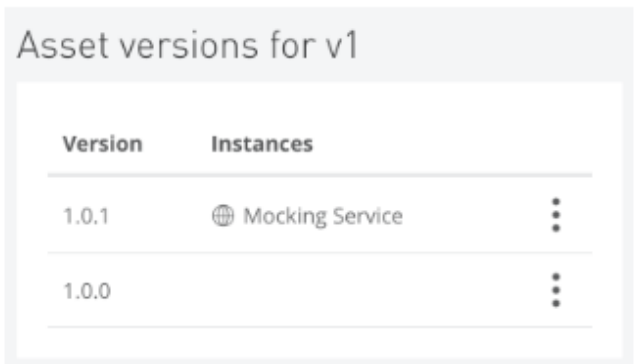
```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
            example: !include examples/AmericanFlightExample.raml
  delete:
    responses:
      200:
        body:
          application/json:
            example:
              message: Flight deleted (but not really)
  put:
    body:
      application/json:
        type: AmericanFlight
        example: !include examples/AmericanFlightExample.raml
    responses:
      200:
        body:
          application/json:
            example:
              message: Flight updated (but not really)
```

45. Click the Publish to Exchange button.

46. In the Publish API specification to Exchange dialog box, look at the asset version.

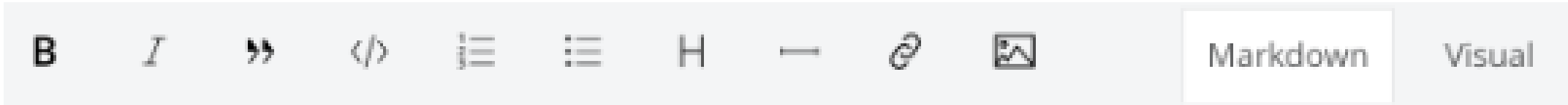


47. Click Publish.
48. In the Publish API specification to Exchange dialog box, click Exchange; Exchange should open in a new browser tab.
49. Locate the asset versions listed for the API; you should see both asset versions of the API listed with an associated API instance using the mocking service for the latest version.



50. Click the Edit button.

51. Add two new operations that delete a flight and update a flight.

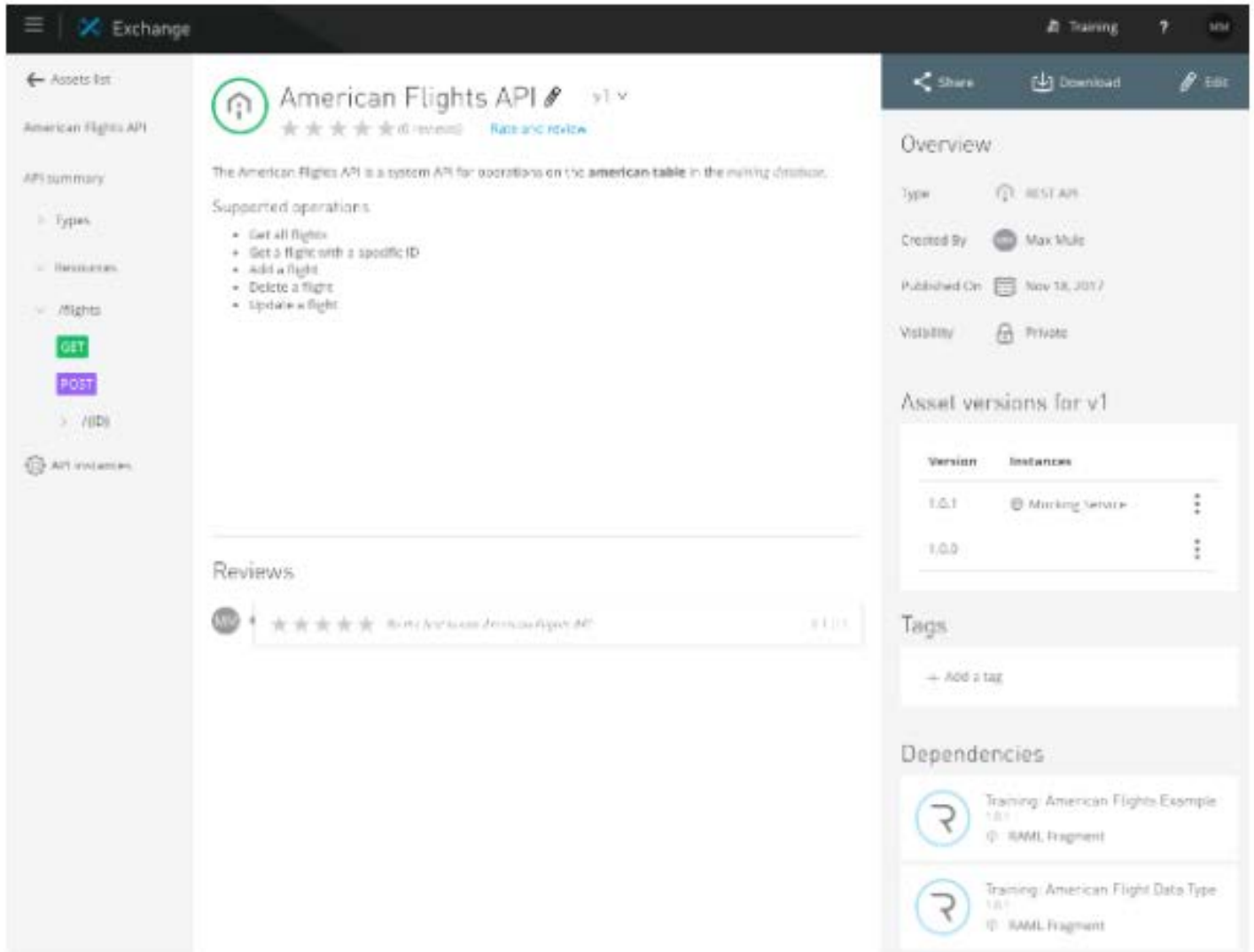



```
The American Flights API is a system API for operations on the **american
table** in the _training database_.

#### Supported operations

- Get all flights
- Get a flight with a specific ID
- Add a flight
- Delete a flight
- Update a flight
```

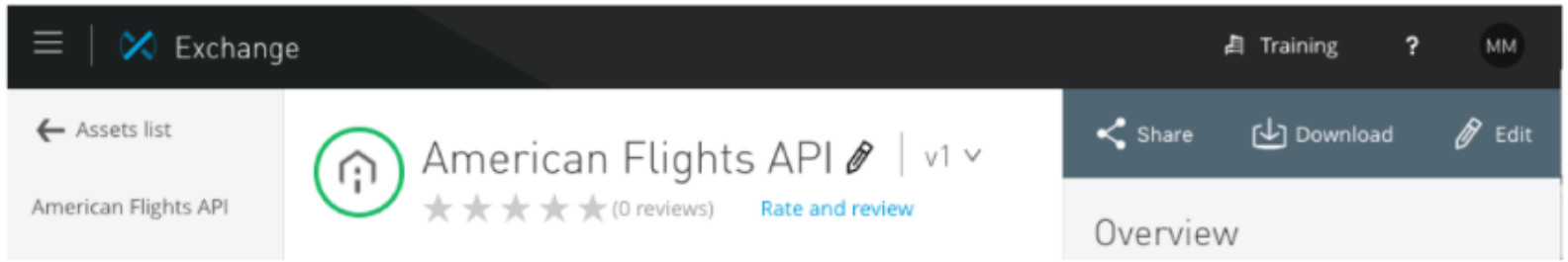
52. Click Save as draft and then Publish.



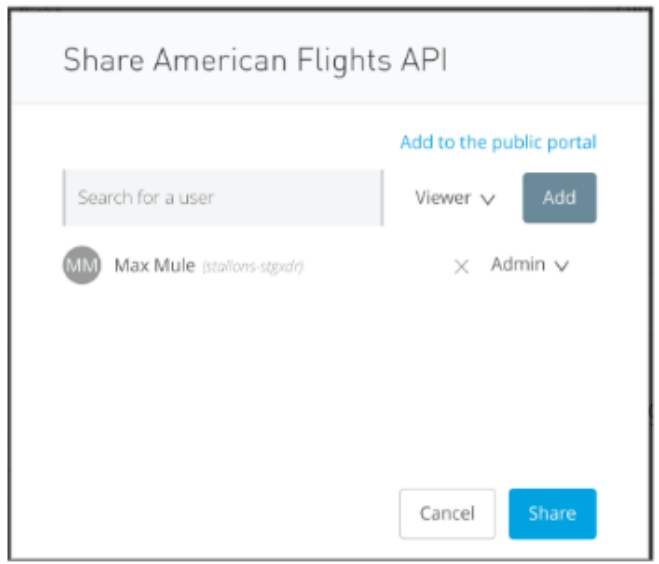
Sharing APIs

Sharing APIs

- You can share an API in Exchange with other internal or external users

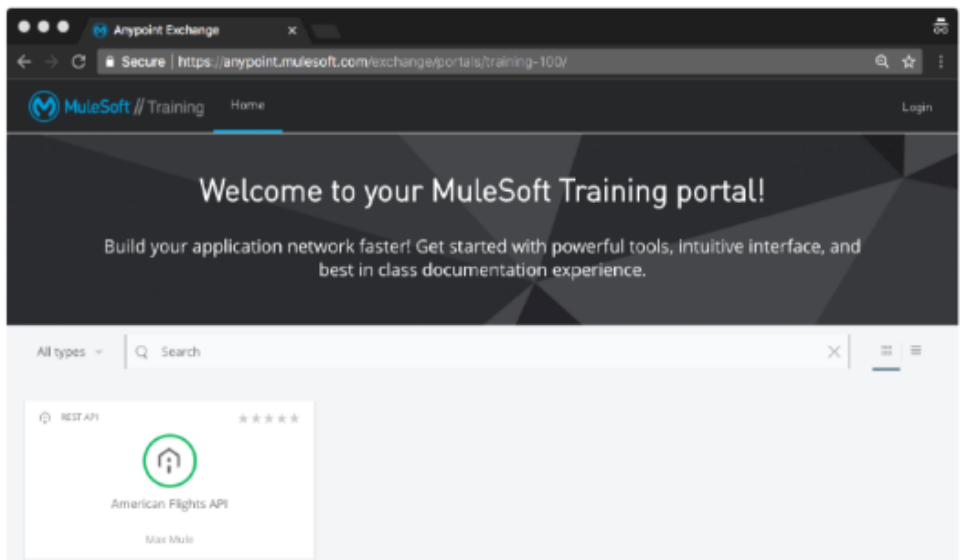


- Share an API within an org through the **private Exchange**
- Share an API with external users in a **public portal** that you create from Exchange



Walkthrough 3-5: Share an API

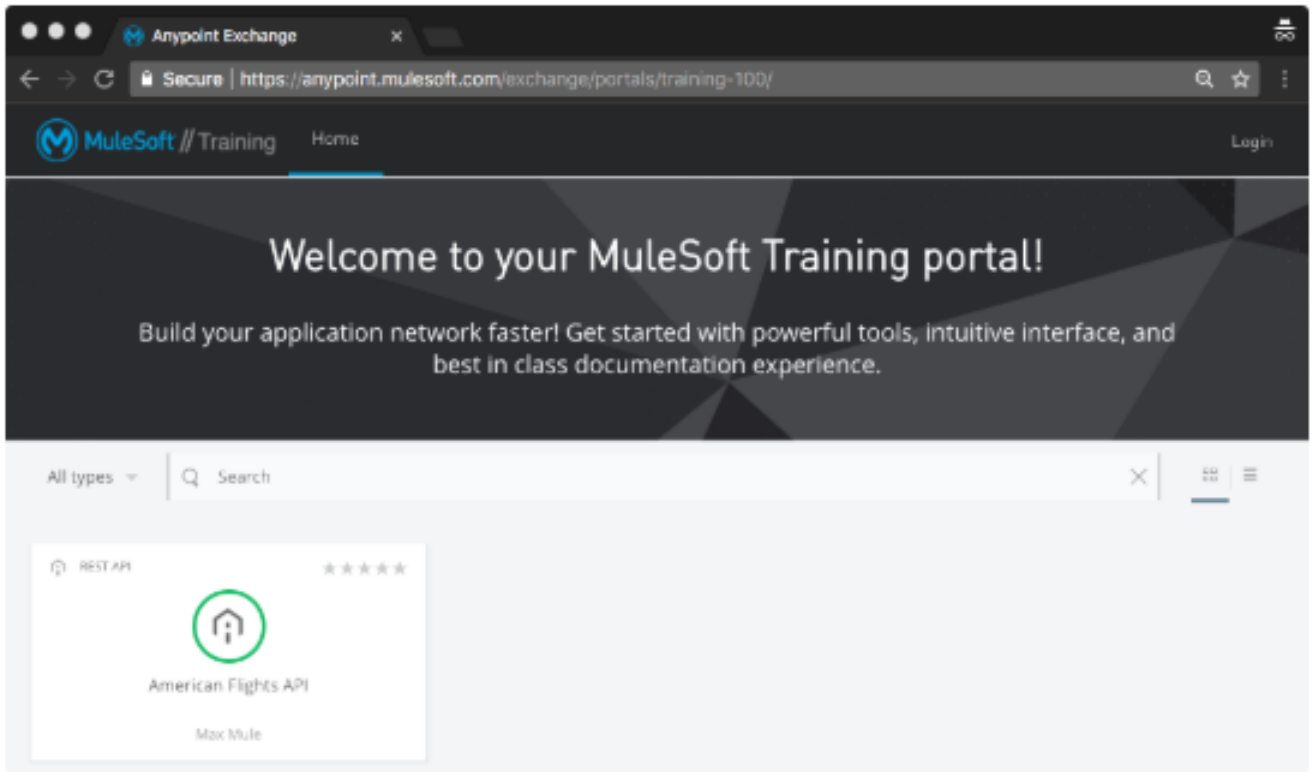
- Share an API within an organization using the private Exchange
- Create a public API portal
- Customize a public portal
- Explore a public portal as an external developer



Walkthrough 3-5: Share an API

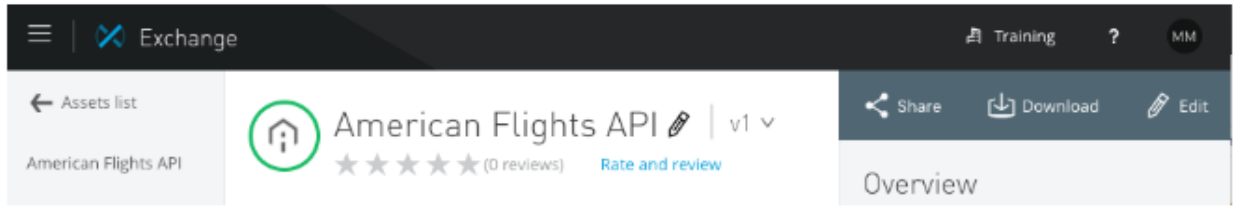
In this walkthrough, you share an API with both internal and external developers to locate, learn about, and try out the API. You will:

- Share an API within an organization using the private Exchange.
- Create a public API portal.
- Customize a public portal.
- Explore a public portal as an external developer.

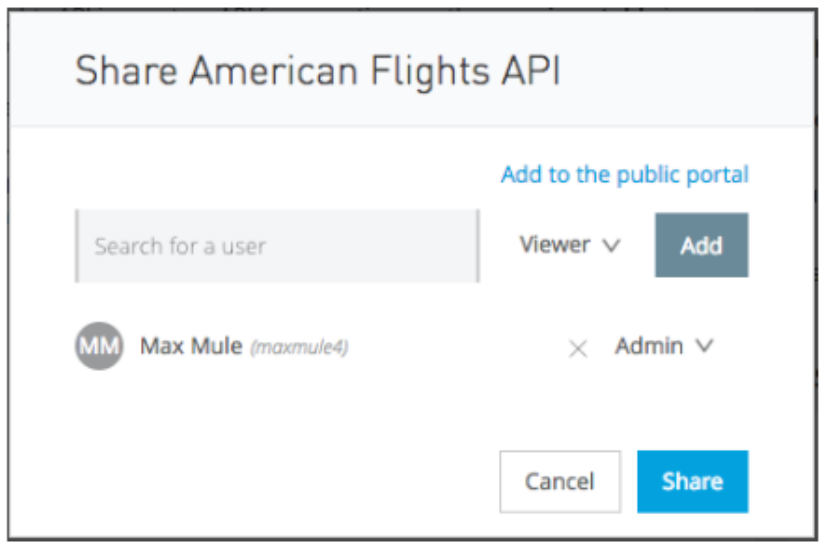


Share the API in the private Exchange with others

1. Return to your American Flights API in Exchange.
2. Click Share.



3. In the Share American Flights API dialog box, you should see that you are an Admin for this API.



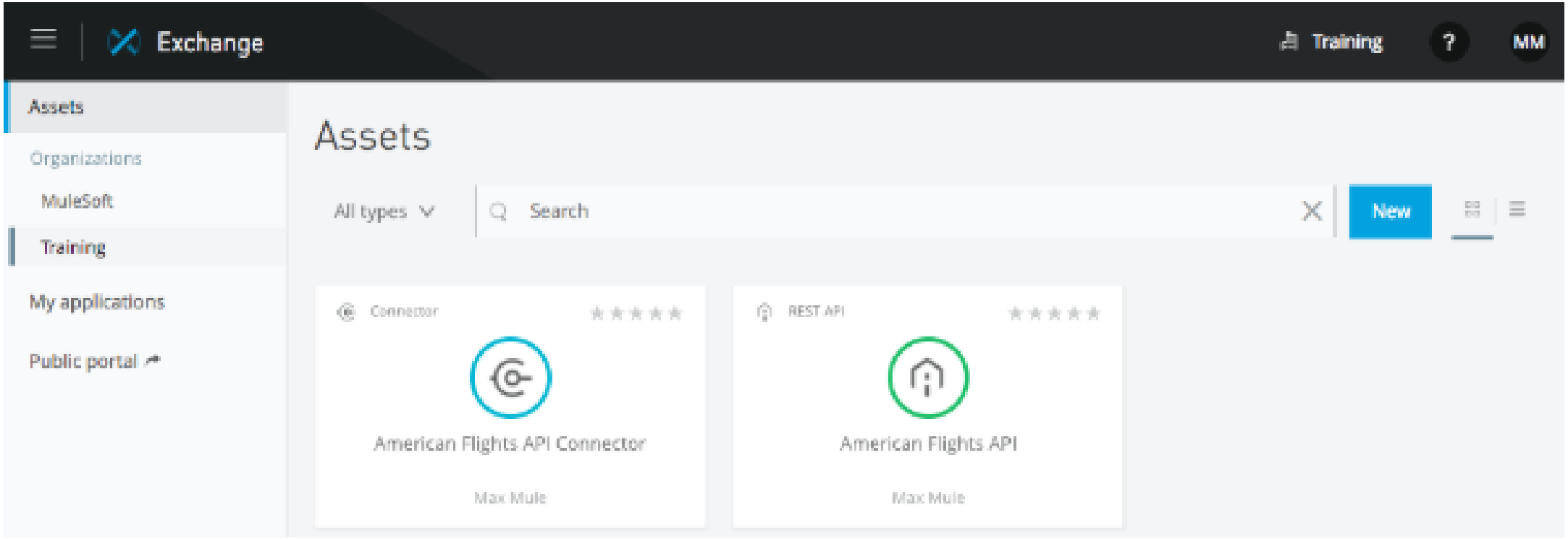
4. Open the Viewer drop-down menu located next to the user search field; you should see that you can add additional users to be of type Viewer, Contributor, or Admin.

   Note: In the future, you will also be able to share an asset with an entire business group.

5. Click Cancel.
6. In the left-side navigation, click Assets list.

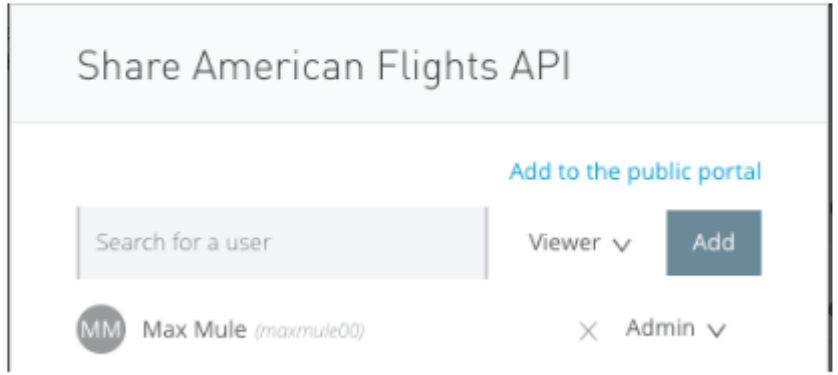
7. In the left-side navigation, select the name of your organization.
8. Click your American Flights API.

Note: This is how users you share the API with will find the API.

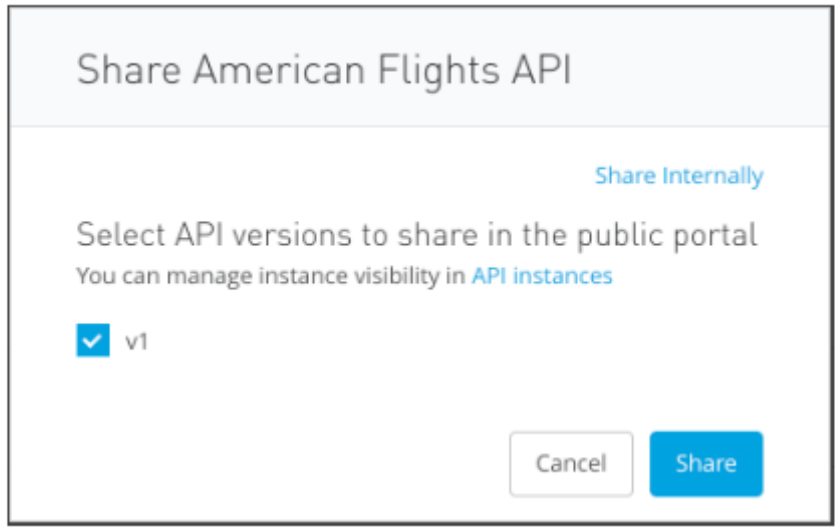


Create a public API portal

9. Click the Share button for the API again.
10. In the Share American Flights API dialog box, click Add to the public portal.

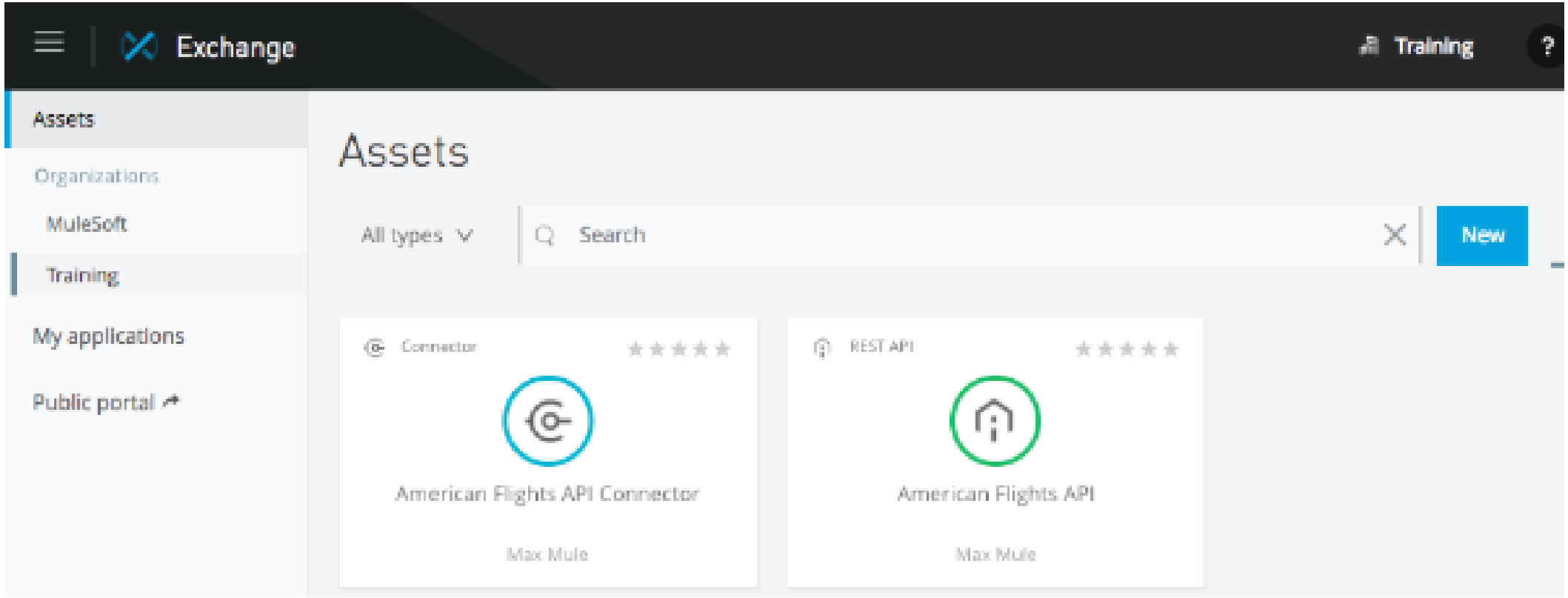


11. In the new dialog box, select to share v1 of the API.
12. Click Share.



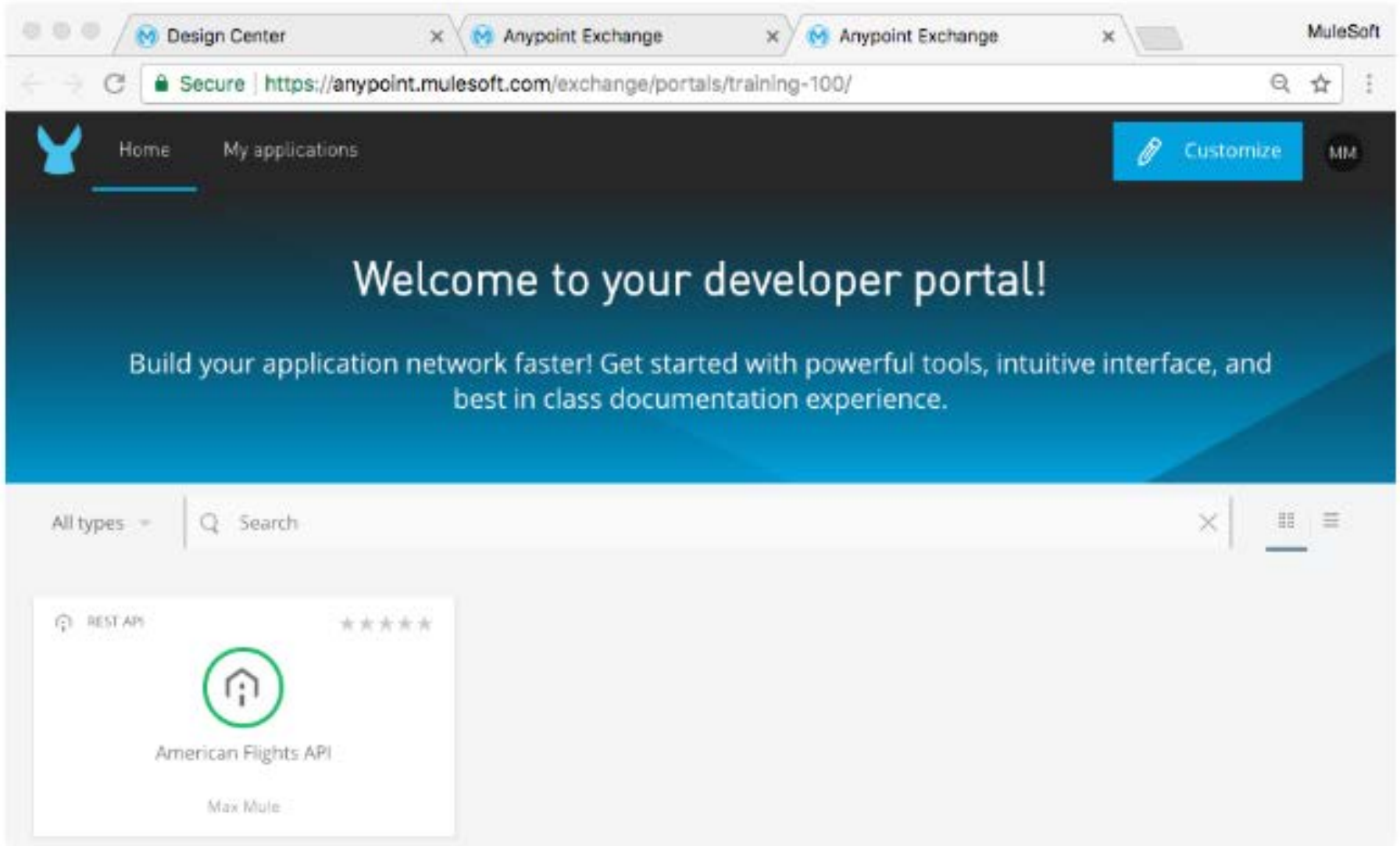
Explore the API in the public portal

13. In the left-side navigation, click Assets list.
14. In the left-side navigation, select Public Portal.

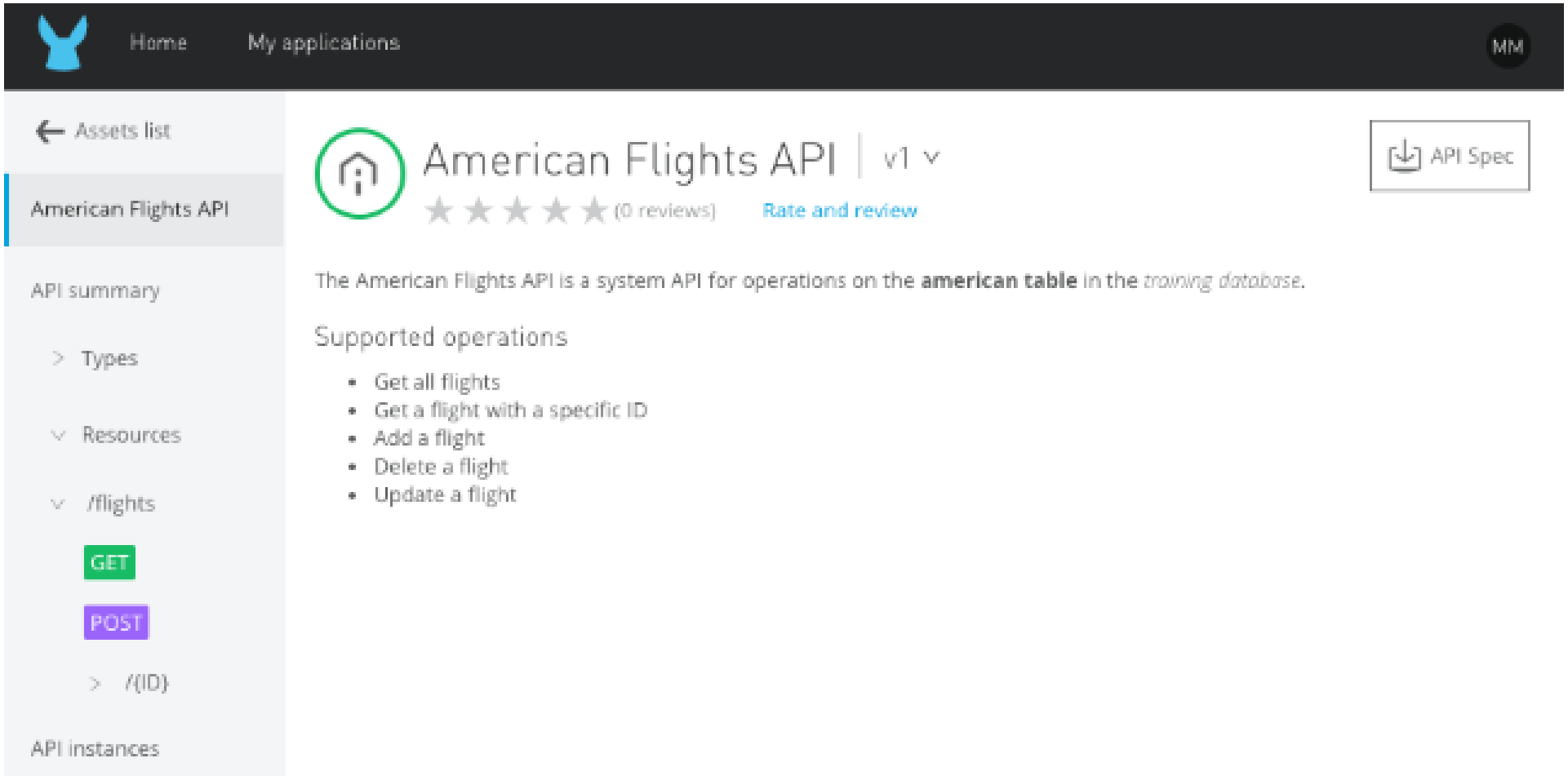


15. Review the public portal that opens in a new browser tab.

16. Look at the URL for the public portal.

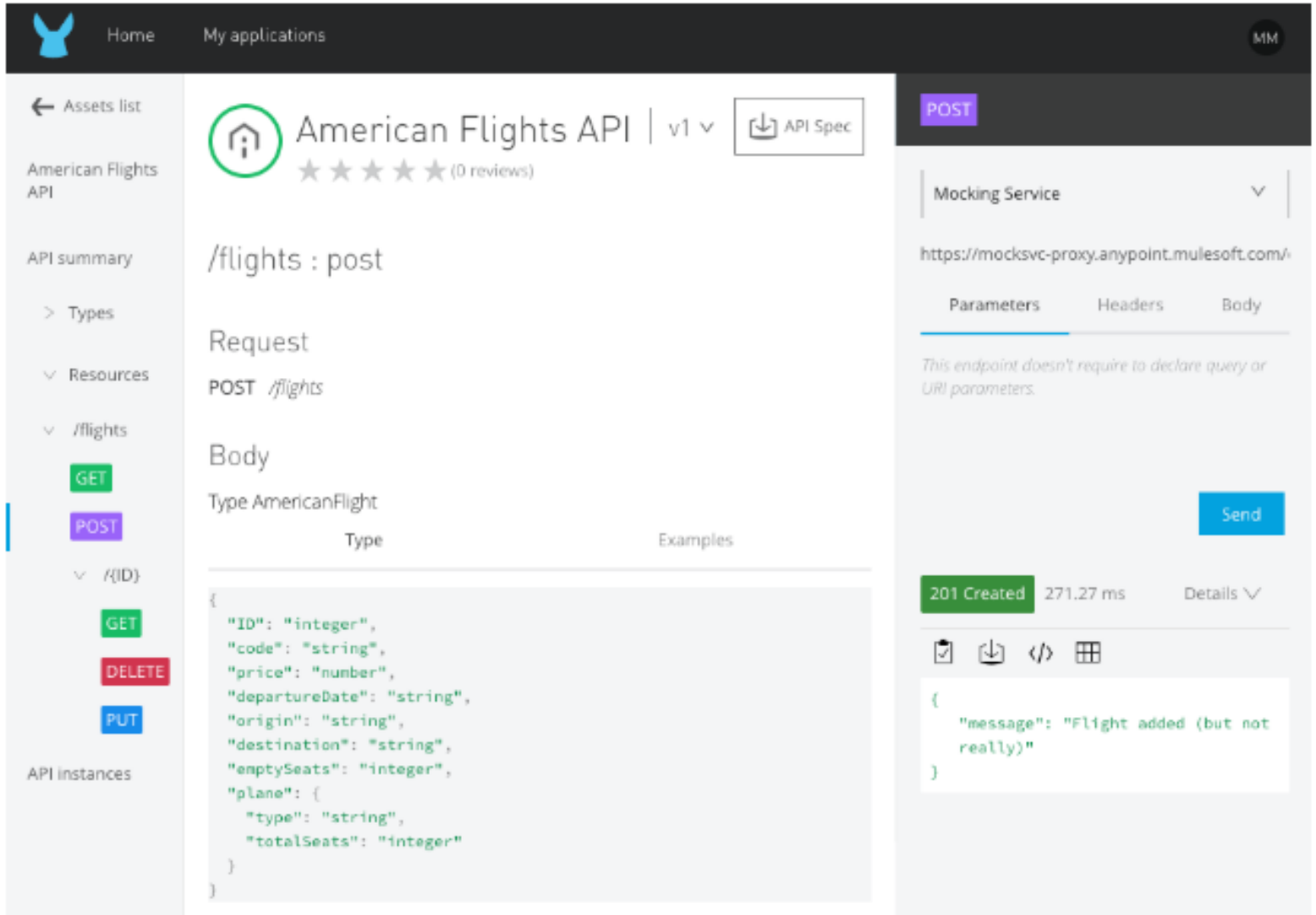


17. Click your American Flights API and review the auto-generated public API portal.



18. In the left-side navigation, click the POST method for the flights resource.
19. Review the body and response information.

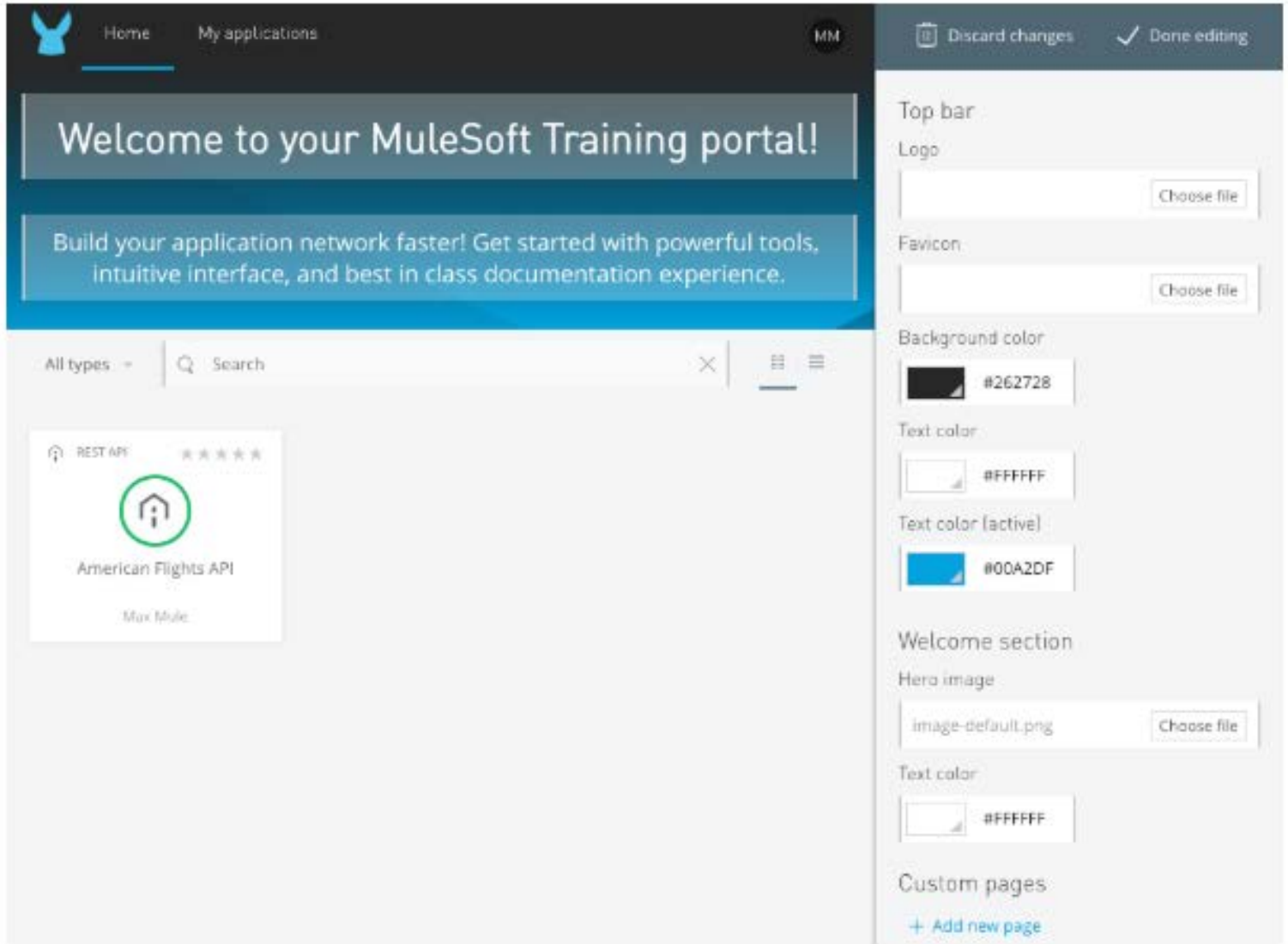
20. In the API console, click Send; you should get a 201 response with the example data returned from the mocking service.



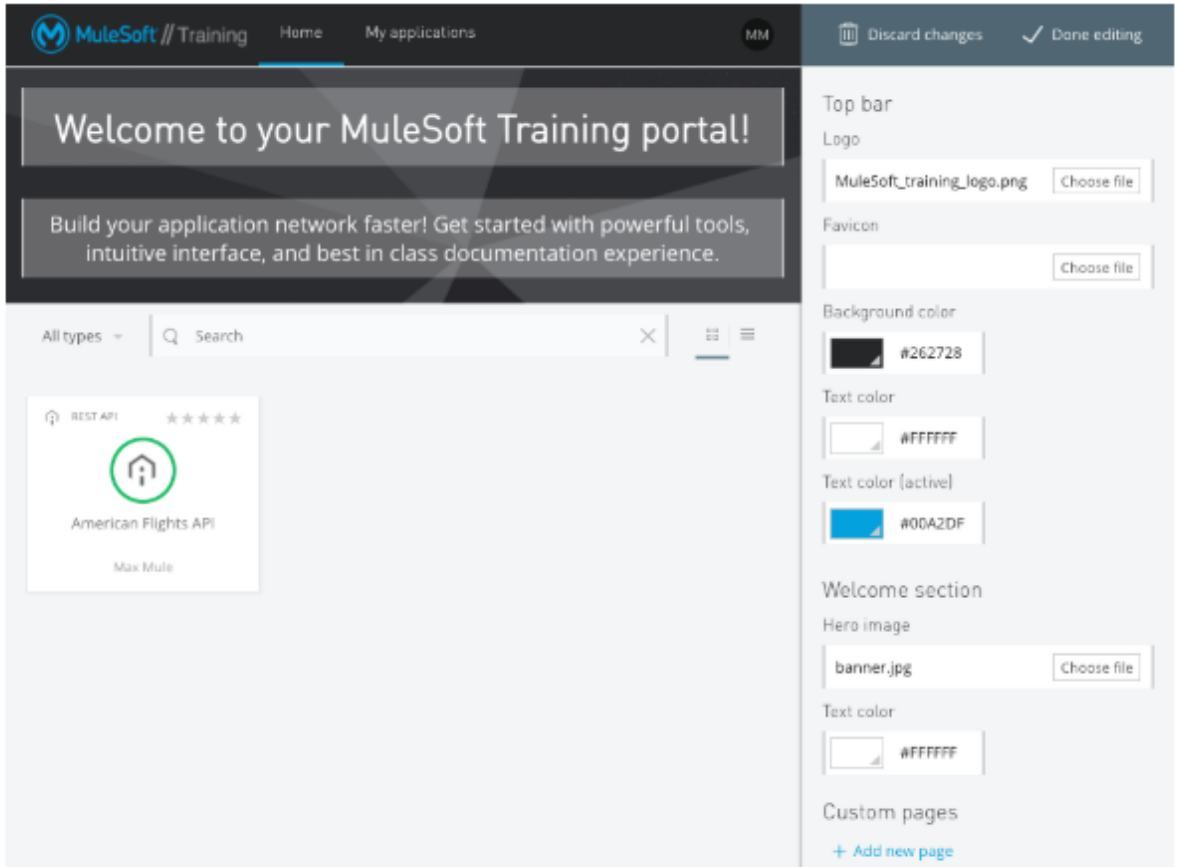
Customize the public portal

21. In the left-side navigation, click Assets list.
22. Click the Customize button.

23. Change the text to Welcome to your MuleSoft Training portal!



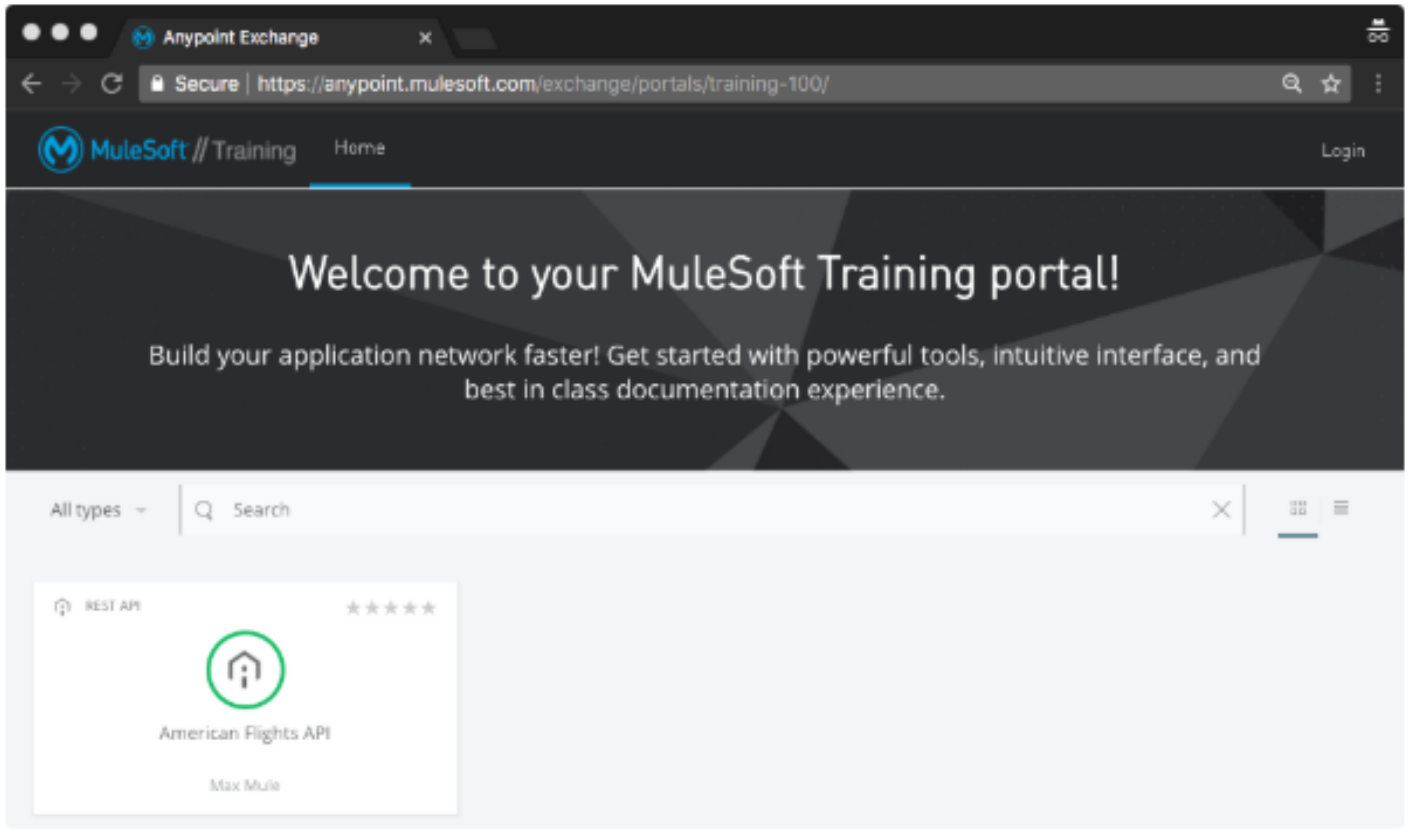
24. In the logo field, click Choose file.
25. In the file browser dialog box, navigate to the student files and locate the MuleSoft_training_logo.png file in the resources folder and click Open.
26. Locate the new logo in the preview.
27. In the hero image field, click Choose file.
28. In the file browser dialog box, navigate to the student files and locate the banner.jpg file in the resources folder and click Open.
29. Review the new logo and banner in the preview.



30. Change any colors that you want.
31. Click the Done editing button.
32. In the Publish changes dialog box, click Yes, publish; you should see your customized public portal.

Explore the public portal as an external developer

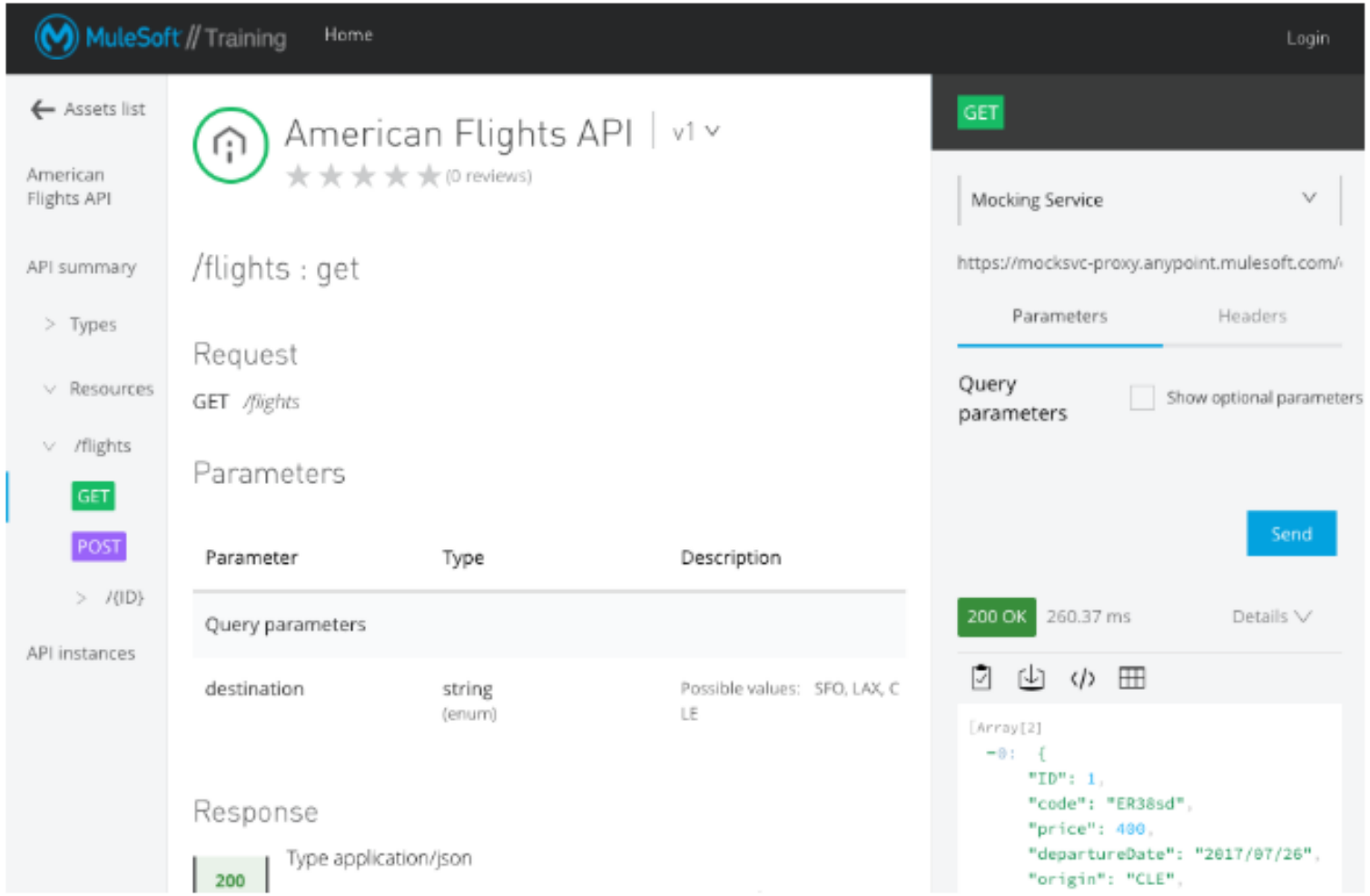
33. In the browser, copy the URL for the public portal.
34. Open a new private or incognito window in your browser.
35. Navigate to the portal URL you copied; you should see the public portal (without the customize button).



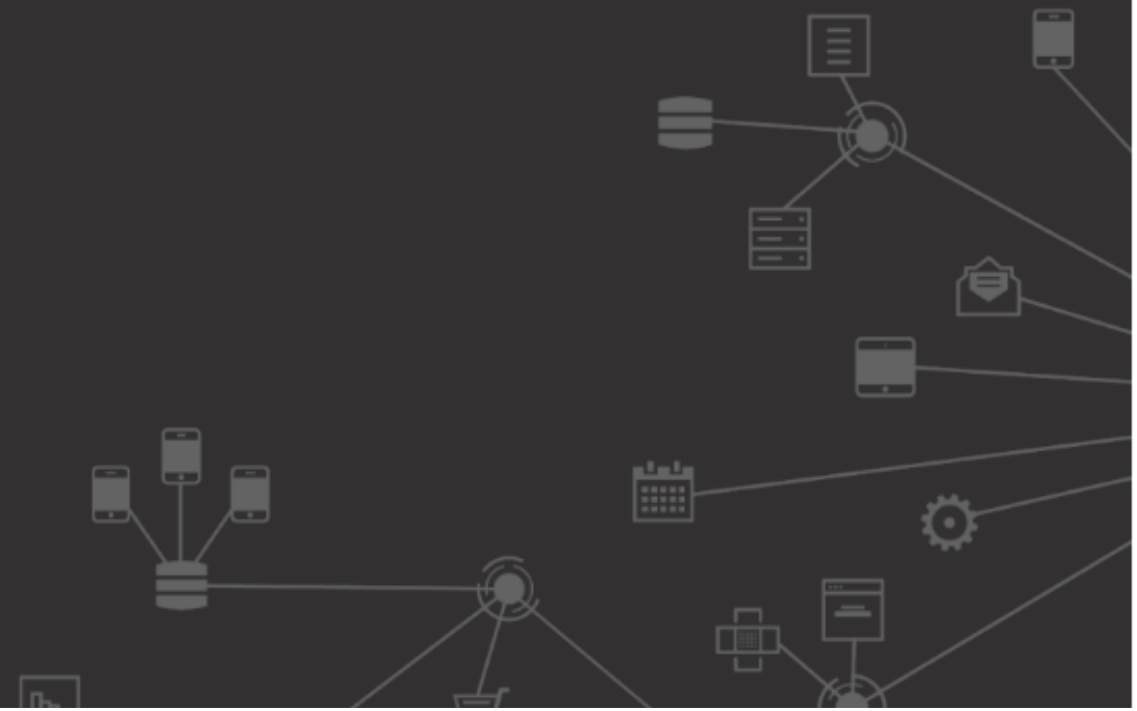
36. Click the American Flights API.
37. Explore the API portal.

38. Make a call to one of the resource methods.

Note: As an anonymous user, you can make calls to an API instance that uses the mocking service but not managed APIs.



Summary



Summary

MuleSoft

- **RAML** is a non-proprietary, standards-based API description language spec that is simple, succinct, and intuitive to use
  - Data structure hierarchy is specified by indentation, not markup characters
- Use **API designer** to write API specifications with RAML
- Documentation is auto-generated from a RAML file and displayed in an **API console**
- A **mocking service** can be used in API console to test an API and return the example data specified in RAML

Summary

- Make an **API discoverable** by adding it to your **private Exchange**
- **API portals** are automatically created for the APIs with
  - Auto-generated **API documentation**
  - An **API console** that provides a way to consume and test an API
  - An **automatically generated API endpoint** that uses a **mocking service** to allow the API to be tested without having to implement it
- API portals can be shared with both internal and external users
  - Selectively share APIs in your org's **private Exchange** with other internal developers
  - Share APIs with external developers by creating and customizing a **public portal** from Exchange and specifying what APIs you would like to include in it

RAML resources

- RAML definitions can be a lot more complex and sophisticated then what we built here

- Training: training.mulesoft.com
  - *Anypoint Platform: API Design*

- Website: raml.org
  - Documentation
  - Tutorials
  - Full spec
  - Resources

DIY Exercise 3-1: Create an API specification with RAML

Time estimate: 1 hour

Objectives

In this exercise, you create an API specification using RAML. You will:

- Appropriately specify GET and POST methods.
- Appropriately define URI parameters, query parameters, and headers.
- Restrict possible values using an enumeration.

Scenario

Your company needs to expose a customer accounts database as a System API for the rest of the organization. The first step is to create the RAML specification and post it to your company's private Anypoint Exchange so all stakeholders can review and provide feedback. This RAML specification needs to separate out data type definitions from the main API RAML file into separate reusable RAML files.

Create the API specification

In Design Center, create a RAML specification called Accounts API with the following requirements:

- The API has a resource called accounts.
- All client requests must include the required header Requester-ID. This header helps identify the person in the organization making account requests.
- The /accounts resource has a GET method with a required queryParameter named type, where type must be either personal or business. The type query parameter is used to return records that match that account's type.
- The GET method for the /accounts resource has two optional string queryParameters named name and country. The name query parameter value is used to filter and return all account records for a particular account owner. The country query parameter value is used to filter and return records matching the account owner's country of residence.
- The GET method's response data is an array of Account objects. To represent the response data, define and use an Account data type that contain the following fields: id, firstName, lastName, address, postal, country, miles, creationDate, and type. The creationDate field is of type datetime and miles is of type integer.
- The API defines and uses a data type for the Account object that is stored in a separate RAML file.
- The Account data type definition has a corresponding example that is included in the main API RAML file or in the Account data type RAML file.
- The /accounts resource has a POST method that accepts an array of Account objects (with no id and no creationDate field) in JSON format and returns a JSON message: {"message": "Account created (but not really)"}. This Account object is represented as a new data type that does not have id or creationDate fields, makes the type field required, and has a corresponding example.
- Each method should have a description.
- Each method should have a custom 400 error status message example in JSON format.

Publish the API to Anypoint Exchange

Publish the API to Anypoint Exchange and then test the mocked endpoint using the API Console and create a public portal for the API.

Verify your solution

Load the solution /files/module03/accounts-mod03-api-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.