Module 3: Designing APIs

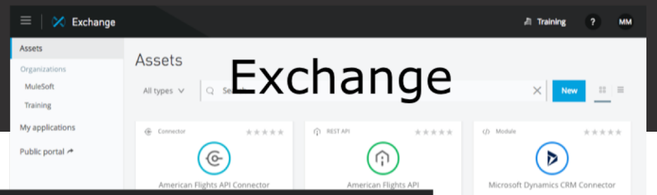
Spec driven development

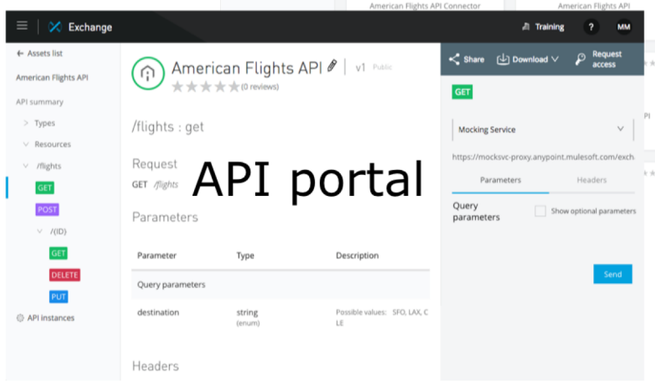
- We discussed in the last modules about the benefits of designing an API first before actually building it

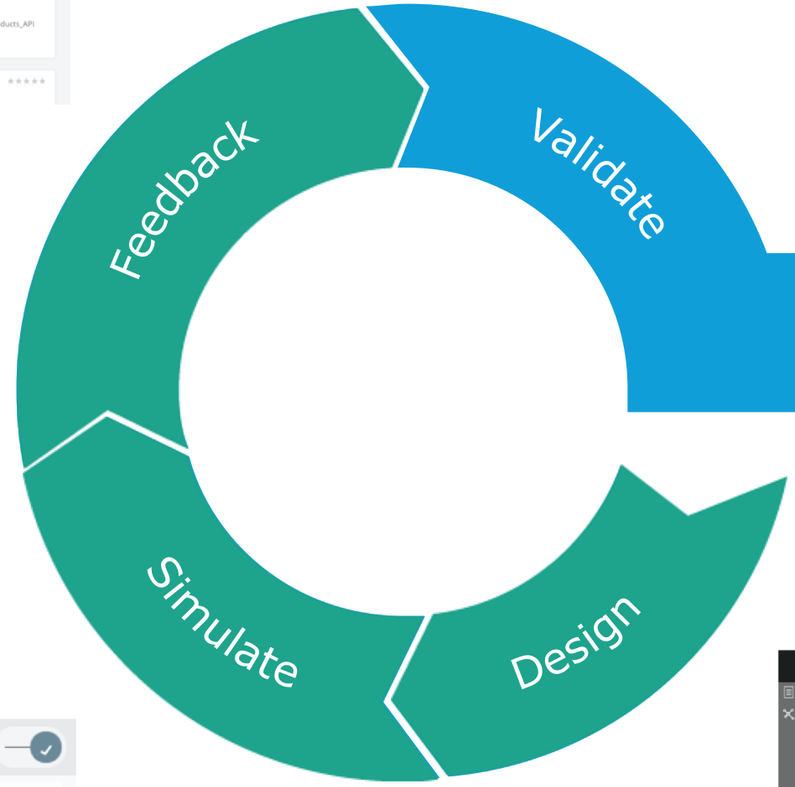
- This is often referred to as **spec driven development**
  - A development process where your application is built in two distinct phases
    - The creation of a spec (the design phase)
    - Development of code to match the spec (the development phase)

- 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

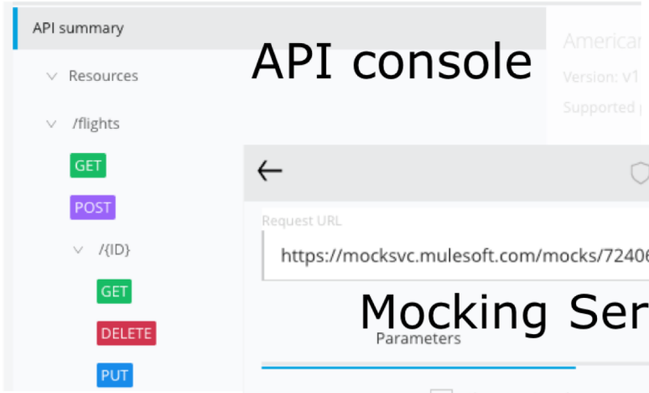
Goal

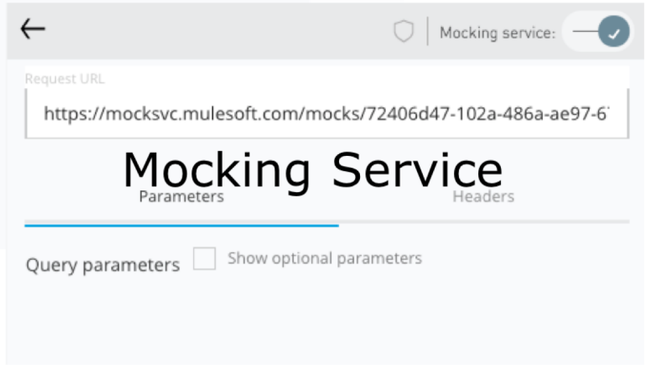


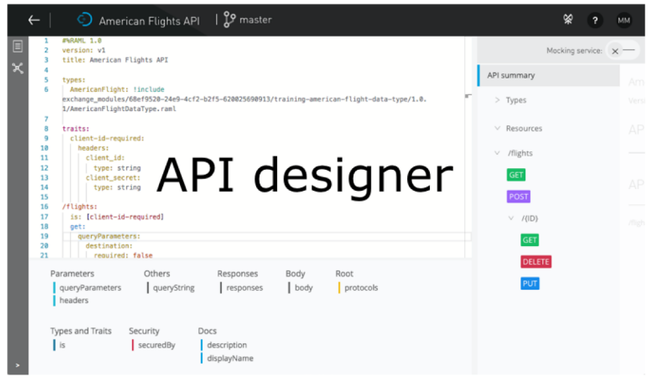




API Spec
(RAML)







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

OpenAPI Spec

RAML

Introducing RAML

RAML: RESTful API Modeling Language

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

http://raml.org

- A non-proprietary, vendor-neutral open spec

- 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

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

/flights:
  get:
  post:
  /{ID}:
    get:
    delete:
    put:
      responses:
        200:
          body:
            application/json:
```

Notice the indentation used to specify to what each line applies

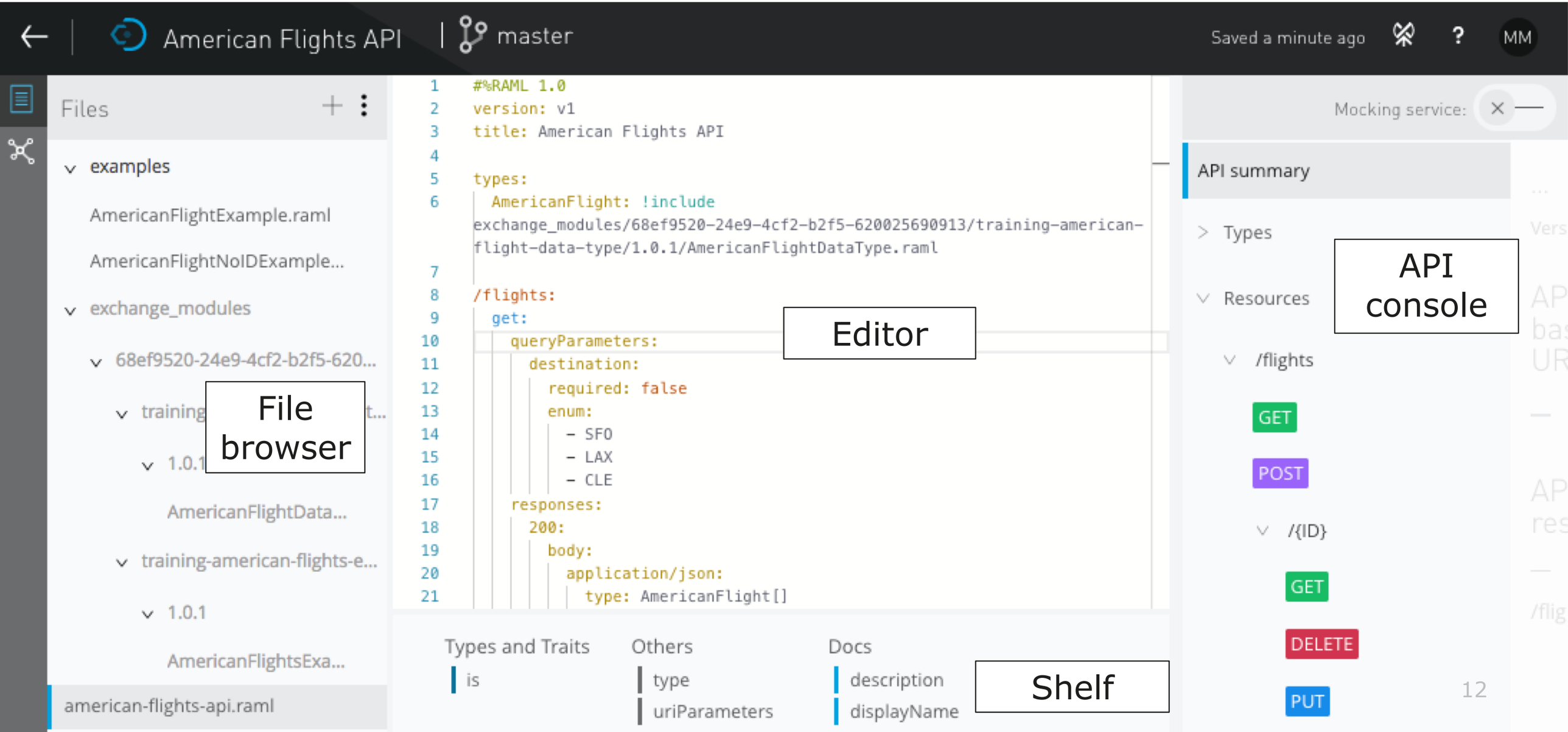
Defining resources and methods

- 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:
```

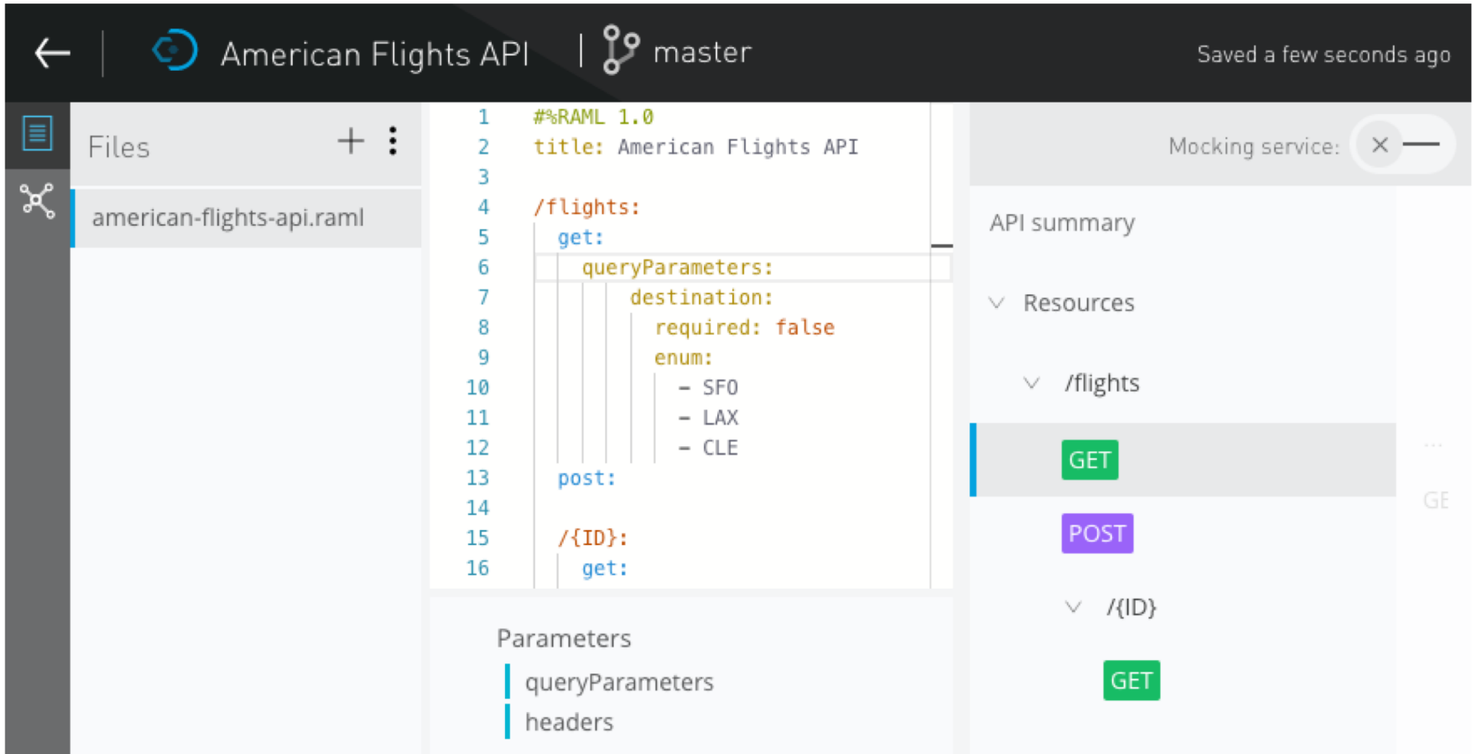
Using API designer to define APIs with RAML

API designer



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

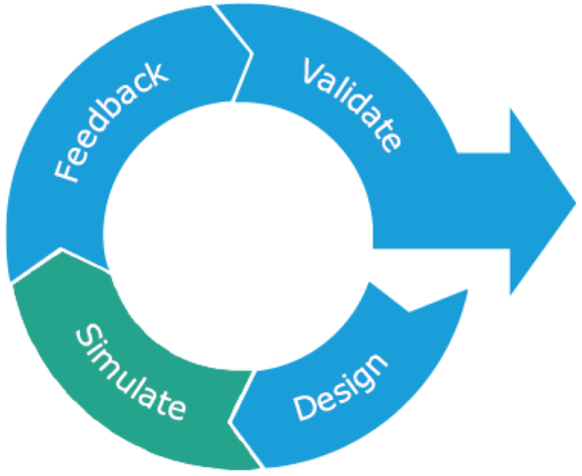
- Define resources and nested resources
- Define get and post methods
- Specify query parameters
- Interact with an API using the API console



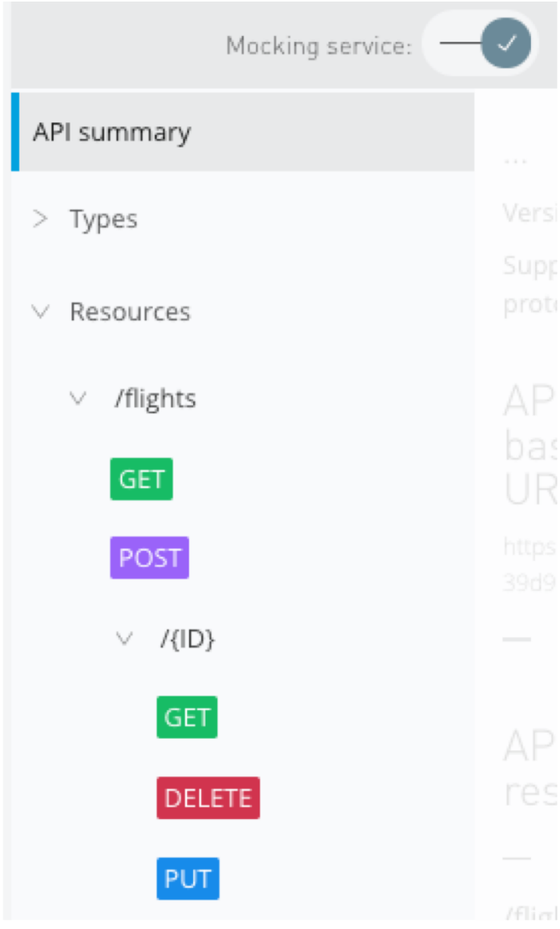
Testing API design without writing code

Simulating an API

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



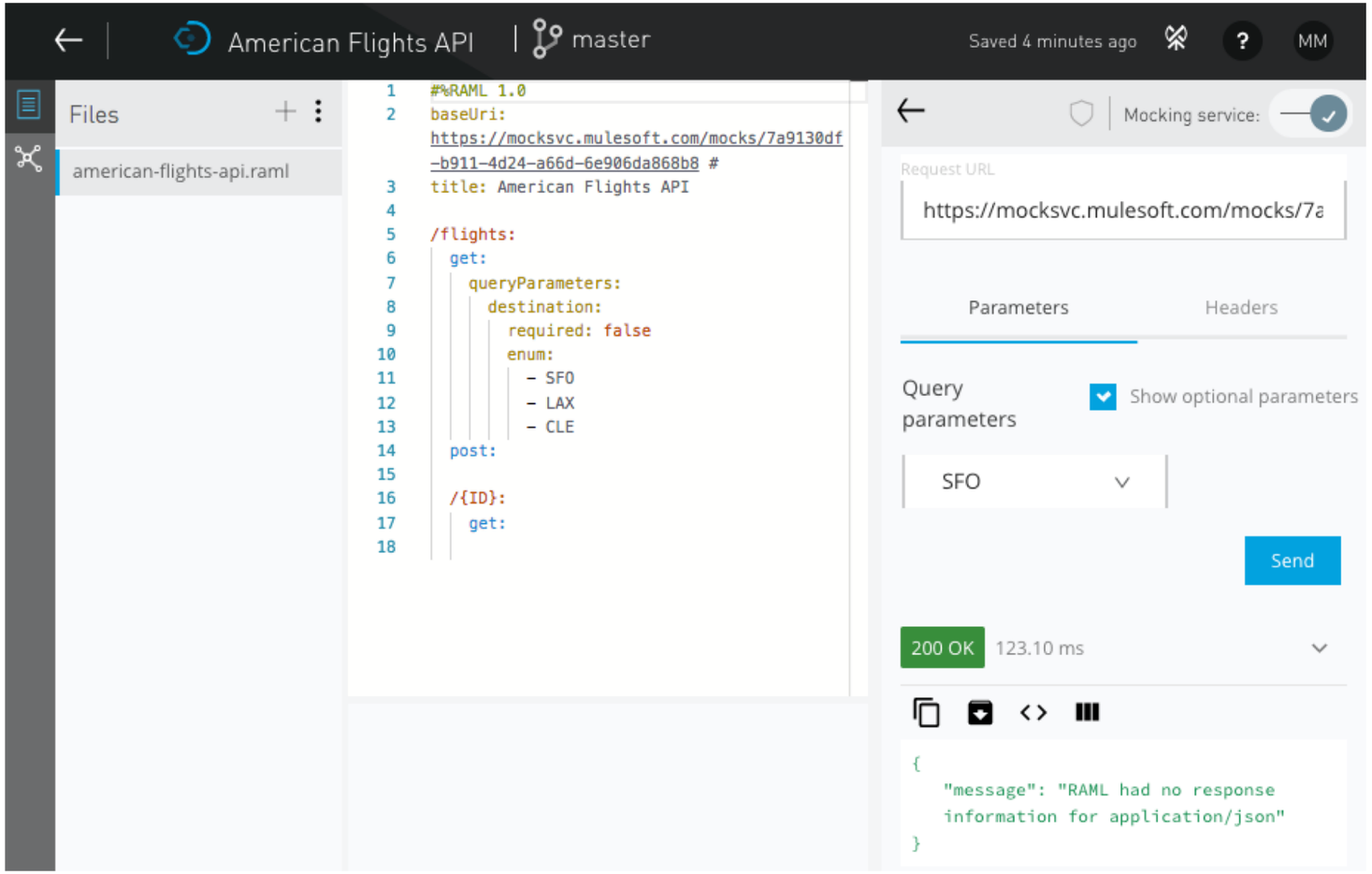
- Use the **API console** and the **mocking service** to run a live simulation
  - Returns sample API responses defined in the API definition



- 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



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

- 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
```

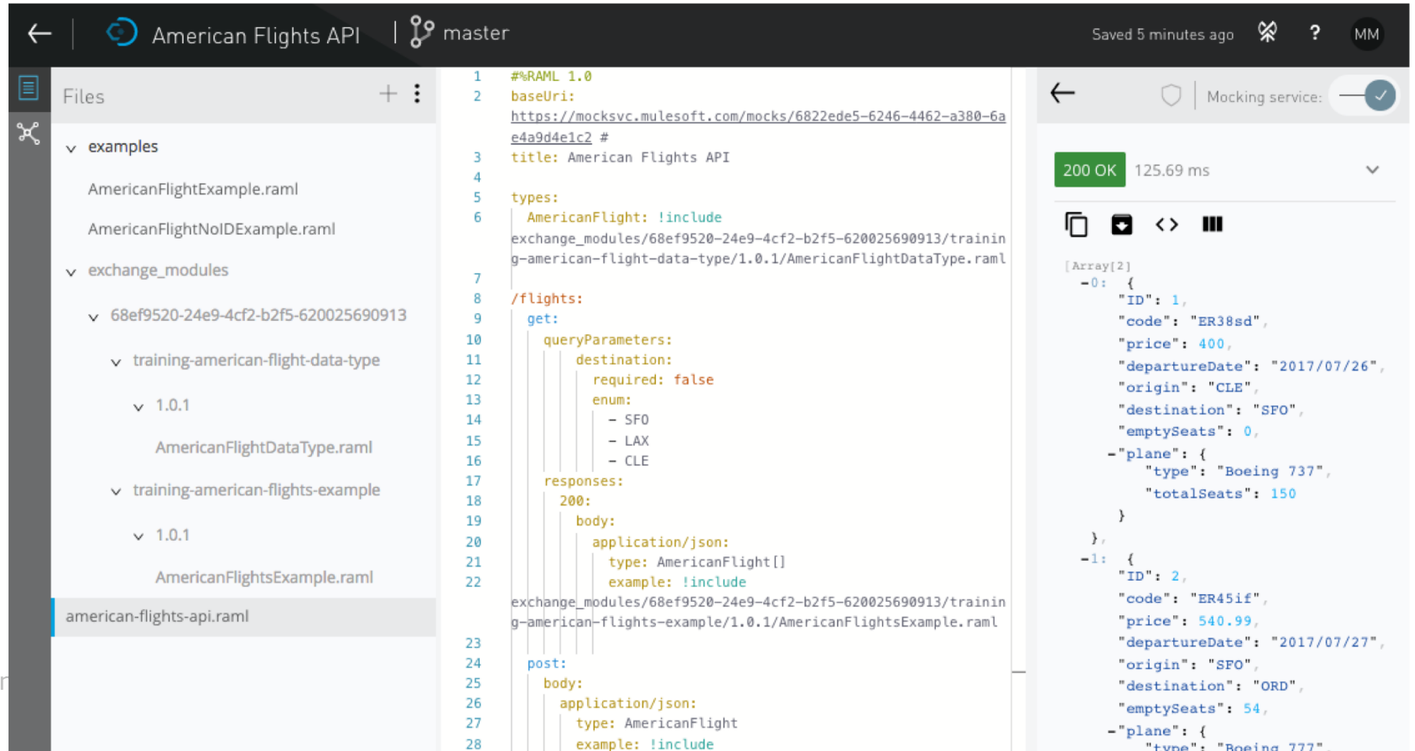
Modularizing APIs

- 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



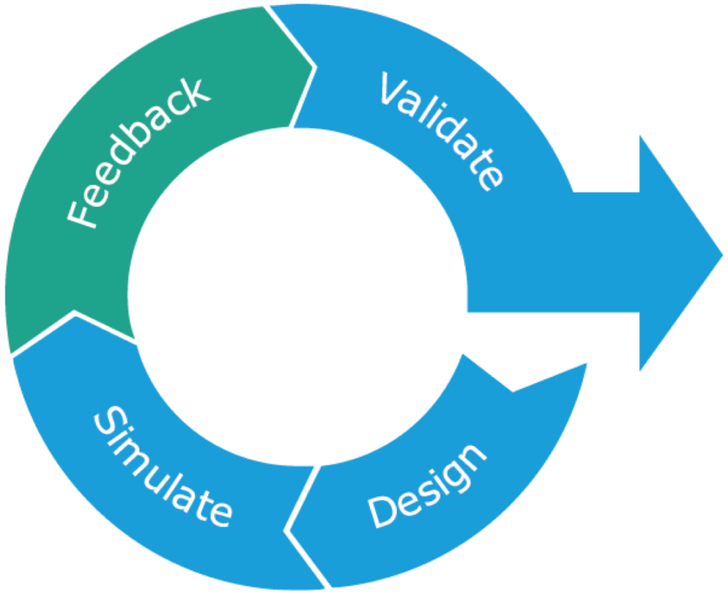
Engaging with users

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

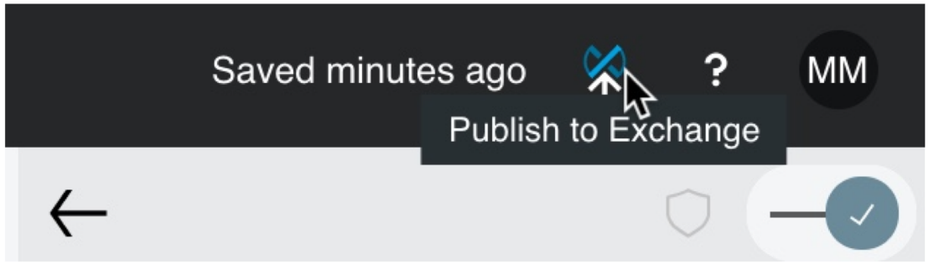
- To do this, you need to provide ways for developers to discover and play with the API

- 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

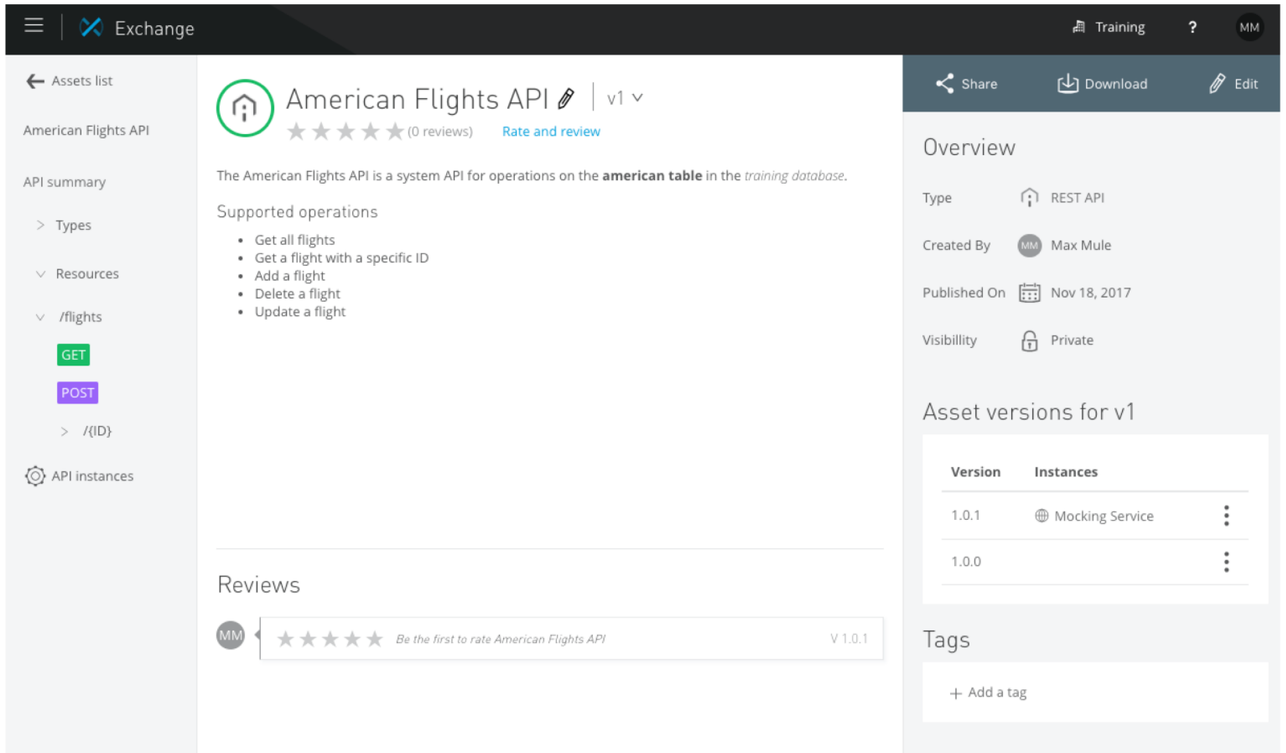


- **API portals** are automatically created for REST APIs added to Exchange
  - An **API console** for consuming and testing APIs
  - An **automatically generated API endpoint** that uses a **mocking service** to allow the API to be tested without having to implement it

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

Walkthrough 3-4: Add an API to Anypoint Exchange

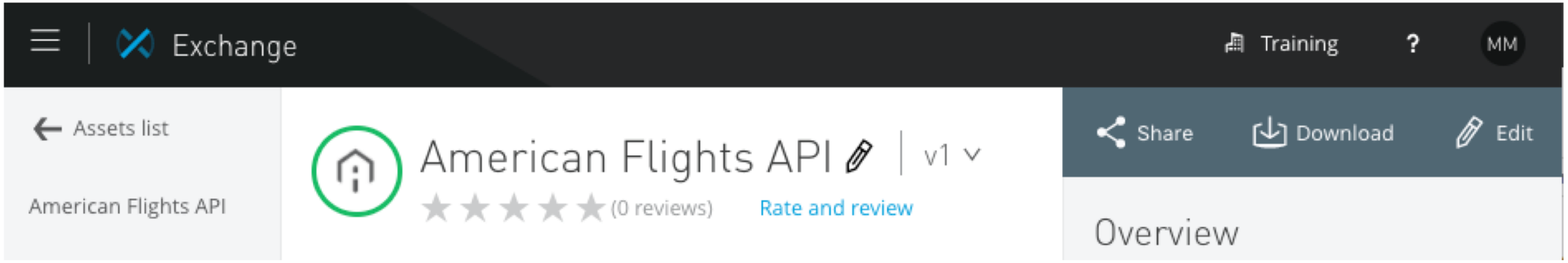
- Publish an API to Exchange from API designer
- Review an auto-generated API portal in Exchange and test the API
- Add information about an API to its API portal
- Create and publish a new API version to Exchange



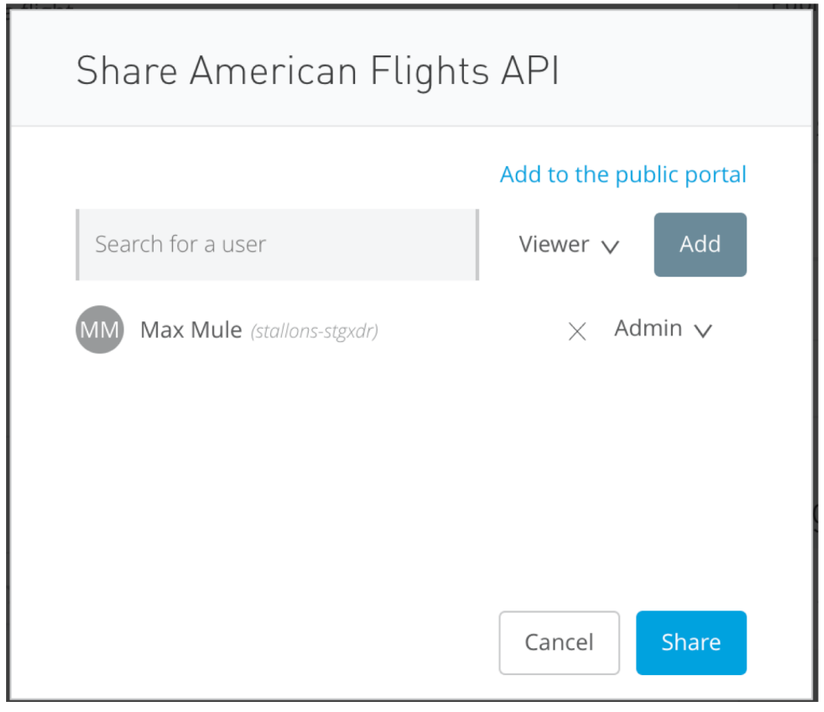
Sharing APIs

Sharing APIs

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

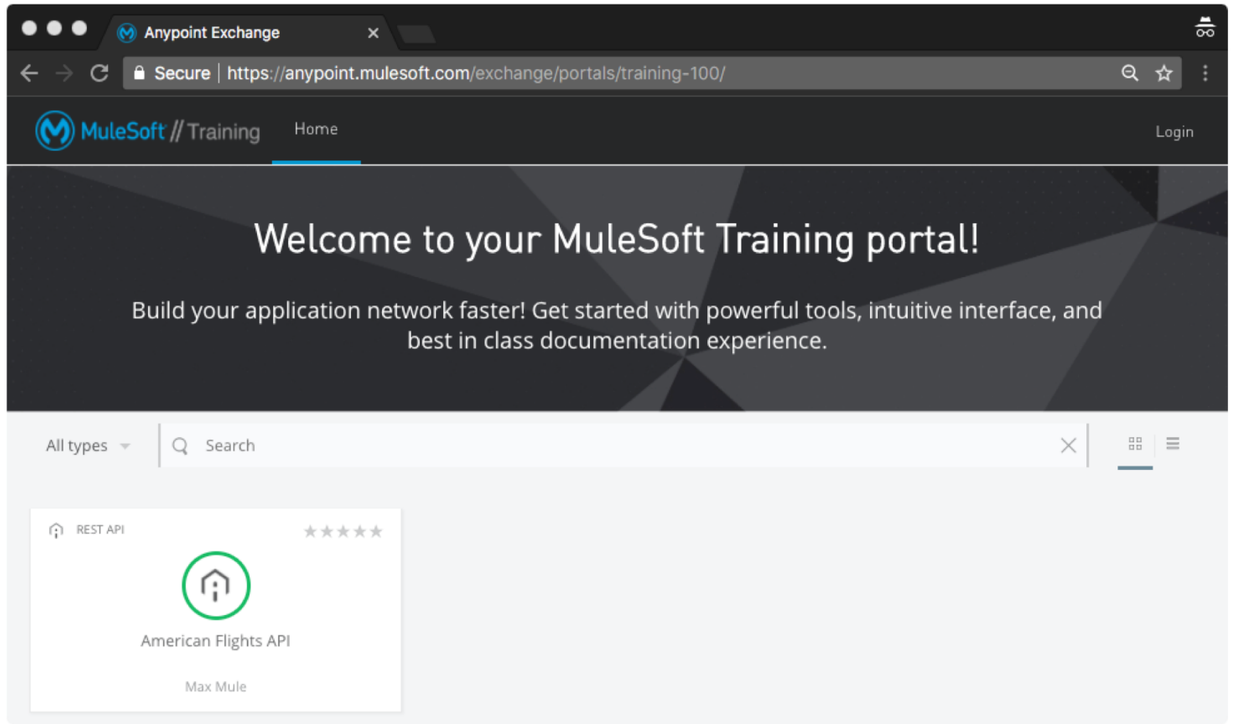


- Share an API within an org through the **private Exchange**
- Share an API with external users in a **public portal** that you create from Exchange



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



Summary

Summary

- **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