



Module 3: Designing APIs

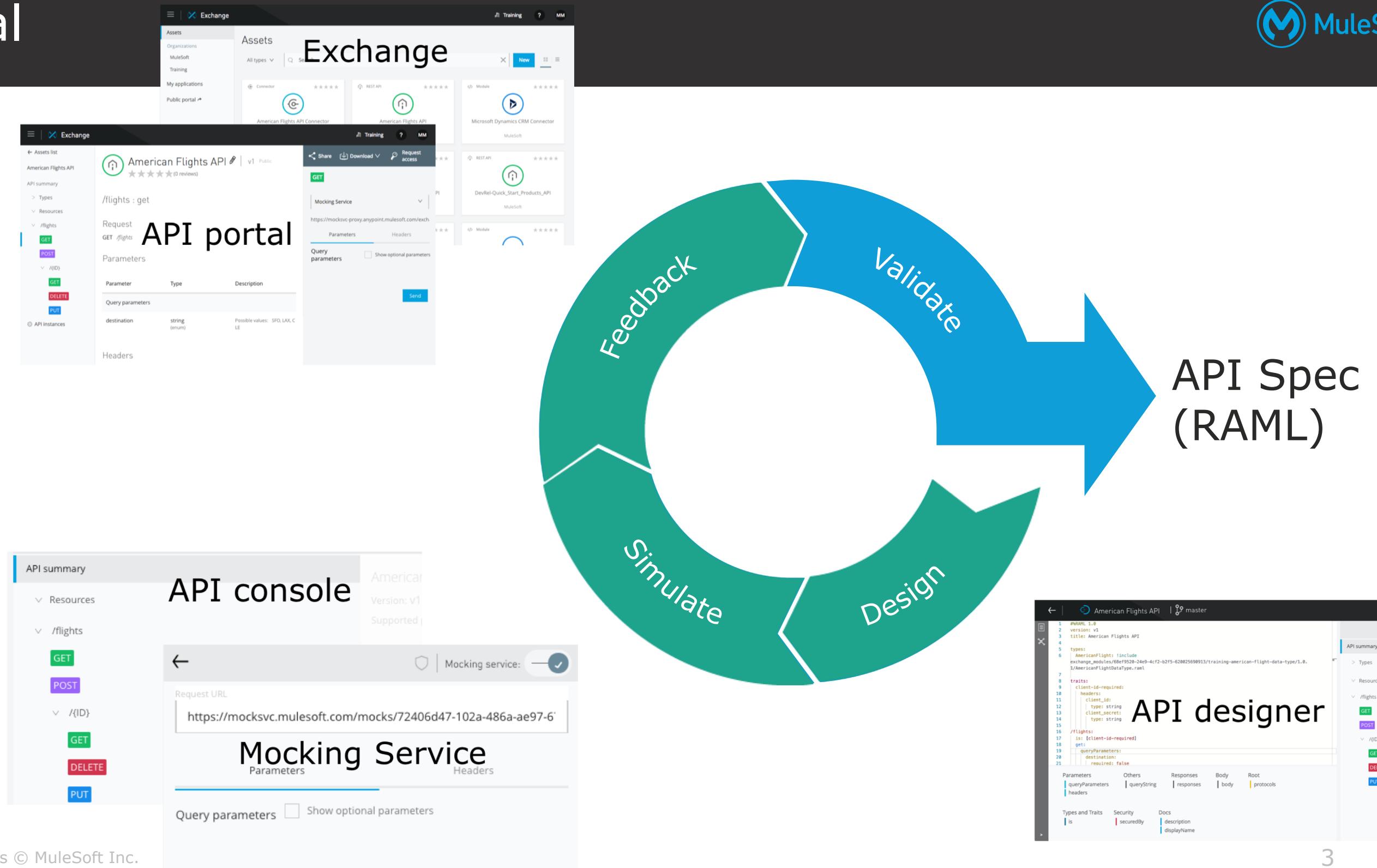


Spec driven development



- 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



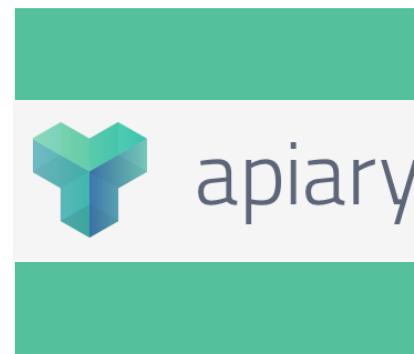
Approaches to API design



Hand coding

```
auditting service present
h: "/pci@0/pci-ata@1/ata-400/00:10:00:00"
IOPathMatch</key>string ID="1">IDeviceType:pci-Hd
reWire GUID = 0x50e4ff:8
sent:8
t device = IOService:/GossamerPE/pci1000000000000000/0
2PCIIBridge/pci-ata@1/CHD64Root/ata-400/D064000000000000
geDriver/IORATABlockStorageDevice/IOBlockStorageDrive/520000
titions
  - UNTITLED_3018
    for 14, minor 9
      created with uid=0 audit=1
```

API Blueprint



OpenAPI Spec



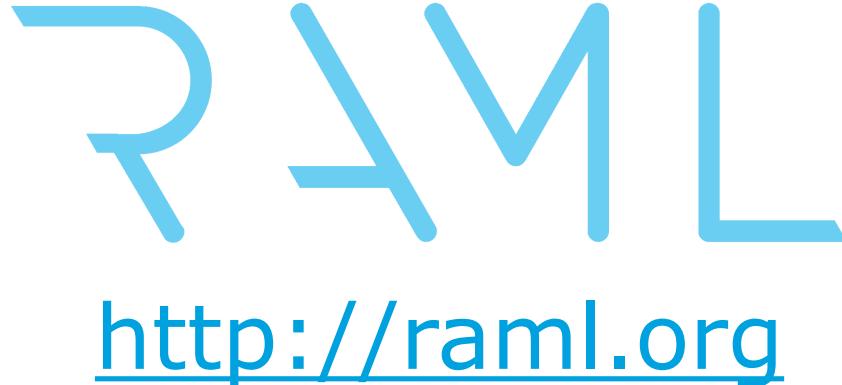
RAML



Introducing RAML



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



American Flights API | master

← | ⚙️ American Flights API | 🔒 master

Files + :

- examples
- AmericanFlightExample.raml
- AmericanFlightNoIDExample...
- exchange_modules
- 68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flight-data-type/1.0.1/AmericanFlightDataType.raml
- training
- 1.0.1
- AmericanFlightData...
- training-american-flights-e...
- 1.0.1
- AmericanFlightsExa...

Mocking service: X —

Editor

```
1  #%RAML 1.0
2  version: v1
3  title: American Flights API
4
5  types:
6    AmericanFlight: !include
7      exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-
8      flight-data-type/1.0.1/AmericanFlightDataType.raml
9
10 /flights:
11   get:
12     queryParameters:
13       destination:
14         required: false
15         enum:
16           - SFO
17           - LAX
18           - CLE
19     responses:
20       200:
21         body:
22           application/json:
23             type: AmericanFlight[]
```

Types and Traits Others Docs

is type description

uriParameters displayName

API summary

> Types

Resources

< /flights

GET

POST

< /{ID}

GET

DELETE

PUT

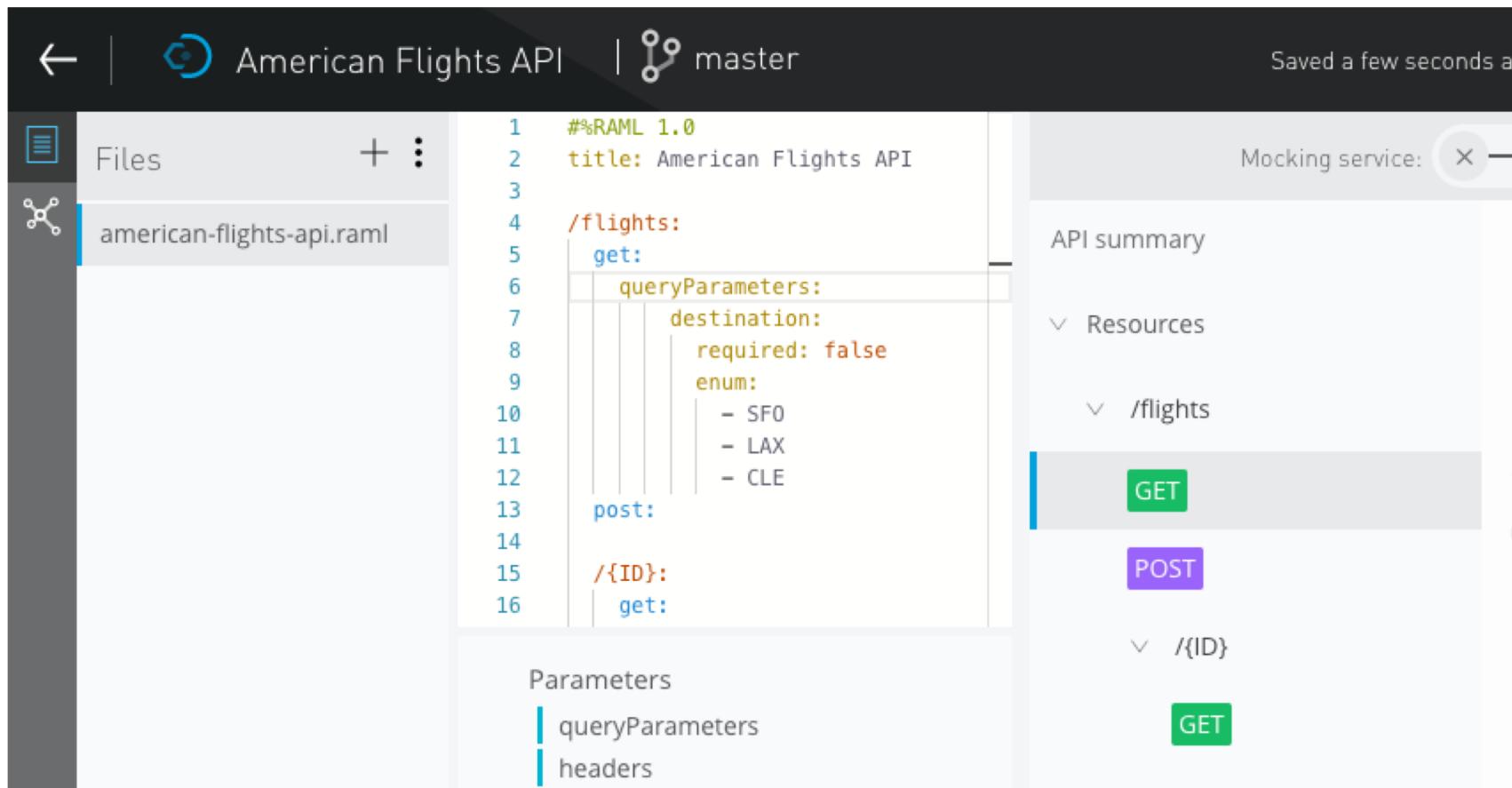
API console

Shelf

12

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



The screenshot shows the MuleSoft API Designer interface. The top navigation bar includes a back arrow, the project name "American Flights API", a branch indicator "master", and a save status "Saved a few seconds ago". The left sidebar has "Files" selected, showing the file "american-flights-api.raml". The main area displays the RAML code:

```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
5    get:
6      queryParameters:
7        destination:
8          required: false
9          enum:
10            - SFO
11            - LAX
12            - CLE
13
14  post:
15  /{ID}:
16    get:
```

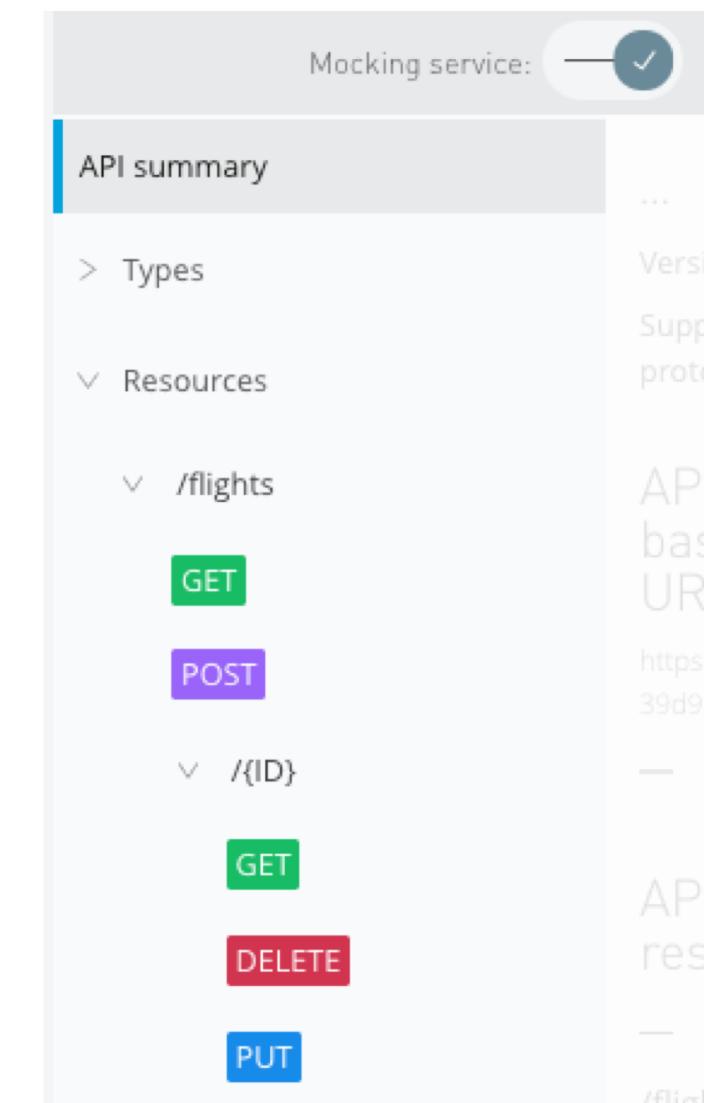
Below the code editor, there's a "Parameters" section with "queryParameters" and "headers". To the right, the "API summary" pane shows the "Resources" section, which details the "/flights" resource with a GET method and a POST method, and the("/{ID}") resource with a GET method.

Testing API design without writing code

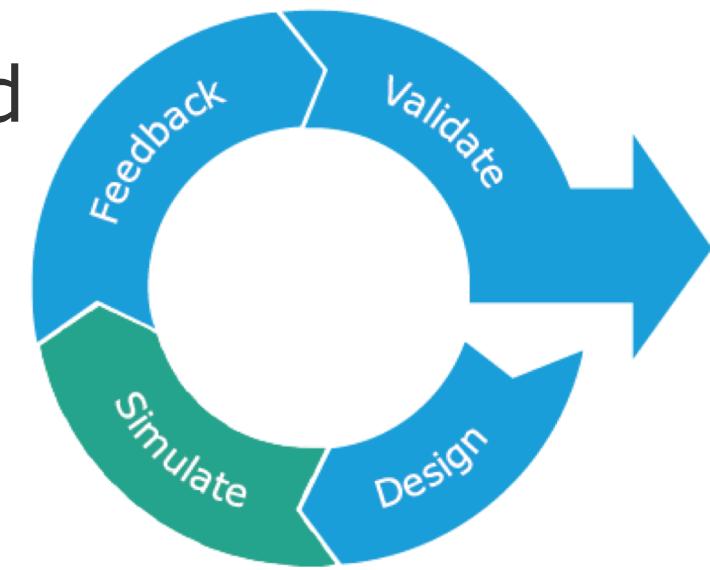


Simulating an API

- 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

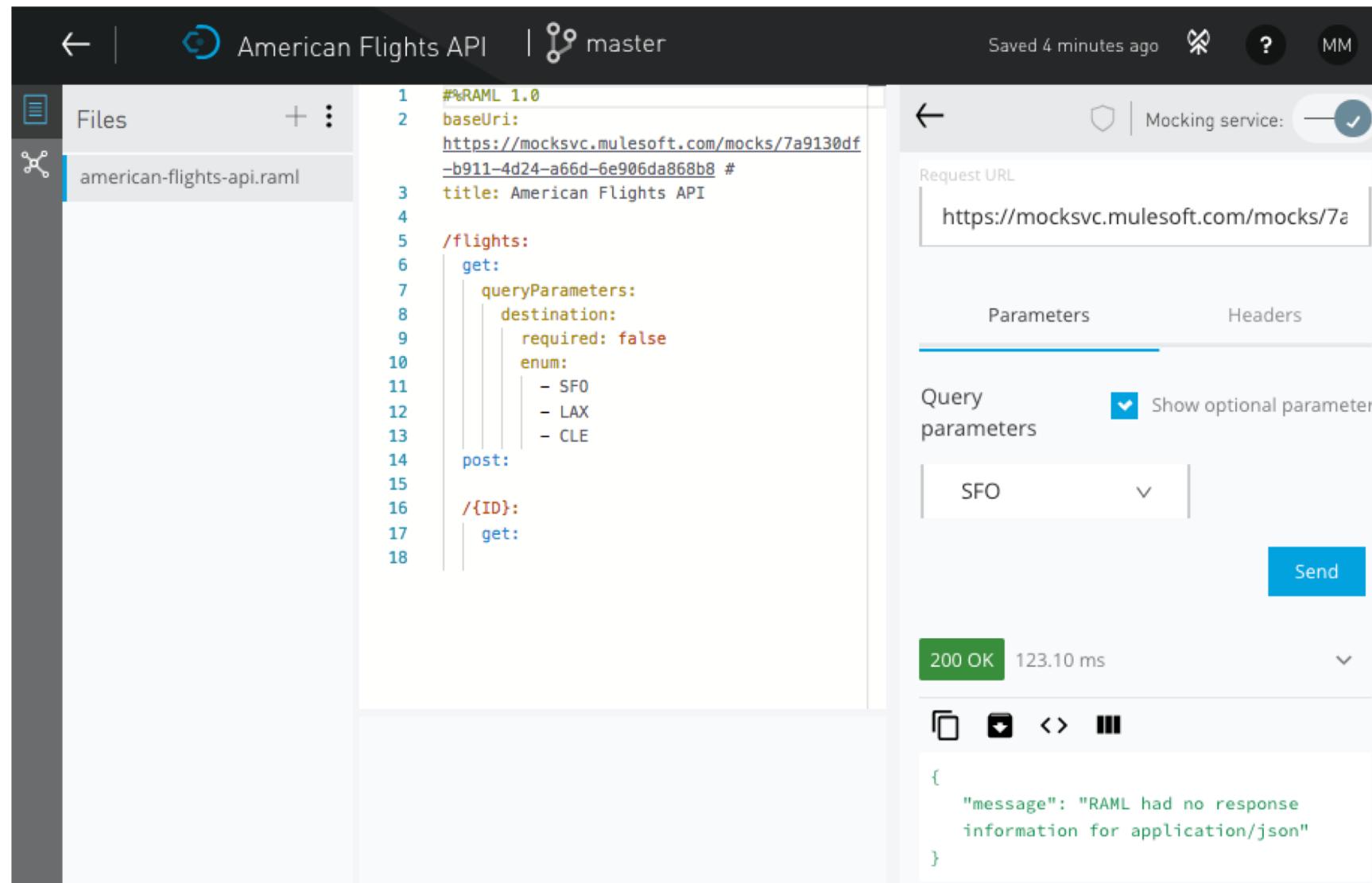


The screenshot shows the API summary page of the MuleSoft API console. At the top, there is a toggle switch labeled "Mocking service:" which is turned on. Below the header, there are sections for "API summary", "Types", "Resources", and "APIs". The "Resources" section is expanded, showing a list of endpoints under the "/flights" resource. The endpoints are: GET (for the root), POST, and another GET endpoint for("/{ID}"). To the right of these endpoints, there are columns for "AP", "base UR", "https", and "39d9". The bottom part of the screenshot is cut off.



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



The screenshot shows the MuleSoft API Mocking Service interface. On the left, a code editor displays a RAML 1.0 file named "american-flights-api.raml". The file defines a base URI and endpoints for flights and flight details. On the right, a request builder interface is shown. It has a "Request URL" field containing "https://mocksvc.mulesoft.com/mocks/7a9130df_b911-4d24-a66d-6e906da868b8 #". Below it, a "Parameters" tab is selected, showing a dropdown menu with "SFO". A "Send" button is at the bottom right. At the bottom, a response card shows "200 OK" and "123.10 ms", with a JSON message indicating no response information.

```
#%RAML 1.0
baseUri: https://mocksvc.mulesoft.com/mocks/7a9130df_b911-4d24-a66d-6e906da868b8 #
title: American Flights API

/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
    post:
      /{ID}:
        get:
```

Request URL: https://mocksvc.mulesoft.com/mocks/7a9130df_b911-4d24-a66d-6e906da868b8 #

Parameters Headers

Query parameters Show optional parameters

SFO

Send

200 OK 123.10 ms

{ "message": "RAML had no response information for application/json" }

Using RAML to define specifications for requests and responses



- 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

```
6   /flights:  
7     get:  
8       responses:  
9         200:  
10           body:  
11             application/json:  
12               example:  
13                 ID: 1  
14                 code: GQ574  
15                 price: 399  
16                 departureDate: 2016/12/20  
17                 origin: ORD  
18                 destination: SFO  
19                 emptySeats: 200  
20                 plane:  
21                   type: Boeing 747  
22                   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
```

- 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

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the project tree displays files like `AmericanFlightExample.raml`, `AmericanFlightNoIDExample.raml`, and `AmericanFlightDataType.raml`. The main editor area shows the `american-flights-api.raml` file with RAML code. The code defines a base URI, title, types, and a /flights endpoint with GET and POST methods. The GET method has query parameters for destination (enum: SFO, LAX, CLE) and responses for 200 OK. The 200 OK response body is defined as an array of flight objects. The POST method also has a similar response definition. On the right, a test result window shows a successful 200 OK response with a duration of 125.69 ms. The response body is displayed as JSON, showing two flight objects with details like ID, code, price, departure date, origin, destination, empty seats, and plane type.

```
#%RAML 1.0
baseUri: https://mocksvc.mulesoft.com/mocks/6822ede5-6246-4462-a380-6ae4a9d4e1c2 #
title: American Flights API
types:
  AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flight-data-type/1.0.1/AmericanFlightDataType.raml
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight[]
            example: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flights-example/1.0.1/AmericanFlightsExample.raml
  post:
    body:
      application/json:
        type: AmericanFlight
        example: !include
```

200 OK 125.69 ms

```
[Array[2]
  -0: {
    "ID": 1,
    "code": "ER38sd",
    "price": 400,
    "departureDate": "2017/07/26",
    "origin": "CLE",
    "destination": "SFO",
    "emptySeats": 0,
    "plane": {
      "type": "Boeing 737",
      "totalSeats": 150
    }
  },
  -1: {
    "ID": 2,
    "code": "ER45if",
    "price": 540.99,
    "departureDate": "2017/07/27",
    "origin": "SFO",
    "destination": "ORD",
    "emptySeats": 54,
    "plane": {
      "type": "Boeing 777"
    }
  }
]
```

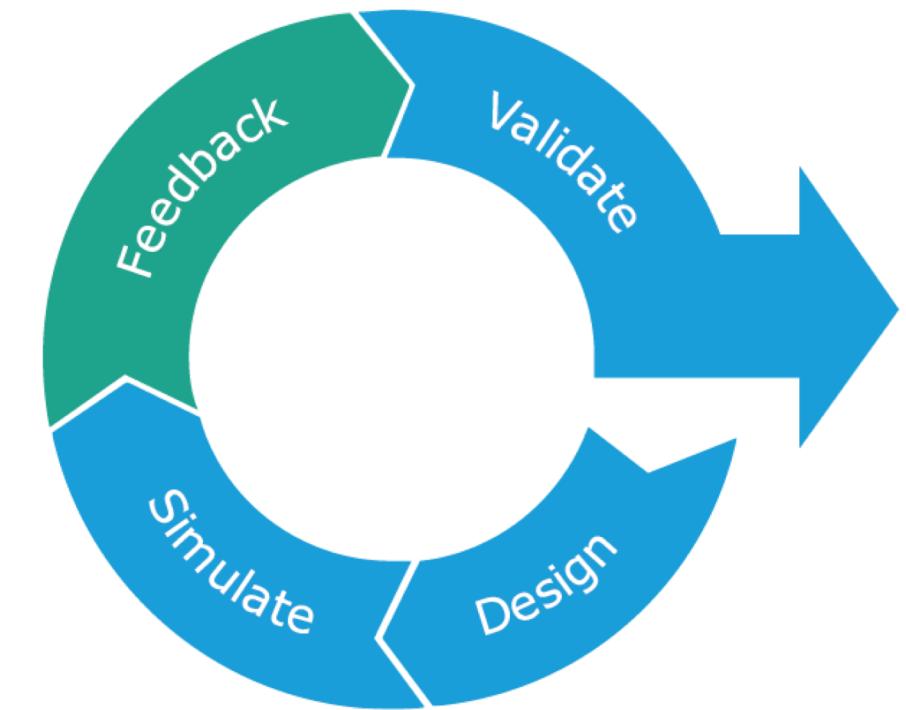
Engaging with users



Engaging users during the API design phase



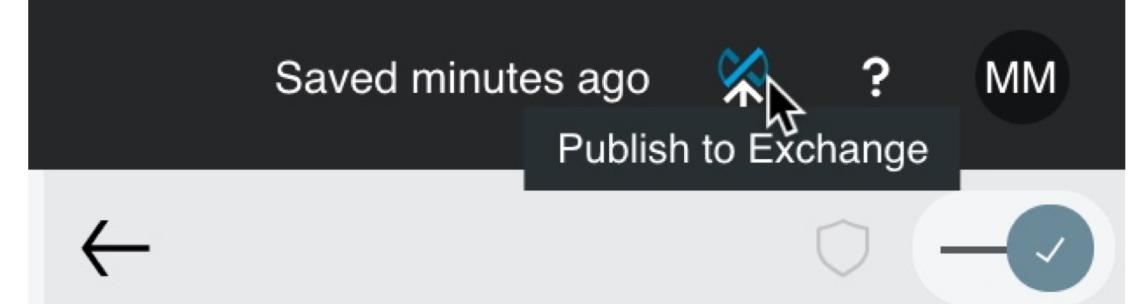
- To build a successful API, you should define it iteratively
 - Get feedback from developers on usability and functionality along the way
- To do this, you need to provide ways for developers to discover and play with the API
- Anypoint Platform makes this easy with **API portals in Exchange**
 - In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

The screenshot shows the Anypoint Exchange interface for the 'American Flights API'. The left sidebar lists the API's structure: 'Assets list', 'American Flights API', 'API summary', 'Types' (selected), 'Resources', '/flights' (selected), and 'API instances'. Under '/flights', there are 'GET' and 'POST' methods. The main content area displays the API's details: title 'American Flights API' (version v1), rating (0 reviews), and a brief description stating it's a system API for operations on the 'american' table in the 'training' database. It lists supported operations: Get all flights, Get a flight with a specific ID, Add a flight, Delete a flight, and Update a flight. The 'Overview' section shows the API is a REST API, created by 'Max Mule' on Nov 18, 2017, and is private. The 'Asset versions for v1' section shows two versions: 1.0.1 (Mocking Service) and 1.0.0. The 'Tags' section has a placeholder '+ Add a tag'.

Sharing APIs



Sharing APIs



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

A screenshot of the MuleSoft Exchange interface. At the top, there's a navigation bar with 'Assets list', 'American Flights API', 'v1', 'Share', 'Download', and 'Edit'. The main area shows the 'American Flights API' with a green circular icon containing a house symbol, a 5-star rating with '(0 reviews)', and a 'Rate and review' button. Below this is an 'Overview' section.

- 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

A screenshot of a 'Share American Flights API' dialog box. It includes a search bar for users ('Search for a user'), a dropdown for roles ('Viewer'), a 'Add' button, a list of users ('Max Mule (stallions-stgxdr) Admin'), and a 'Cancel' and 'Share' button at the bottom right.

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

A screenshot of a web browser displaying the 'Anypoint Exchange' portal at the URL https://anypoint.mulesoft.com/exchange/portals/training-100/. The page title is 'MuleSoft // Training'. The main content area features a dark background with a geometric pattern. It displays a welcome message: 'Welcome to your MuleSoft Training portal!' and a sub-message: 'Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience.' Below this, there is a search bar with 'All types' and a 'Search' button, and a grid view showing one item: 'American Flights API' by 'Max Mule'. The API icon is circled in green.

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

- 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 definitions can be a lot more complex and sophisticated than what we built here
- Training: training.mulesoft.com
 - *Anypoint Platform: API Design*
- Website: raml.org
 - Documentation
 - Tutorials
 - Full spec
 - Resources

