Mastering API Design

RAML, Fragments, Mocking & APIKit

API Design Lifecycle

Design → Simulate → Build

Topics Covered:

RAML Syntax & Fragments · Traits & Types
Mocking Service · APIKit Router Internals

Prepared for:
MuleSoft Developer
December 9, 2025

Contents

Chapter 1

RAML Fundamentals

1.1 Definition Purpose

RAML (RESTful API Modeling Language) is a YAML-based language for describing RESTful APIs.

- **Purpose:** It serves as the contract between the API provider and the consumer. It defines *what* the API does before a single line of code is written (Design-First Approach).
- **Why use it?** It is human-readable, supports modularity (reusability), and integrates natively with the Anypoint Platform to generate scaffolding.

1.2 Core Structure

A RAML file has a hierarchical structure based on indentation (spaces, not tabs).

1. **Root Section:** Metadata (Title, Version, Base URI).
2. **Resources:** The URL paths (e.g., ‘/users‘).
3. **Methods:** HTTP Verbs (GET, POST, PUT, DELETE).
4. **Parameters & Body:** Query params, URI params, JSON payloads.

1.3 Writing Your First Endpoint

Below is a standard RAML definition for a generic "System API".

```
#%RAML 1.0
title: Customer System API
version: v1
baseUri: http://localhost:8081/api
mediaType: application/json

/customers:
  get:
    description: Fetch all customers
    queryParameters:
      region:
        type: string
        required: false
        example: "NA"
    responses:
      200:
        body:
          application/json:
            example: [{"id": 1, "name": "Alice"}]

  post:
    description: Create a new customer
    body:
      application/json:
        example: {"name": "Bob", "email": "bob@test.com"}
    responses:
      201:
        body:
          application/json:
            example: {"message": "Created"}
```

Listing 1.1: simple-api.raml

Chapter 2

Fragments: Traits & Data Types

2.1 The Concept of Fragments

In a large enterprise API, writing everything in one file is unmanageable. **Fragments** allow you to externalize reusable code into separate files.

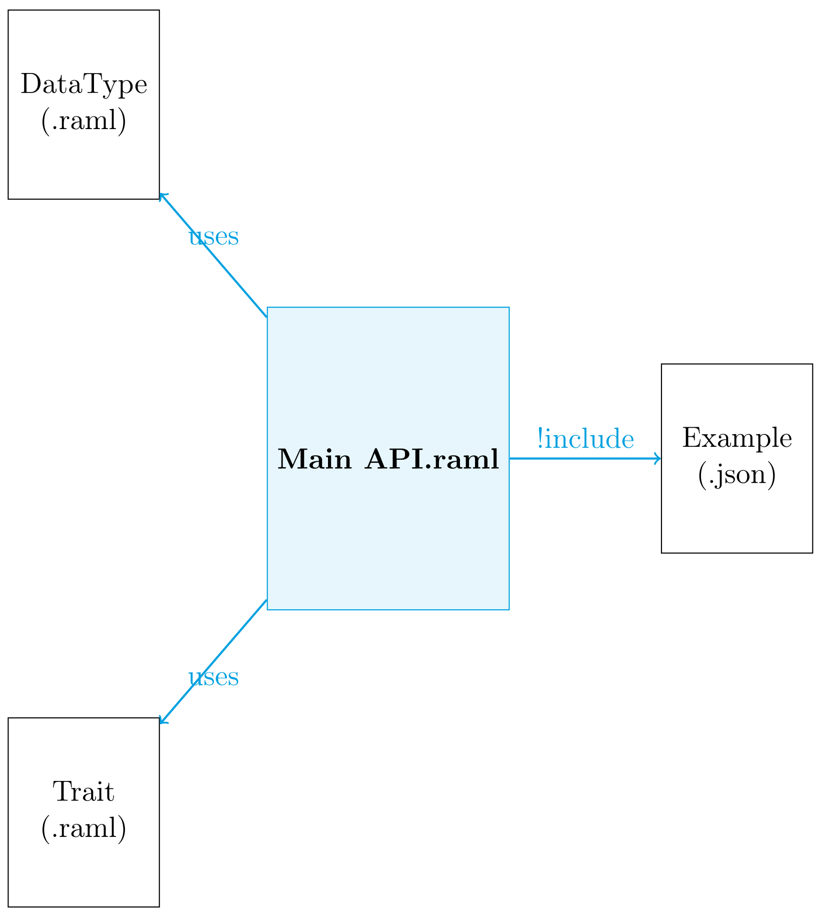

Figure 2.1: RAML Modularization Architecture

2.2 Traits (Reusable Request Logic)

Definition: A Trait defines reusable *operation* characteristics, such as Query Parameters, Headers, or Error Responses. **Use Case:** Pagination (offset/limit) or Authorization headers used across 20 different endpoints.

```
#%RAML 1.0 Trait
queryParameters:
  page:
    type: integer
    default: 1
  size:
    type: integer
    default: 10
```

Listing 2.1: traits/pagination.raml

Usage in Main RAML:

```
traits:
  pageable: !include traits/pagination.raml

/products:
  get:
    is: [pageable]  # Applies the query params here
```

2.3 Data Types (Reusable Body Logic)

Definition: Defines the schema/structure of the JSON object. **Use Case:** A "Customer" object is used in GET (response) and POST (request).

```
#%RAML 1.0 DataType
type: object
properties:
  id:
    type: integer
    required: false
  name: string
  email: string
```

Listing 2.2: types/Customer.raml

Chapter 3

Mocking Service

3.1 What is it?

The Mocking Service simulates the API behavior before the actual implementation is built. It intercepts requests sent to the RAML definition and returns the ‘example‘ data defined in the file.

3.2 Why use it?

- **Parallel Development:** Frontend teams can build their UI using the mock URL while Backend teams build the implementation.
- **Feedback:** Stakeholders can test the API design and provide feedback on the JSON structure early.

3.3 How to use in Design Center

1. Open Anypoint Design Center.
2. Create a RAML Specification.
3. Ensure your responses have ‘example:‘ defined.
4. Toggle the **Mocking Service** switch on the top right.
5. It generates a temporary public URL (e.g., ‘https://mocksvc.../api/customers‘).

Chapter 4

APIKit Router

4.1 Definition

The **APIKit Router** is a specialized Mule component that acts as the "Traffic Cop" of your application. It serves as the bridge between the Interface (RAML) and the Implementation (Flows).

4.2 How it works Internally

When a request hits the Mule Runtime:

1. **Validation:** APIKit validates the incoming request (Headers, Body, Query Params) against the RAML contract.
2. **Routing:** It looks at the HTTP Method and Resource Path (e.g., ‘GET /customers‘).
3. **Dispatch:** It routes the message to a specific flow named ‘get::api-config‘.
4. **Serialization:** It ensures the response matches the generated output format.

4.3 Architecture Diagram

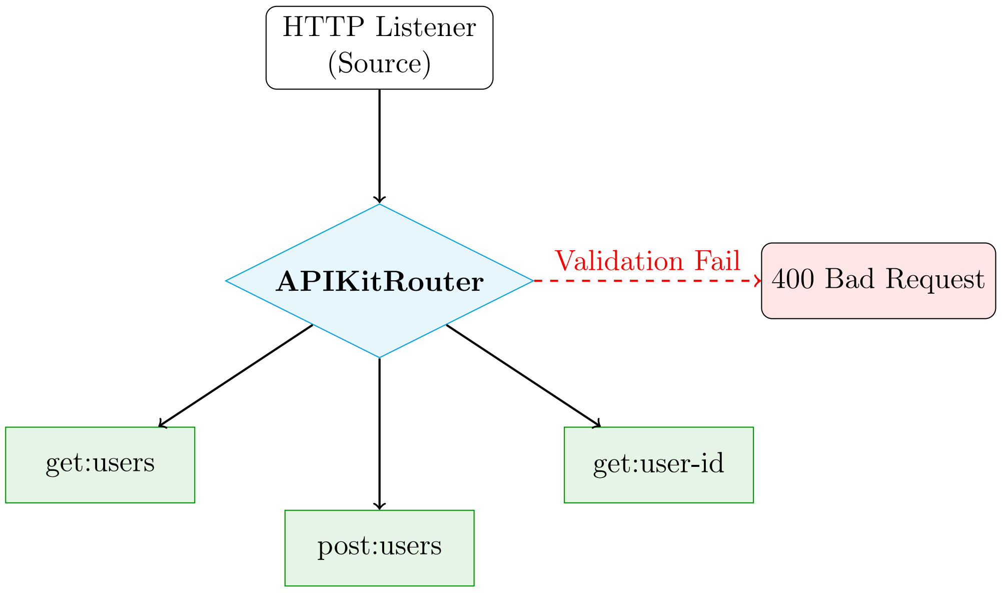

Figure 4.1: APIKit Routing Mechanism

4.4 APIKit Error Handling

The Router automatically generates global error handlers for:

- **APIKIT:BAD_REQUEST (400):** Schema validation failed (e.g., missing required field).
- **APIKIT:NOT_FOUND (404):** The resource path doesn't exist.
- **APIKIT:METHOD_NOT_ALLOWED (405):** Calling POST on a GET-only endpoint.
- **APIKIT:NOT_ACCEPTABLE (406):** Invalid Accept header.
- **APIKIT:UNSUPPORTED_MEDIA_TYPE (415):** Sending XML when JSON is expected.

4.5 Configuration in Mule XML

When you import RAML into Anypoint Studio, this is generated automatically (Scaffolding).

```
<apikit:config name="my-api-config" api="resource::.../my-api.zip" >
    <apikit:flow-mappings>
        </apikit:flow-mappings>
</apikit:config>

<flow name="main-api-flow">
    <http:listener path="/api/*" config-ref="HTTP_Config"/>

    <apikit:router config-ref="my-api-config" />

    <error-handler>
        <on-error-propagate type="APIKIT:BAD_REQUEST">
            <ee:transform>
```

```
                <ee:message>
                    <ee:set-payload>{"message": "Invalid Data"}</ee:set-payload>
                </ee:message>
            </ee:transform>
        </on-error-propagate>
    </error-handler>
</flow>
```

Listing 4.1: apikit-config.xml

Chapter 5

Interview Guide

5.1 Common Questions

5.1.1 Q1: What is the difference between a Trait and a Resource Type?

Answer:

- **Trait:** Encapsulates *auxiliary* details (QueryParams, Headers). "Is-a" relationship (This method *is* pageable).
- **Resource Type:** Encapsulates the *entire structure* of a resource (Methods GET/POST and their standard responses). "Has-a" relationship (This resource *has* a standard collection behavior).

5.1.2 Q2: How does APIKit Router validate requests?

Answer: It checks the incoming HTTP request against the RAML definition. If a field marked ‘required: true‘ in RAML is missing in the JSON body, the Router throws ‘APIKIT:BAD$_R EQUEST‘beforethef$

5.1.3 Q3: Can we modify the flows generated by APIKit?

Answer: Yes. You implement the logic inside the flows (e.g., ‘get::config‘). However, you should generally **not** modify the main flow containing the Router and Error Handler, as regenerating the API from Design Center might overwrite manual changes in the main scaffolding logic.

5.2 Mini Project: Employee API

Task

Create a RAML for ‘/employees‘.

1. Create a **DataType** ‘Employee.raml‘ (id, name, role).
2. Create a **Trait** ‘Traceable.raml‘ (header: ‘x-correlation-id‘).
3. Define ‘GET /employees‘ using the Trait.
4. Define ‘POST /employees‘ using the DataType.
5. Import into Studio and observe the 4 generated flows.

5.3 Summary

- **RAML** is the contract. It allows for Design-First development.
- **Fragments** (Traits/Types) prevent code duplication.
- **Mocking Service** allows testing before coding.
- **APIKit Router** automates validation, routing, and error mapping, saving developers hours of manual coding.