API Designing

Ruchi Saini

Section 1 - Introduction

Introduction

Ruchi (Instructor - Corporate Trainer)

Freelancer

MuleESB and Spring Instructor

About

Around 18 years of Experience in IT industry.

Instructor in Following technologies :-

Mule ESB
Mule Basics, Using REST,SOAP,JMS,Database,Dataweave,Batch Processing,Error Handling,MUnit Testing,API Design,API Management etc,

Spring -
Core Spring, Spring Boot , Microservices

Previously Worked in HCL Technologies.

Article Contributions under following link-
---
https://www.mindsmapped.com/category/integration-platforms/
1.Introduction to Anypoint Platform and Mule ESB - part 1
2.Introduction to Anypoint Platform and Mule ESB - part 2
3.Transaction Management with Mule ESB

Awards and Recognition

1.O2 Award'11 For being rated Outstanding Performer consecutively in 2010-11
2.O2 Award'10 For being rated Outstanding Performer consecutively in 2009-10
3.Excellence Award by GE in 2009
4.Team Player Award by HCL in 2007
5.Appreciation Award by BEA for Technical Abilities in Feb'2005
6.Excellence Award by BEA for Technical Abilities in January'2004
7.Team of Quarter Award by HCL in May'2004
8.Excellence Award by HCL Technologies for Technical Abilities in July'2003

TOC

- **Introduction**
  - Introduction to API
  - Introduction to REST and RAML
  - Introduction to API Designer
- **Design API**
  - Create RAML API Specs in API Designer
  - Define Resources,Methods,Query Parameters,Headers
  - Define Requests and Responses
  - Use HTTP Status codes for responses
  - Define Datatypes for request/response bodies
  - Create Datatype fragments
  - Create Example fragments

TOC

- **Testing :**
  - Test using Mocking service
- **Documentation:**
  - Create Documentation Fragments
  - Add Description and DisplayNames
- **Reusability and Modularization**
  - ResourceTypes
  - Traits
  - Libraries

TOC

TOC

- **Version APIs**
  - Publish New Versions
  - Create Branches for API versions
  - Deprecate Versions

Introduction to API

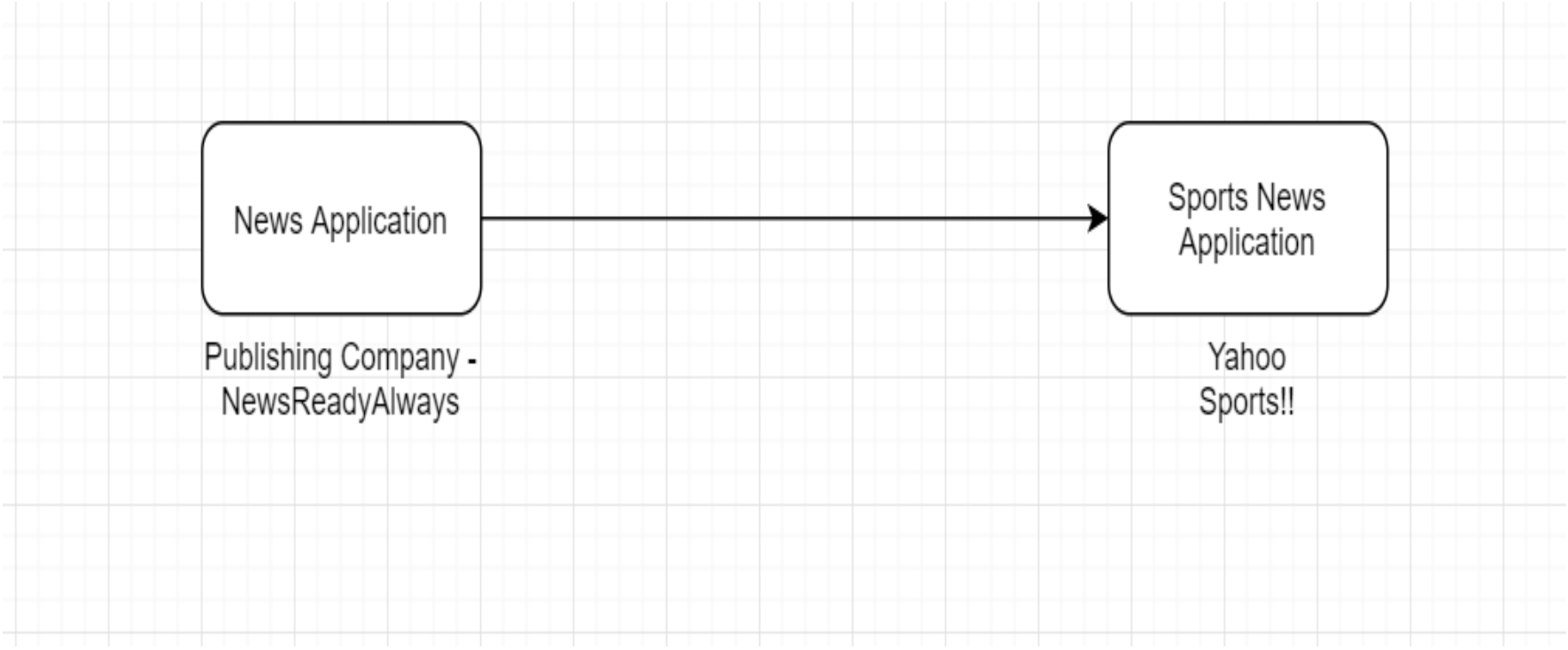
Ruchi Saini

Introduction to API

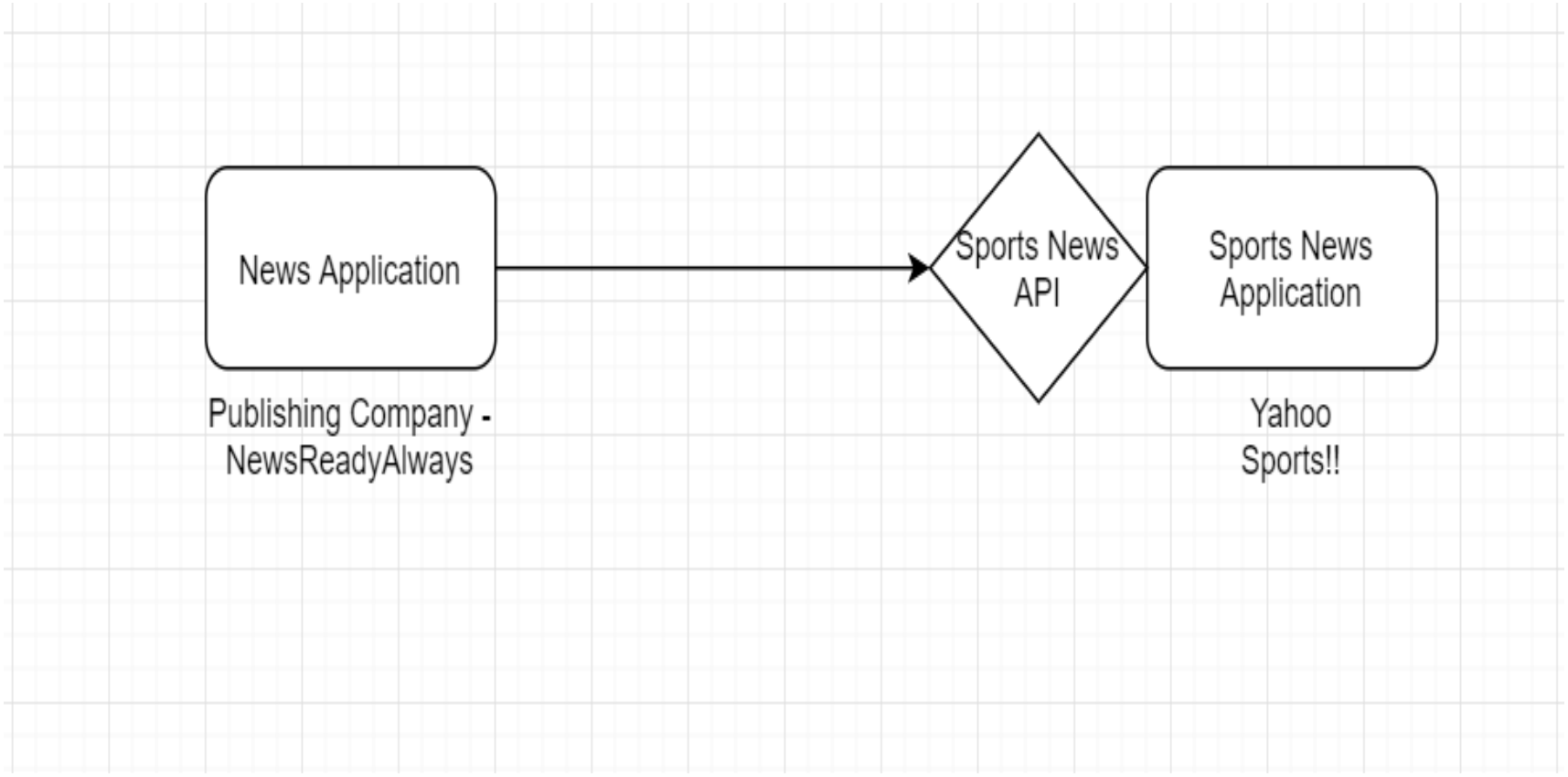
- API(Application programming interface) is an interface of your software application for external world.

- An external application uses your API to send requests to your application and get responses from your application.

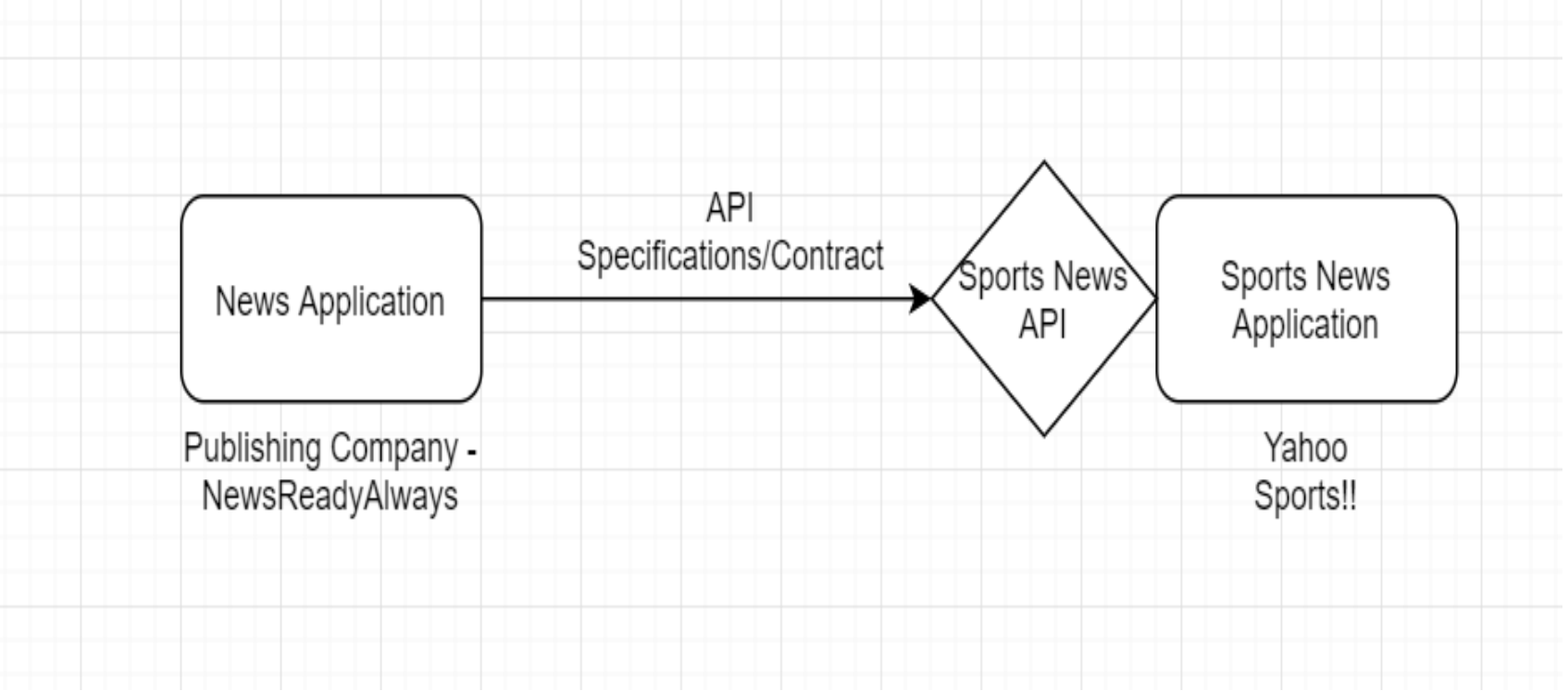
Introduction to API



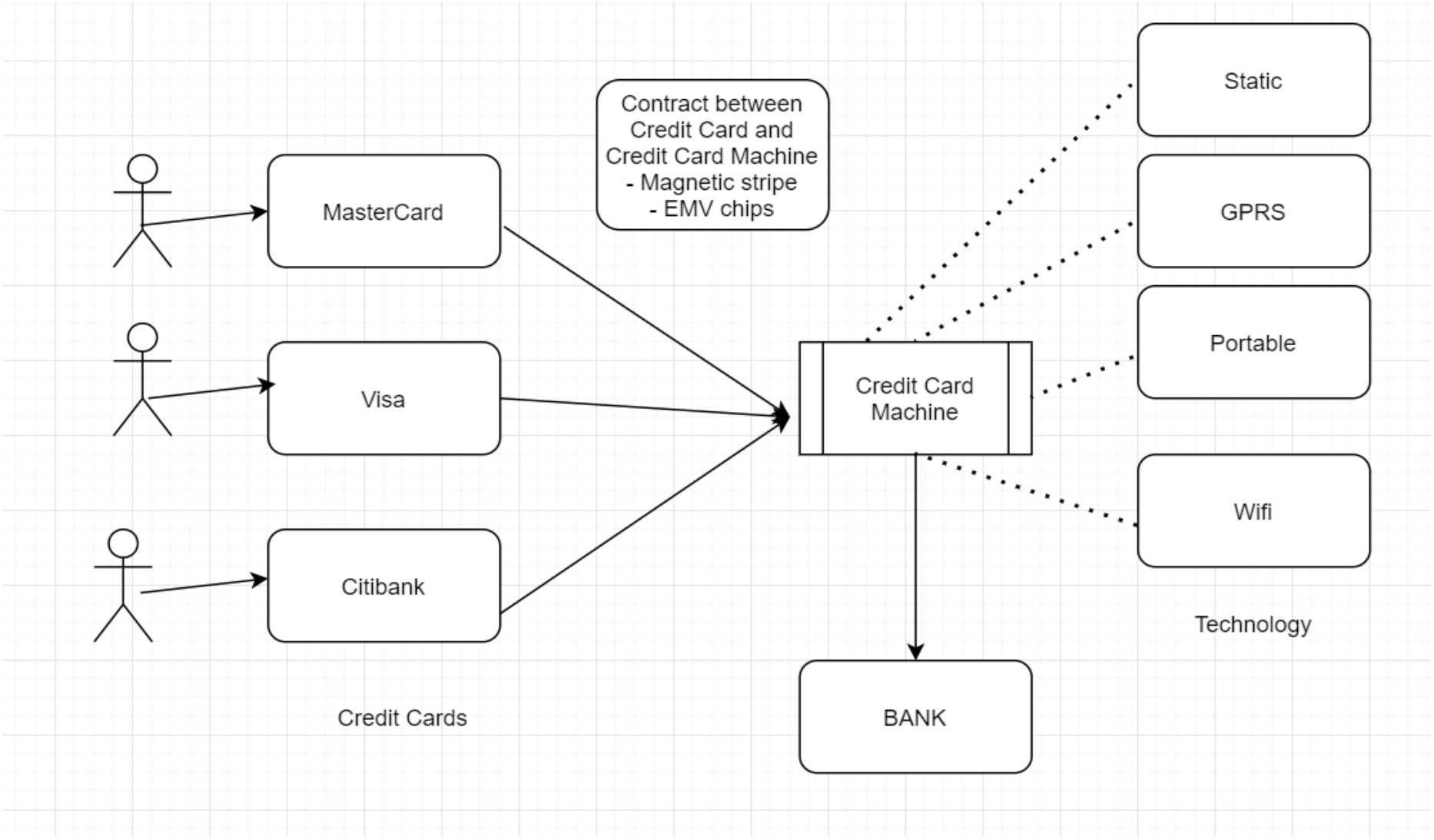
Introduction to API



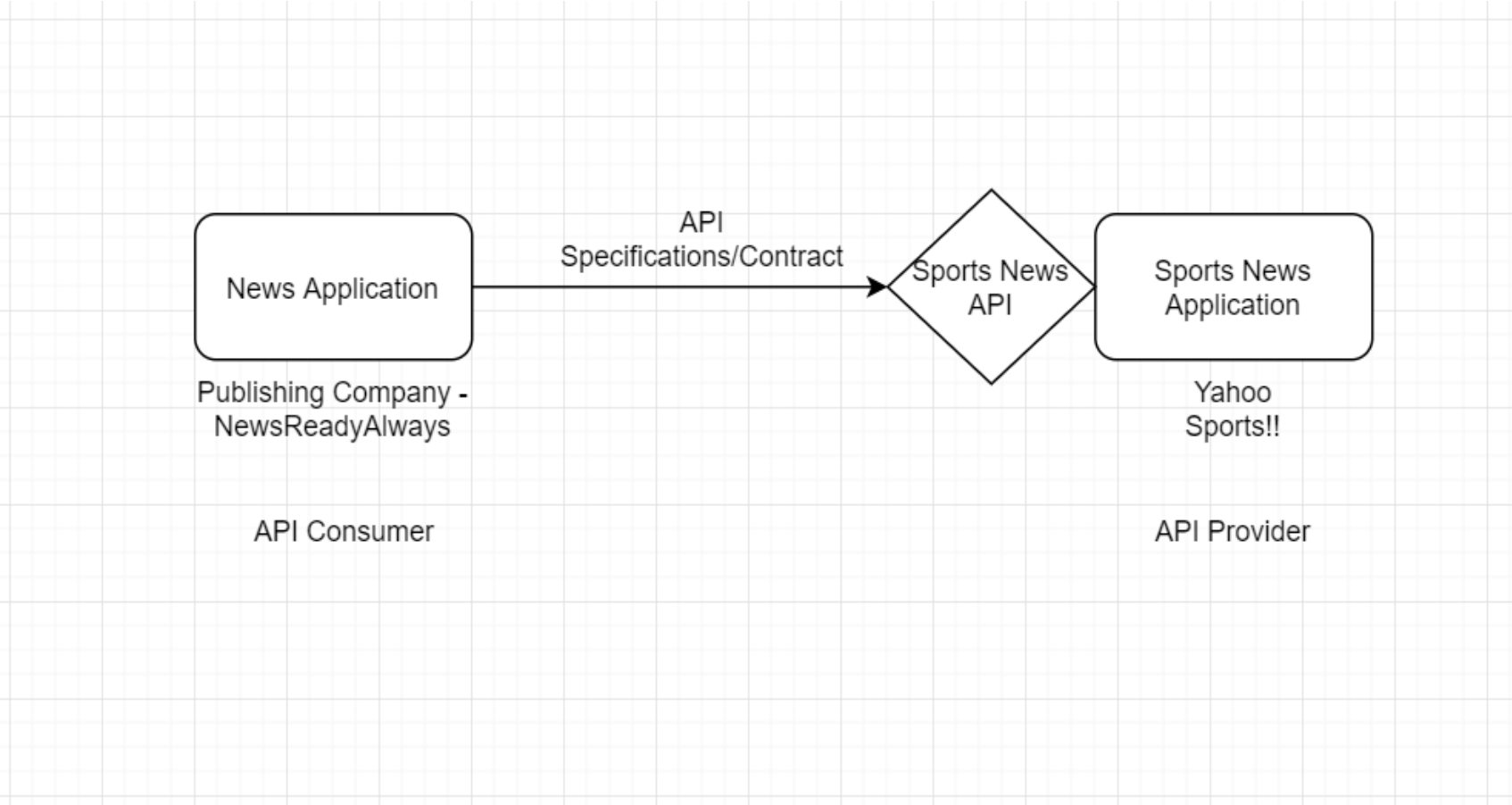
Introduction to API



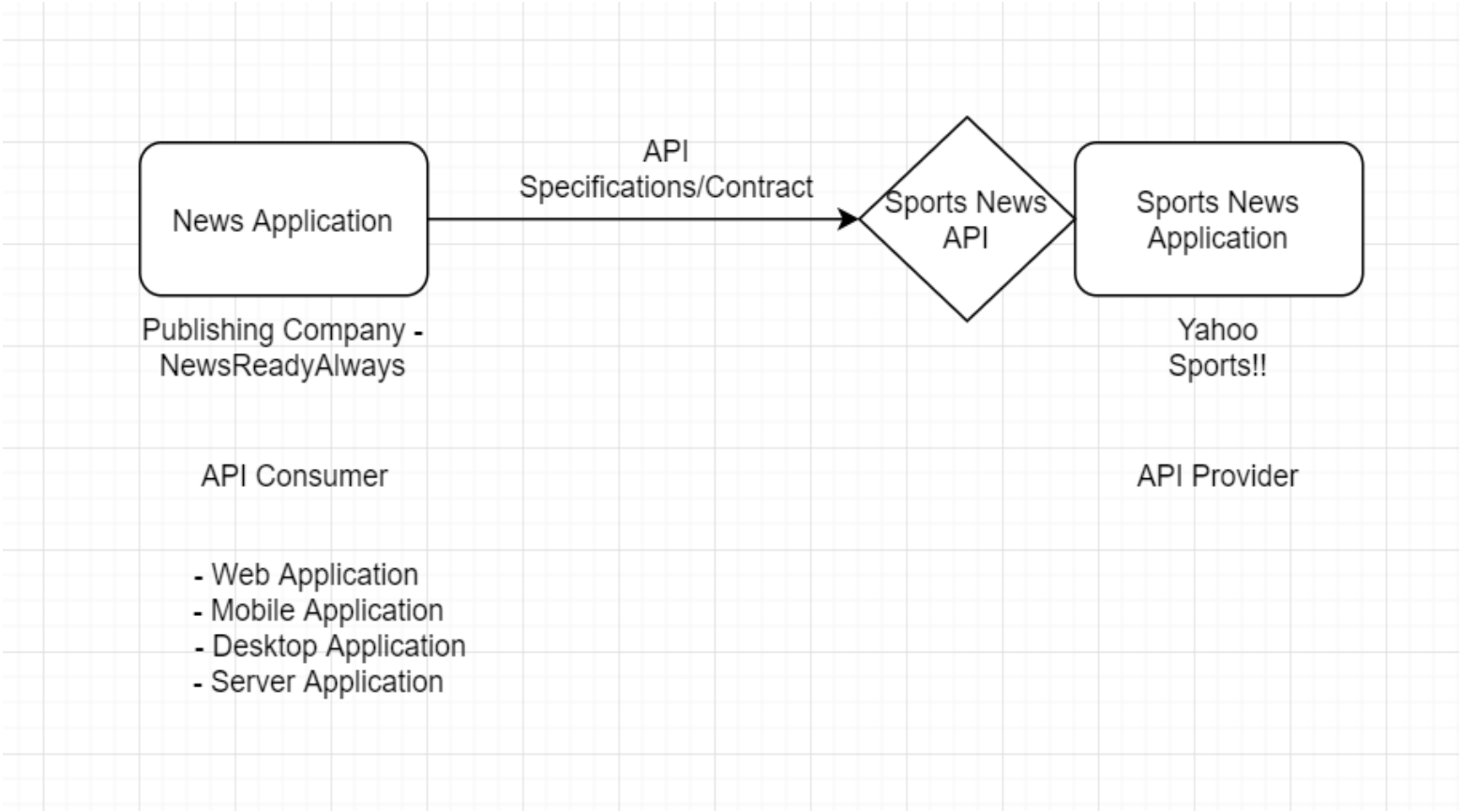
Real world Analogy



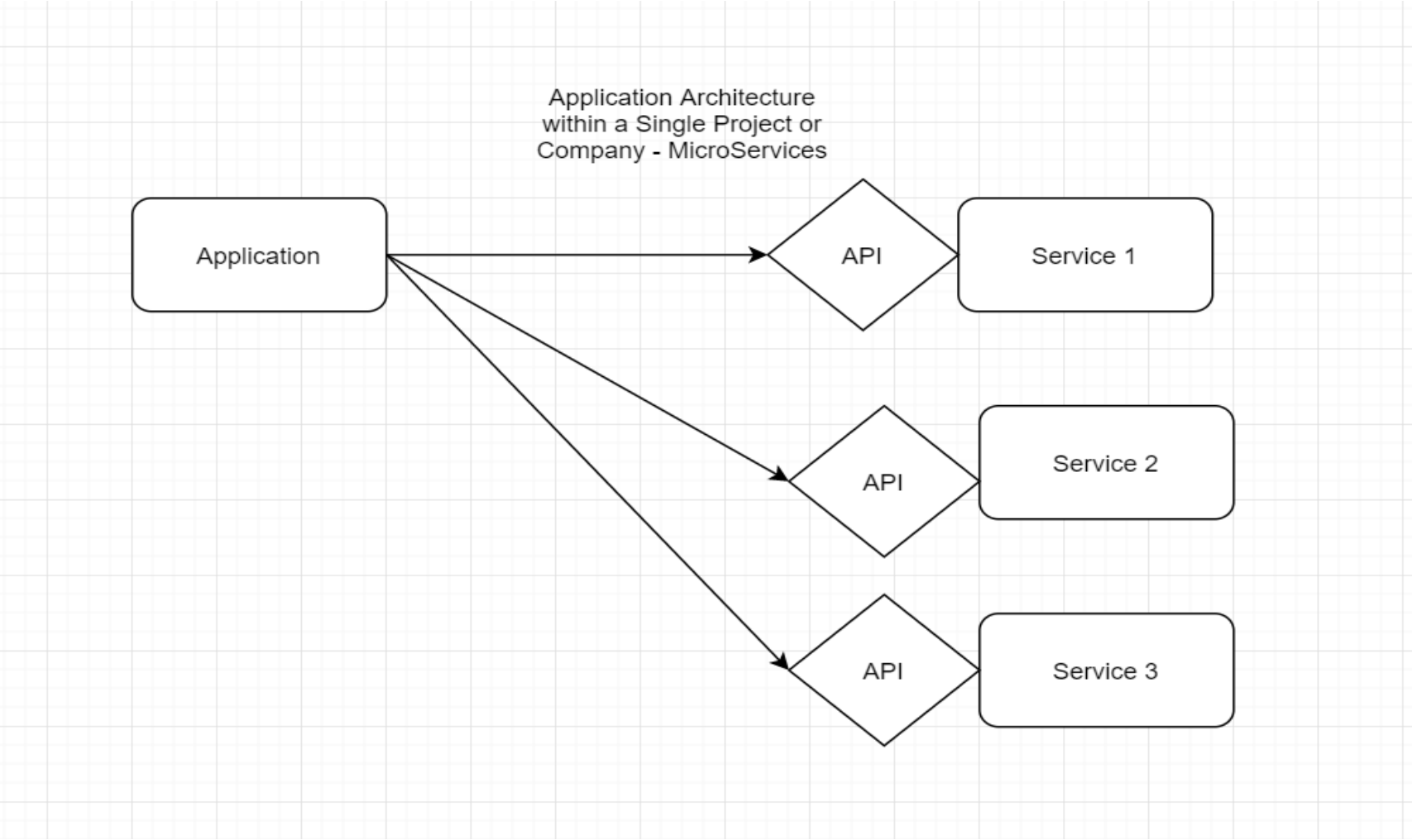
API Consumer and Provider



Different Types of Consumers



MicroServices

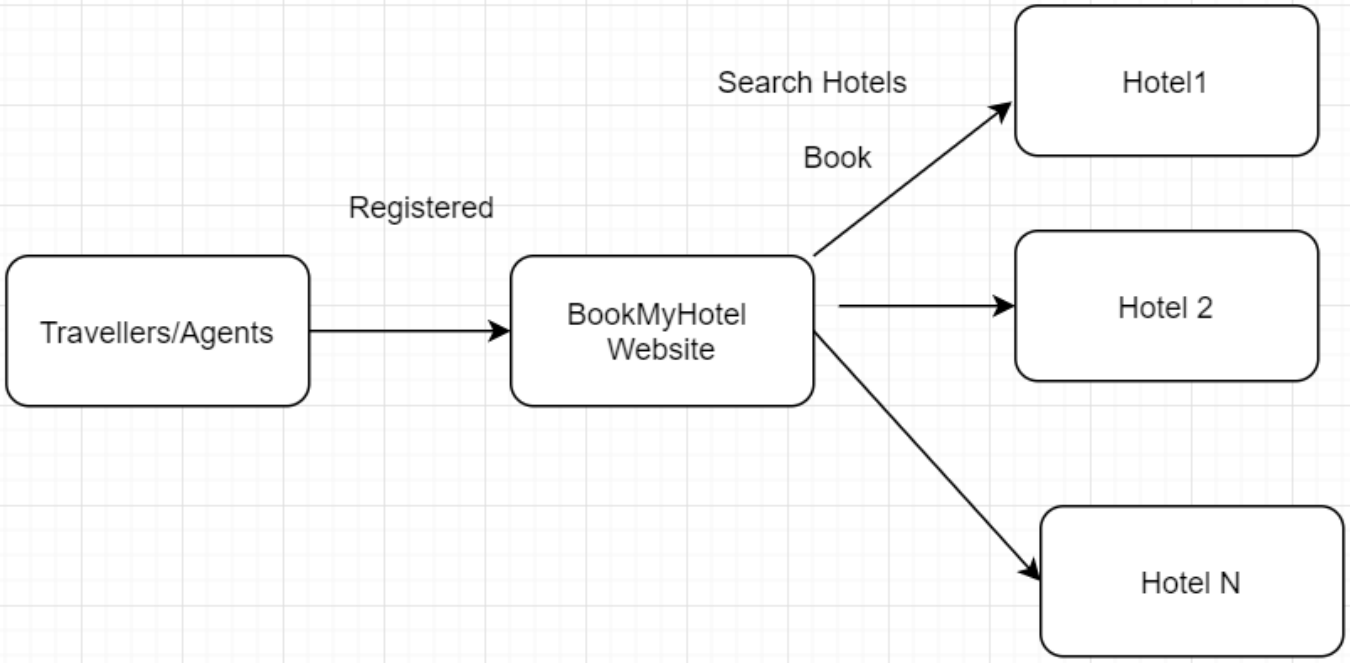


Use Case

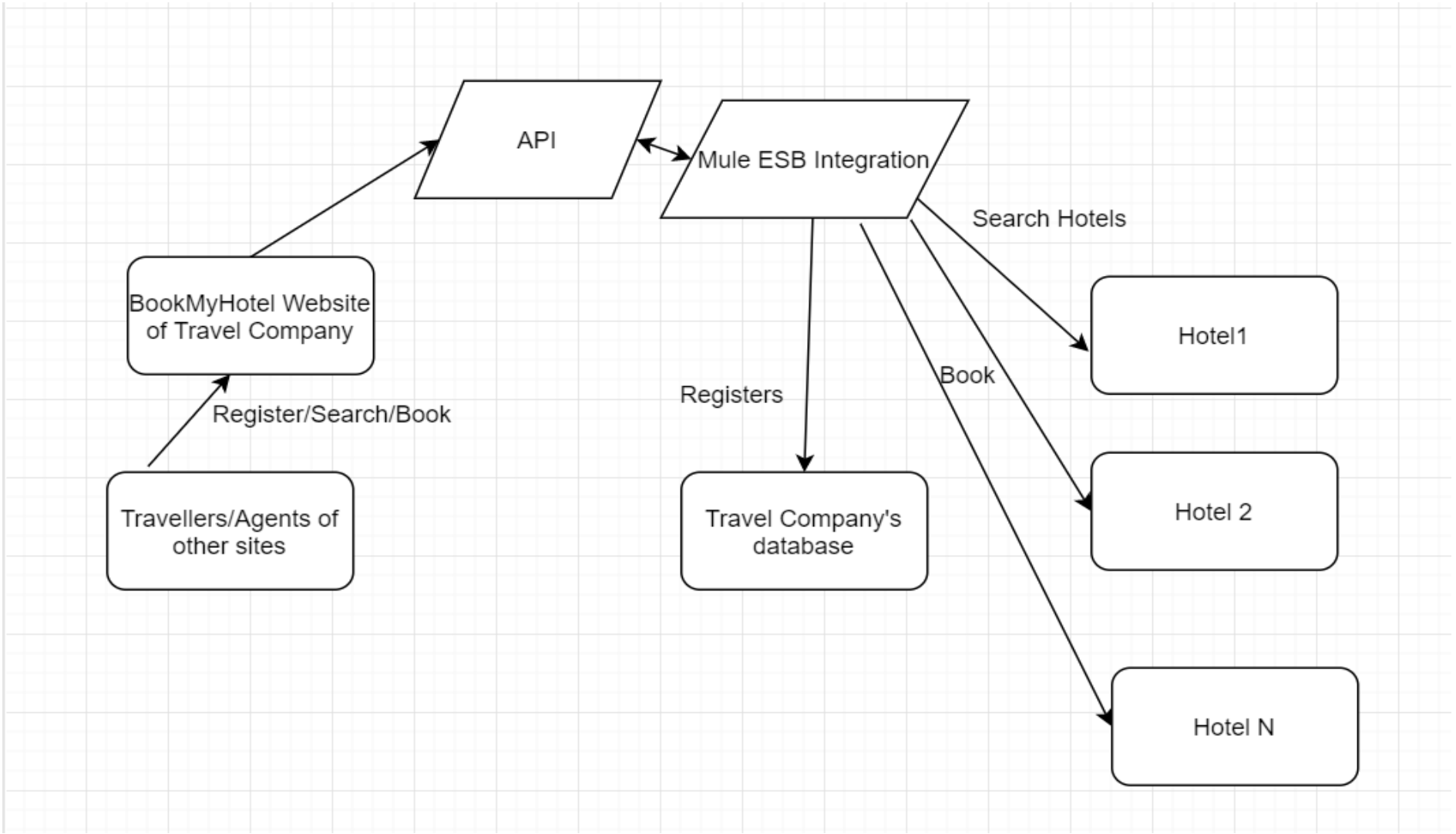
Use Case - BookMyHotel

- A Travel Company wants to create BookMyHotel website
- This website will be used to search for hotels and book the hotel.
- Search of Hotels are done using City/State/Country
- MuleESB integration application will be used to connect the website and different hotels.

Use Case - BookMyHotel



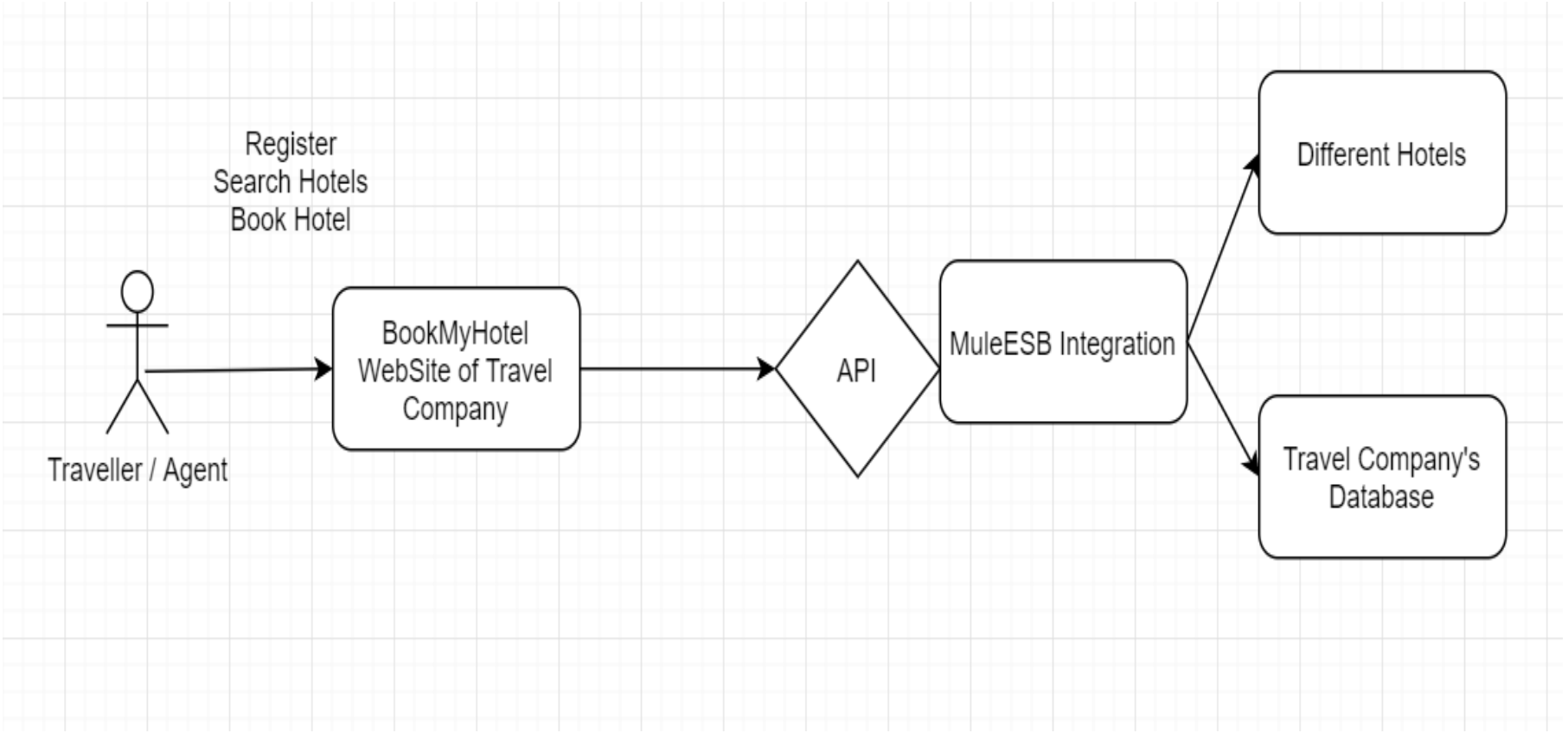
Use Case - BookMyHotel



Introduction to REST/RAML

Ruchi Saini

UseCase


Register
Search Hotels
Book Hotel
Traveller / Agent
BookMyHotel
WebSite of Travel
Company
API
MuleESB Integration
Different Hotels
Travel Company's
Database

Different Types of APIs

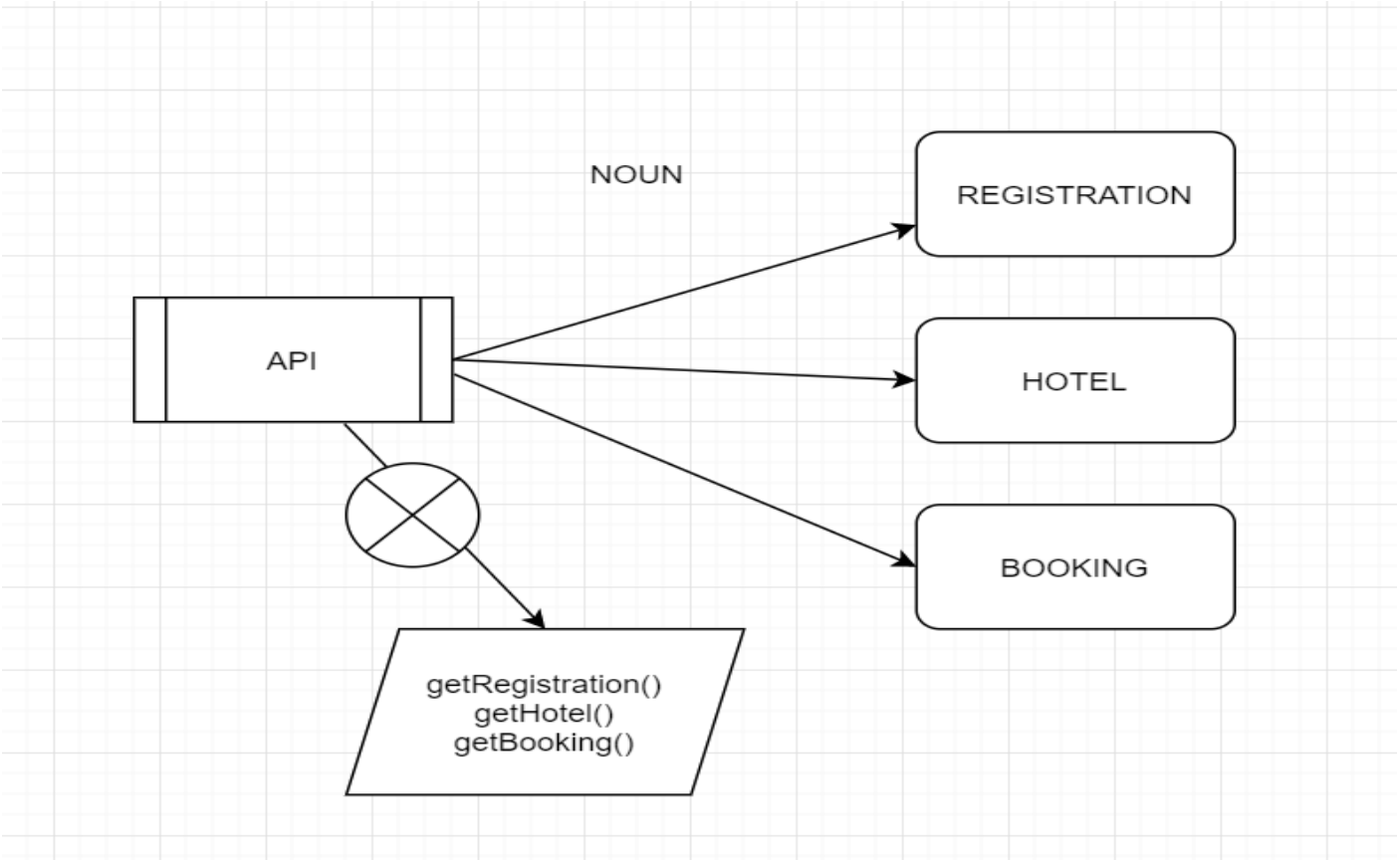
- SOAP APIs
- RPC APIs
- REST APIs

Design Principles of REST APIs

- HTTP as Application Protocol
- Resource Based
- Representation
- Uniform Interface
- Stateless

Resources

- Nouns
- Resource based rather than Action Based



Resources

- Collection Resources
  - Registrations
  - Hotels
  - Bookings
- Nested Resources
  - Registration
  - Hotel
  - Booking

Uniform Interface

- Specify actions in terms of HTTP Methods ->
  - GET
  - POST
  - DELETE
  - PATCH
  - PUT
- Responses as -
  - HTTP response -> status and body

Representation

- Representations exchanged between consumer and provider
- Resource representation can be in different formats – XML,JSON etc
- Resource – Hotel, Method – GET, Representation – Hotel details in json format

```
{
"hotelId": "h2",
"hotelName": "Imperial Hotel",
"rating": "5 star",
"price": 50000,
"food": [
 "veg",
 "non-veg"
],
"languagesSpoken": [
 "english",
 "hindi"
]
}
```

- Supported Representation/media type specified in API design
- Consumer uses Accept Header and Provider sends accordingly

Resource URIs

- Resources are identified by URIs
- Collection Resources
  - Registrations
    - http://bookmyhotel.com/registrations
- Nested Resources
  - Registration
  - http://bookmyhotel.com/registrations/1

Stateless

- No state is maintained
- Requests are independent
- Any server machine can handle any request

Map Actions to HTTP Methods

Resource	Action	Method	Resource Path
REGISTRATIONS	Get a list of registrations	GET	/registrations
REGISTRATIONS	Register a traveller	POST	/registrations
REGISTRATION	Get a specific registration	GET	/registrations/{registrationId}
REGISTRATION	Delete a specific registration	DELETE	/registrations/{registrationId}
REGISTRATION	Update specific registration	PATCH	/registrations/{registrationId}

Map Actions to HTTP Methods

Resource	Action	Method	Resource Path
HOTELS	Get a list of hotels	GET	/hotels
HOTELS	Register a hotel	POST	/hotels
HOTEL	Get a specific hotel	GET	/hotels/{hotelId}
HOTEL	Delete a specific hotel	DELETE	/hotels/{hotelId}
HOTEL	Update specific hotel	PATCH	/hotels/{hotelId}
BOOKINGS	Get a list of bookings for a hotel	GET	/hotels/{hotelId}/bookings

Map Actions to HTTP Methods

Resource	Action	Method	Resource Path
BOOKINGS	Get a list of bookings	GET	/bookings
BOOKINGS	Add a new booking	POST	/bookings
BOOKING	Get a specific booking	GET	/bookings/{bookingId}
BOOKING	Delete a specific booking	DELETE	/bookings/{bookingId}
BOOKING	Update specific booking	PATCH	/bookings/{bookingId}

RAML

- API is designed and described in API Specifications
- API Specifications/Contract of REST API -> RAML, Open API
- RAML – RESTful API Modeling Language
- RAML is built on top of YAML
- Like a plain text , easy to learn and consume

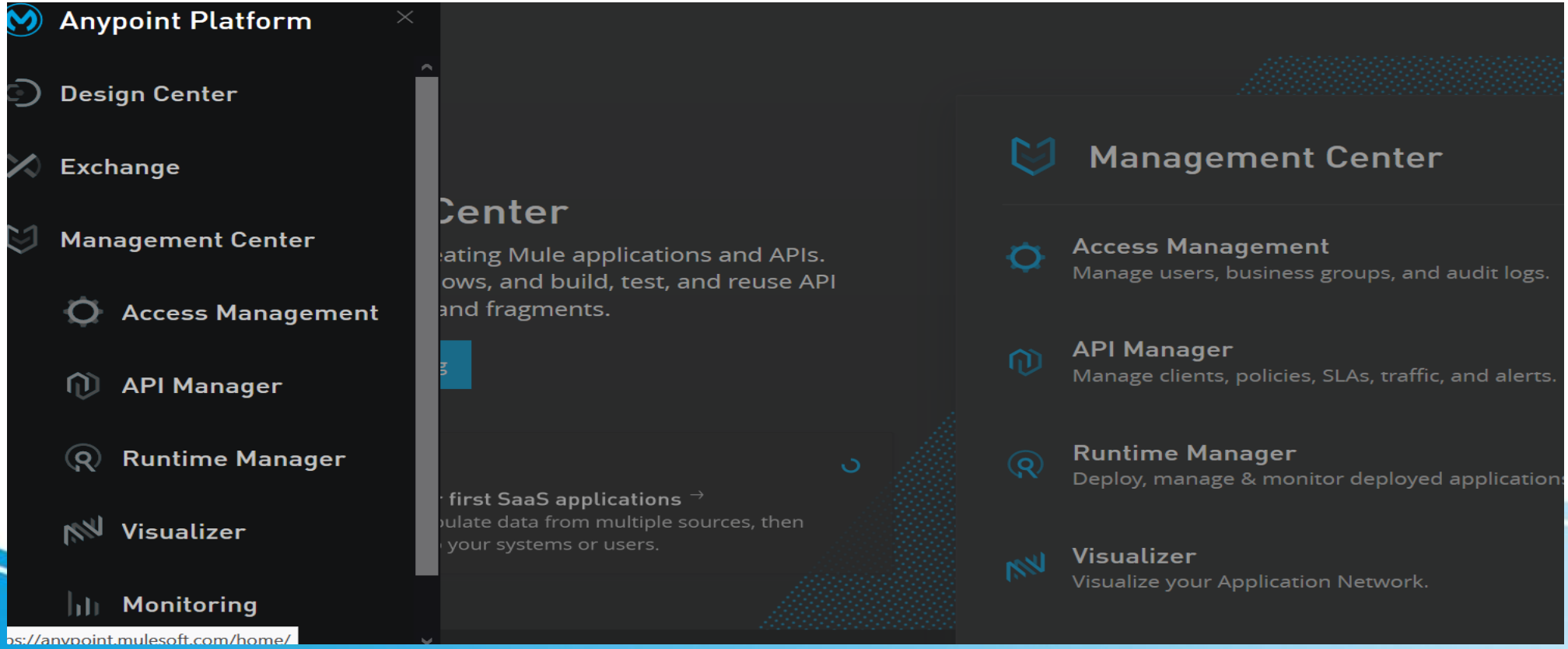
YAML Example :-

```
databasename:  muletrainingdb
username: root
password: root
tables:
   - customer
   - bank
   - transactions
```

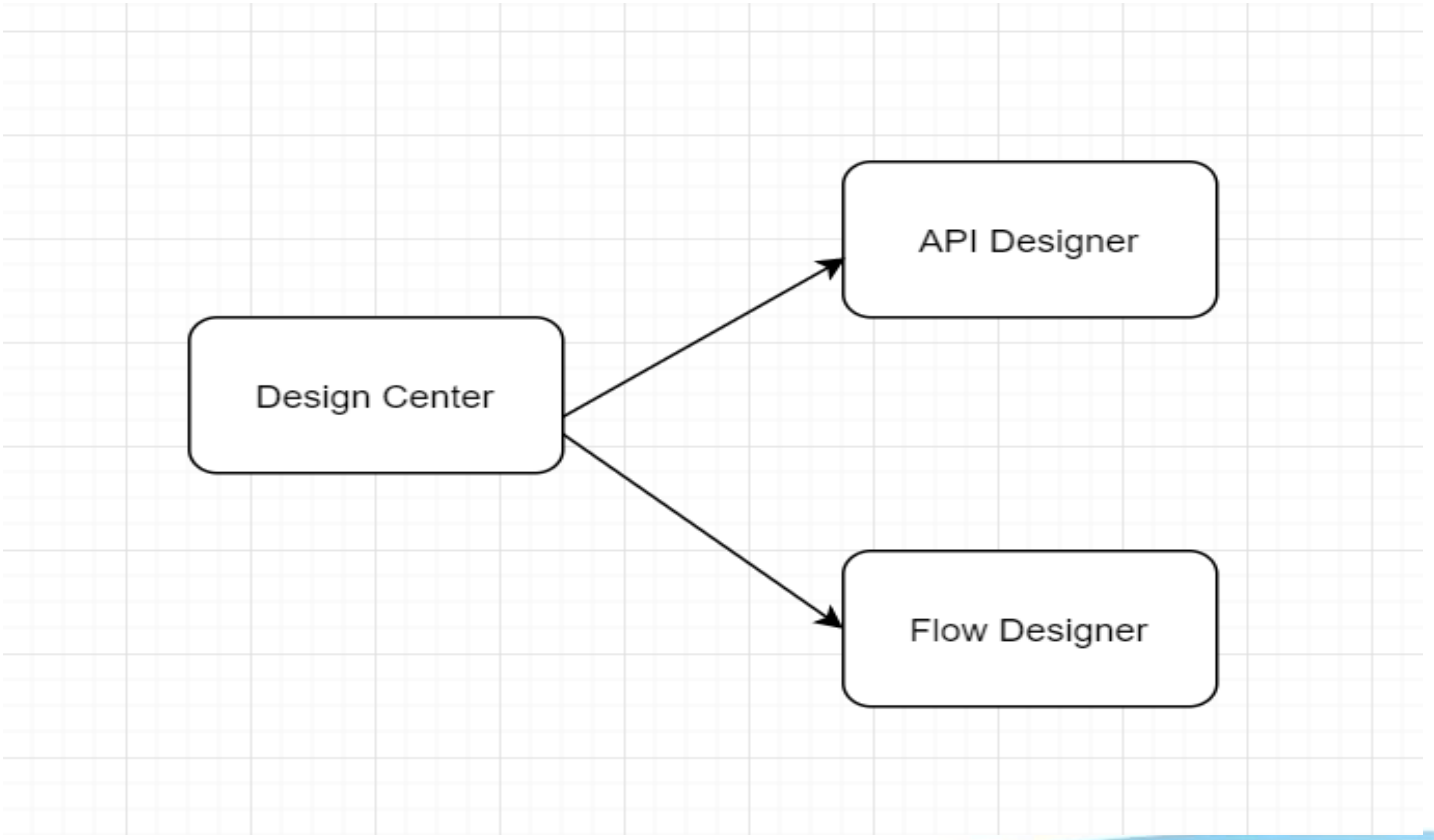
API Designer

Anypoint Platform

- Login to Anypoint Platform
  - https://anypoint.mulesoft.com/login/



Design Center



API Designer

- Create API Specifications with RAML
- Reusability
- Documentation
- Validation
- Exchange
- User Feedback

Demo

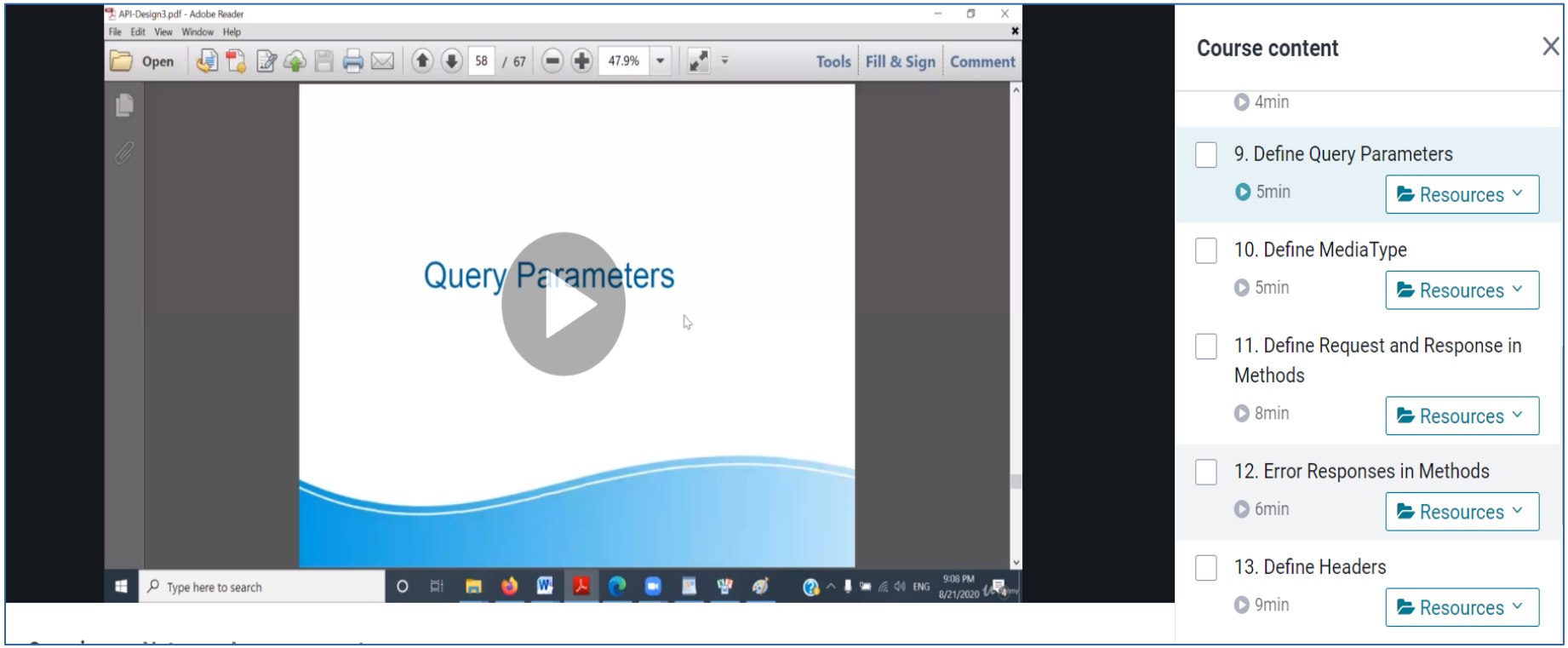
- Login to Anypoint Platform
  - https://anypoint.mulesoft.com/login/

API Specification Project Zip

Ruchi Saini

API Specification Project Zip

- Project Zip Attached in the resources section



Resources

Resources

- Collection Resources
  - Registrations
  - Hotels
  - Bookings
- Nested Resources
  - Registration
  - Hotel
  - Booking

Resources

- Resources are identified by URIs
- BaseUri : http://bookmyhotel.com

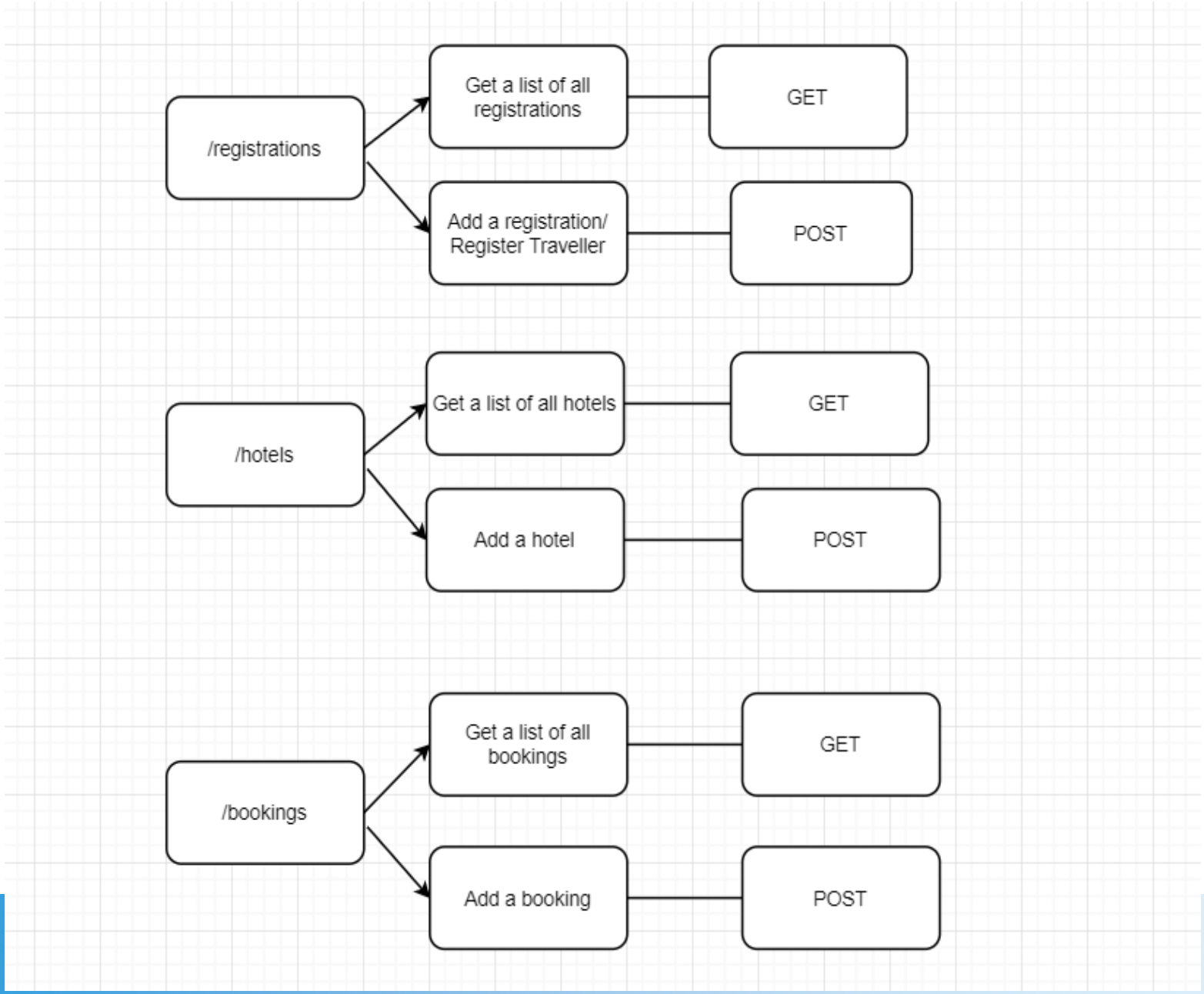
Resource	**Resource Path**
Registrations	/registrations
Hotels	/hotels
Bookings	/bookings
Registration	/registrations/{registrationId}
Hotel	/hotels/{hotelId}
Booking	/bookings/{bookingId}

Demo

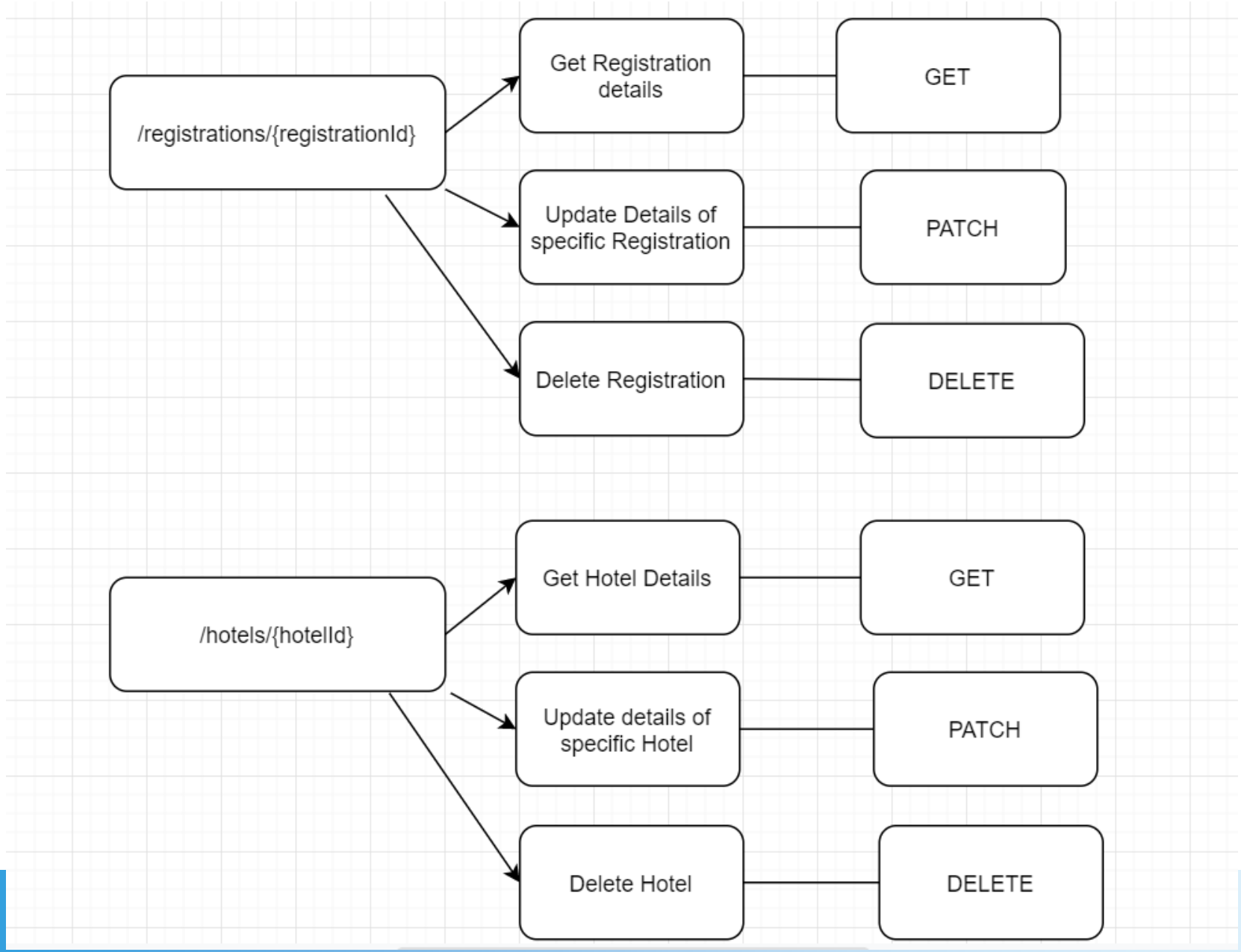
- Define Resources in RAML

Methods

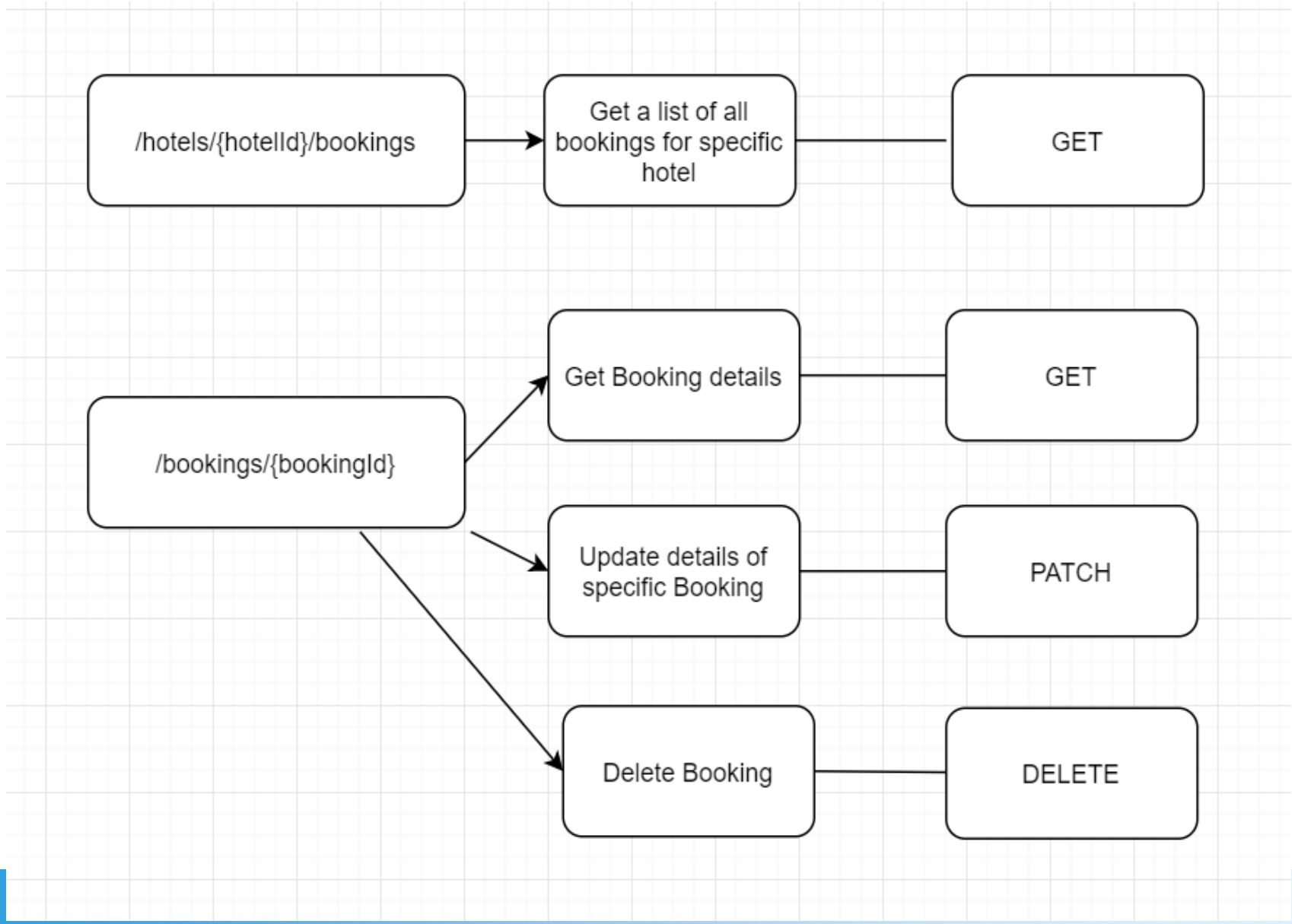
Mapping Actions to Methods



Mapping Actions to Methods



Mapping Actions to Methods

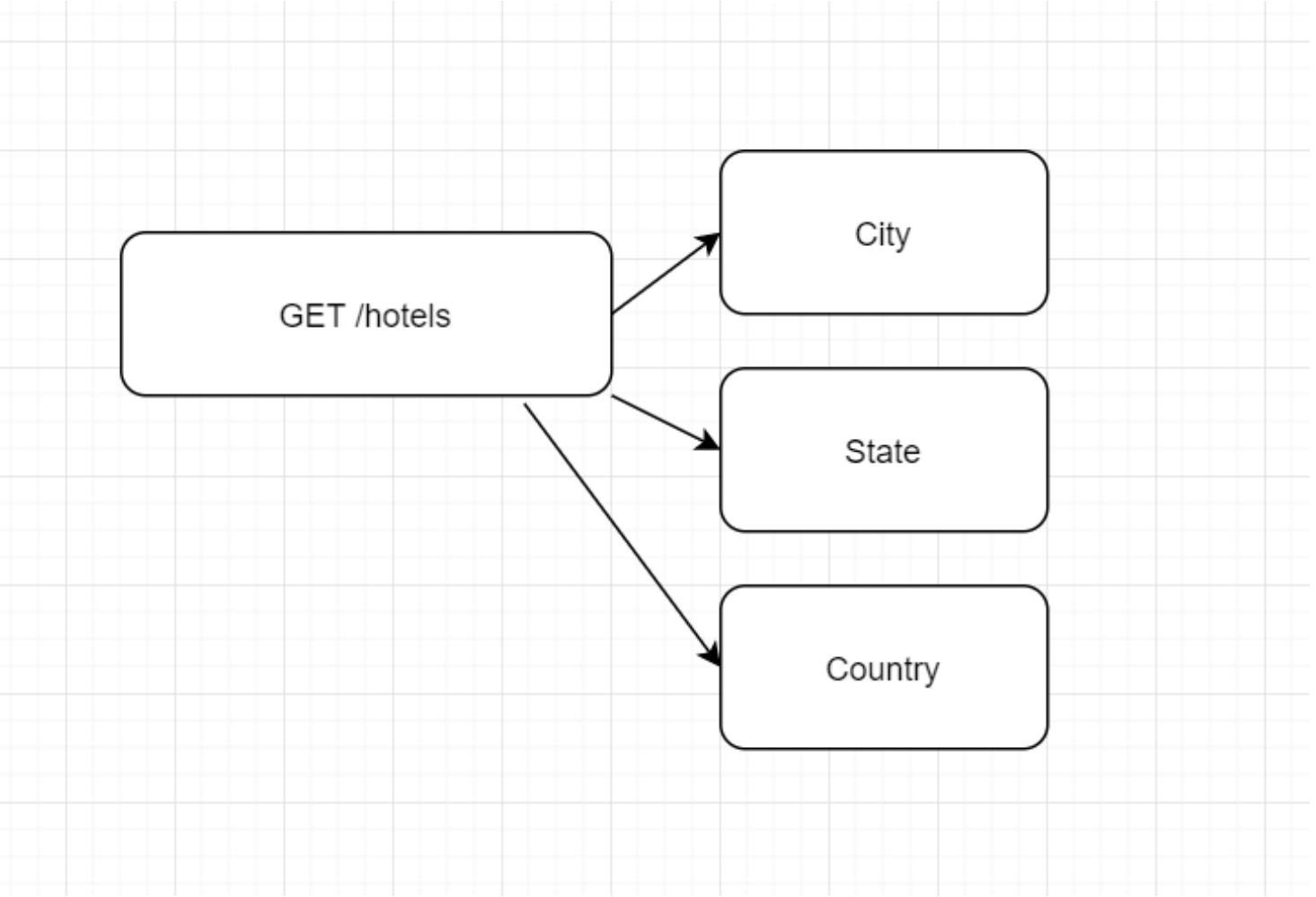


Demo

- Define Methods in RAML

Query Parameters

Query Parameters

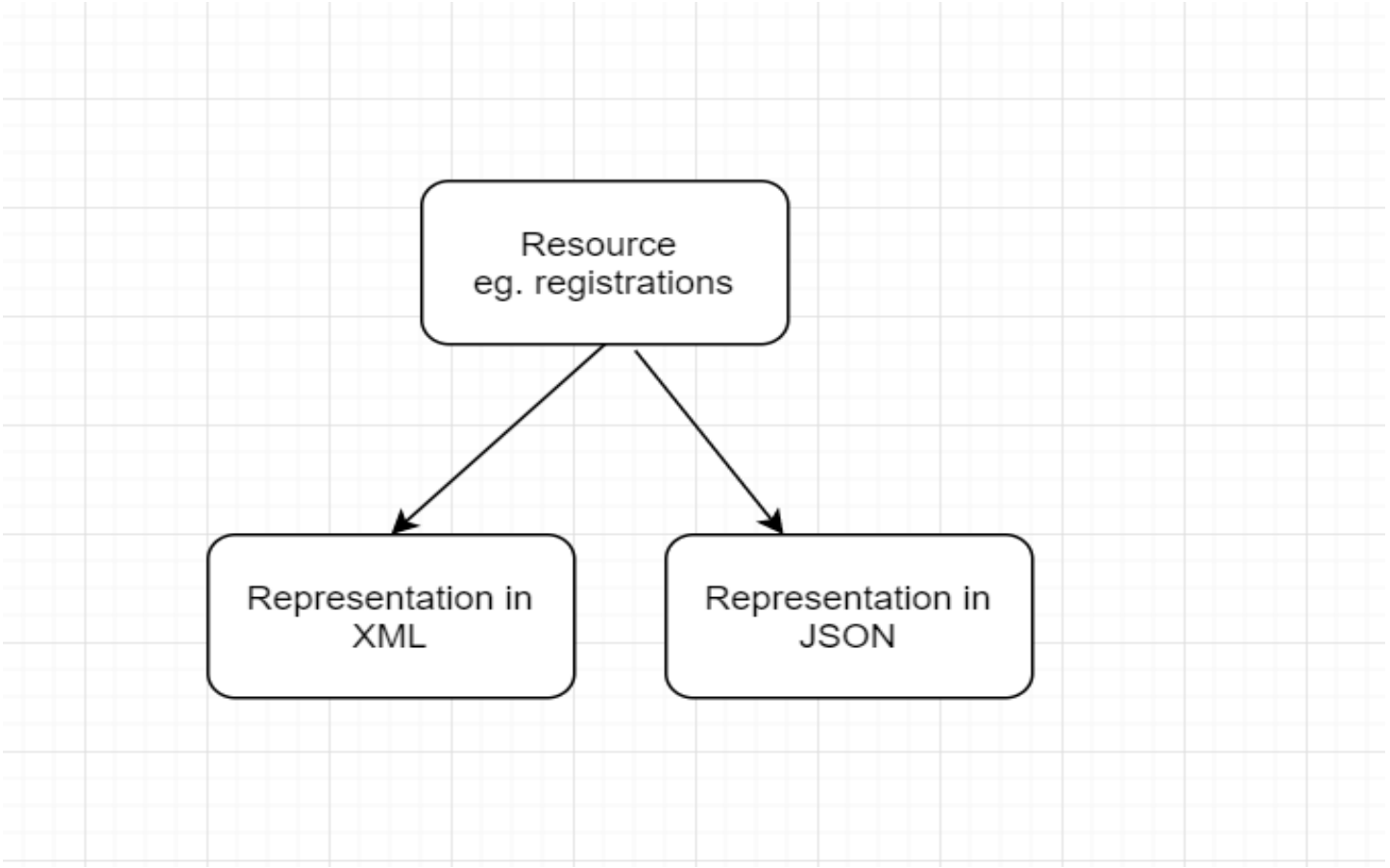


Demo

- http://bookmyhotel.com/hotels?city=noida&state=up&country=india
- Define QueryParameters for HOTELS resource in RAML

Media Type

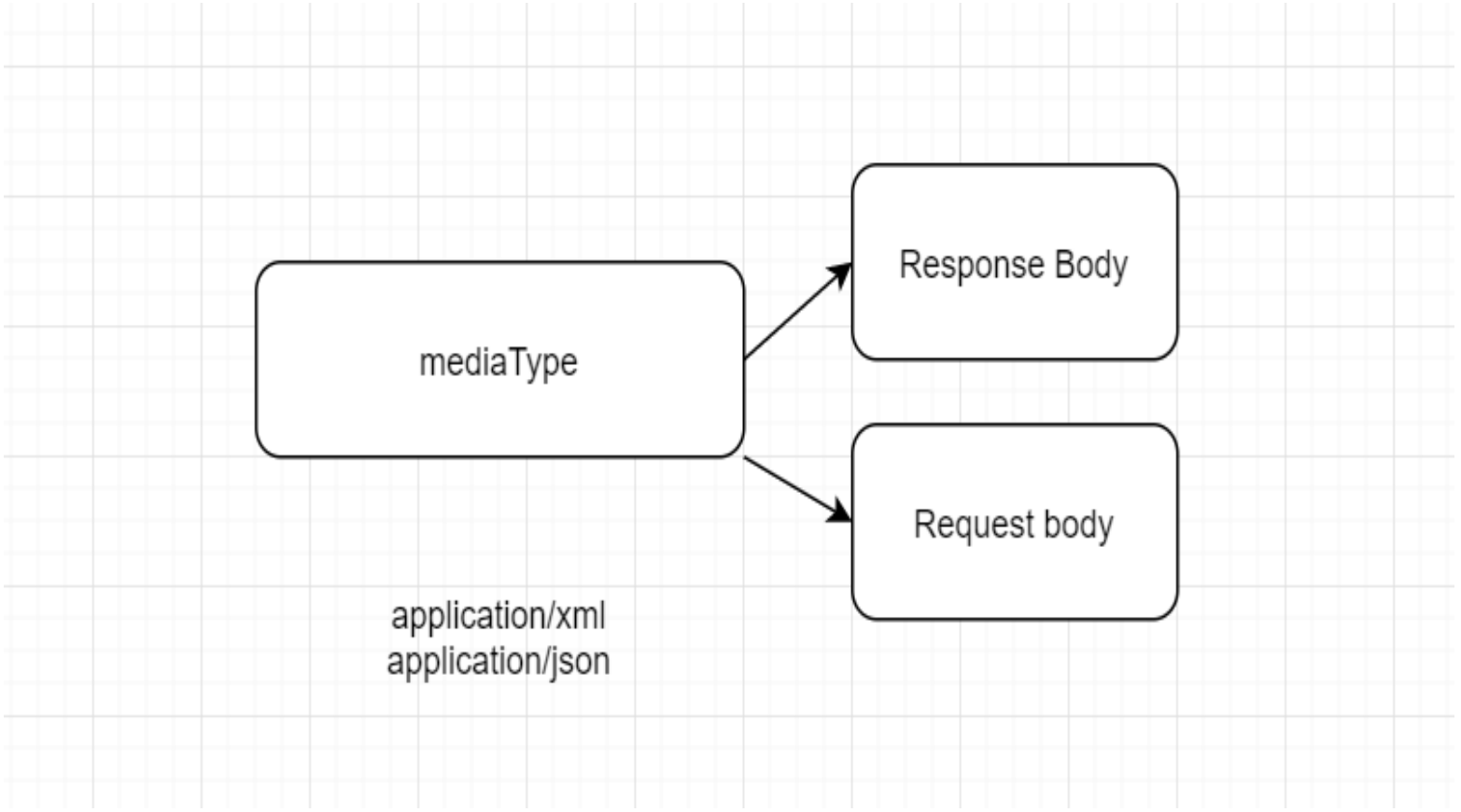
Resource Representation



Media Type

- Resource Representation → Media Type
- Media Type value needs to conform to the media type specification in RFC6838.
- Example :
  - application/json
  - application/xml

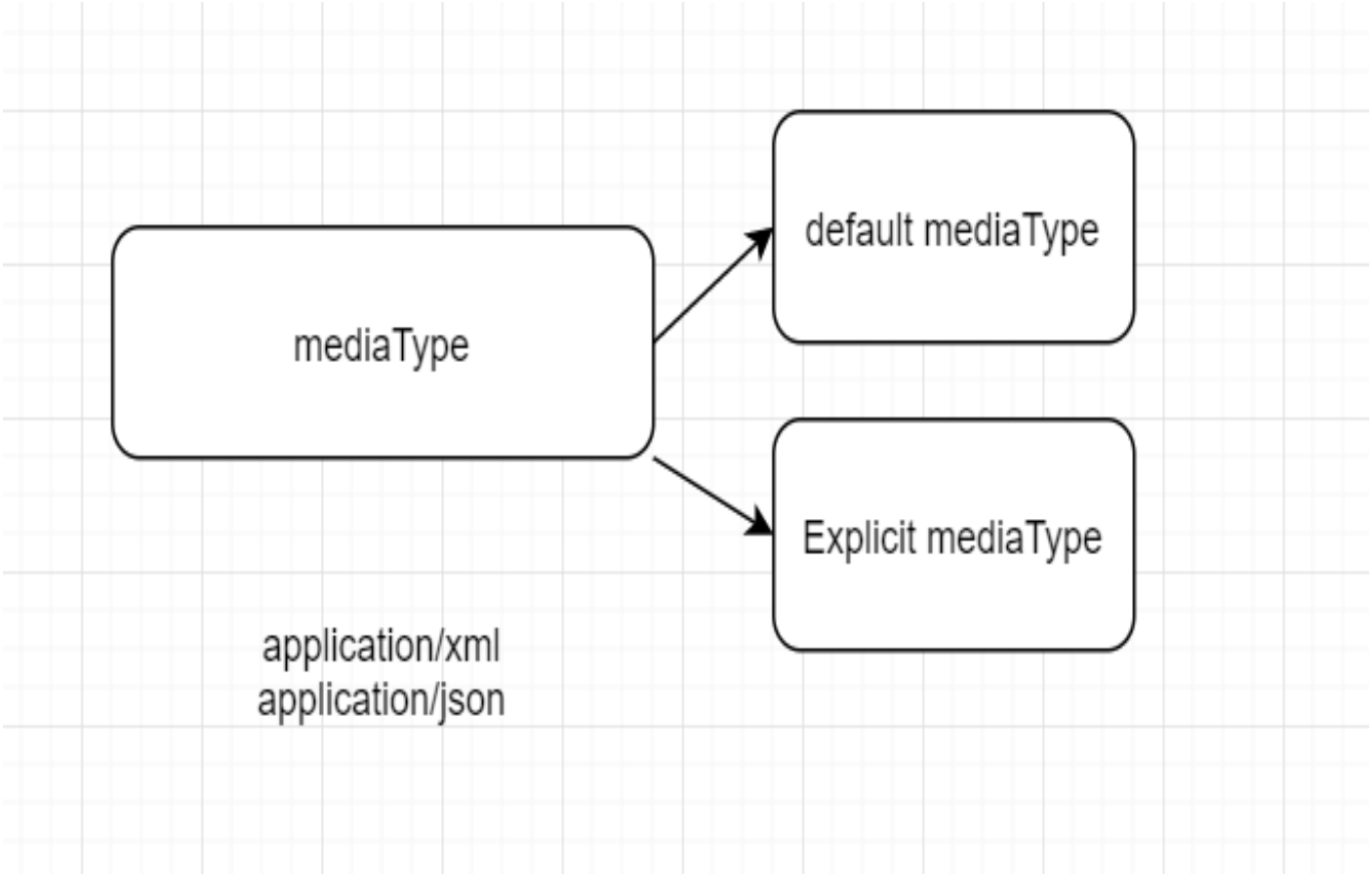
Media Type


mediaType
Response Body
Request body
application/xml
application/json

Media Type

- Headers -
  - Content-Type: application/json
  - Accept: application/json

Media Type

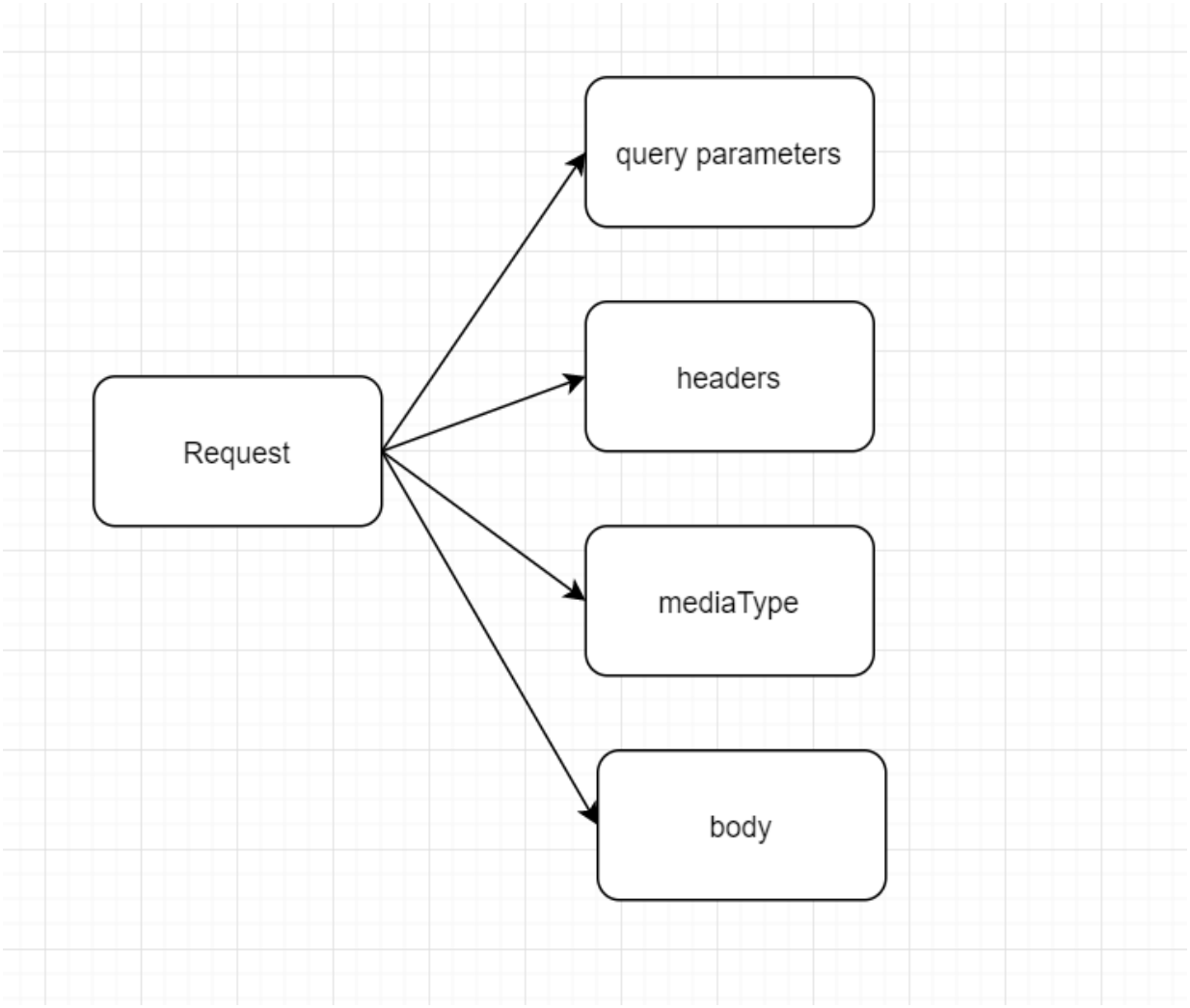

mediaType
default mediaType
Explicit mediaType
application/xml
application/json

Demo

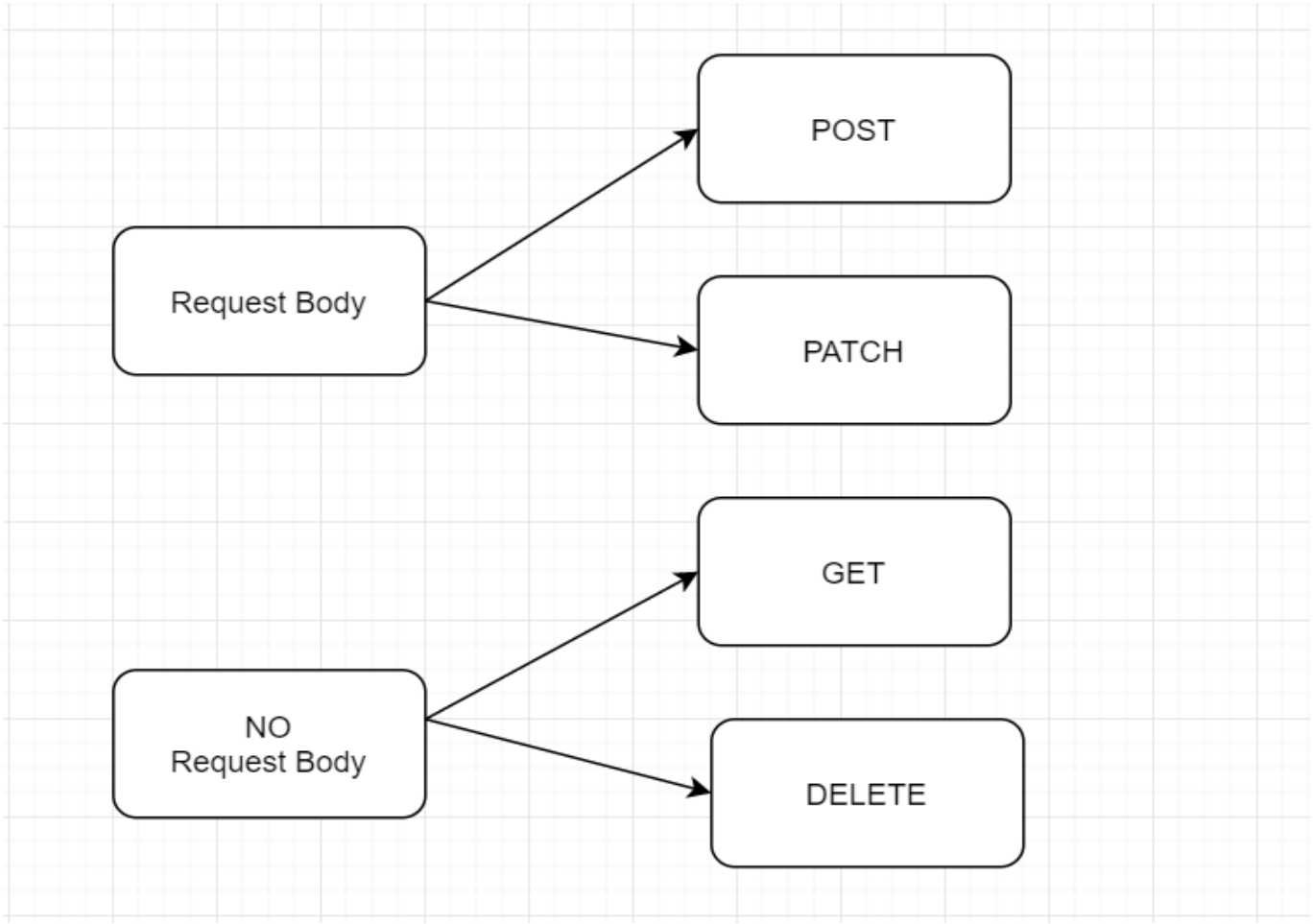
- Define mediaType in RAML

Request and Response

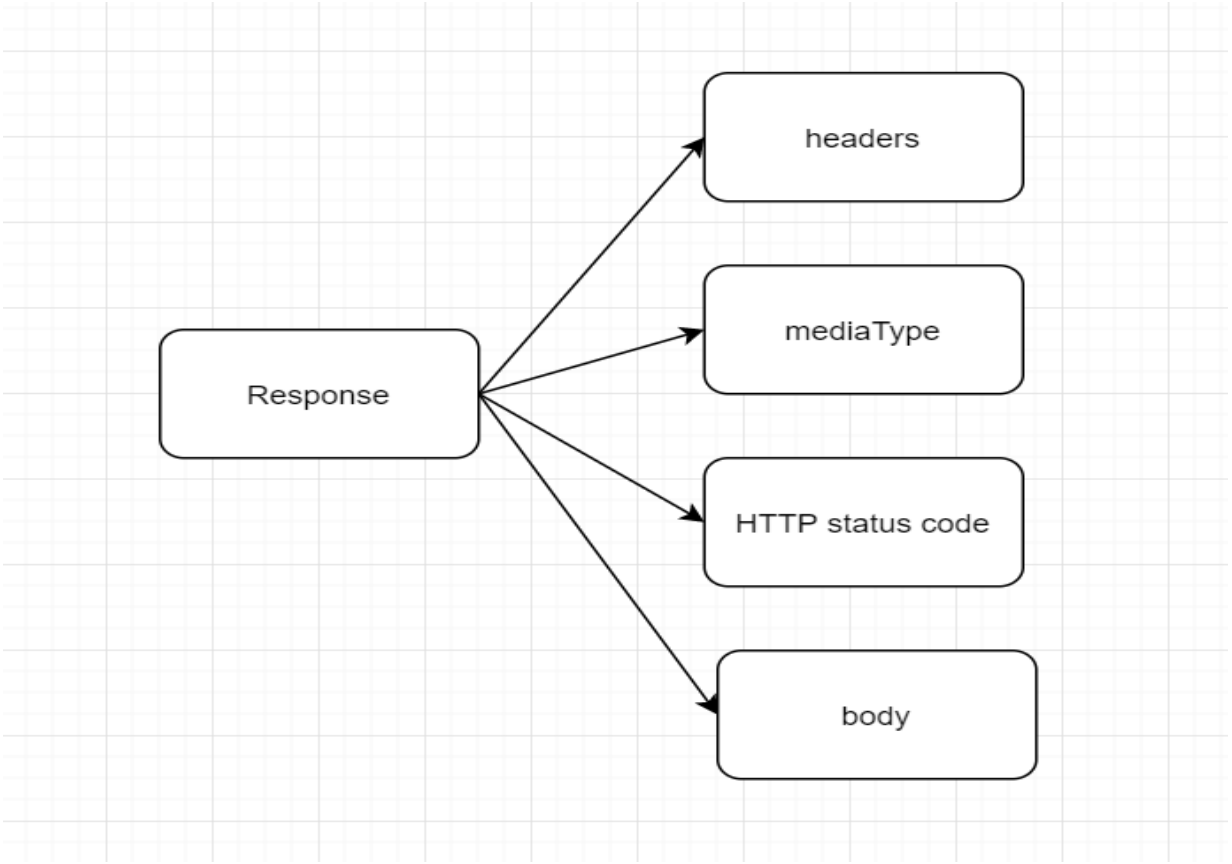
Request



Request Body



Response



Response Codes

- HTTP Status Codes
- Reference : https://www.restapitutorial.com/httpstatuscodes.html
- https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

1xx Informational

100 Continue
101 Switching Protocols
102 Processing (WebDAV)

2xx Success

★ 200 OK
★ 201 Created
202 Accepted
203 Non-Authoritative Information
★ 204 No Content
205 Reset Content
206 Partial Content
207 Multi-Status (WebDAV)
208 Already Reported (WebDAV)
226 IM Used

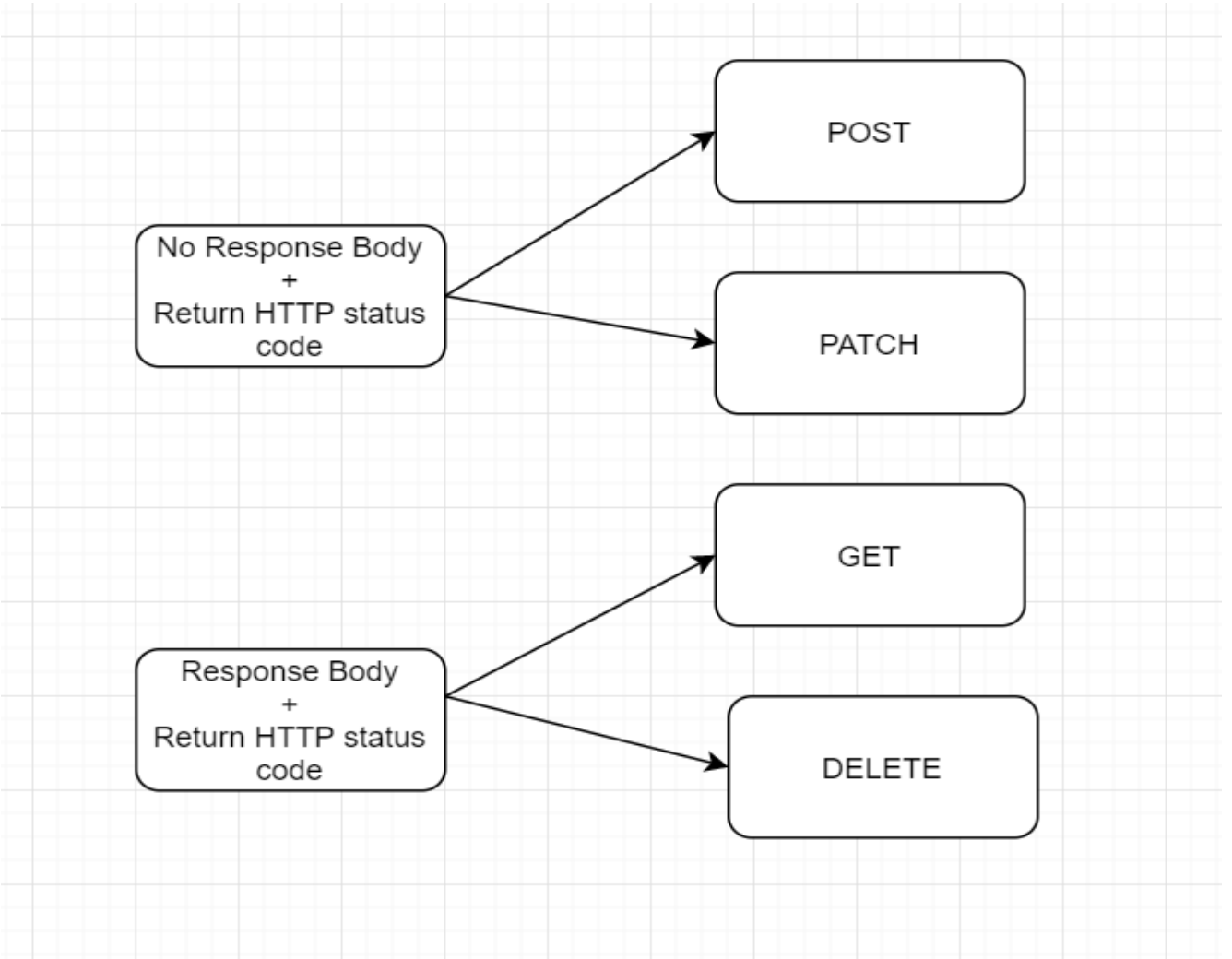
3xx Redirection

300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
★ 304 Not Modified
305 Use Proxy
306 (Unused)
307 Temporary Redirect
308 Permanent Redirect (experimental)

4xx Client Error

★ 400 Bad Request
★ 401 Unauthorized
402 Payment Required
★ 403 Forbidden
★ 404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
★ 409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot (RFC 2324)
420 Enhance Your Calm (Twitter)
422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)
424 Failed Dependency (WebDAV)
425 Reserved for WebDAV
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 No Response (Nginx)
449 Retry With (Microsoft)
450 Blocked by Windows Parental Controls (Microsoft)
451 Unavailable For Legal Reasons
499 Client Closed Request (Nginx)

Response Body

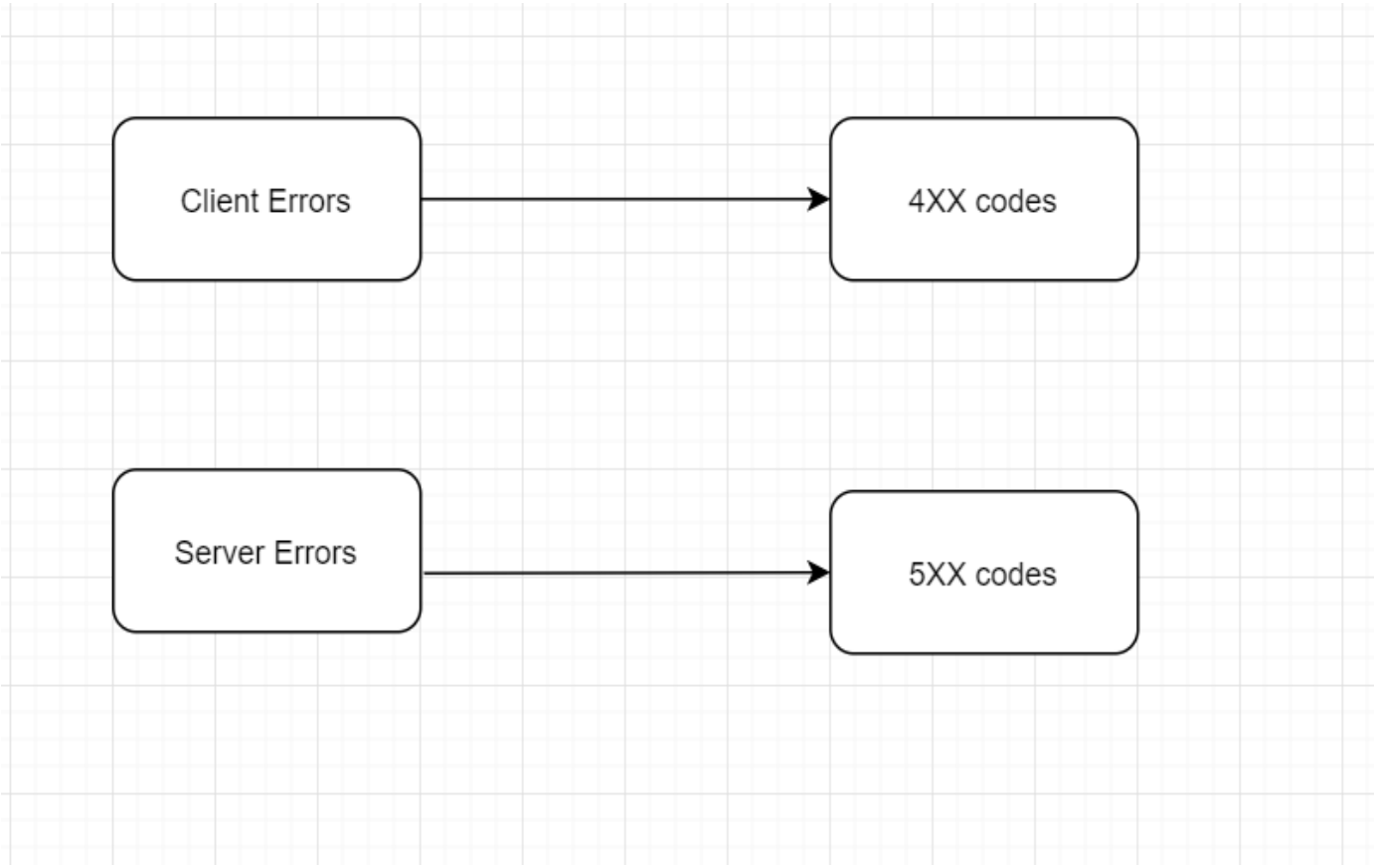


Demo

- Define Request bodies for POST and PATCH
- Define response status codes and body as applicable-
- 200 OK - for GET
- 201 Created - for POST
- 204 No Content - for PATCH
- 200 OK - for DELETE

Error Responses

Error Responses



Error Responses

4xx Client Error

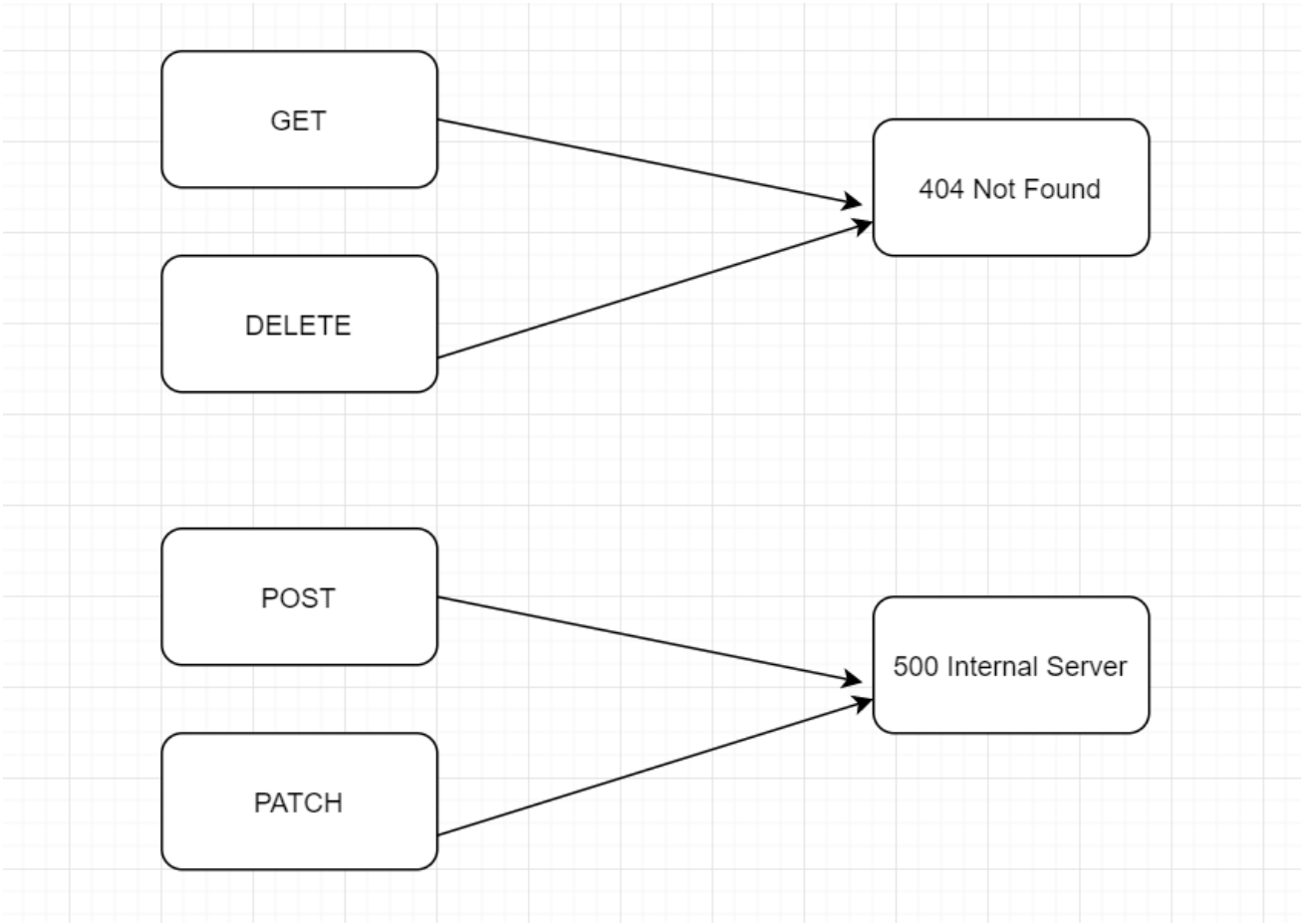
★ 400 Bad Request
★ 401 Unauthorized
402 Payment Required
★ 403 Forbidden
★ 404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
★ 409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot (RFC 2324)
420 Enhance Your Calm (Twitter)
422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)
424 Failed Dependency (WebDAV)
425 Reserved for WebDAV
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 No Response (Nginx)
449 Retry With (Microsoft)
450 Blocked by Windows Parental Controls (Microsoft)
451 Unavailable For Legal Reasons
499 Client Closed Request (Nginx)

5xx Server Error

★ 500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
506 Variant Also Negotiates (Experimental)
507 Insufficient Storage (WebDAV)
508 Loop Detected (WebDAV)
509 Bandwidth Limit Exceeded (Apache)
510 Not Extended
511 Network Authentication Required

Ruchi Saini

Error Responses

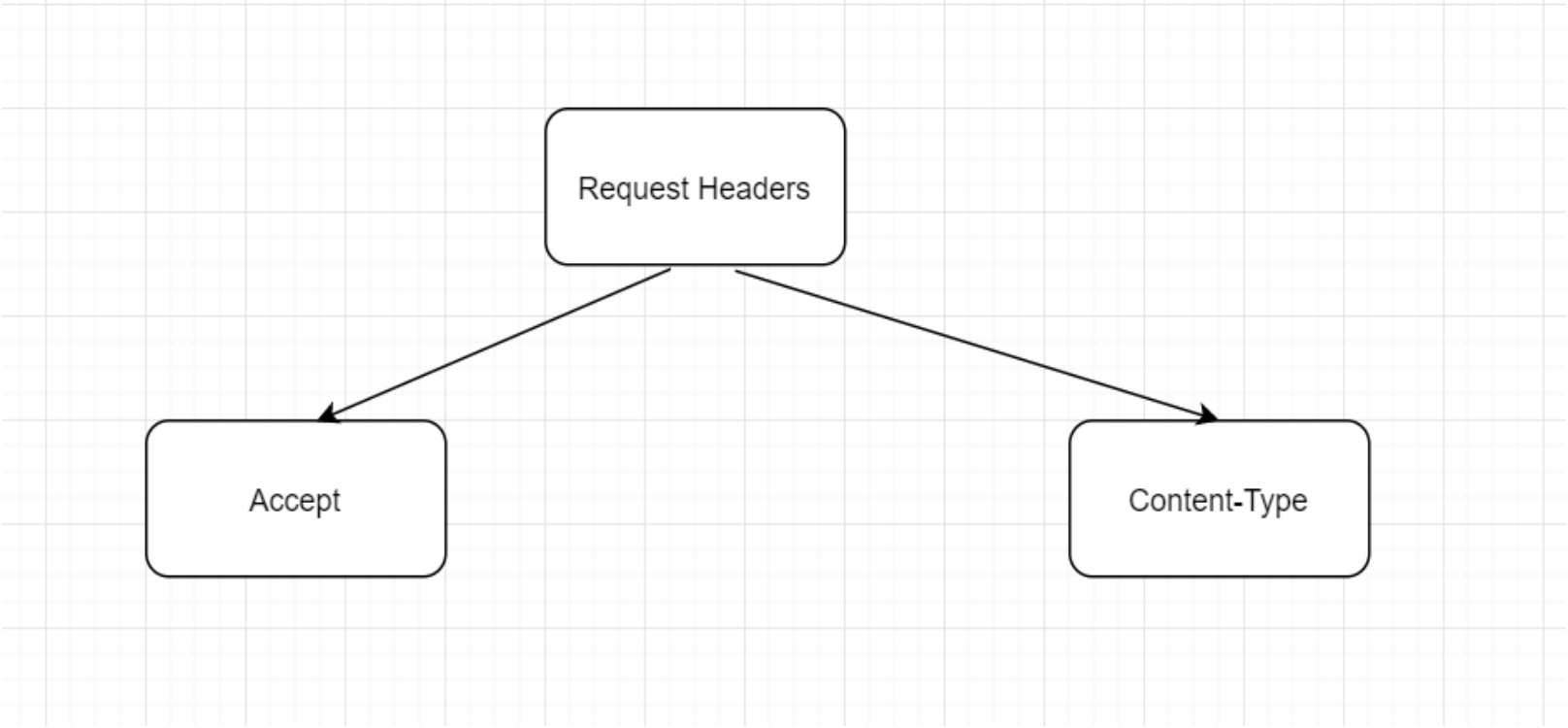


Demo

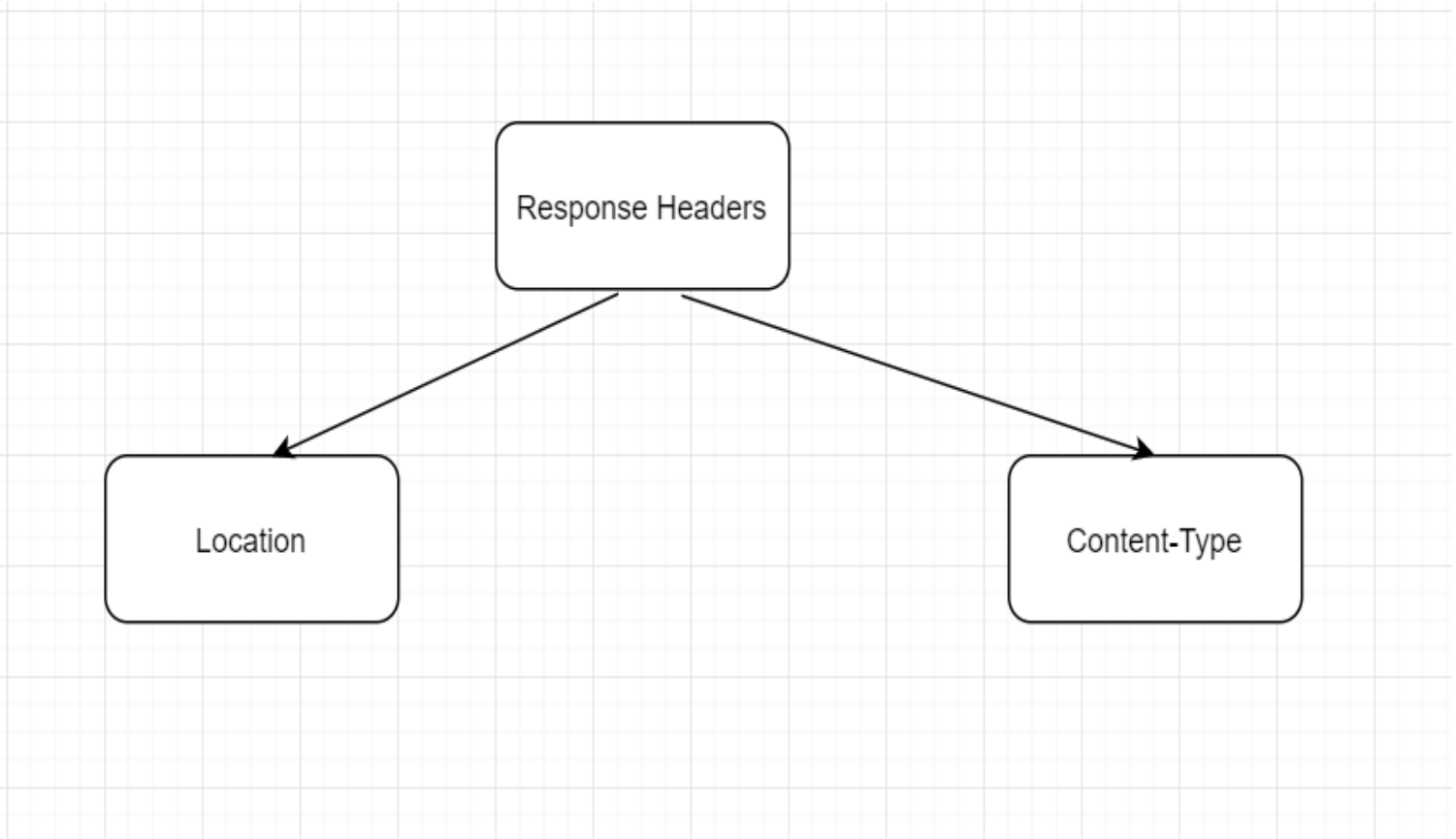
- Define HTTP Status codes for Error Responses

Headers

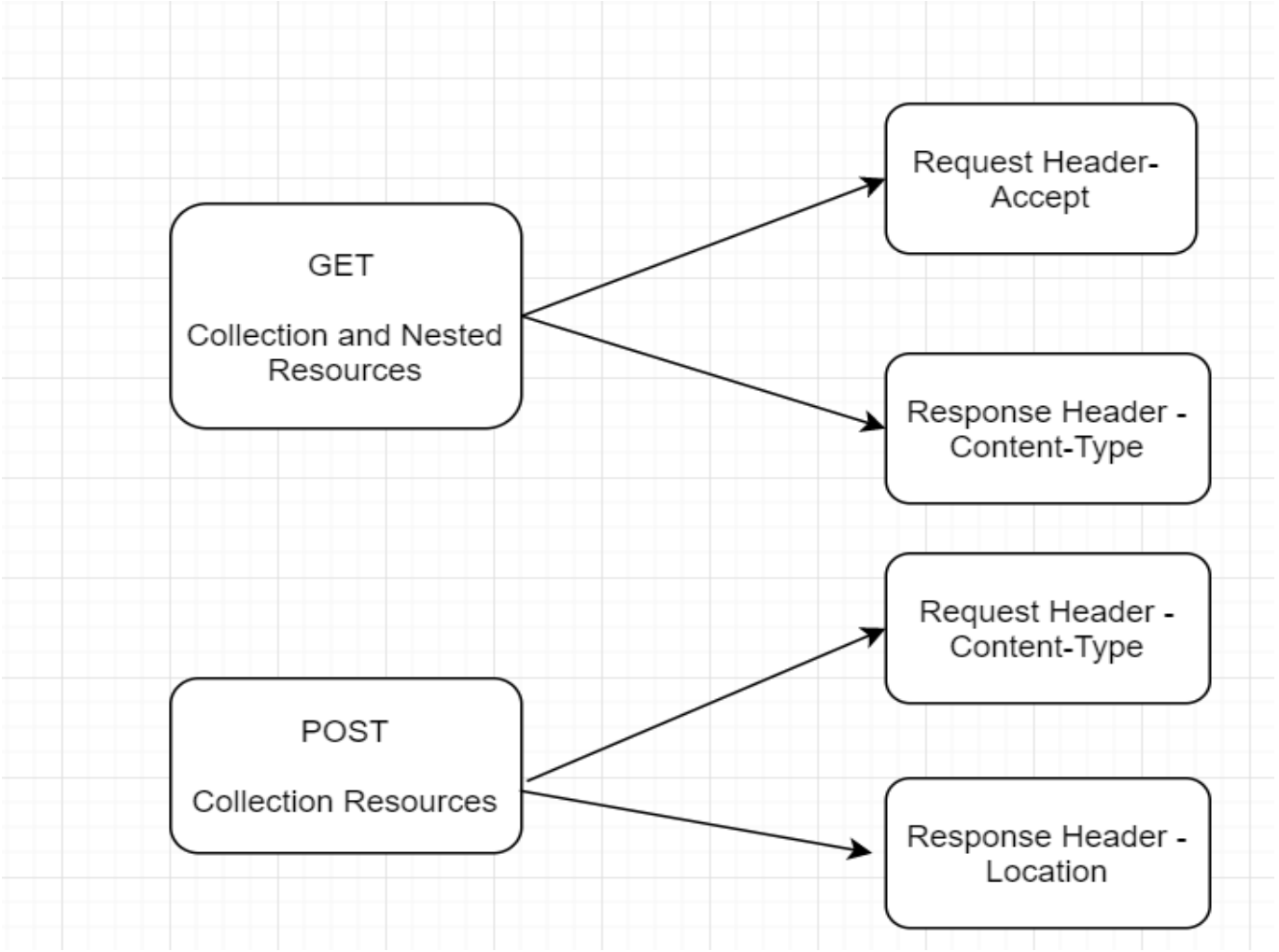
Request Headers



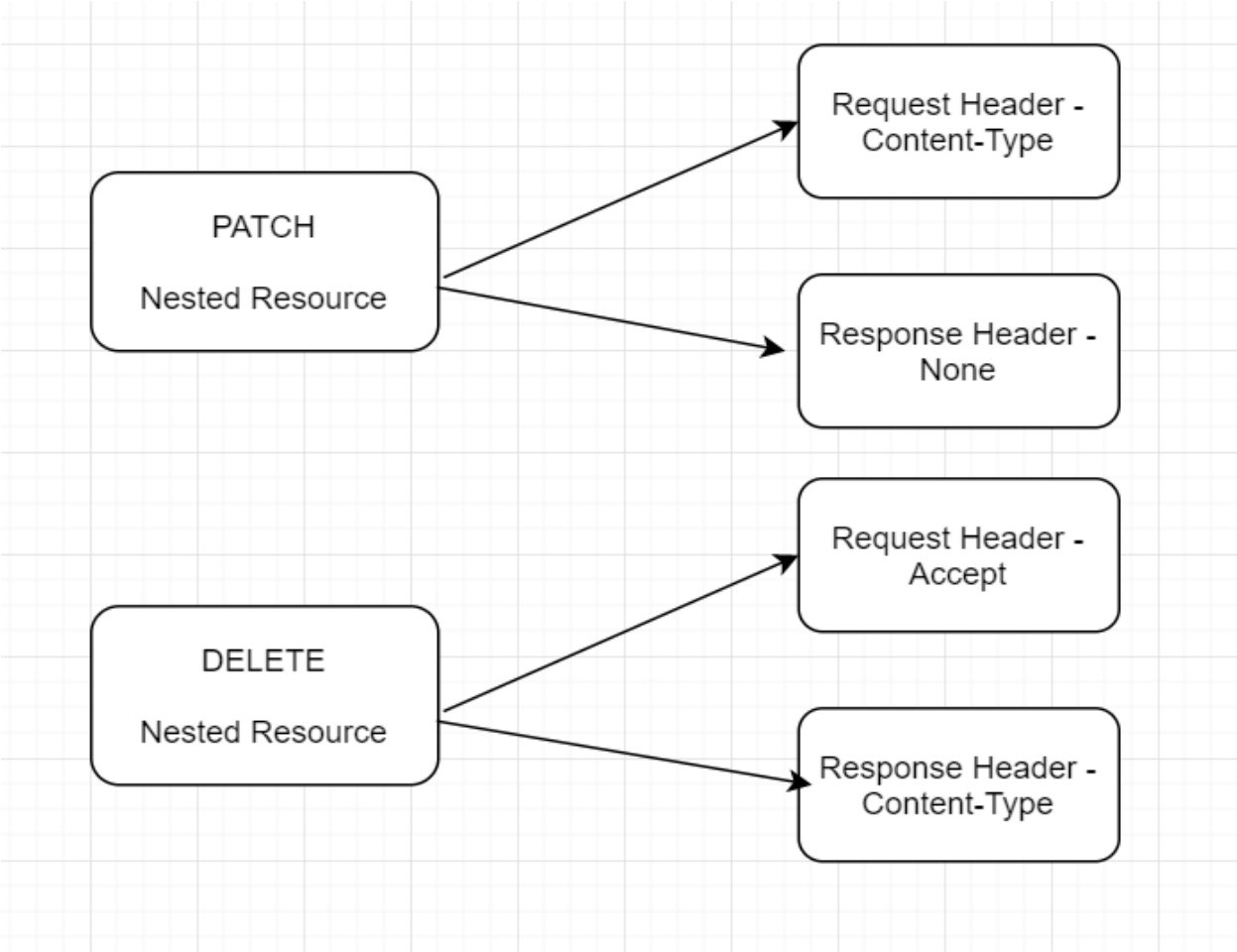
Response Headers



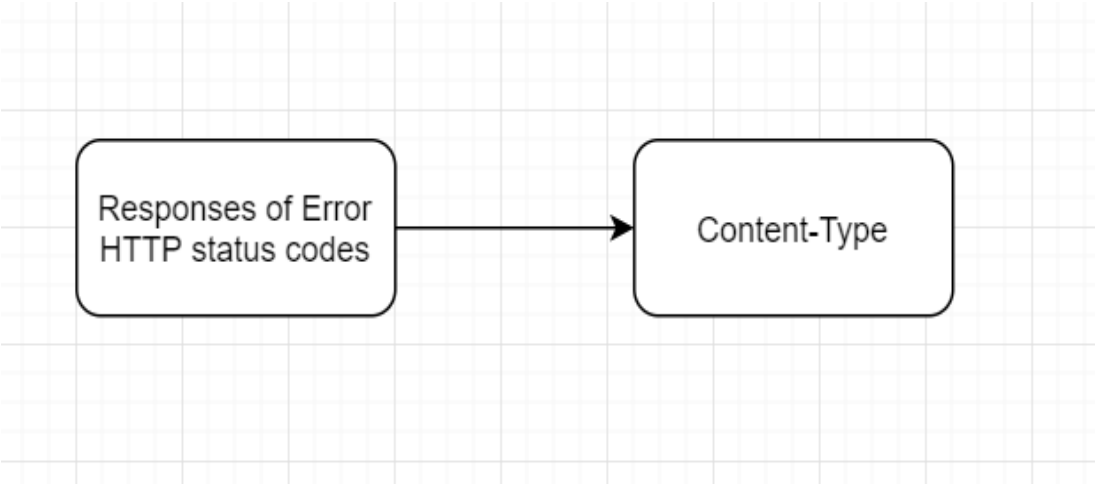
Headers


GET
Collection and Nested Resources
Request Header- Accept
Response Header - Content-Type
POST
Collection Resources
Request Header - Content-Type
Response Header - Location

Headers


PATCH
Nested Resource
Request Header - Content-Type
Response Header - None
DELETE
Nested Resource
Request Header - Accept
Response Header - Content-Type

Headers



Demo

- Define Headers in RAML

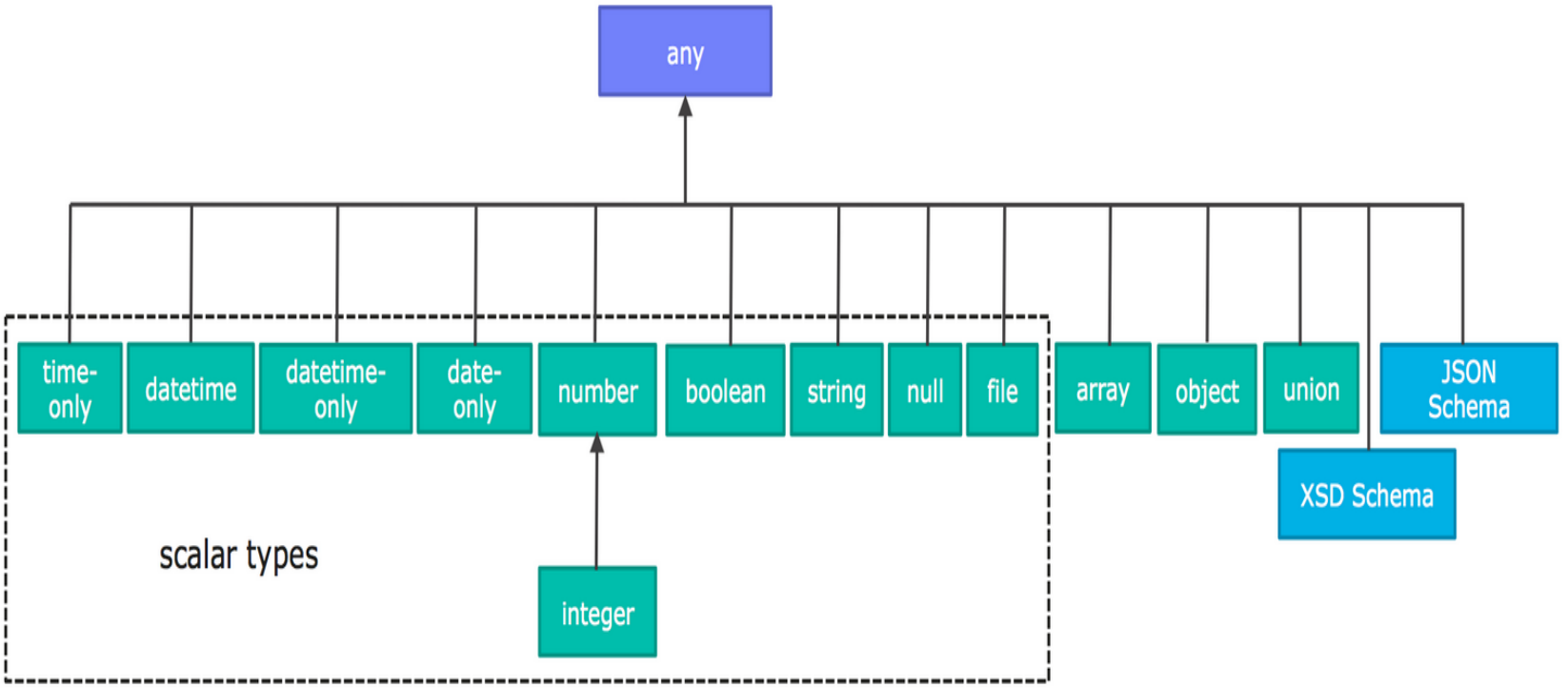
Introduction to RAML Data Types

Ruchi Saini

Data Types

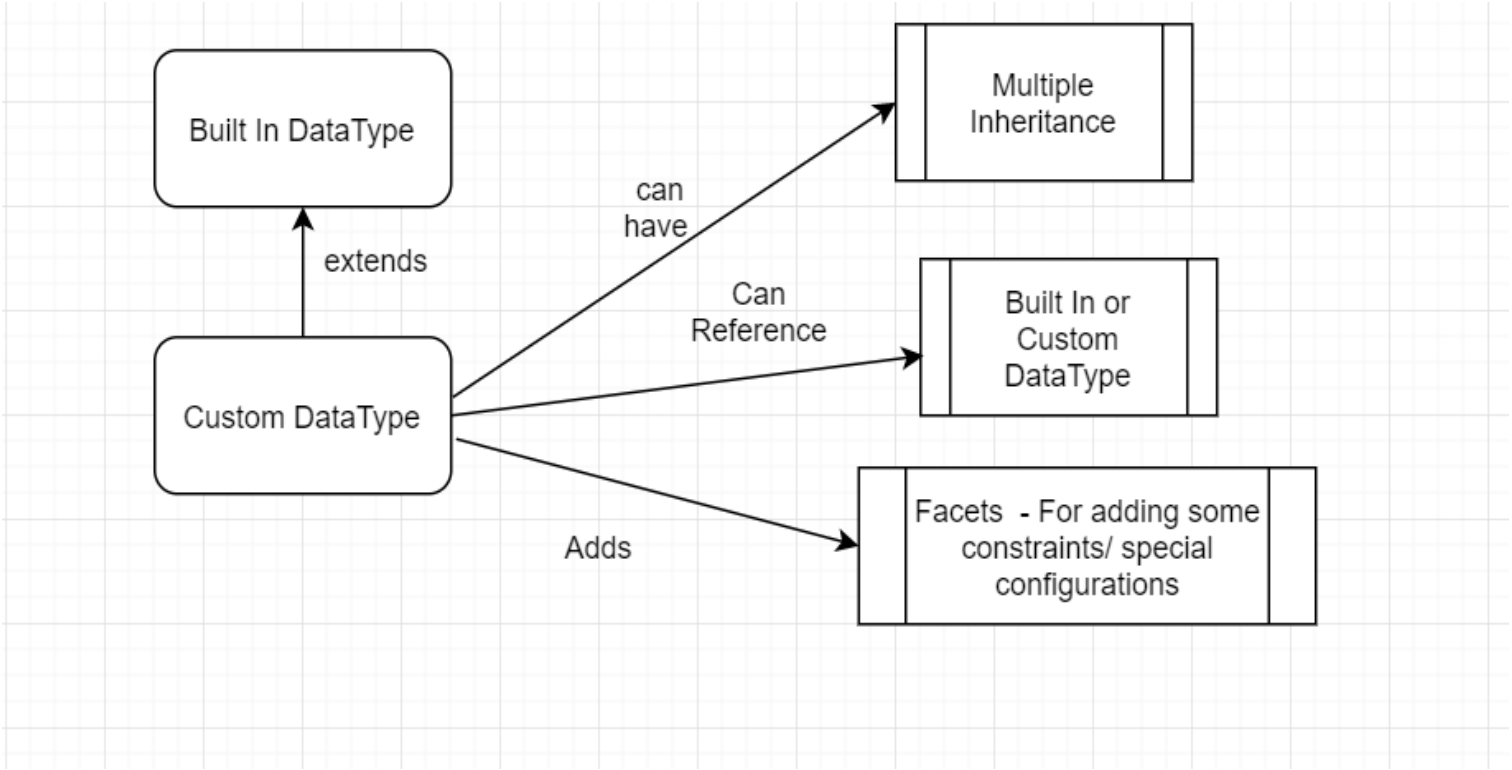
- Describes the Data in an API
- Add Rules to validate data against a type declaration
- Built-In Data types
- Custom Data types

Built In Data Types



Custom Data Types

- Created by extending Built In DataTypes



Example of Custom Data Type

- types:
- Error: # key name
- # value is a type declaration

- RAML Code Snippet : -

```
types:
  Error:
    type: object
    properties:
      code: integer
      reasonPhrase: string
      details: string
```

```
/registrations:
  get:
    headers:
      Accept?:
    responses:
      200:
        headers:
          Content-Type:
        body:
      404:
        headers:
          Content-Type:
        body:
          Error
```

Facets

```
types:
  Error:
    type: object
    description: Defines Error
    properties:
      code:
        type: integer
        minimum: 400
        maximum: 599
      reasonPhrase: string
      details: string
```

Facets

- Few Common Facets for all type declarations
  - eg. default,description,enum,example
- Some are specific to the type
  - Object : properties, minProperties,maxProperties etc
  - Array: uniqueItems,minItems,maxItems etc
  - String : pattern,minLength,maxLength etc

Datatype Examples

```
types:
  Age:
    type: integer
    minimum: 3
    maximum: 5
    format: int8
    multipleOf: 1
```

```
types:
  Email:
    type: string
    minLength: 2
    maxLength: 6
    pattern: ^note\d+$
```

```
types:
  Email:
    type: object
    properties:
      name:
        type: string
  Emails:
    type: array
    items: Email
    minItems: 1
    uniqueItems: true
```

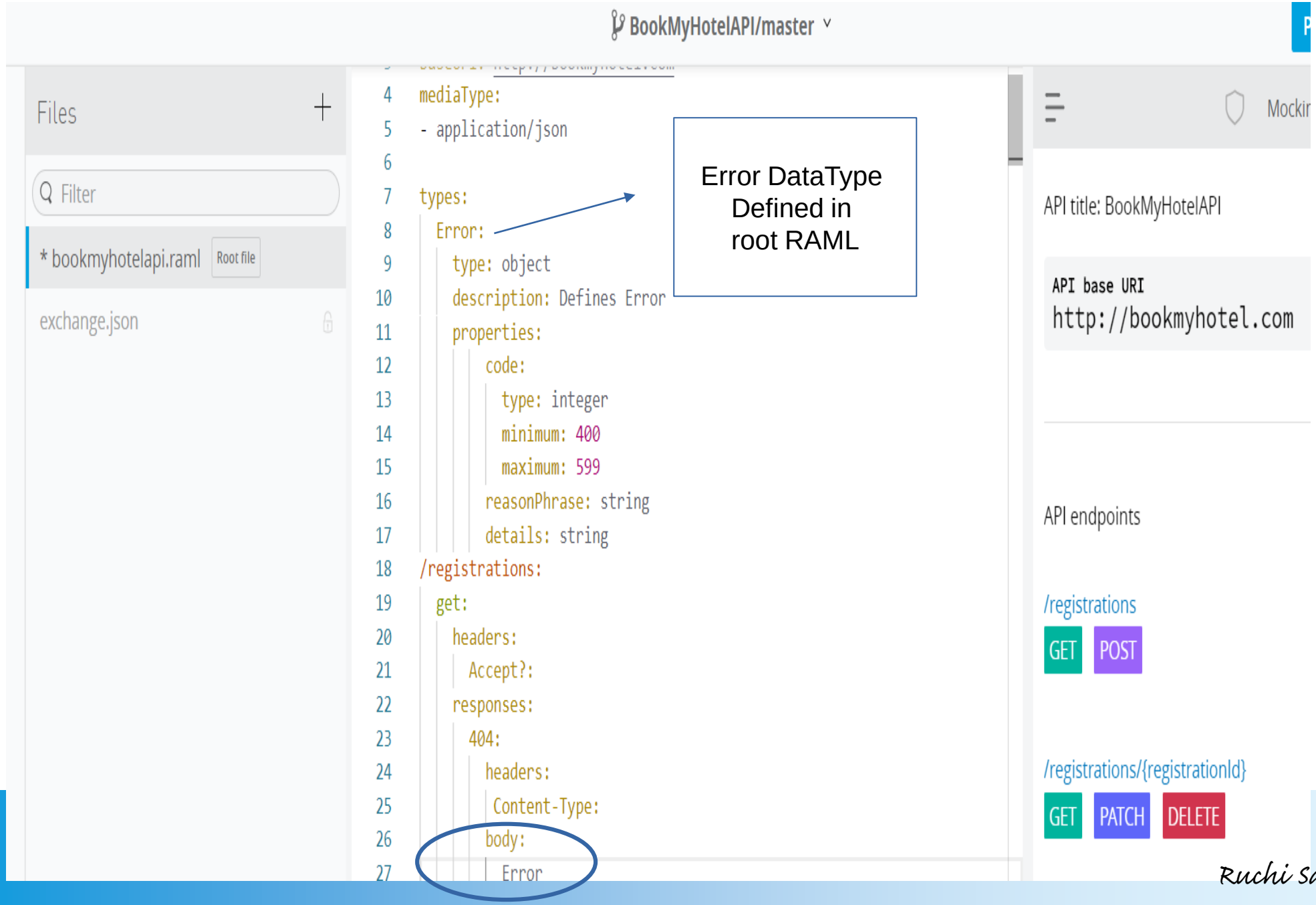
- Using Email[] is equivalent to using type: array.
- The items facet defines the Email type as the one each array item inherits from.

Demo

- Next Video

Datatype Fragments

Data Types defined Inline



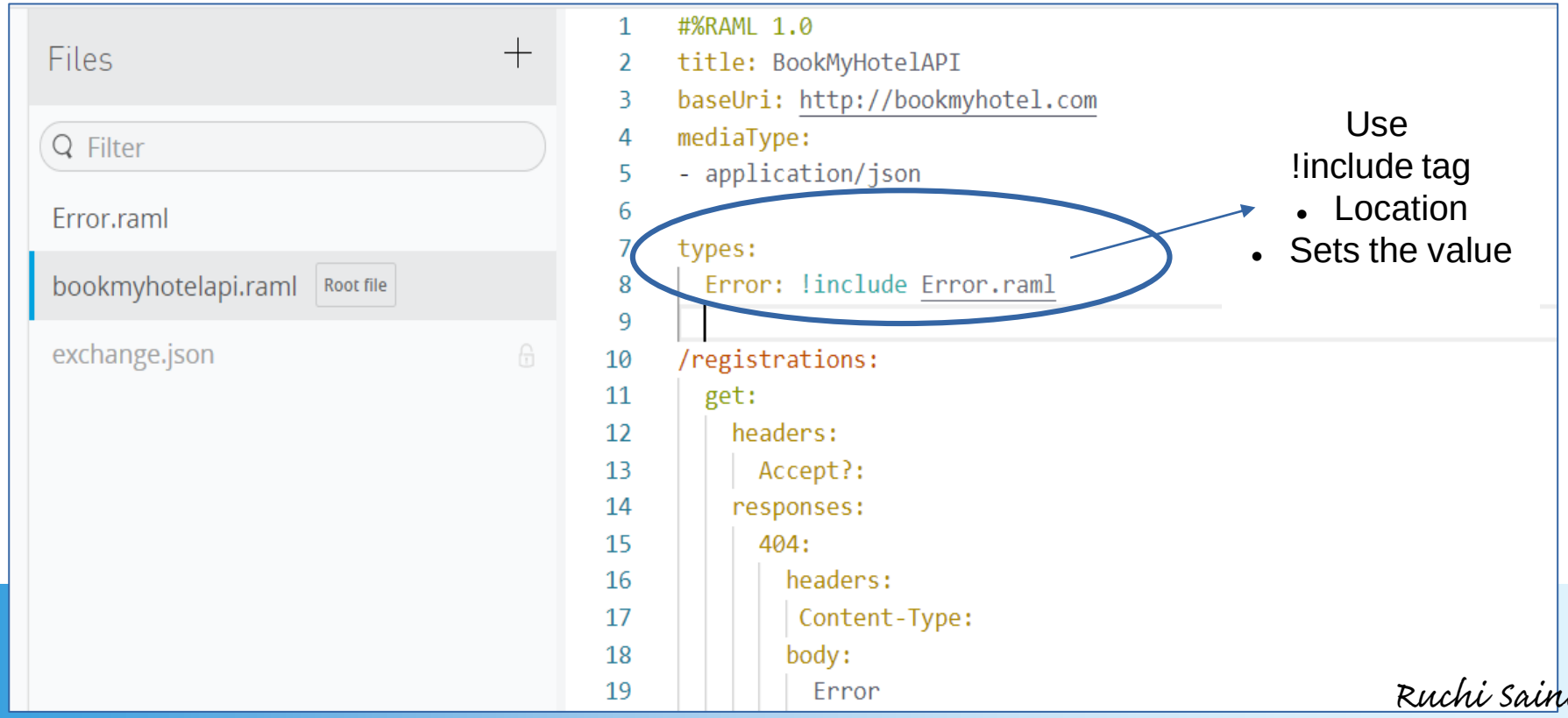
Ruchi Saini

Data Type Fragments

Fragment Identifier

DataType in separate/standalone RAML

- Broken into smaller components
- Readable
- Reusable

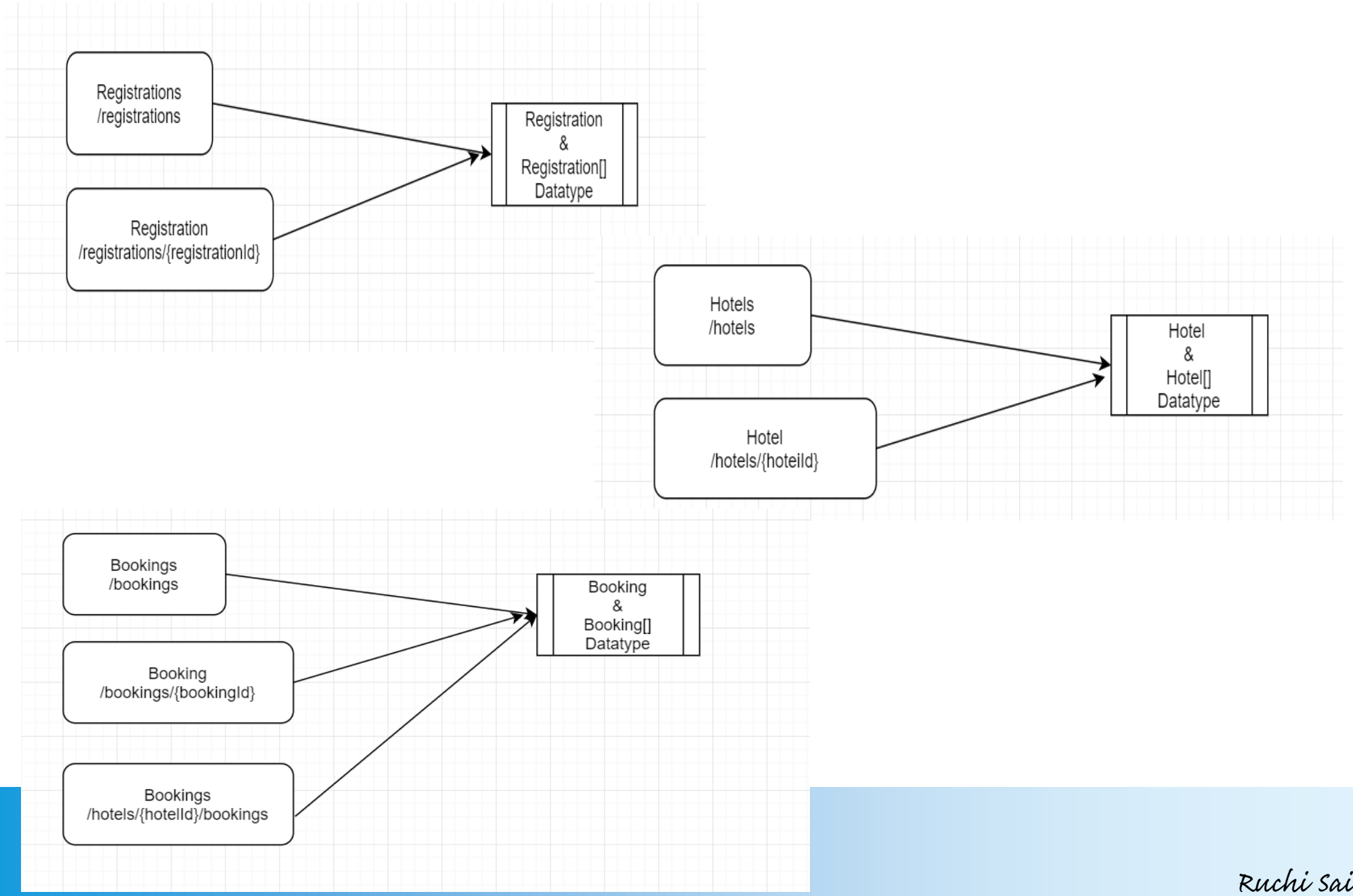


Use !include tag

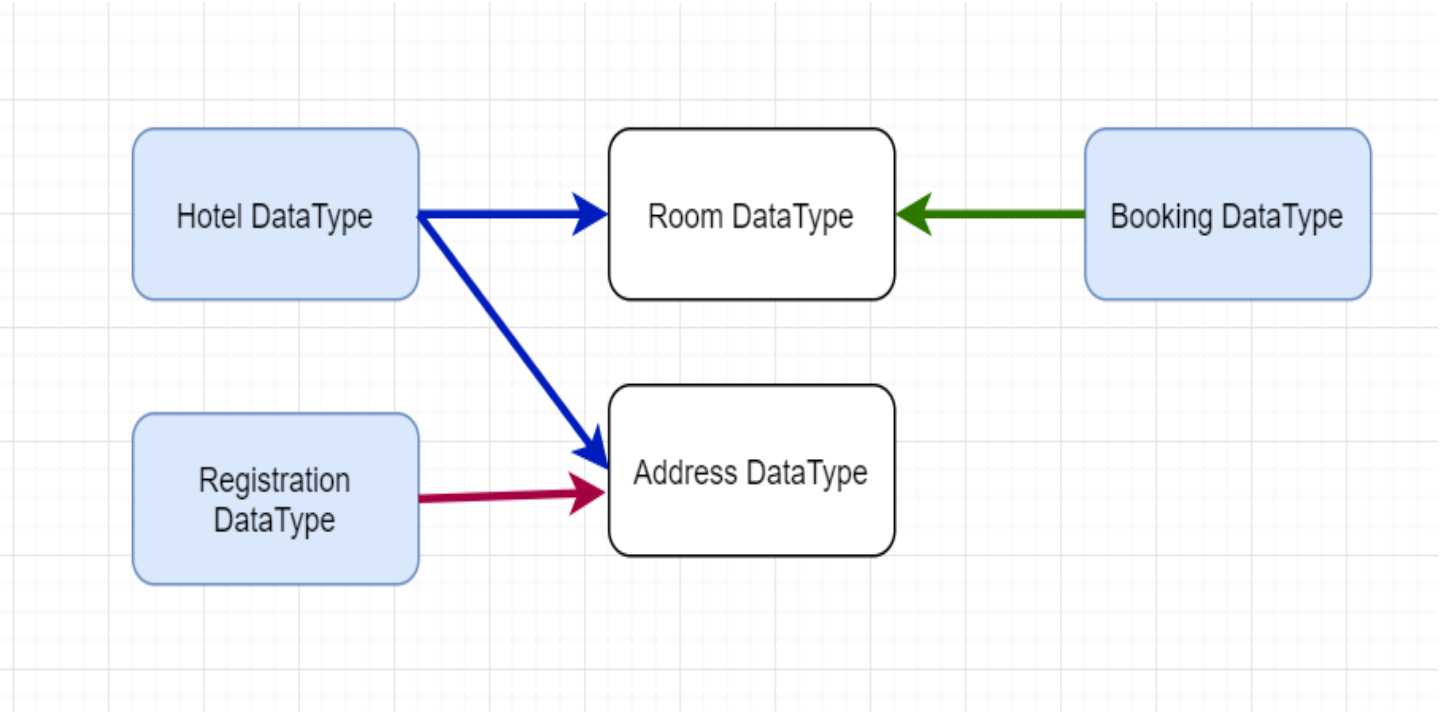
- Location
- Sets the value

Ruchi Saini

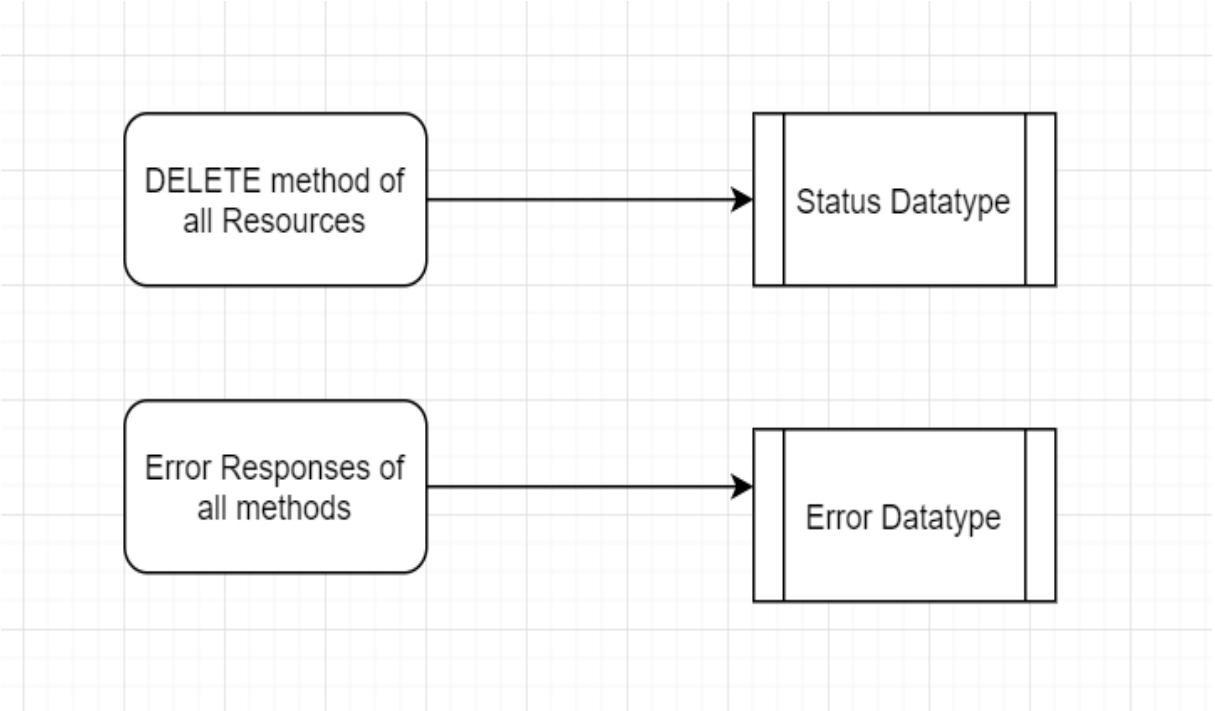
Data Types - Use Case



Data Types – Use Case



Data Types – Use Case

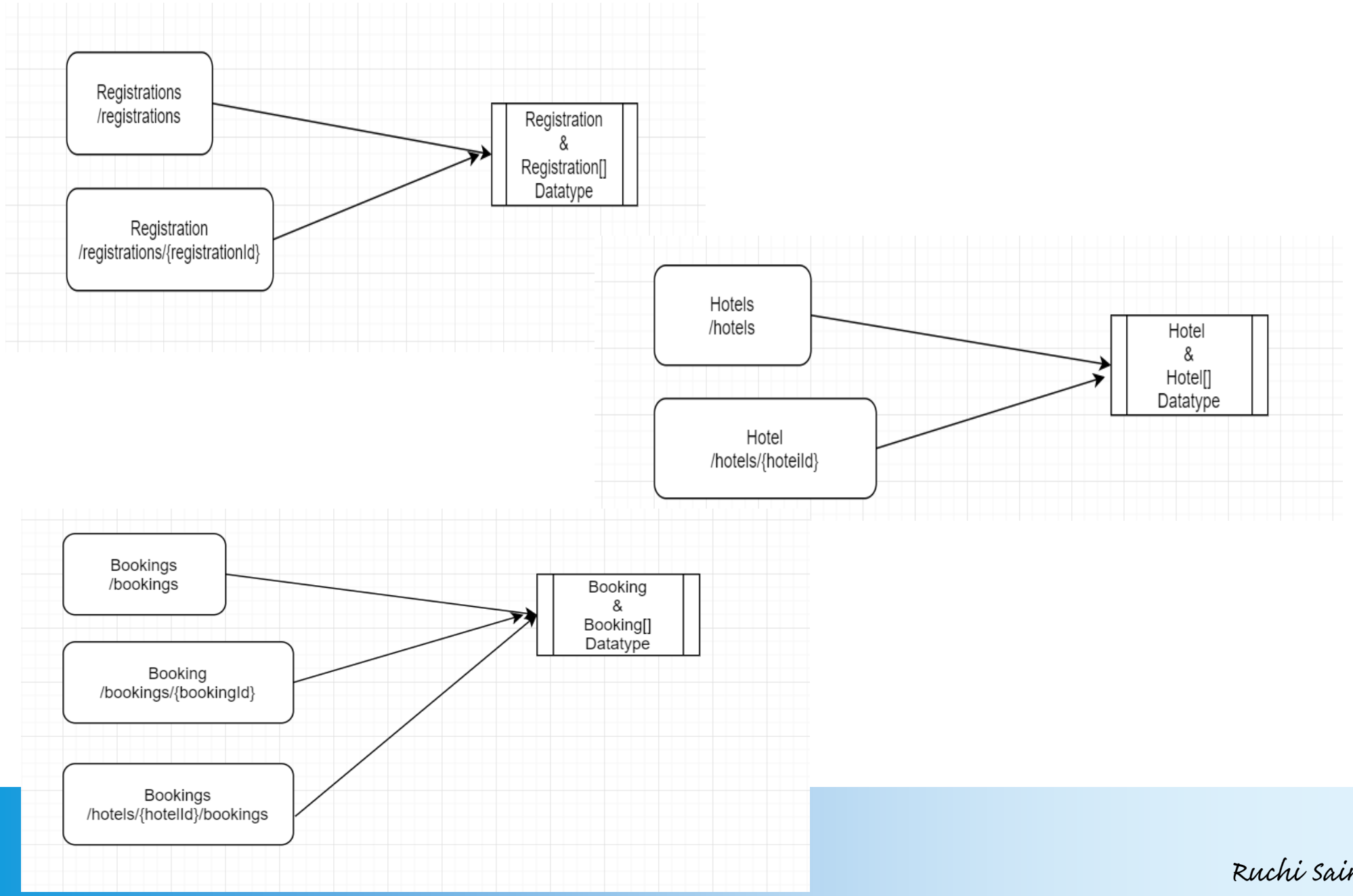


Demo

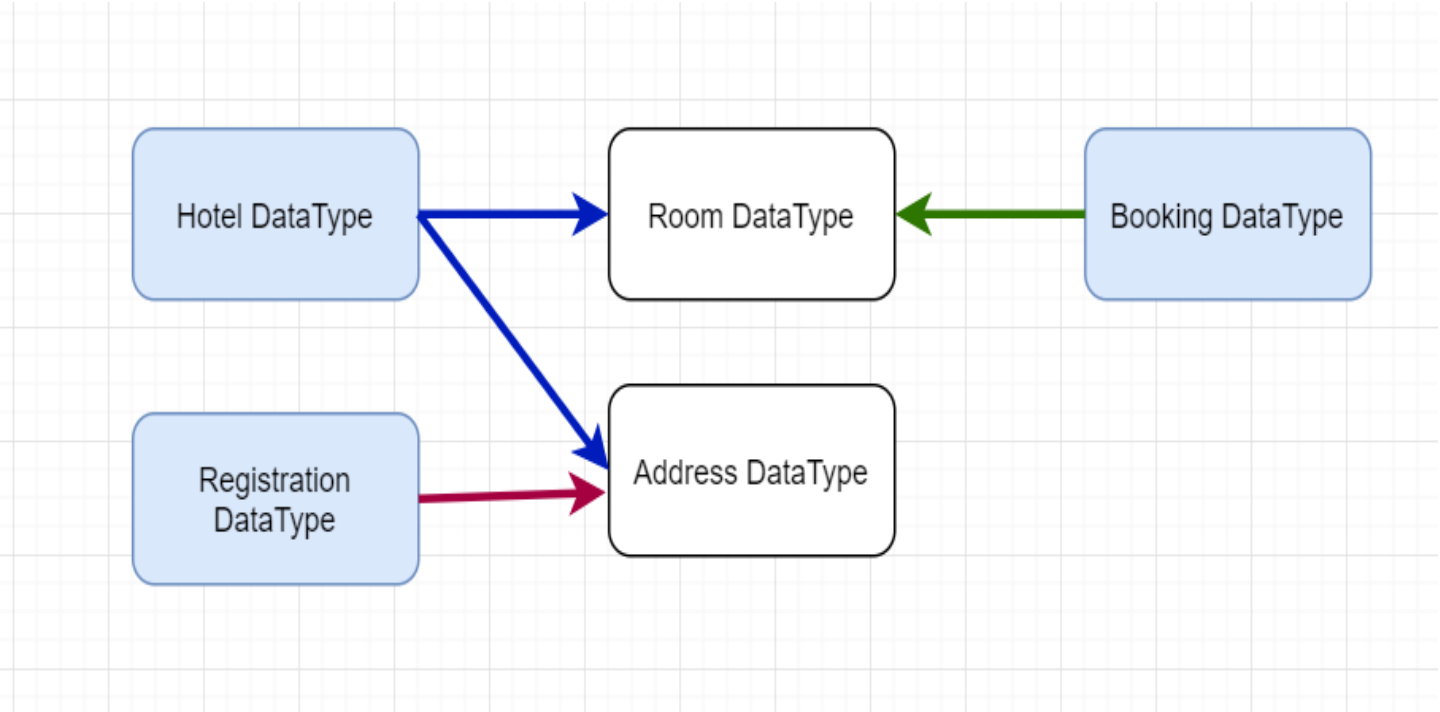
- Define Error DataType Fragment
- Include this fragment in root raml file

Create Datatype Fragments

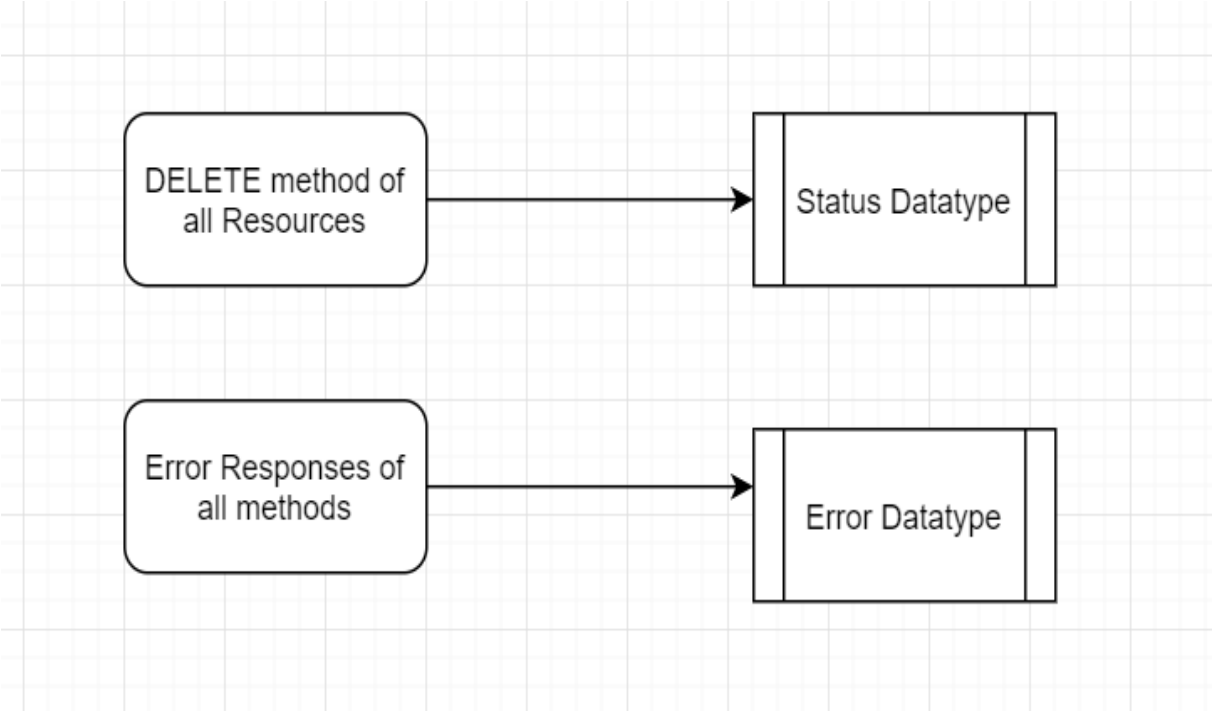
Data Types - Use Case



Data Types – Use Case



Data Types – Use Case



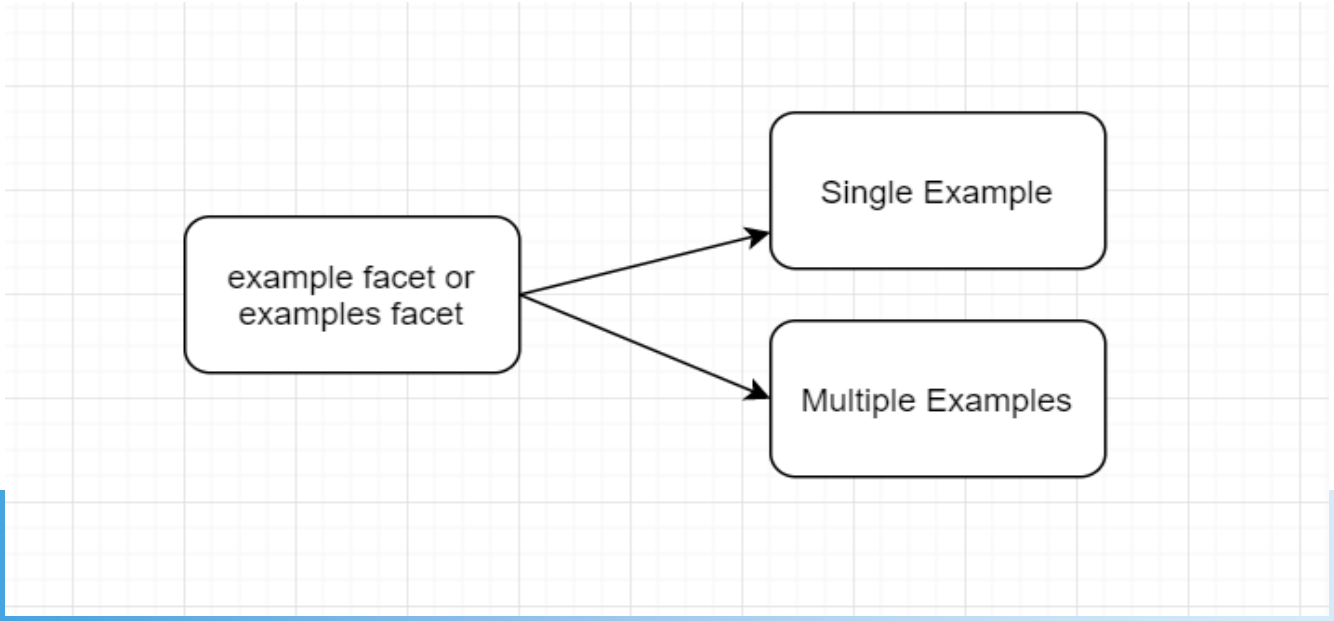
Demo

- Define ALL DataType Fragments and Create DataTypes
- Use DataTypes to define request and response bodies of methods

Example Fragments

Example

- Define example for an instance of datatype declaration
- Attach single or multiple examples to datatype declaration
- Documentation
- Readability
- Helpful to API Consumers



Example Fragment

```
1   #%RAML 1.0 DataType
2   type: object
3   description: Defines Error
4   properties:
5     code:
6       type: integer
7       minimum: 400
8       maximum: 599
9     reasonPhrase: string
10    details: string
```

```
1   #%RAML 1.0 NamedExample
2   value:
3     code: 404
4     reasonPhrase: Not Found
5     details: Resource Not Found
```

```
1   #%RAML 1.0 DataType
2   type: object
3   properties:
4     code:
5       type: integer
6       minimum: 400
7       maximum: 599
8     reasonPhrase: string
9     details: string
10  example: !include example/ErrorExample.raml
```

Demo

- Create Example Fragments for Datatypes
- Include Example Fragments in DataType Fragments

Introduction to Mocking Service

Ruchi Saini

Mocking Service

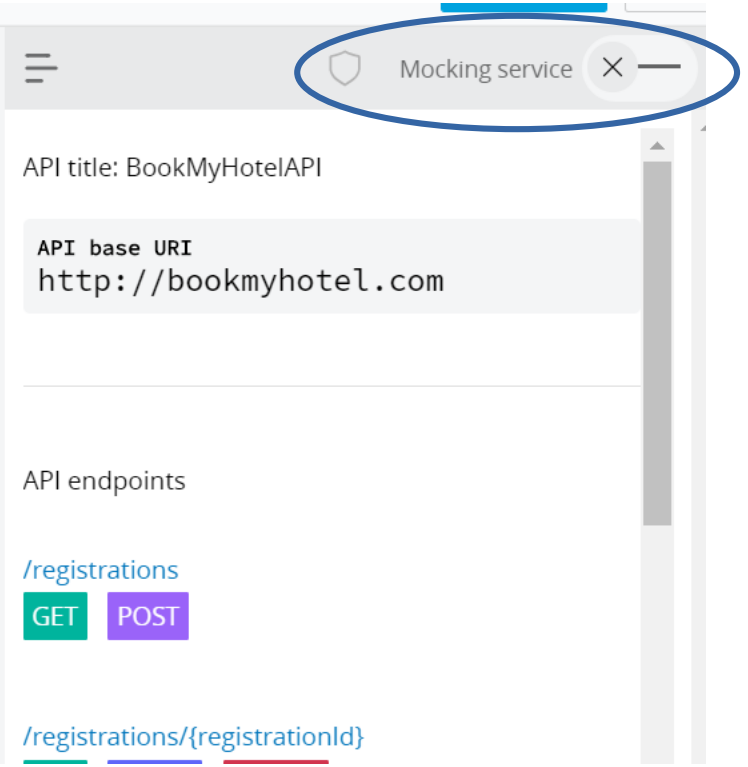
- Activate the Mocking service
  - Exposes mock service
  - Provides public Link
- Returns
  - HTTP Status Code
  - Defined Examples in responses
  - Reacts as per API Specification configuration
- Test
  - Test/Simulate in API Designer
  - Exchange after publishing
  - Browser/Postman etc
- Use
  - Testing/Simulating API Calls
  - Exploring API

Mocking Service

```
#%RAML 1.0
title: BookMyHotelAPI
baseUri: http://bookmyhotel.com
mediaType:
- application/json

types:
  Error: !include datatypes/Error.raml
  Hotel: !include datatypes/Hotel.raml
  Registration: !include datatypes/Registration.raml
  Booking: !include datatypes/Booking.raml
  Status: !include datatypes/Status.raml


/registrations:
  get:
    headers:
      Accept?:
    responses:
      200:
        headers:
          Content-Type:
```

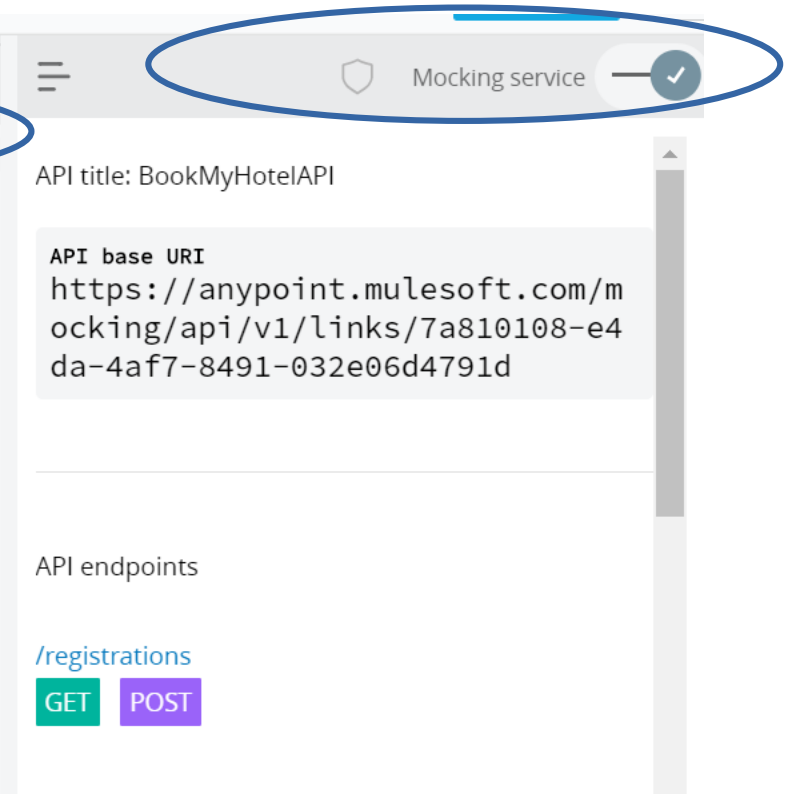


Mocking Service

```
#%RAML 1.0
title: BookMyHotelAPI
baseUri: https://anypoint.mulesoft.com/mocking/api/v1/links/
7a810108-e4da-4af7-8491-032e06d4791d/ # baseUri: http://bookmyhotel.com
mediaType:
- application/json

types:
  Error: !include datatypes/Error.raml
  Hotel: !include datatypes/Hotel.raml
  Registration: !include datatypes/Registration.raml
  Booking: !include datatypes/Booking.raml
  Status: !include datatypes/Status.raml


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



Ruchi Saini

Demo

- Enable Mocking Service
- Test Resource Methods

Documentation Fragment

Documentation

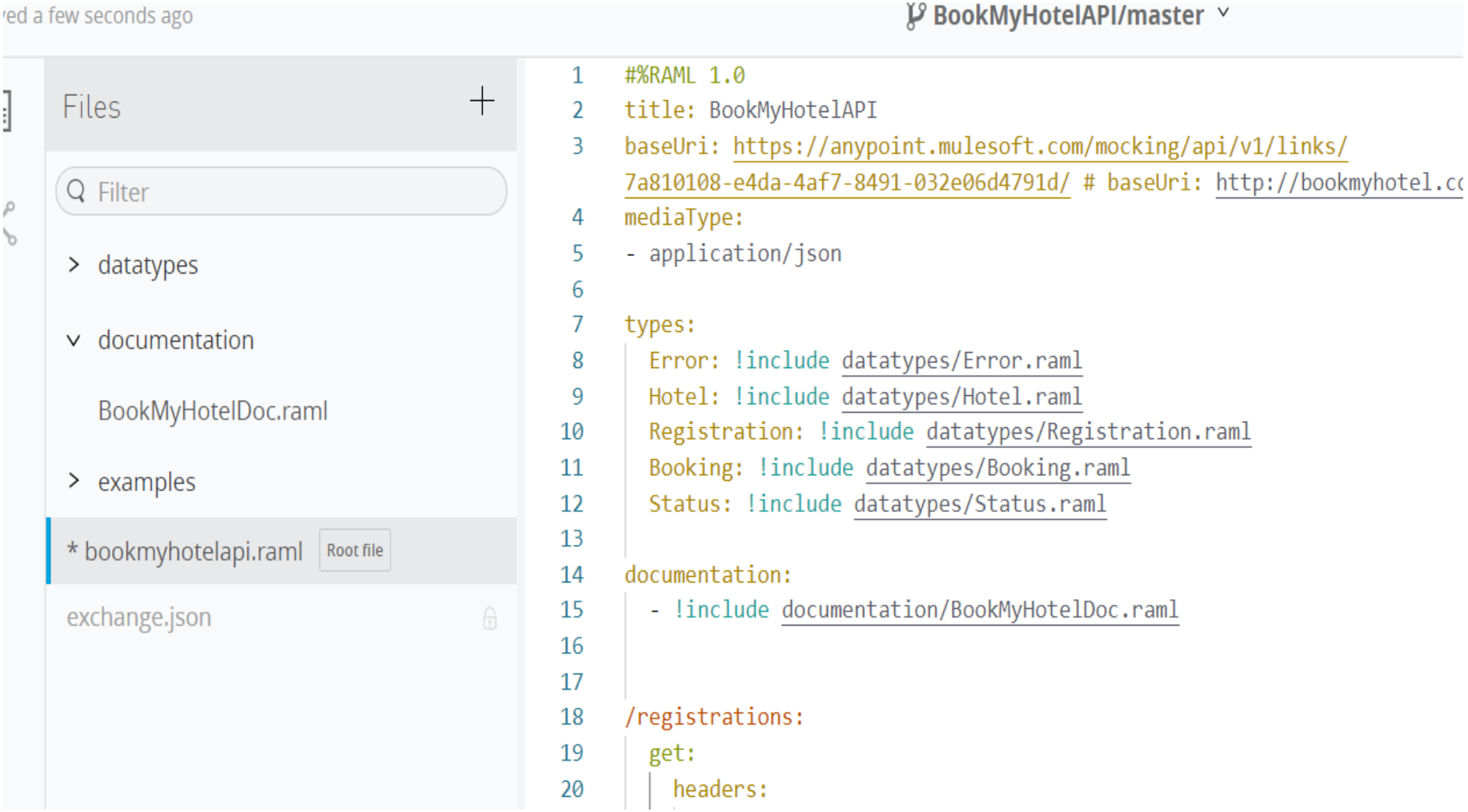
- Reference Guide for the API
- Functional and Business Context
- How API works

Using Documentation

```
1   #%RAML 1.0
2   title: BookMyHotelAPI
3   version: v3
4   baseUri: https://anypoint.mulesoft.com/mocking/api/
    v1/links/bbbbad6b-92bc-40c0-ae06-49d66445ff93/ #
    baseUri: http://localhost:9091/api
5   mediaType: application/json
6
7
8
9
10  documentation:
11    - title: BookMyHotel Documentation
12      content: BookMyHotel is a website to view
    available hotels and book the desired hotel by
    travellers.
13
14  uses:
15    bookmyhotellib: library/BookMyHotellib.raml
```

- Documentation node is a sequence of one or more documents.
- Each document is a map that MUST have exactly two key-value pairs

Using Documentation Fragment



Demo

- Create Documentation Fragment
- Include the documentation fragment in root RAML

Description and DisplayNames

DisplayName and Description

- displayName? - An alternate, human-friendly name
- description? - A substantial, human-friendly description
- Can be defined at multiple places , some of the examples are below -

Resource and Nested Resource Like /hotels. /hotels/{hotelId}	Value is Map of key –value pairs	Some of the properties defined as keys of this map :-get,put,patch , displayName and description etc
Methods: The OPTIONAL properties get, patch, put, post, delete, head, and options of a resource define its methods;	Value of these methods is a map of key –value pairs	Some of the properties defined as keys of this map :- queryParameters,headers,responses,body,displayName and description etc
DataTypes	Common Facets	displayName and description

Demo

- Define displayName and description

ResourceTypes

Repetitive/Bulky RAML

- Problem -
- Repetitive and Tedious code in RAML

Review Collection Resources

```yaml
/registrations:
  displayName: Registrations
  description: All Registrations
  get:
    displayName: GET Registrations
    description: Retrieve a list of all Registrations
    headers:
      Accept?:
    responses:
      200:
        headers:
          Content-Type:
        body:
          Registration[]
      404:
        headers:
          Content-Type:
        body:
          Error

  post:
    displayName: POST Registration
    description: Add a new Registration
    headers:
      Content-Type:
    body:
      Registration
    responses:
      201:
        headers:
          Location:
            example: http://bookmyhotel.com/registrations/r1
      500:
        headers:
          Content-Type:
        body:
          Error
```

```yaml
/bookings:
  displayName: Bookings
  description: All Bookings
  get:
    displayName: GET Bookings
    description: Retrieve a list of all Bookings
    headers:
      Accept?:
    responses:
      200:
        headers:
          Content-Type:
        body:
          Booking[]
      404:
        headers:
          Content-Type:
        body:
          Error
  post:
    displayName: POST Booking
    description: Add a new Booking
    headers:
      Content-Type:
    body:
      Booking
    responses:
      201:
        headers:
          Location:
            example: http://bookmyhotel.com/bookings/b1
      500:
        headers:
          Content-Type:
        body:
          Error
```

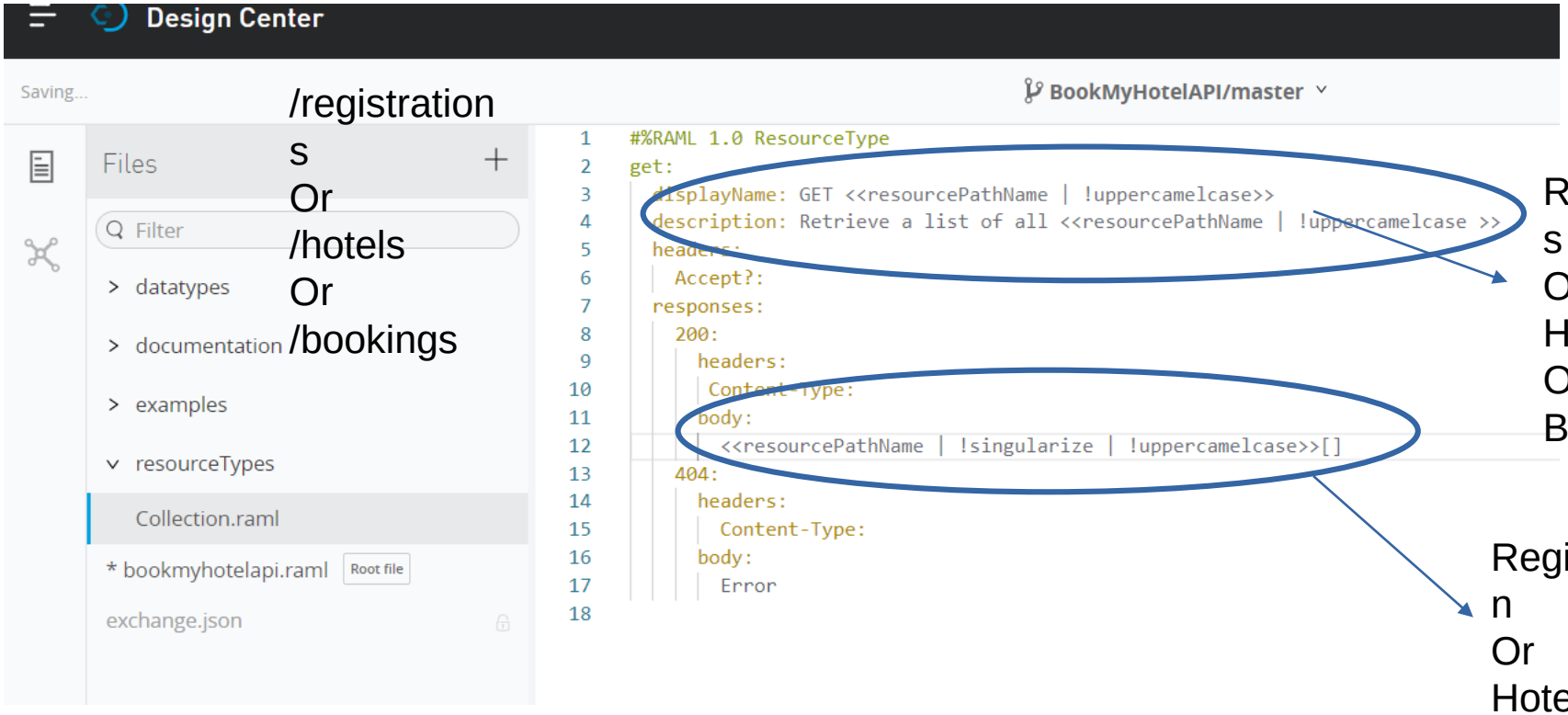
ResourceTypes

- How to solve -
  - Check emerging pattern in resource definitions
- Resource Types -
  - Create ResourceTypes using Patterns
  - Resource definitions uses the ResourceType
  - Reduces Complexity
  - Encourages Consistency

Resource Types

- A resource that uses a resource type inherits its nodes
- A resource type, like a resource, can specify
  - Methods
  - Description
  - Display Name
  - Security Schemes etc

ResourceType Example




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

/registrations:
  displayName: Registrations
  description: All Registrations
  type: collection
  get:
  post:
    displayName: POST Registration
    description: Add a new Registration
```

```
/bookings:
  displayName: Bookings
  description: All Bookings
  type: collection
  get:
  post:
    displayName: POST Booking
    description: Add a new Booking
    headers:
      Content-Type:
    body:
      Booking
    responses:
      201:
```

Parameters

- Double angle brackets enclose a parameter name in resource type
- User Defined Parameters
- Reserved Parameters
  - <<resourcePath>>
  - <<resourcePathName>>
-

URI	*ResourcePath*	*ResourcePathName*
/hotels	*/hotels*	*hotels*
/hotels/{hotelId}	*/hotels/{hotelId}*	*hotels*

Parameter Function

- !singularize
  - for example: hotels --> hotel
- !pluralize
  - for example: hotel --> hotels
- !uppercase
  - for example: hotel --> HOTEL
- !lowercase
  - for example: HOTEL --> hotel
- !uppercamelcase
  - for example: hotelId --> HotelId
- !lowercamelcase
  - for example: HotelId --> hotelId

Parameter Function

- !upperunderscorecase
  - for example: hotelId --> HOTEL_ID
- !lowerunderscorecase
  - for example: hotelId --> hotel_id
- !upperhyphencase
  - for example: hotelId --> HOTEL-ID
- !lowerhyphencase
  - for example: hotelId --> hotel-id

Inheritance

- A ResourceType can inherit from another ResourceType

```
#%RAML 1.0 ResourceType
get:
headers:
  Accept?:
responses:
  200:
    headers:
      Content-Type:
    body:
      <<resourcePathName | !singularize | !uppercamelcase>>[]
  404:
    headers:
      Content-Type:
    body:
      Error
```

ReadOnlyCollection
ResourceType with name → collection

Modifiable Collection
ResourceType

```
#%RAML 1.0 ResourceType
type:  collection
post:
  displayName: Post <<resourcePathName | !singularize>>
  description: Add a new <<resourcePathName | !singularize>>
  body:
    <<resourcePathName | !singularize | !uppercamelcase>>
  responses:
    201:
      headers:
       Location:
            description: URI to new <<resourcePathName | !singularize>>
            example: http://localhost:9090/api/<<resourcePathName>>/r1
```

Demo

- Define ResourceType Fragments and use it
-

Traits

Traits

- Extract patterns from method defintions like GET,POST etc
- Create Trait and reuse it across methods

Traits

- Traits are created using patterns from method definitions like GET,POST etc
- Method level nodes :
  - description
  - headers
  - query parameters
  - body
  - response etc

Traits

- Parameters -
  - Reserved Parameters
    - <<resourcePath>>
    - <<resourcePathName>>
    - <<methodName>>
  - User Defined Parameters
- Inheritance -
  - Inheritance is possible with Traits
- Apply Traits to -
  - Methods within Resources and Resource Types

Traits

```
#%RAML 1.0 ResourceType
get:
  headers:
    Accept?:
  responses:
    200:
      headers:
        Content-Type:
      body:
        <<resourcePathName | !singularize | !uppercamelcase>>[]
    404:
      headers:
        Content-Type:
      body:
        Error
```

- Error HTTP Status codes with Error body is used in all methods

Traits

```
traits:
  hasErrorResponse:
    responses:
      <<errorCode>>:
        headers:
          Content-Type:
        body:
          Error
```

Traits

```
get:
   displayName: Get <<resourcePathName | !singularize>>
   description: Get a <<resourcePathName | !singularize>>
   is:
     - hasErrorResponse: {errorCode: 404}

   headers:
    Accept?:
   responses:
    200:
      headers:
          Content-Type:
      body:
         <<resourcePathName | !singularize | !uppercamelcase>>[]
```

Demo

- Define and use Trait Fragments

Library

Library

- Library has the collection of declarations of any or all of following -
  - Data type declarations
  - Resource type declarations
  - Trait declarations etc
- Library helps in -
  - Defining common declarations in external file or inline
  - Modularize any number and combination of data types, resource types, traits etc.
  - Reusable

Library

```
1   #%RAML 1.0 Library
2   usage: Define all Traits
3   traits:
4     hasErrorResponse:
5       responses:
6         <<errorCode>>:
7           headers:
8             Content-Type:
9           body:
10            error
11
12    hasAcceptHeader:
13      headers:
14        Accept?:
15          description: expected mediaType of response
16          example: application/json
17
```

Library Fragment

Library

```
uses:
 traitsLib: library/BookMyHotelLib-Traits.raml
```

```
post:
  displayName: Post <<resourcePathName | !singularize>>
  description: Add a new <<resourcePathName | !singularize>>
  body:
    <<resourcePathName | !singularize>>
  is:
    - traitsLib.hasErrorResponse: {errorCode: 500}
  responses:
    201:
      headers:
        Location:
```

ResourceType

Demo

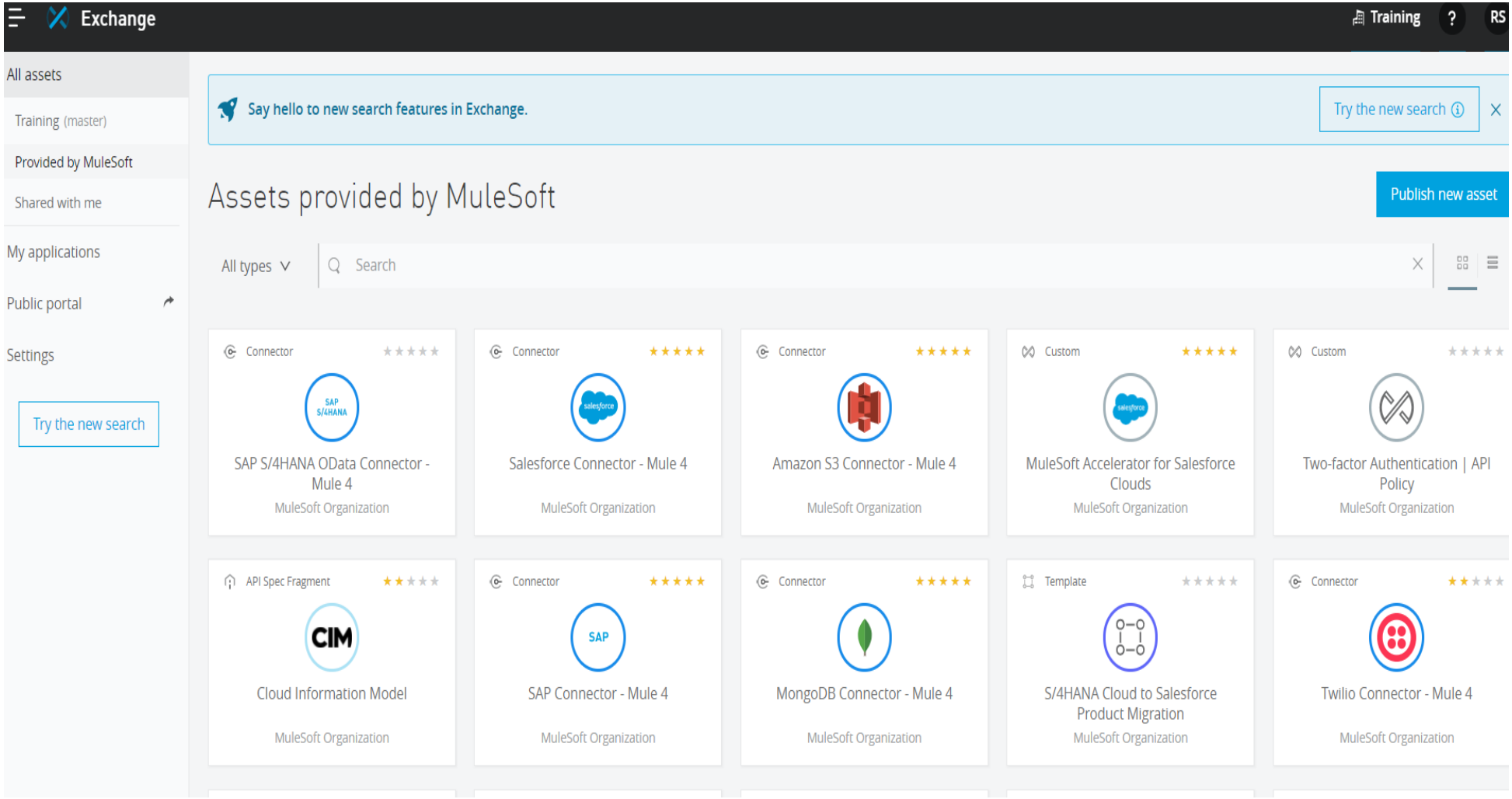
- Create and use Library Fragments

Anypoint Exchange

Exchange

- MarketPlace
- Assets are published and reused by Organization
- Asset is a maven artifact
- Assets
  - APIs
  - API Group
  - Fragments
  - Connectors
  - Examples
  - Templates
  - Custom

Exchange



Ruchi Saini

Exchange

- Exchange
  - Publish
    - Design Center
    - Maven
    - API Manager
    - Directly from Exchange
    - etc
  - Consume
    - Anypoint Studio (connectors, templates, and examples),
    - API Manager (APIs, API Groups, and policies),
    - etc
- Exchange
  - Public - By MuleSoft
  - Private – within organization/Business Group
  - Shared with me

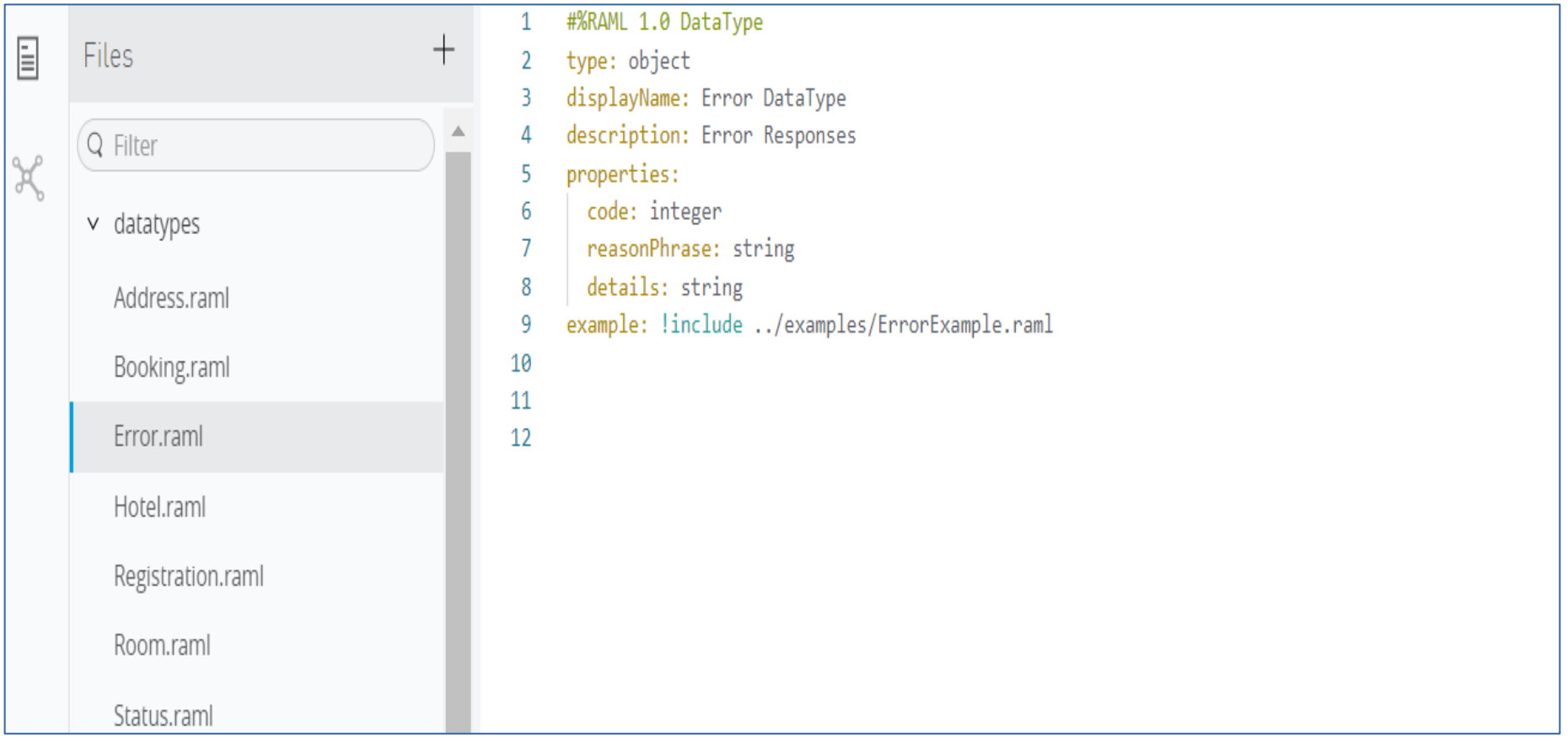
Publish and Consume API Fragment

Ruchi Saini

API Fragments

- An API fragment is a RAML document but is not in itself a complete RAML specification
- API fragments are of types -
  - Type
  - Resource Type
  - Library
  - Trait
  - User Documentation
  - Example
  - etc

DataType Fragments



Demo

- Publish Error DataType Fragment
- Consume Error Fragment
- Republish Fragment and change version in API Specs

Publish API Specification To Exchange

Ruchi Saini

Demo

- Publish API Spec to Exchange

Sharing Asset within Organization

Ruchi Saini

Share Asset

- Asset can be shared with -
  - Another user in your organization
  - Your Whole organization
  - External organization

Demo

- Share BookMyHotel API to others users within same organization

Public Portals

Public Portals

- Private Exchange → Share Assets Within the Organization
- Public Portal → Share API Assets With External Users
- Add/Delete API Asset from Public Portal

Demo

- Share BookMyHotel API to public portal

API Notebook

Ruchi Saini

API Notebook

- API Notebook
  - Web based tool
  - Generates a Client for API using RAML definition
  - Calls endpoints on Mocking Server or Real Server
  - JavaScript
- Helpful
  - Interactive Tutorial
  - Examples
  - Use Cases

API Notebook

- Create an API client in a Notebook using the global method API.createClient
- Path Segments of Resources become nested objects
- Hotels with relative URI - /hotels
  - {clientName}.hotels
- Hotel with relative URI -/hotels/{hotelId}
  - {clientName}.hotels.hotelId({hotelIdValue})
  - For eg. Client.hotels.hotelId(‘h1’)

API Notebook

- Call methods on Object
  - {clientName}.hotels.get()
- Request Parameters and Headers can be passed
- Response body, status and headers are received in the response

Demo

- Create API Notebook

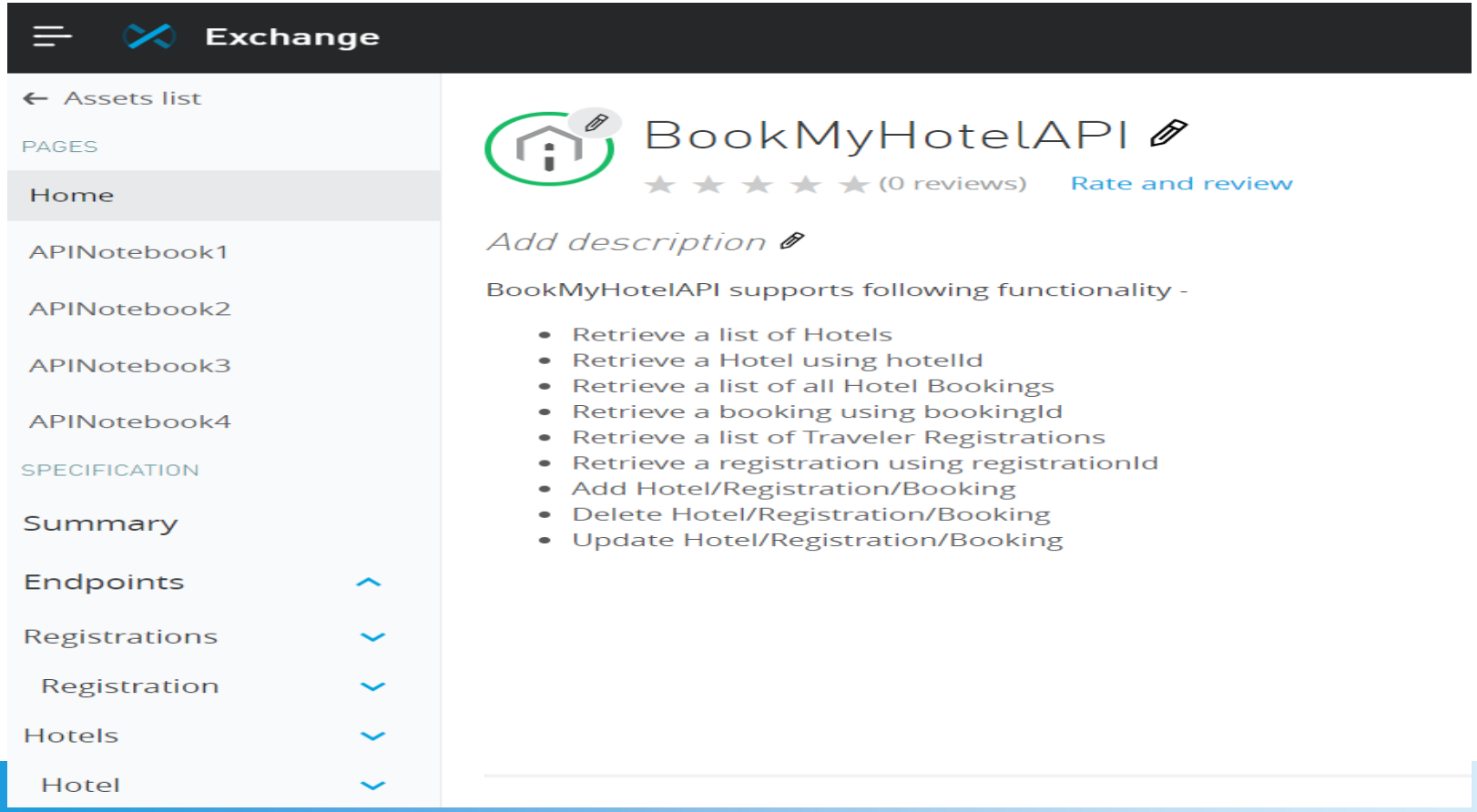
Versioning APIs

Versioning API

- When to Version API?
- Version Change
  - If changes break/fails the consumer and provider communication
  - For example :
  - Deleting a resource or method from existing API interface and implementation
  - Change in the representation format
- No Version Change
  - No version change during technology change in backend application or data model changes
  - No version change when new resources or methods are added

Versioning API

- Publishing new version to Exchange
  - Carry forward the documentation of previous version
  - Make any new changes as per requirement



Versioning API

- Deprecate the older API versions in exchange
- Deprecation helps in preventing further usage of older versions

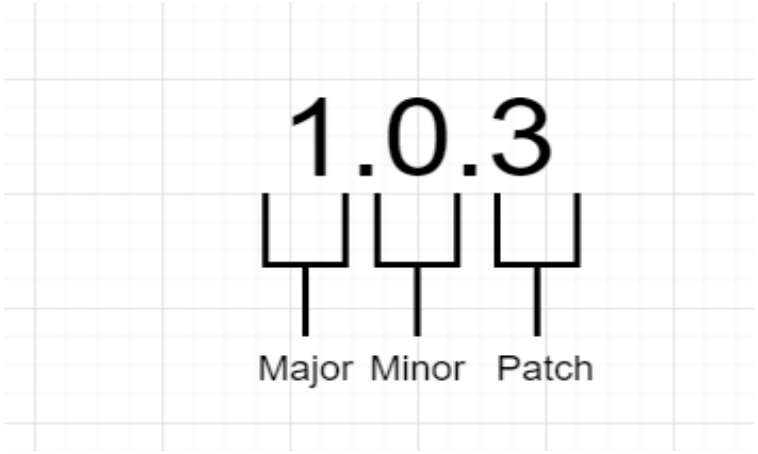
Versioning API

- Each API has API version
- Each Exchange asset has Asset version
- For each API version, multiple asset versions can be published.
-

API Version	Asset Version
v1	1.0.0
v1	1.0.1
v1	1.1.1
v2	2.0.0
v2	2.0.2
v3	3.1.0

Versioning API

- Asset Version ->
  - Exchange asset versions follow the Semantic Versioning
  - Semantic Version refers to model of major, minor, and patch releases
  - Format : A.B.C where A stands for major version, B for minor version and C for patch i.e. Major.Minor.Patch
  - For example, if an asset is of version 1.0.3,



Versioning API

- Major versions
  - When introducing a change in the structure of the API
  - Requires the user of the API to adapt the interface on the consumer side
- Minor versions
  - Introducing a backward compatible changes in the API
  - Some new feature which does not break the API
  - Does not require an API user to change
  - Eg: new optional element
- Patch versions
  - Backward compatible
  - Eg: bug fix

Demo

- Version API