[Mule 4: Design API with RAML 1.0](https://deloittedevelopment.udemy.com/course/mule-api-design-with-raml/)

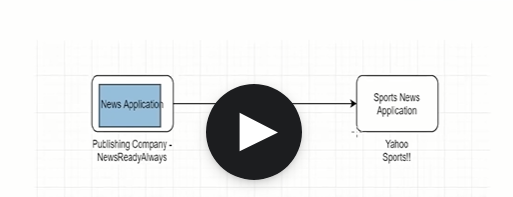
-by Ruchi Saini

* **Intro to API**

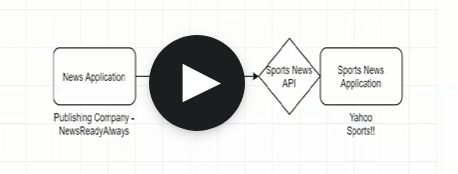
-API stands for Application Programming Interface

-API Is an interface of your software application for external world.

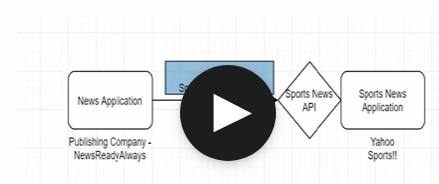
-An external application uses your API to send request to your software app and gets response from your Software app.



In above digram,let say NewsReadyAlways is a news publishing company which hosts News Application in which they provide content related to politics,Hollywood.bollywood,National,Sports.For Sports News they don.t want to create an app which shows news about sports but they want to use other companies in this case Yahoo Sports’s Sports News Application to show sports news on NewsReadyAlways’s News Application so for that they need to discuss with Yahoo Sports and need to create an API(Sports News API) which takes all sports related news from Yahoo Sports’s Sports News App to NewsReadyAlways’s News App.



To Connect with Sports News ApI the News application of NewsReadyAlways’s app needs to know what kind of Request Sports News Accepts,What Parameters it accepts ,What Response the Sports Api sends back to News App.All this will be decided n documented in a doc called API Specification between NewsReadyAlways co n Yahoo Sports co.This is mutual contract between those co which we can called it as Api Specification.

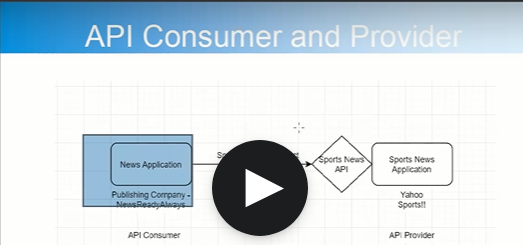


So basically here An API is a interface of an your software app that means Your Sport News Api is an interface of Sports News App which allows communication i.e Request and Response between two systems here it is News App n Sports News App.To get the information from Sports Api,extrenal app here News App should know what type of request needs to be sent to Sports Api and what type of Response it expects from Sports Api.

* **API Consumer and API Provider**

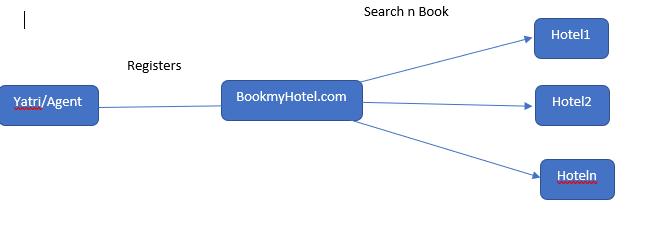
API Consumer is a external system which consumes or gets the response from the API here NewsReadyAlways is a API Consumer.

API Provider is a system which provides or Accepts the request from extrenal system thourgh its API.Here Yahoo Sports is a API Provider

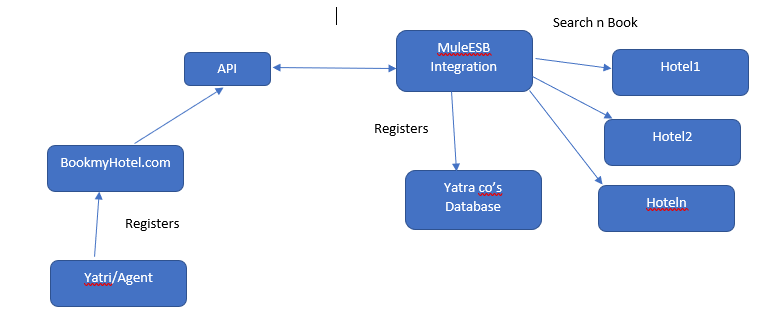


* **UseCase**

A travel co called Yatra want to build a website called as BookmyHotel.com where a yatri who wants to do yatra will register himself and then he can search n book hotel.He can search n book hotel by using city/state/country filter.The integration between the BookmyHotel.com and different hotels is done by MuleEsb Integration .



The Whole picture looks like below img:-



* **Intro to Rest/RAML n Identify Resources for the BookmyHotel.com- Use case**

**-**Types of diff API Standards

Soap API

Rest API

RPC API

-REST API stands for Representational State Transfer

-REST is Http Application Protocol means accepts request in HTTP methods like get,post,put,delete,patch n provides response in HTTP Status code like 201,200,400,405,401 etc.

-Rest is Resource based n not action based that means it doesn’t use methods like gethotel() rather it uses noun like hotel,registration,booking etc

-Rest is Uniform interface where

Request can be

* GET is used for getting current status of resource
* POST is used to add resource
* DELETE is used to delete resource
* PATCH is used partially update the resource
* PUT is used to completely update the resource

Response can be

* HTTP status and Body

-Rest is representational that is we can specify any format like xml,json etc. let say we have resource=Hotel Method-Get Representation or media-type=application/json.

-Rest can be stateless,no state is maintained at server side.

* **RAML**

-RAML stands for RestFul API Modeling Language

-API designed n described by api specification and API Specification for REST API is created by RAML ,Open API

-RAML is built on YAML(Yet another Markup Language)

-Older version was RAML v 0.8

-Latest version is RAML v 1.0 which uses YAML 1.2

* **To Identify the resources in the BookmyHotel.com use case**

1. Collection Resource:Represents collection like

* Registrations- represents collection of all registrations
* Bookings- represents collection of all the bookings
* Hotels- represents collection of all the hotels

1. Nested Resource:Represent specific resource like

* Registration: represents specific registration based on specific identifier like registration\_id
* Booking: represents specific booking based on specific identifier like Booking\_id
* Hotel represents specific hotel based on specific identifier like hotel\_id
* MAP actions to HTTP Methods

|  |  |  |  |
| --- | --- | --- | --- |
| Resource | Action | Method | Resource Path |
| Registrations | Get list of all registration | GET | /Registrations |
| register new yatri/agent | POST | /Registrations |
| Get particular registration | GET | /Registrations/{registration\_id} |
| Delete particular Registration | DELETE | /Registrations/{registration\_id} |
| Update particular resource | PATCH | /Registrations/{registration\_id} |
| Hotels | Get list of all Hotels | GET | /HOTELS |
| register new Hotel | POST | /HOTELS |
| Get particular hotel | GET | /HOTELS/{hotel\_id} |
| Delete particular hotel | DELETE | /HOTELS/{hotel\_id} |
| Update particular hotel | PATCH | /HOTELS/{hotel\_id} |
| Bookings | Get a list of bookings for a particular hotel | GET | /HOTELS/{hotel\_id}/Bookings |
| Bookings | Get list of all Bookings | GET | /Bookings |
|  | register new Booking | POST | /Bookings |
|  | Get particular Booking | GET | /Bookings/{booking\_id} |
|  | Delete particular Booking | DELETE | /Bookings/{booking\_id} |
|  | Update particular Booking | PATCH | /Bookings/{booking\_id} |
|  |  |  |  |

* **API Designer**

-Anypoint platform provides tools for design,manage,deploy the API.

-Design center provides tool for designing the API n Designing the integration solution.

-API Designer helps to create the API Specification

-Flow Designer helps to design the integration solution.

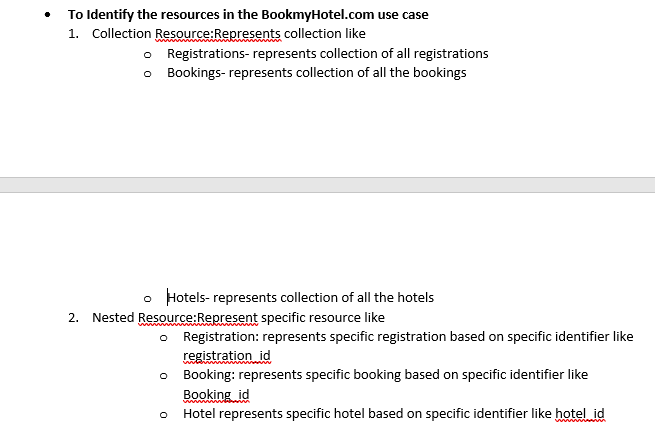
-API Designer helps in

* Creating API Specification
* Reusability-Provides concepts like traits,resource types, fragments
* Validation-Validates api before the implementation by mocking service
* Documentation-creates documentation
* EXCHANGE-Used for publishing the API
* User Feedback-Used for taking the user feedback

XXXXXXXXXXXXXXXXXXX LAB1.PDF XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

* **Define Resources**

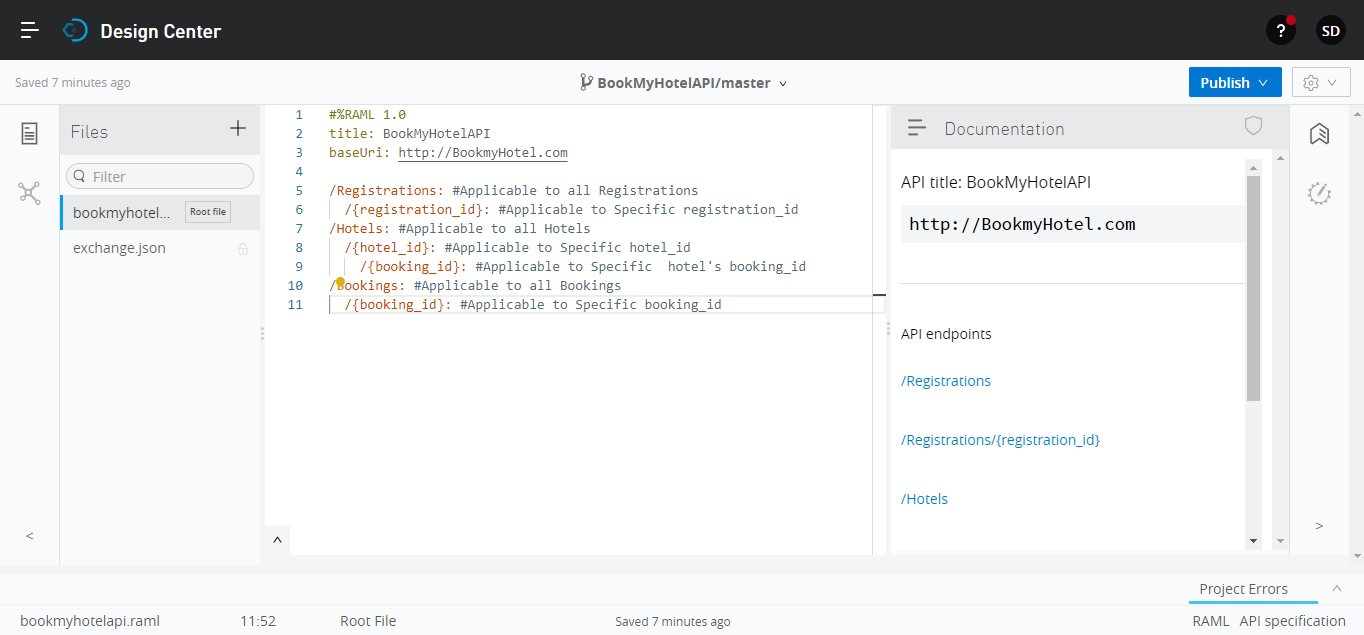
-As per our use case we identified resources that we need



-Resources are identified by URIs

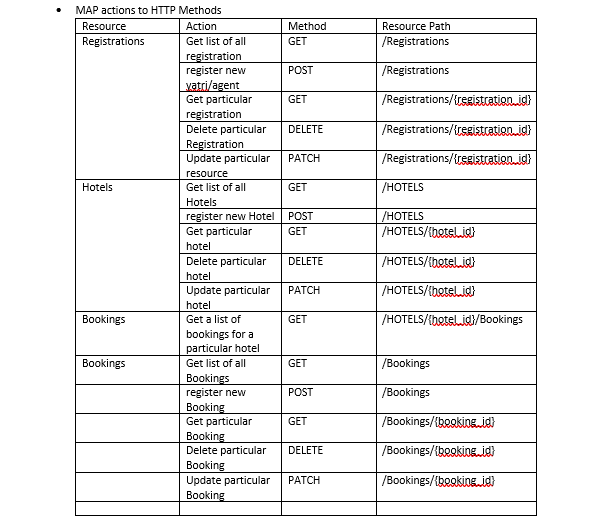
Baseuri:-http://localhost:8080/

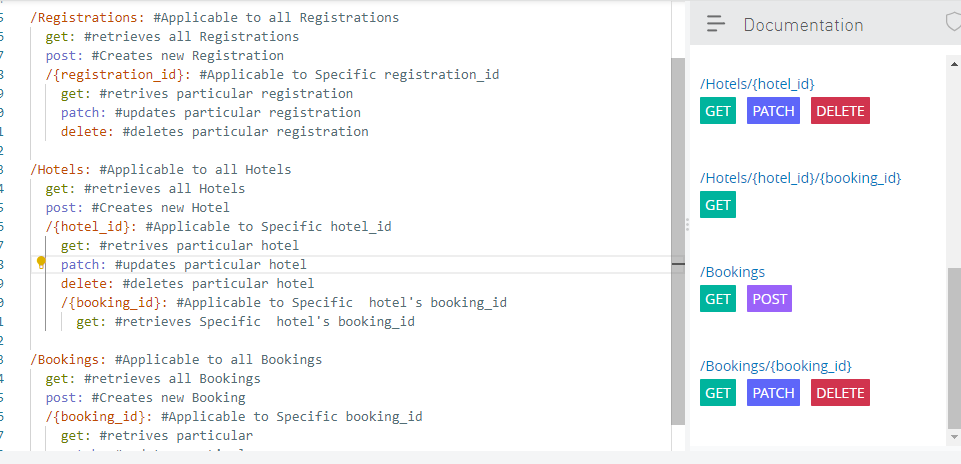
|  |  |
| --- | --- |
| Resource | Resource Path |
| Registrations | /Registrations |
| Hotels | /Hotels |
| Bookings | /Bookings |
| Registration | /Registrations/{registration\_id} |
| Hotel | /Hotels/{hotel\_id} |
| Hotel’s Booking | /Hotels/{hotel\_id}/{booking\_id} |
| Booking | /Bookings/{booking\_id} |



Note:whatever we specify on the root portion of RAML file is applicable to all by default if not specified at that level.# is used for single line comment in RAML .yaml or RAML doesn’t support multiline comments

* **Define Methods**





Note:Method names should be in small case.

* **Media-Type**

-Representation of the Resource is done by Media-Type.

-Media-Type can be in JSON ,XML or any other format.

-Eg: application/json,application/xml etc

-Media-Type can be defined for Request Body and Response body in RAML.

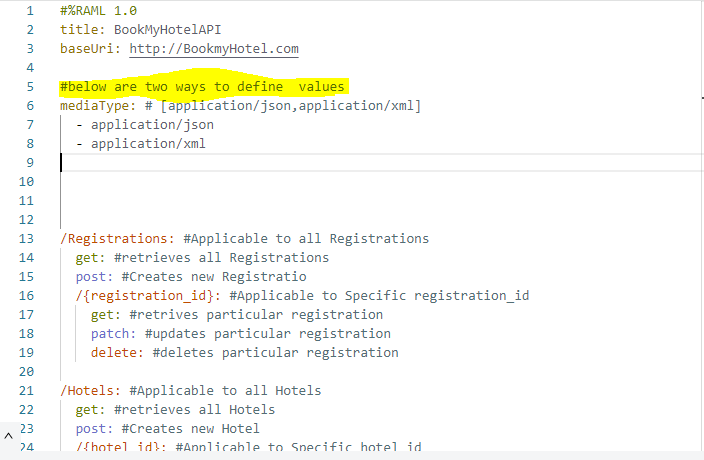
-Media-Type headers are:

* Content-type: application/json

Indicates that the request is in json format.

* Accept:application/json

Indicates that the response should be in json format



* **Query Parameters**

-Query Parameter is used to sort or filter the resource.

-Query Parameter is mentioned by ?param1=value1&param2=value2&paramn=valuen

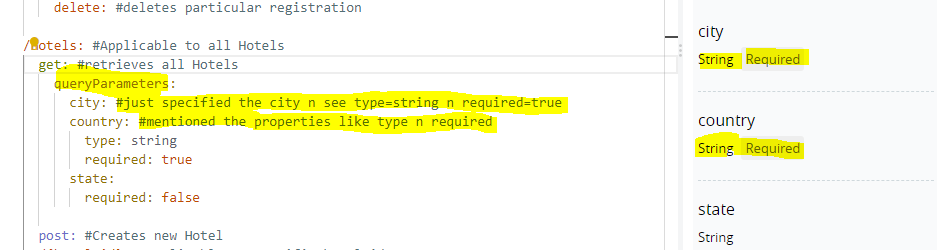
For eg:Base URI=http://BookmyHotel.com/Hotels?city=bangalore

-It is part of the URL

-It is optional.

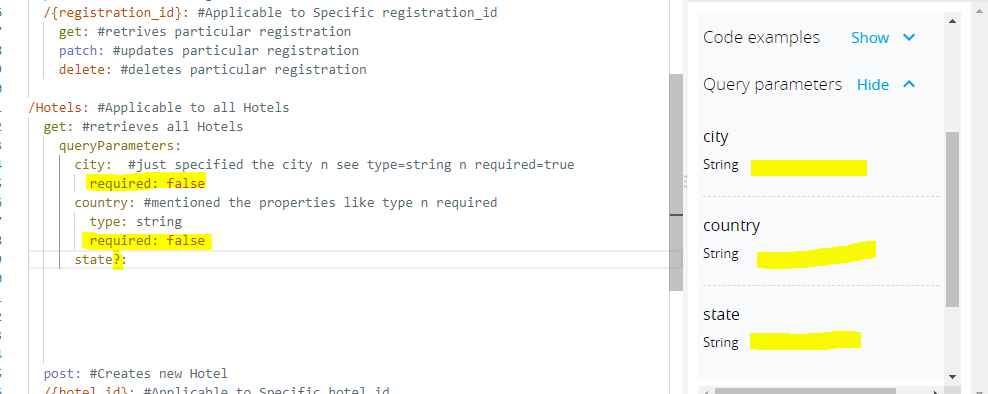
-In our BookmyHotel use case the 3rd requirement was like yatri/agent should be able to search a hotel with country,city,state filter so for doing this we have to add below query parameters city,country,state in /hotels resource in get method

-Ways to define query parameter

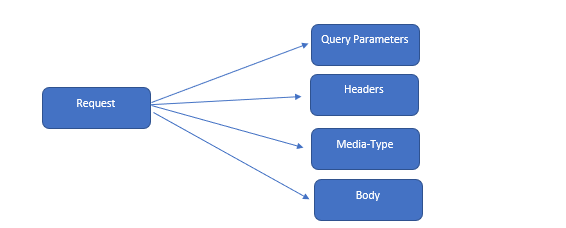


Note:- By default all query parameters are of datatype string(type=string) and they are mandatory(Required=true)

* Two Ways to make required=false is below

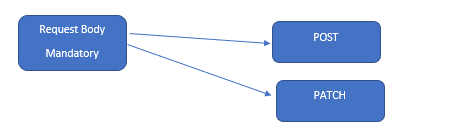


* **Defining Request and Response in Method**

-Request can be made in Query Parameters,Headers,Media-type,Body etc. These are optional.

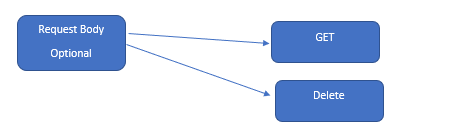
-Request Body can be mandatory in case of we are defining POST,PATCH method,this is because

* For POST,we are creating a new record so the API expects record details which consumer needs to be send with request
* For PATCH,we are updating a record so the API expects RECORD DETAILS which consumer needs to send with request

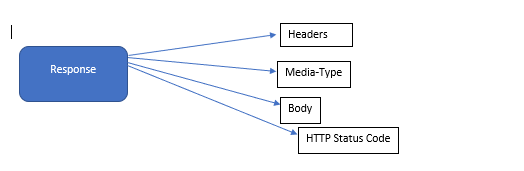


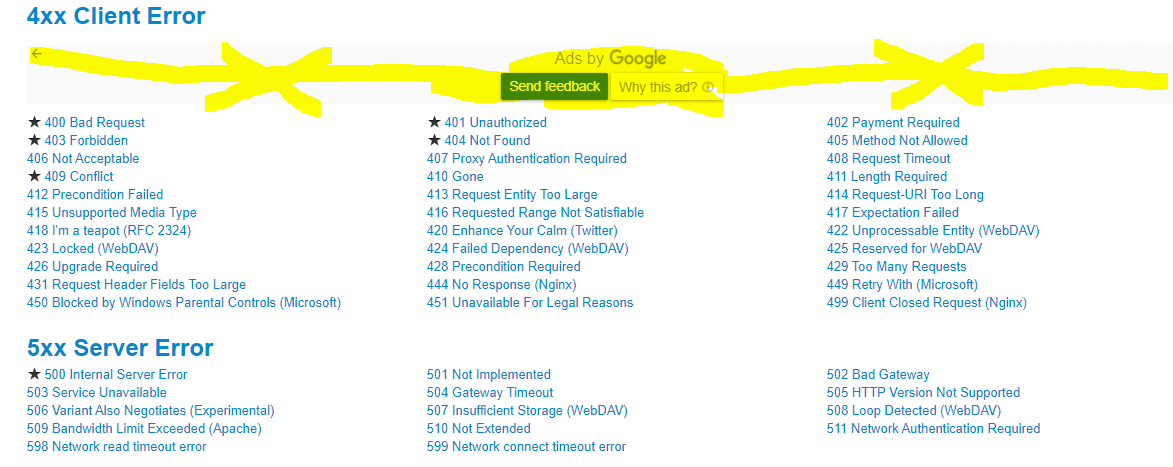
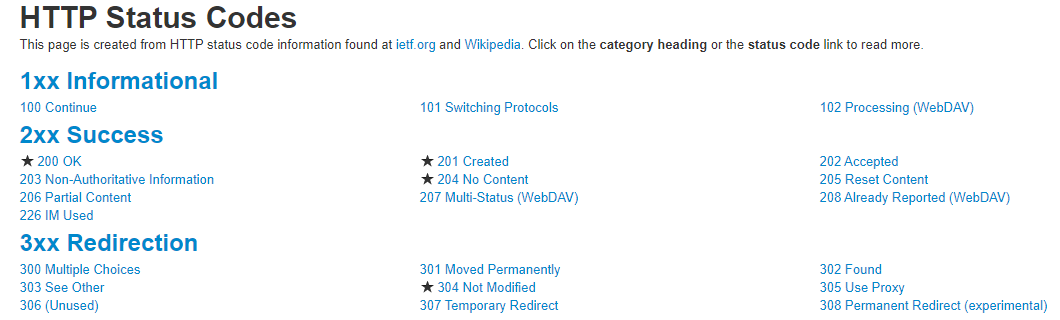
-Request body can be optional if we are defining GET,Delete this is because

* For GET,we are RETRIVING THE VALUE of record so the API doesn’t expects record details which consumer needs to be send with request, we can send particular Record ID with URI Parameter.
* For Delete,we are deleting a record so the API doesn’t expects RECORD DETAILS which consumer needs to send with request,we can send particular Record ID with URI Parameter.



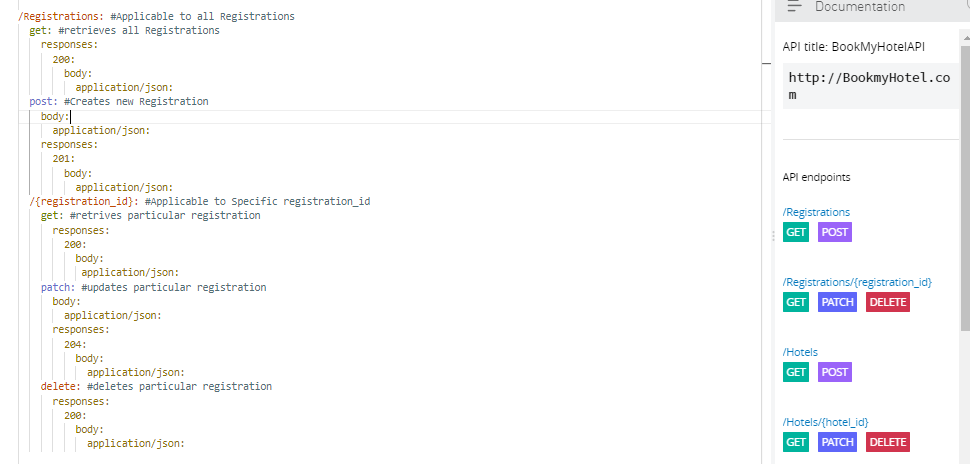
-Response can be made in Query Headers,Media-type,Body,HTTP Status code etc.HTTP Status Code is Must all can be optional





- HTTP Status Code is mandatory in each method ,whether it is GET,DELETE,POST,PATCH but the Response Body can be only mandatory in case of we are defining GET,DELETE method,this is because

* For GET,we are RETRIVING THE VALUE of record(URI Parameter) so the API provider needs to send the response based on Record ID with URI Parameter.
* For Delete,we are deleting a record (URI Parameter) so the API provider needs to send the response based on Record ID with URI Parameter..



Note:-

Request Body is must in POST ,PATCH.

Response Body is must in GET,DELETE.

HTTP STATUS CODE must be there in every Response.

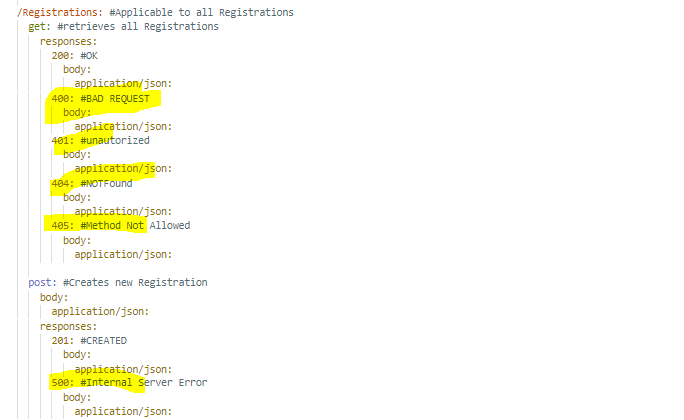
* **Error Response**

**-**Error Response can be of

Client Error—4xx Status Code eg:400,401,405 etc

-Occurs when the request from consumer to Provider is not correct

Server Error-5xx Status Code eg: 500,501,502 etc.

-Occurs when error happened at provider end . 

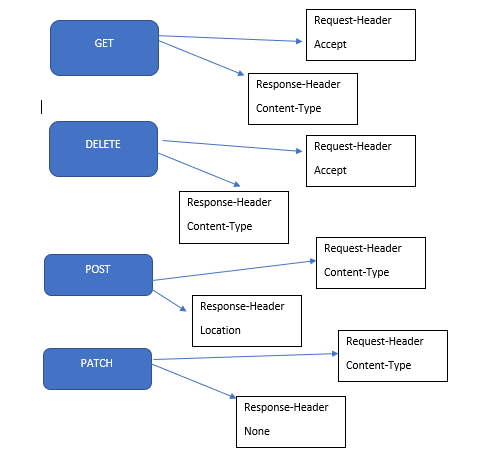
* **Define Headers**

-Request Headers are of two type

* Accept-Tells the provider the response is expected in this value format.eg: Accept:application/json .can be used for GET,DELETE
* Content-Type-tells the provider the format of request.eg: Content-Type:application/json.Can be used for POST,PATCH

-Response Headers are two type

* Location-specifies uri of newly created resource.Can be used for POST
* Content-Type-specifies media-type of response.can be used for GET,DELETE





* **DATA-TYPES:**

-Describes data in API.

Eg:-Assume we create a Person Datatype which has collection of fields of different datatype such as name as string,Age as number,address as string

Person

{

Name:string

Age:number,

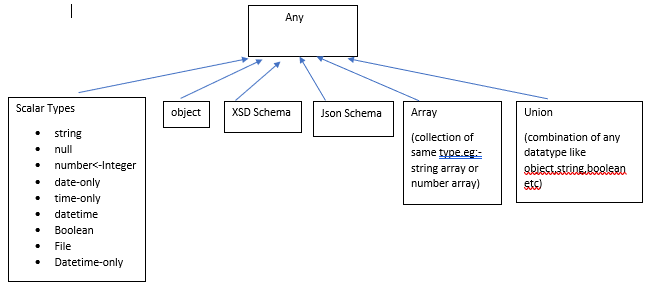
Address:string

}

-Add rules to validates data against a type declaration.

So any Person data cannot be having value like phoneno or email id,it will have only name,age,address data assosciated with Person datatype.

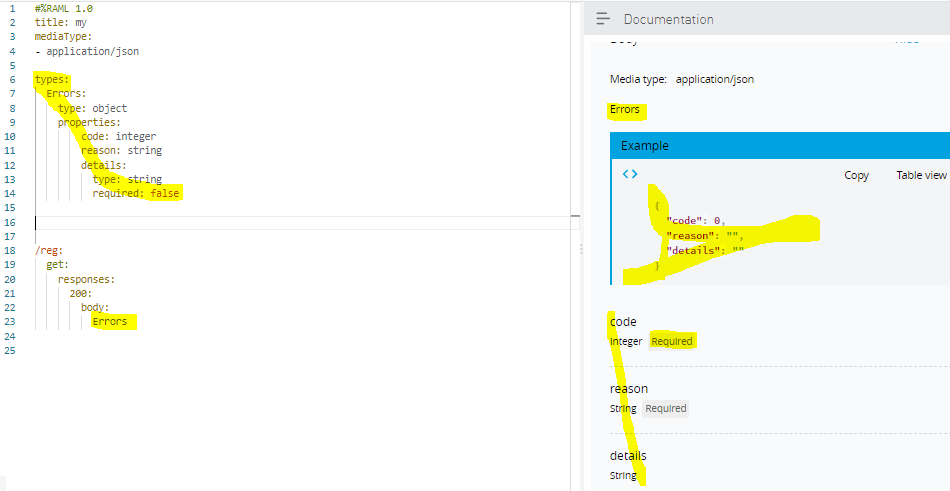
-Built-In Datatypes



-Custom Data Type creates by extending built in Datatype

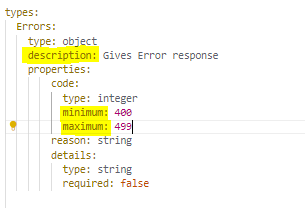
-Example of Custom datatype

Raml code:



* **Facets**

-Facets are used to add some constarints or special configurations eg description,minimum,maximum etc.



-Few common facets for datatype declaration is default,description,enum,example etc

-some are specific to type

Object-Properties,minproperties,maxproperties etc.

Array-uniqueitems,minitems,maxitems

String-pattern,minlength,maxlength etc

* **Fragments**

**-**Fragment is portion of Api Specification (part of Raml Code) which we do not keep in ROOT RAML file to provide good reusability,readability, n clear Root File

-use **!include location\_of\_fragment\_file** to add reference of fragment file in Root API Specification File

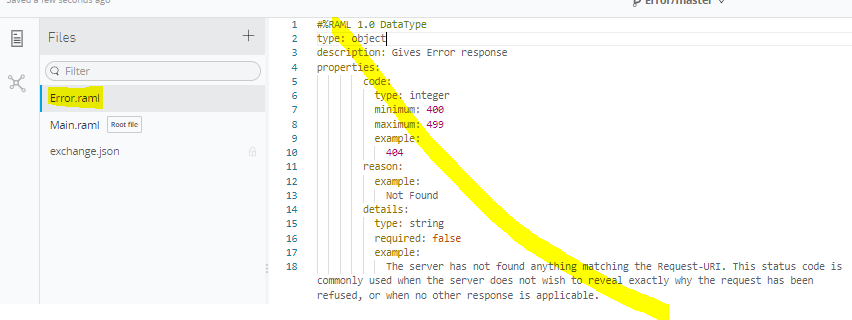
-If you create a fragment file,first line indicates which type of Fragment file is that with #%RAML 1.0 Fragment\_Identifier eg:

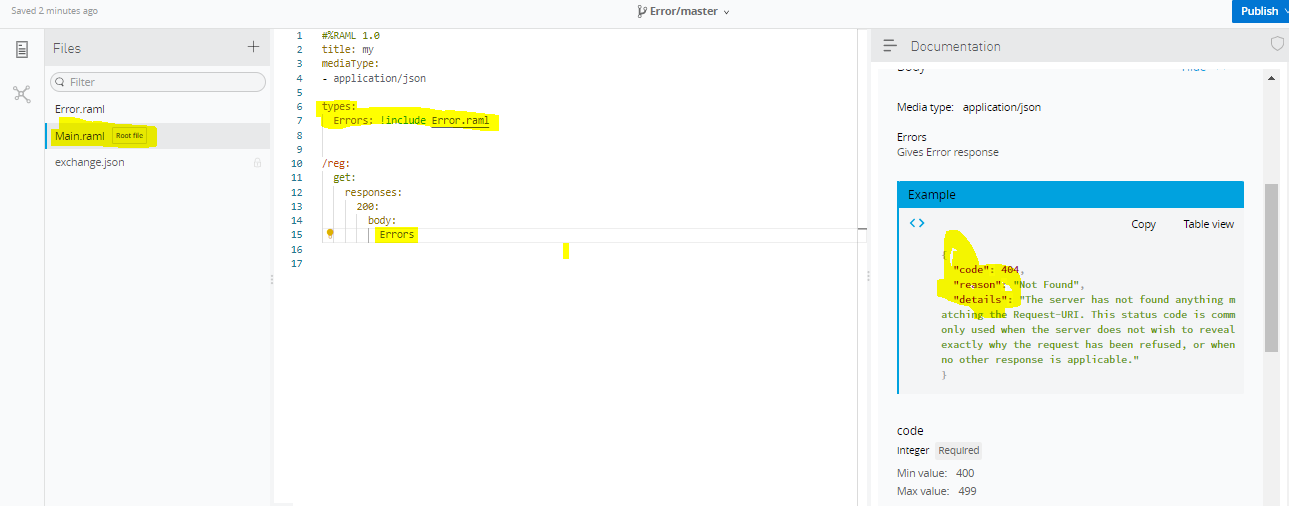
#%RAML 1.0 DataType

-Eg for **RAML Fragment using datatype fragment**

Fragment file name: Error.raml

Main Root File: Main.raml





* **Use Case Datatype**

****

* **Example Fragment**

-Defines example for instance of datatype declaration.

-We can attach single or multiple examples using Examples,Example.

-How to define example?

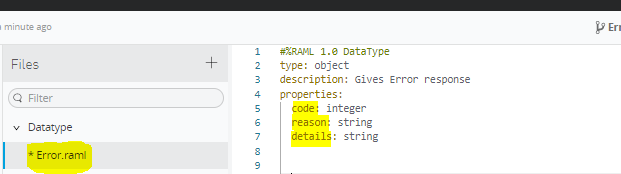
>Create Example fragment n

>then in datatype fragment, we use example or examples facets with !include location\_of\_Example\_Fragment

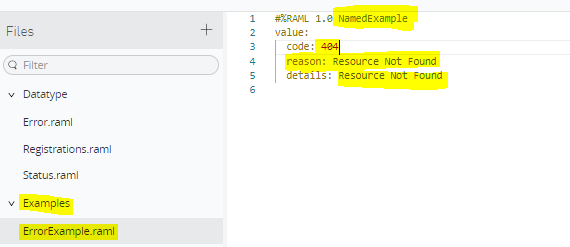
-We can use either use example or examples facets both can not be used

Example Fragment ---Error Datatype

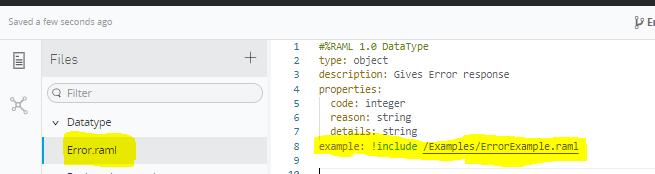
Error.raml datatype fragment file before ErrorExample.raml example fragment file



ErrorExample.raml example fragment file



Error.raml datatype fragment file after ErrorExample.raml example fragment file

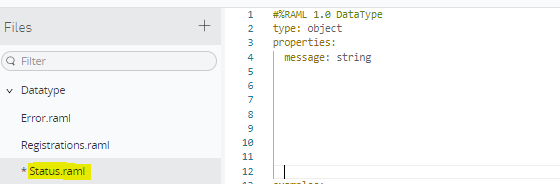


Output of Example in Root Main.raml file

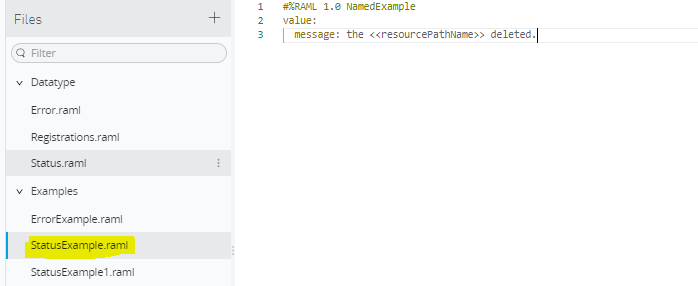
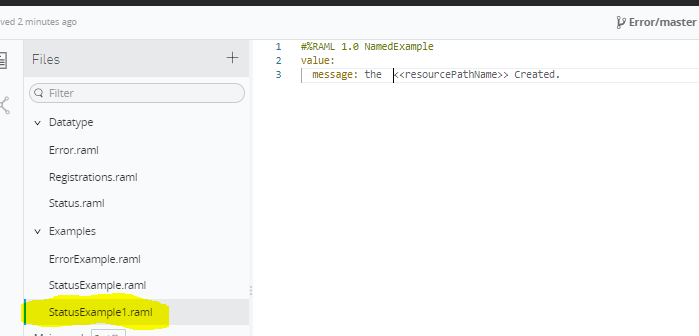


Examples Fragment

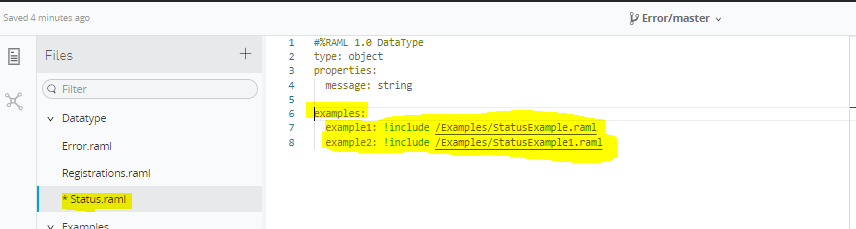
Status.raml before creation of Status Examples file



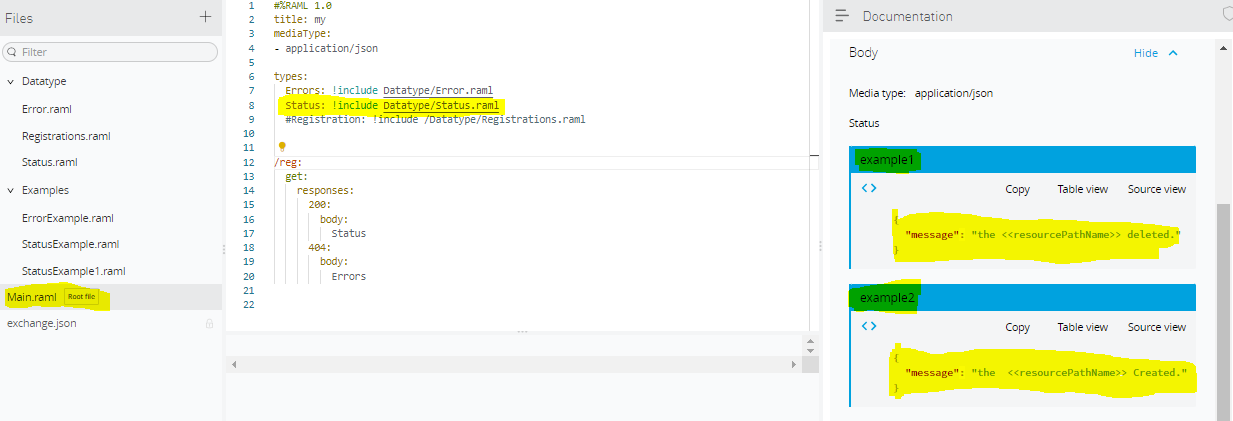
Creation of Example files



Adding examples reference in Status.Raml atatype file



Output in Main.raml Root file

Note: While creating Example file the **value:** key only should be used otherwise will get error like required field’s value is needed.

* **Intro to Mocking Service**

-Mocking Service means without actual implementation of an API we can test our API with Mocking service by using API Specification data like HTTP Status Code,Defined examples in response.

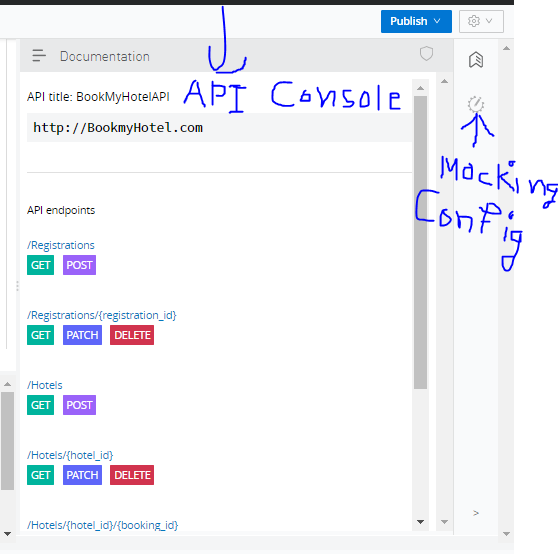
-We can test our API with API Designer,Postman,Browser,Command Prompt,Anypoint Exchange

-Why we use Mocking Service is for testing api calls n exploring API

* **Testing Mocking Service**

-Earlier we need to slide Mocking service to enable the mocking service so that it comments our mentioned baseuri n provides new proxy BaseURI for testing.

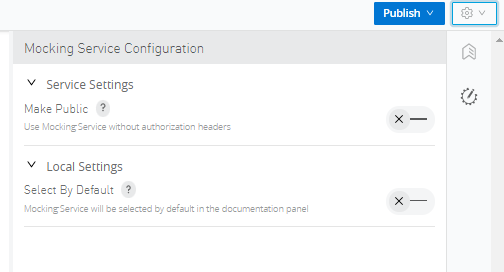
-Now we get mocking service permanently enabled so that we don’t see slider option to enable mocking service/Disable the service + we don’t get Mocking service baseuri in API Specification.



-Two options in Mocking configuration item

1.Service Setting(Make Pubic):we can enable this service n the generated url we can use for testing the API outside the Anypoint platform like on Postman,ARC(Advanced Rest Client).Each time e enable n disable the service new url is generated so use latest url

2.Local Settings(Select by default):we can enable this service so that each time we click on documentation icon by defalut Mocking service is selected instead of Specified baseuri mentioned in API Specification.



* **Documentation**

**-**Documentation is reference Guide for API

-We can add functional n business context in documentation

-It can be used to specify how the developed API Works

-We can add Documentation in our Root Raml file but It is a best practice to create Documentation fragment n use that reference in Root Raml file.

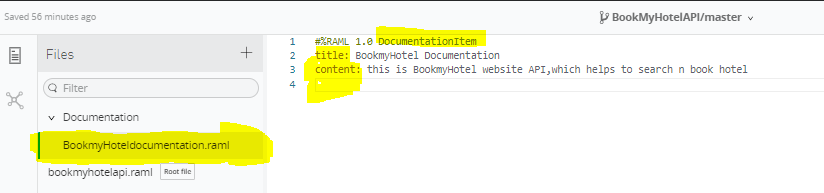
-For documentation to use ,we must specify the two values

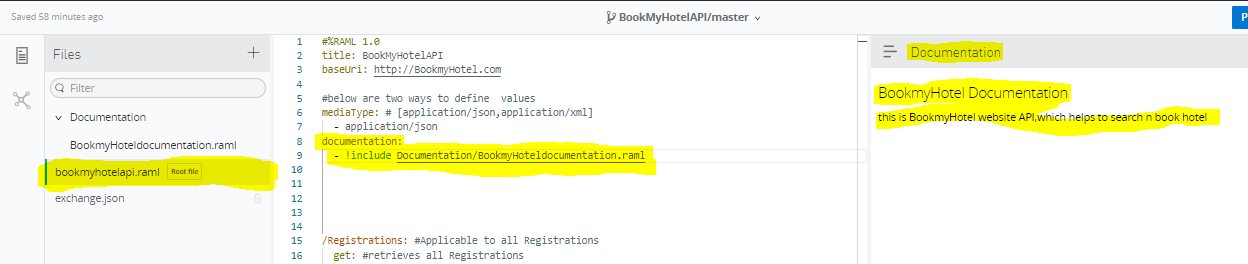
1.title: specify doc title

2.content:Specify Content

-To create documentation we have to select user documentation for creating the this type of fragment

-Fragment Identifier= DocumentationItem





Displayname:an alternate,user friendly name

Description:human friendly description

-we can specify Description n Displayname at Resource level,Nested Resource Level ,Method Level,DataType Level,ResourceTypes Level

Eg:

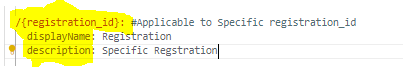
At resource Level,



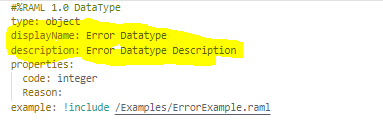
At Method Level,



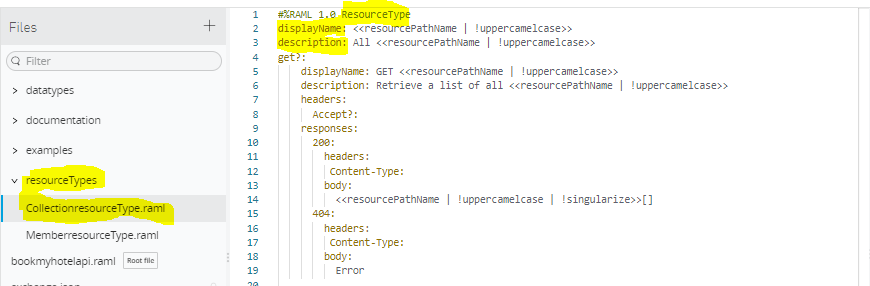
At Nested Resource Level



At DataType Fragment Level



At ResourceTypes Level,



* **Resource Types**

-is used to eradicate tedious n repetitive code from RAML.

-ResourceTypes is like resource in that it can specify the descriptions, methods, and its parameters. Resource that uses resourceTypes can inherit its nodes. ResourceTypes can use and inherit from other resourceTypes.

ResourceType is basically a template that is used to define the descriptions, methods, and parameters that can be used by multiple resources without writing the duplicate code or repeating code.

Let's consider an example. Resource songs want to implement two methods like GET and POST. Each HTTP method will have responseTypes, description, usage, etc.

Now, we have other requirements that resource artists want to implement two methods like GET and POST. So, we will make use of resourceTypes to implement the GET and POST methods and that can be used by both resources songs and artist and it can even be used by other resources in the future.

-Resource Types are used to remove the complexity n encourages consistency.

-Resource types helps to zip the line of code in a Root RAML file by extracting the same patterns in Resources.

-Parameters are specified by <<Parameter\_Name>> i.e double angle brackets encloses a parameter name in resourcetypes

-We can define the parameters i.e user defined parameters

-Reserved Parameters

1.<<resourcePath>>:Provides Resource Path

eg: if URI is /hotels then <<resourcePath>>==/hotels

if URI is /hotels/{hotelid} then <<resourcePath>>==/hotels/{hotelid}

2.<<resourcePathName>>: Provides Resource Path Name

Eg: if URI is /hotels then .<<resourcePathName>>: ==hotels

if URI is /hotels/{hotelid} then <<resourcePathName>>==hotels

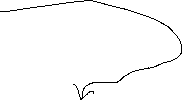
-Parameter Function are used on parameter value let say we used <<resourcePathName>>

|  |  |  |
| --- | --- | --- |
| Parameter Function Name | Before Using Parameter Function | After using Parameter Function |
| !singularize | hotels | hotel |
| !pluralize | hotel | hotels |
| !uppercase | hotel | HOTEL |
| !lowercase | HOTEL | hotel |
| !uppercamelcase | hotel | Hotel |
| !lowercamelcase | Hotel | hotel |
| !upperunderscorecase | hotelNumber | HOTEL\_NUMBER |
| !lowerunderscorecase | hotelNumber | hotel\_number |
| !upperhyphencase | hotelNumber | HOTEL-NUMBER |
| !lowerhyphencase | hotelNumber | hotel-number |

How to use:

For example for /hotels:

<<resourcePathName | !singularize | !uppercase>>



OUTPUT will be



resourcePathName( of /hotels)=hotels



!singularize(hotels)= hotel



!uppercase(hotel)=HOTEL

Final OUTPUT will become HOTEL for this <<resourcePathName | !singularize | !uppercase>>

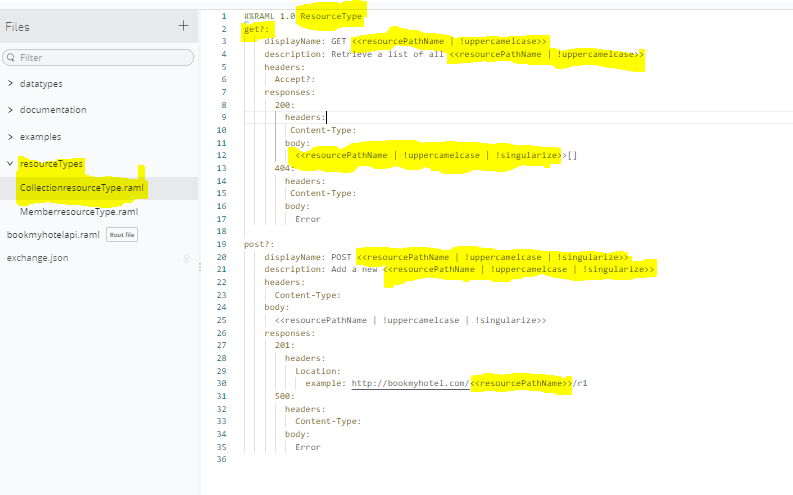


ResourceType Example:-

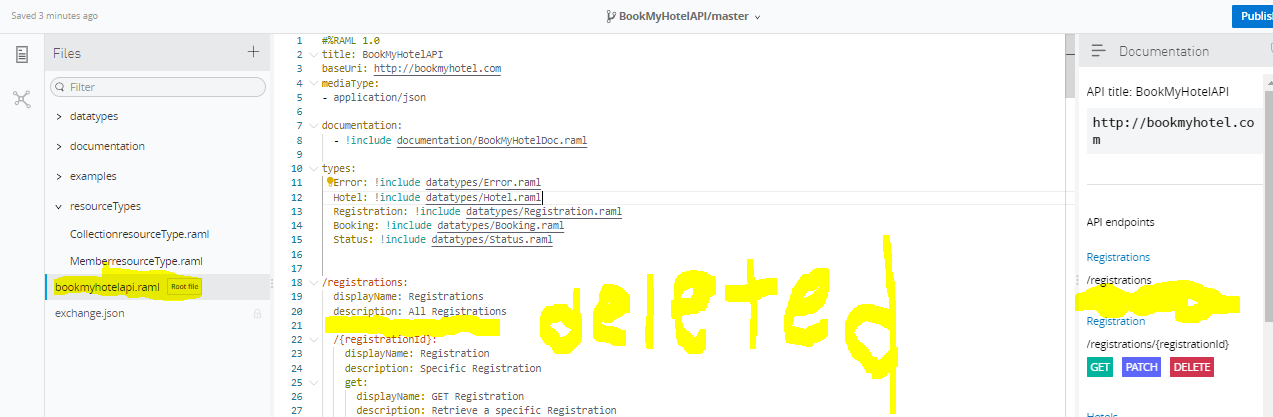
Root Raml File before ResourceType Declarartion: for Collection Resources like Registrations,Hotels,Bookings



After creating ResourceType Fragment file for collection resource



Before including the Resourcetype file in Root File



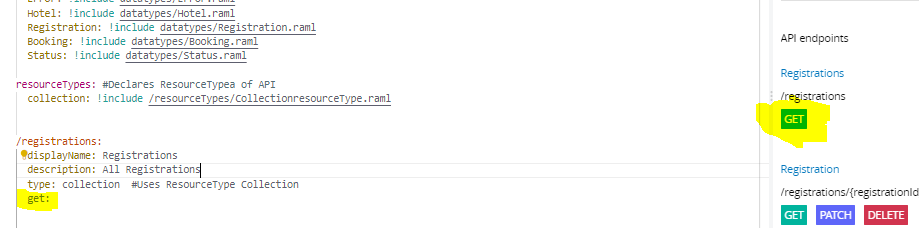
After including n using collection resourceType

Still not able to see Get n post methods as API Endpoints in API Console,which we defined in Collection ResourceType file this is because we have marked get? N post? That means those are optional methods so if we want to add one of them or either both of them as API Endpoints in Root Raml File we need to declare the method below

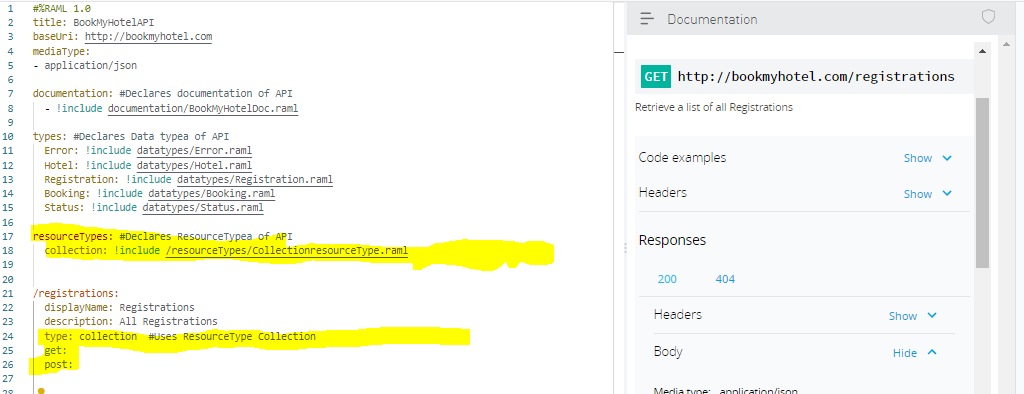
type: collection

get:

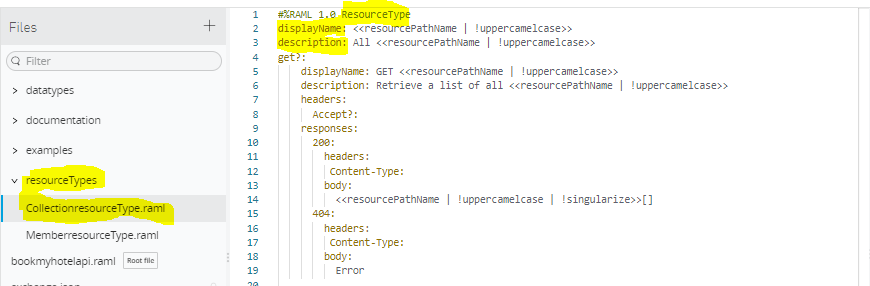
post:



So finally see that by declaring the resouretypes we have removed no of lines of code from root raml file for /registrations.This will helps to reduce complexity,readable n consistent



-We can define displayname n description at resourcetypes level also



-We can override the values of resourcetypes declaration by specifically declaring the values in Root Raml file

* **Traits**

.-Traits is like function and is used to define common attributes for HTTP method (GET, PUT, POST, PATCH, DELETE, etc) such as whether or not they are filterable, searchable, or pageable.

-Extracts patterns from Methods definitions like GET,POST etc.

-We can create traits and use across methods

-Parameters for Traits

User defined Parameters are there.

Reserved Parameters:

1.<<resourcePath>>

2.<<resourcePathName>>

3.<<methodName>>

-Traits are applied to method within Resources as well as ResourceTypes

-Root raml file before creating traits for Error response



* Library

-a library is collection of decalrations of any or all of the following RAML Concepts like

1.Datatype Declaration

2.ResourceType Declaration

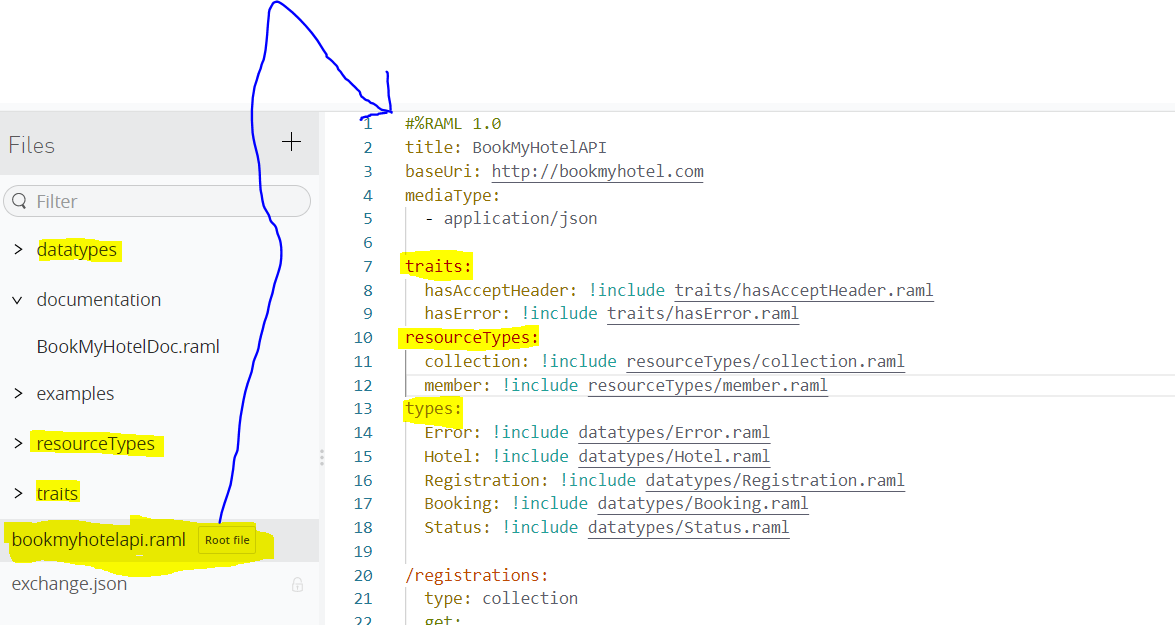
3.Trait Declaration

-Defines common declarations in an external file(Fragment File) or an inline

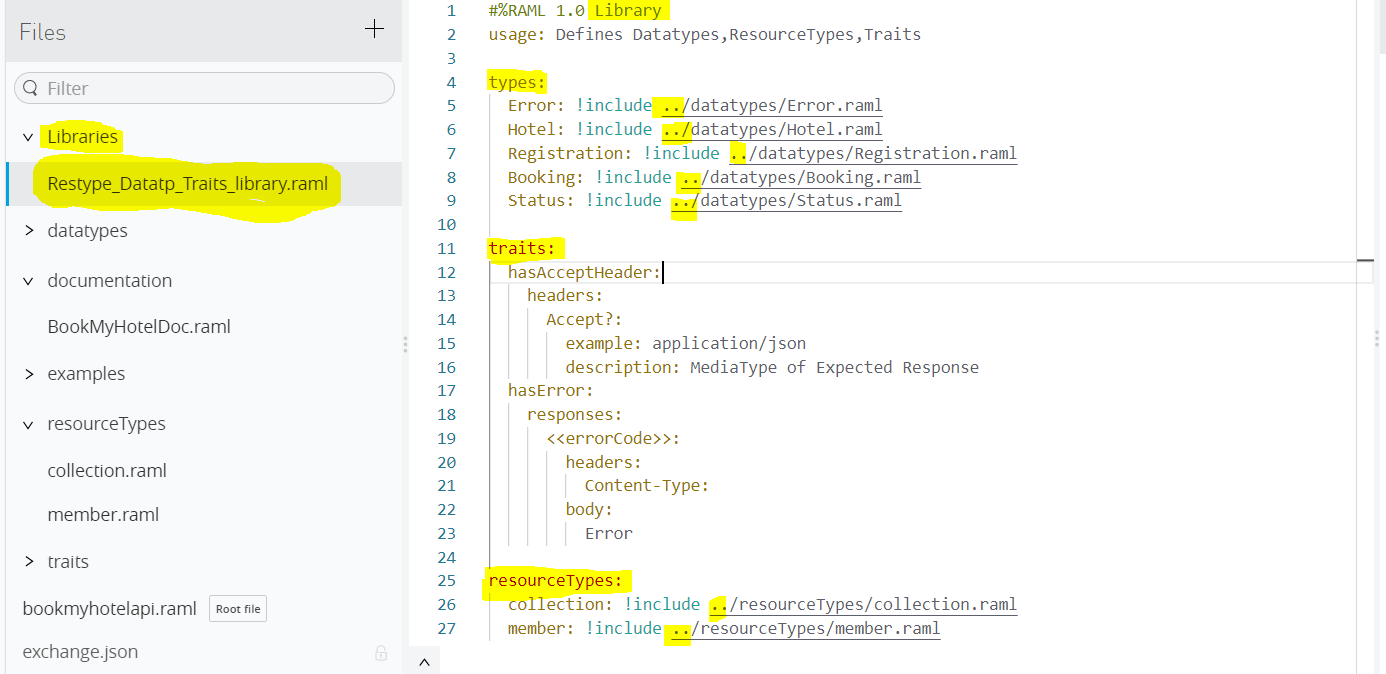
-If we create a Traits fragment file then in that file we can define only one trait but if we use library to define the trait then we can define as many traits as we want.

-Reusable

Root Raml File before Library declaration



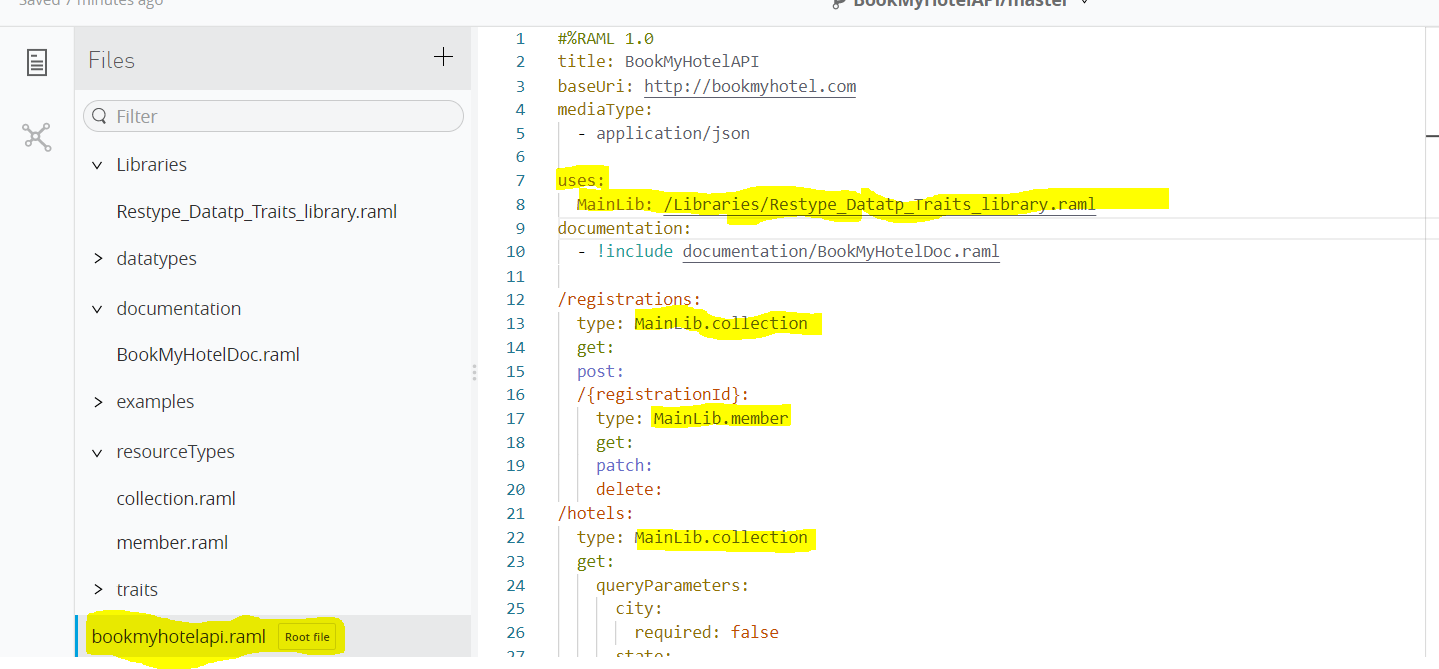
By defining Library file, here I am declaring resourcetypes,Datatypes and Traits in one library file only



Here, after iInclude ../ we have used here because it says relative path that means the library folder n other folders are different folders here so remove the error we have given relative path instead of !include /d………

In library file,we can include the whole definition(Traits in above case) or just providing the reference to definition file like(resourcetypes n datatypes in above case)

The root file after declaring library file reference in Root File



We use ‘uses:’ keyword for referencing the library file in root raml with Key n value field where key=Any name value =location of Library file

Note:-We don’t use !include sign to include the reference of library file

With the key name that we provided in this case ‘MainLib’ we can reference the content that we want to provide in our all files like let say we have ERROR datatype in value field n as we know we defined all datatypes,resourcetypes,traits in one raml file n we are referencing that file with ‘MainLib’ so what we can do is wecan write MainLib.ERROR in value field to remove the error.

* **Anypoint Exchange**

-Anypoint Exchange is like Marketplace where we can published our assets which can be reused by organization.

-Asset is a maven Artifact that

-Assets can be

1.API:REST API with RAML Specification or SOAP API with WSDL

2.API Group:group of API Instances

3.Fragments: An API fragment is a RAML document that has a version and an identifier, but is not in itself a complete RAML specification. API fragments are reusable component of RAML to make the design and build of a reusable API even quicker and easier. Another advantage of building an API spec out of reusable API fragments is that consistency of definitions reduces the effort of implementing APIs.

4.Connectors: are used for connecting to service or a system like fb connector.

5.Examples:helps to understand the concepts

6.Templates:similar like examples but they are based on some standards or best practice so u can utilize those templates for developing an app.

7.Custom:can be video,documentation that u can share with your organization

-Exchange can be used for Publishing the Assets.Different ways of publishing involves

publishing thru

1. Design Center

2.Maven

3.API Manager

4.Directly from Exchange

-Exchange helps to consume the assets.Assets can be consumable by

1.Anypoint Studio(Connectors,Examples,templates)

2.API Manager(APIs,API Groups,Policies)

-Assets can be public/Private

1.Public Assets:Provided by Mulesoft

2.Private Assets:Within organization/business groups

3.Shared with me

* **API Fragment Publish n Consume**

-API Fragment is a RAML Doc but not complete RAML Specification.

-API Fragments can be:

1.Datatypes

2.Library

3.Resource Type

4.Traits

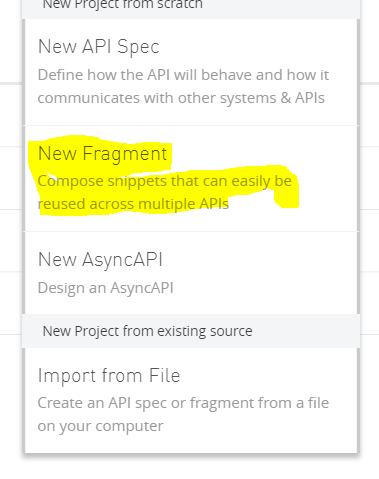
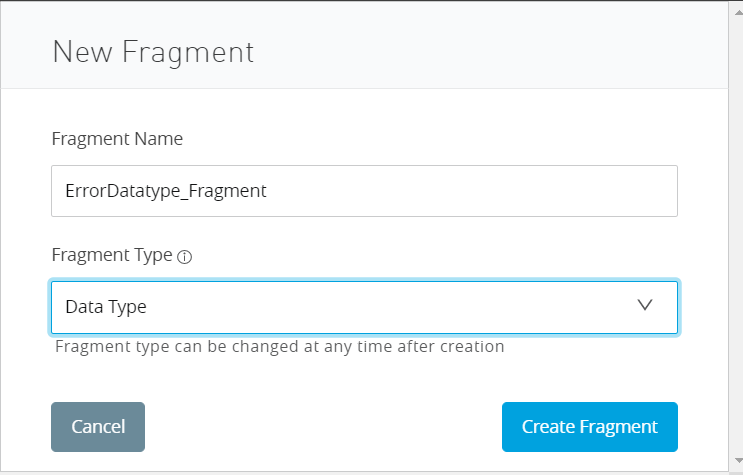
5.User Documentation

6.Example

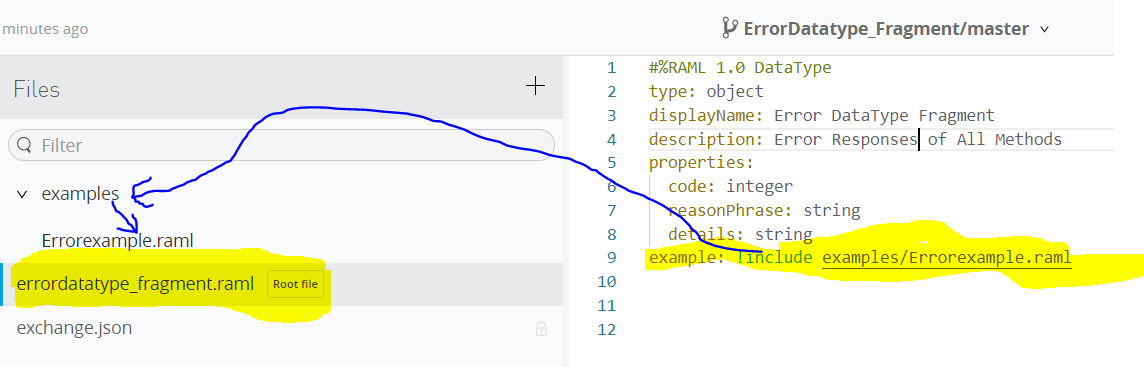
-How to Publish n consume Asset

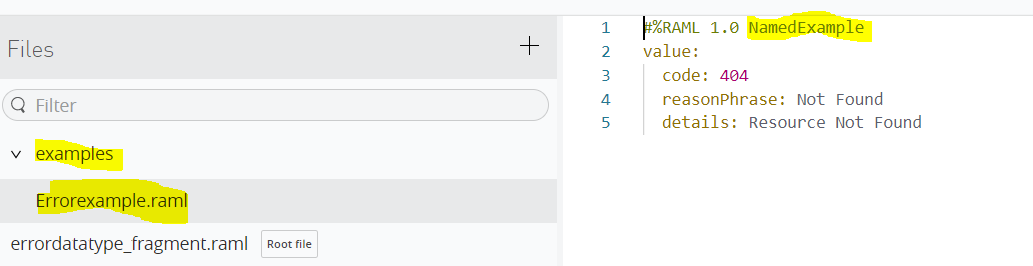
for eg let consider ERROR Datatype API Fragment

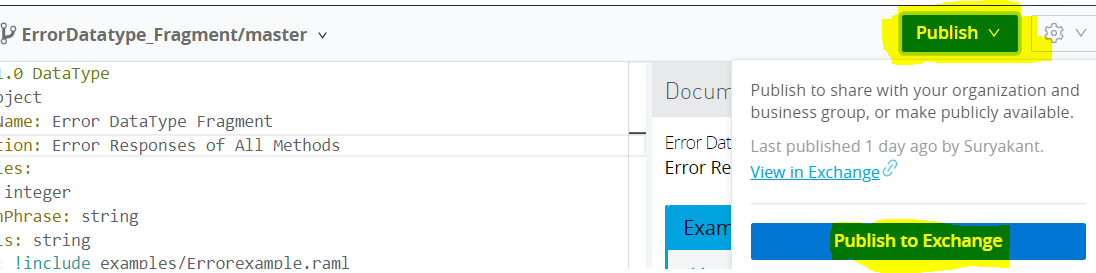
First create Error API Fragment in design center

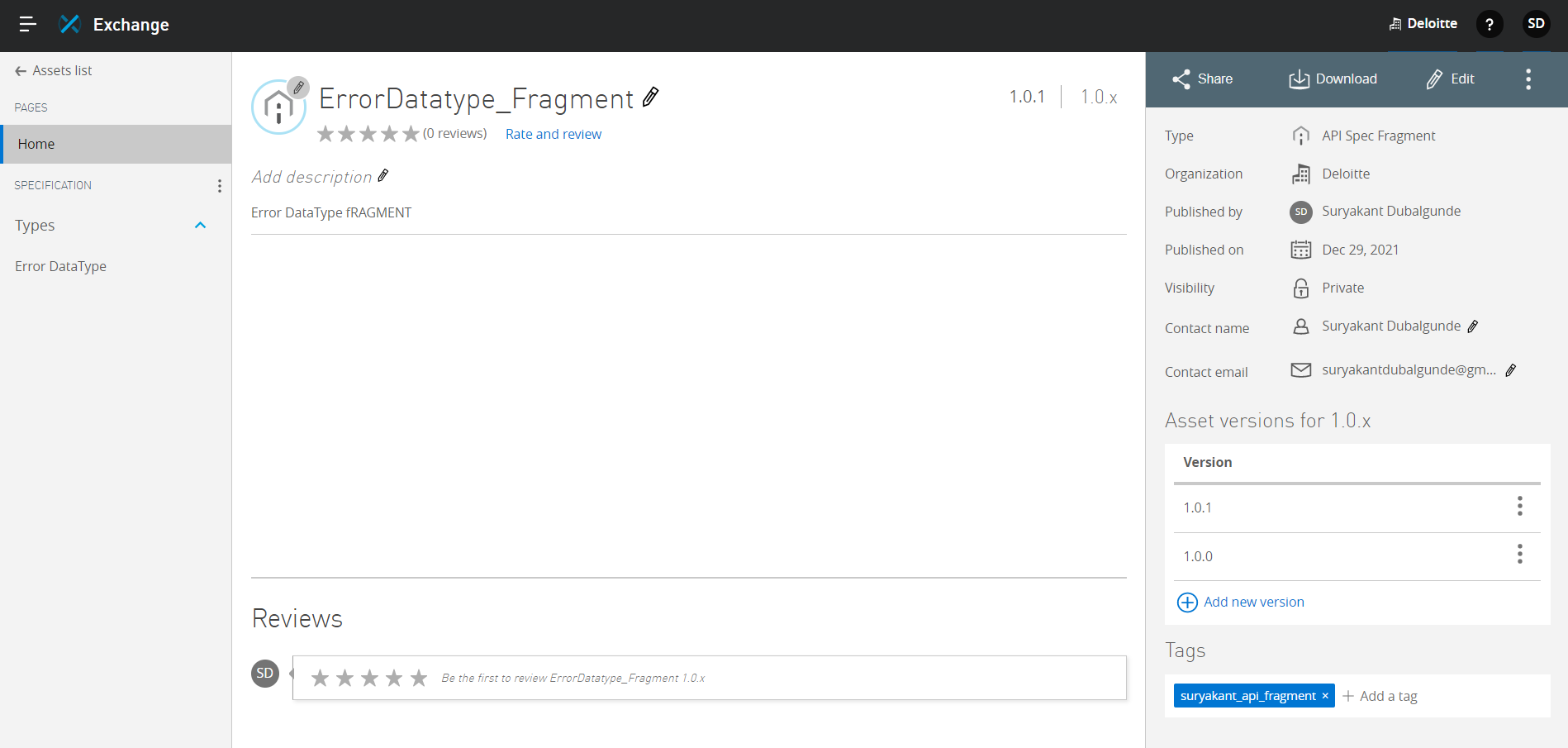
After this write the complete code like below





Once all code looks good then publish the Fragment in Exchange by clicking on publish(If you have published already the fragment ,u will get below message like view in Exchange)

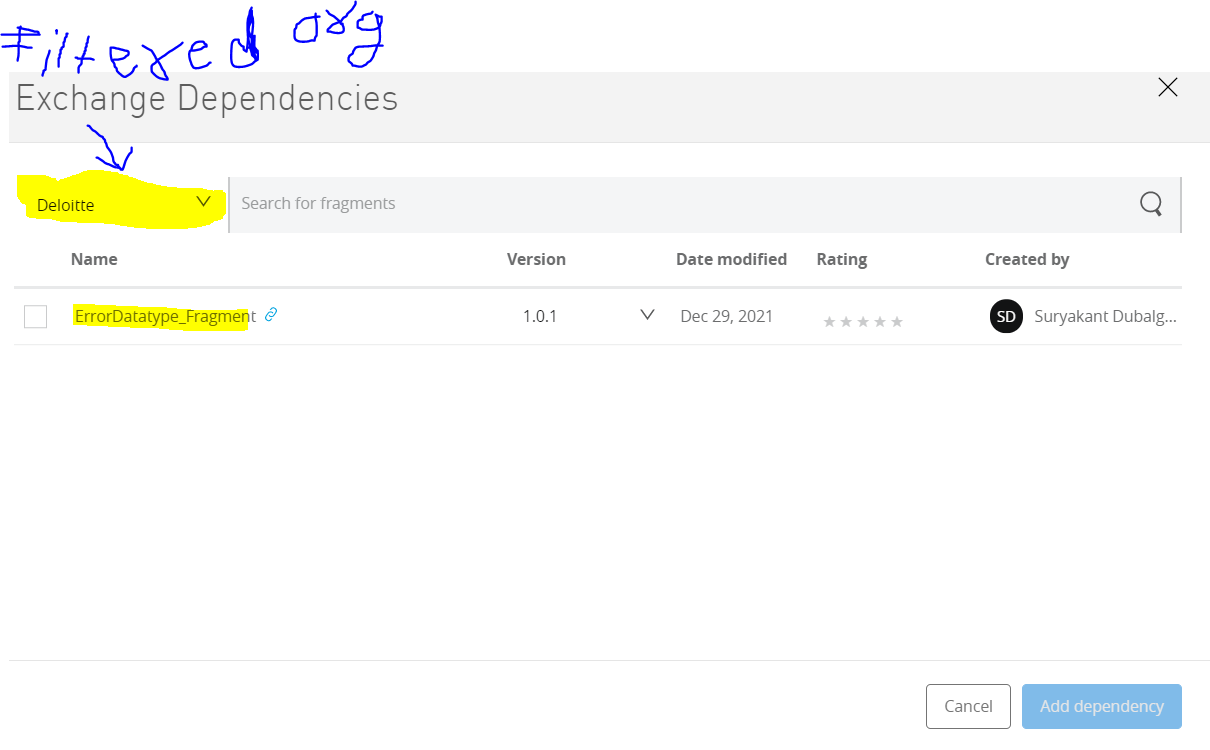
After publishing the asset to Anypoint Exchange,API Portal is created automatically where you can add description, change the icon of asset,change the name,give review,contact name,contact email,add tag etc.



-Consume the recently published Error Datatype Fragment in other app like BookmyHotel Api.

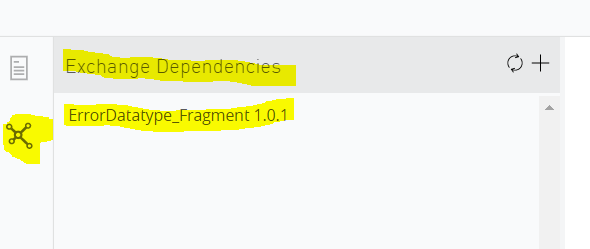
Since we have added the Error Datatype in Exchane now we can refer that Error datatype fragment directly from an other app by using Exchange dependencies button



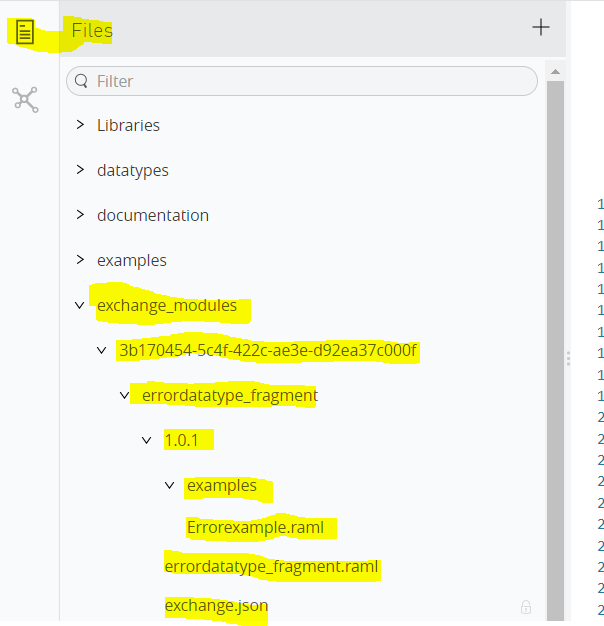
In above screenshot we can defineI

In above screenshot we can define the version of asset we want to add ,here we are using 1.0.1 to add an that code dependency

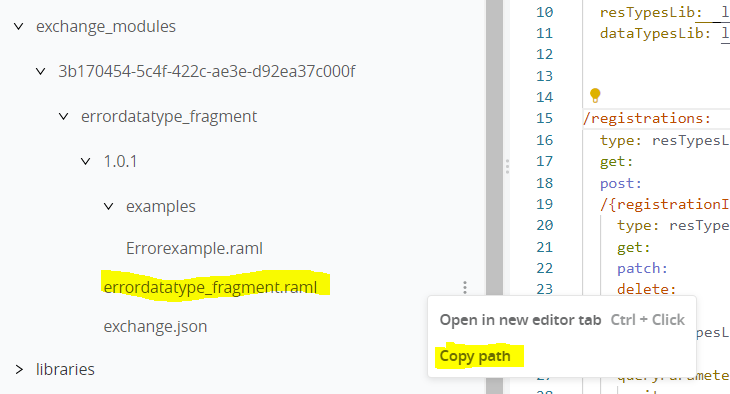
After clicking on add dependency button,in exchange dependency panel we can see the Asset name with version selected is added.

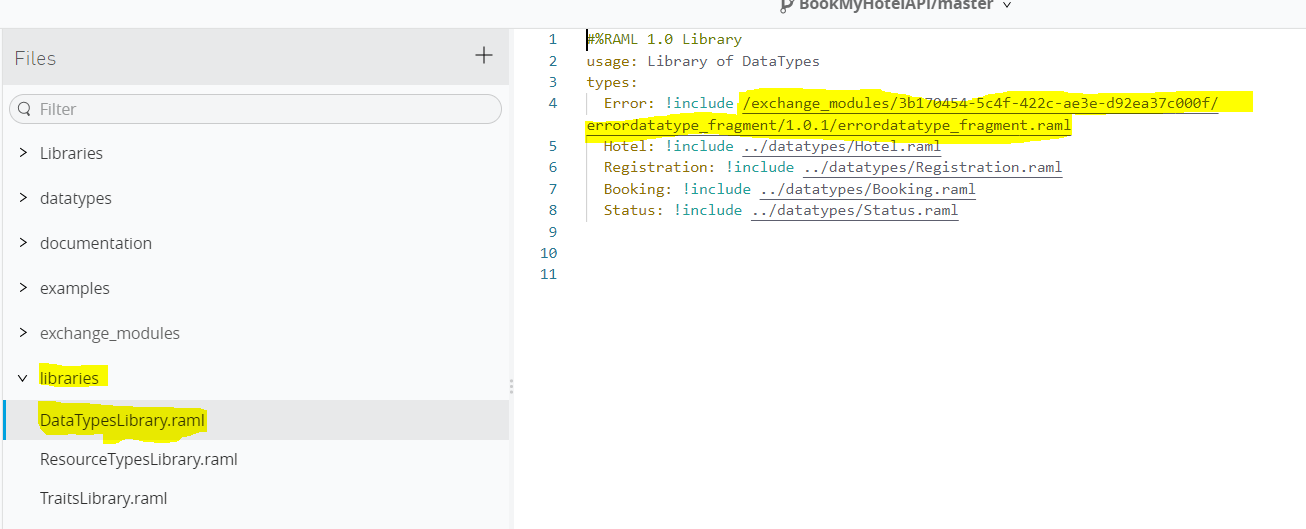


in files menu we can see exchange\_modules node created and we can see below nodes also.



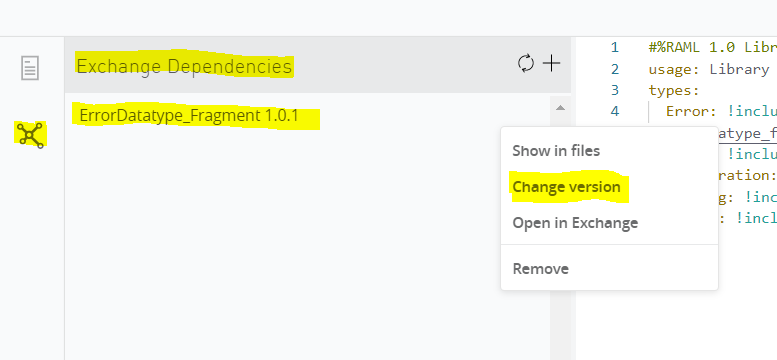
To use Error datatype fragment in this project we have to copy the path of Error Datatype fragment root files path like below:-

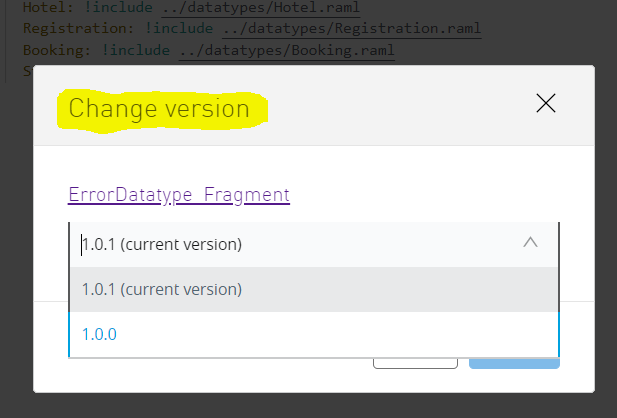


Then add this copied path in mail project’s file wherever applicable,in this case we have to use this path in Library, datatypeslibrary raml 

So this is about publishing n cnsuming the asset.

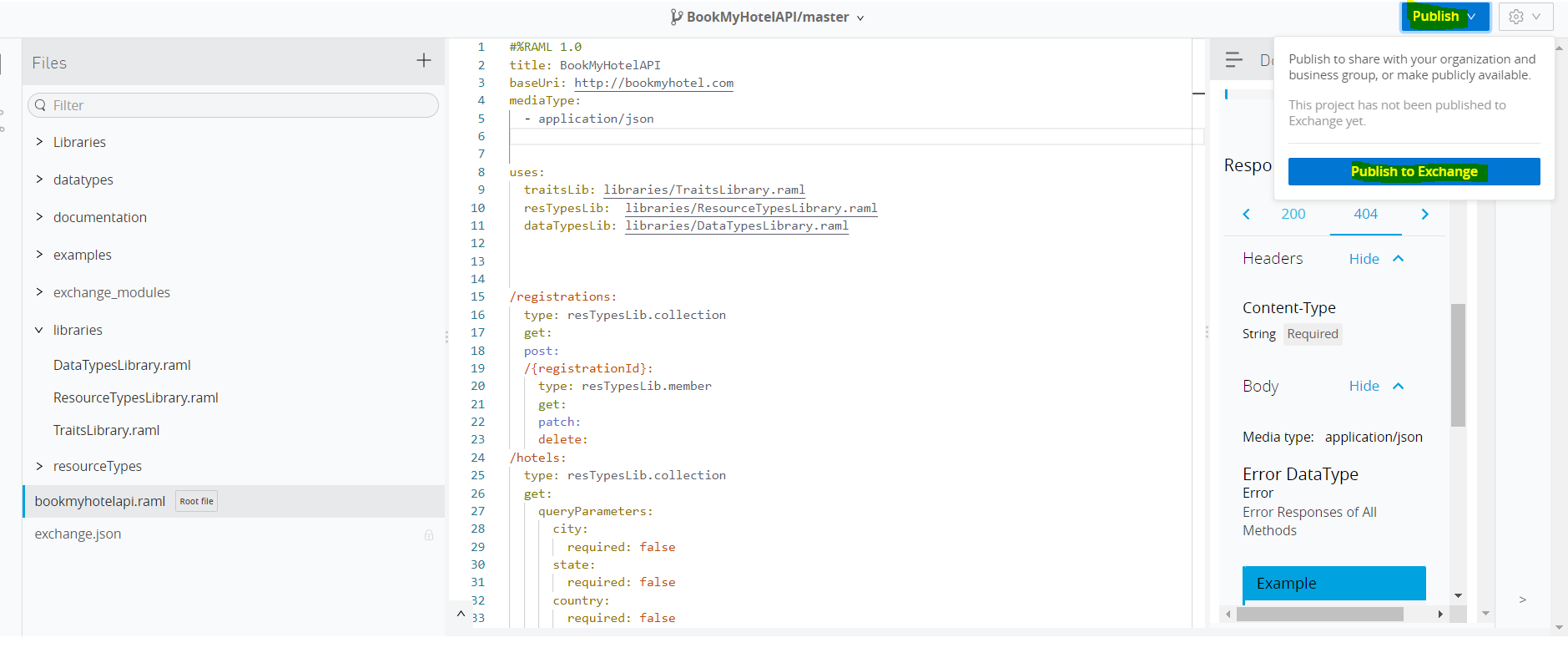
Let say we got an update to do in Error Datatype Fragment n then after publishing its version will change (from 1.0.1 to 1.0.2)n if we want 1.0.2 to be used in our BookmyHotel API how we can do is like below

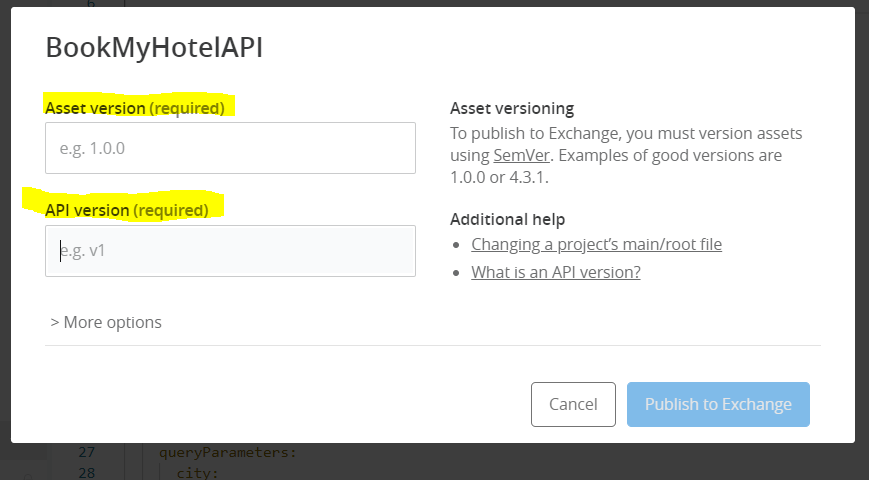
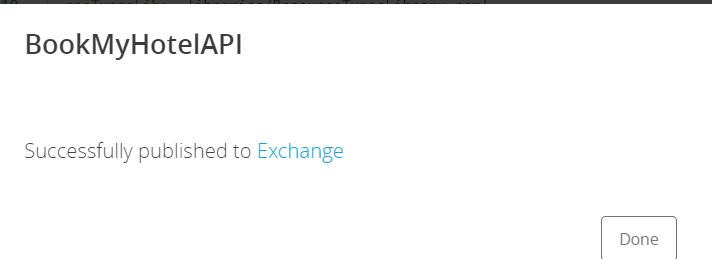
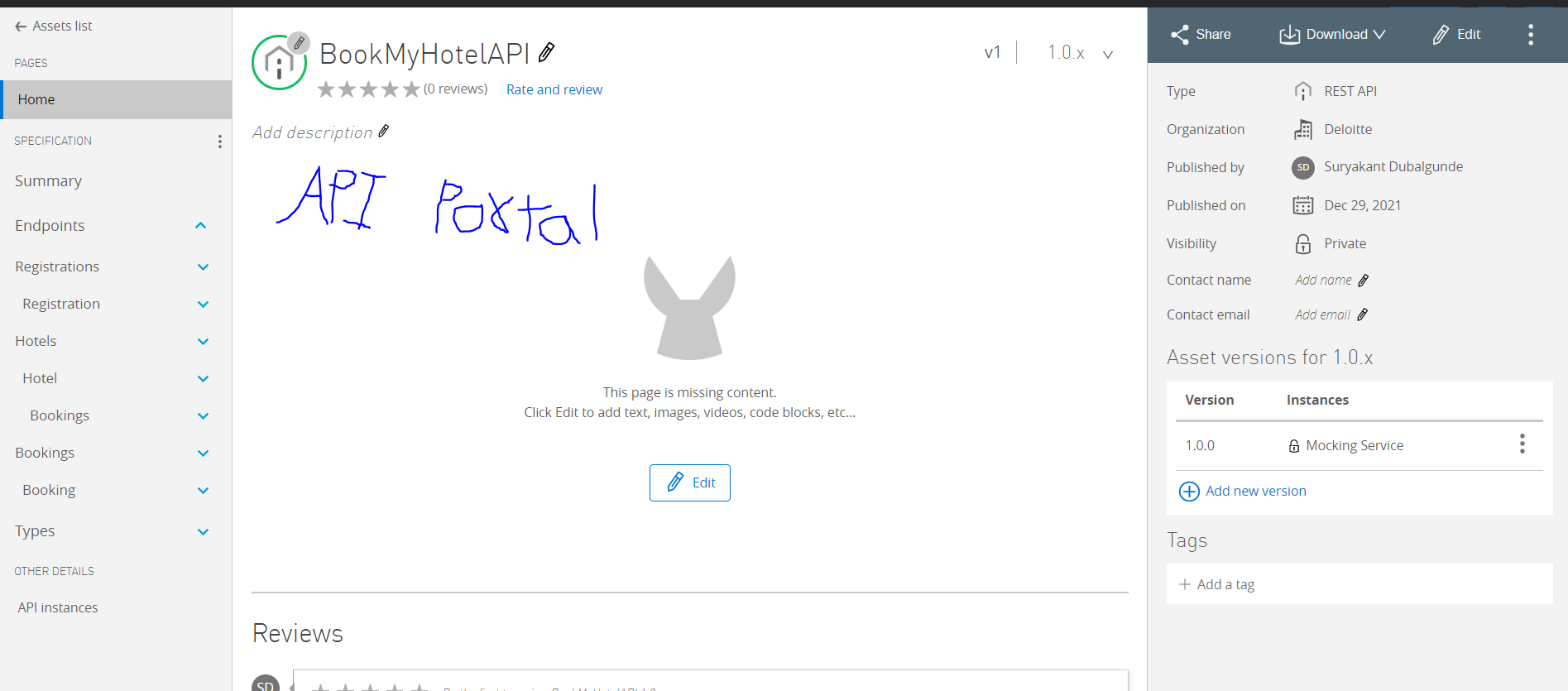
Go to your project->go to Exchange dependency button->select the old asset file which needs to be updated->click on three dots n select change version 

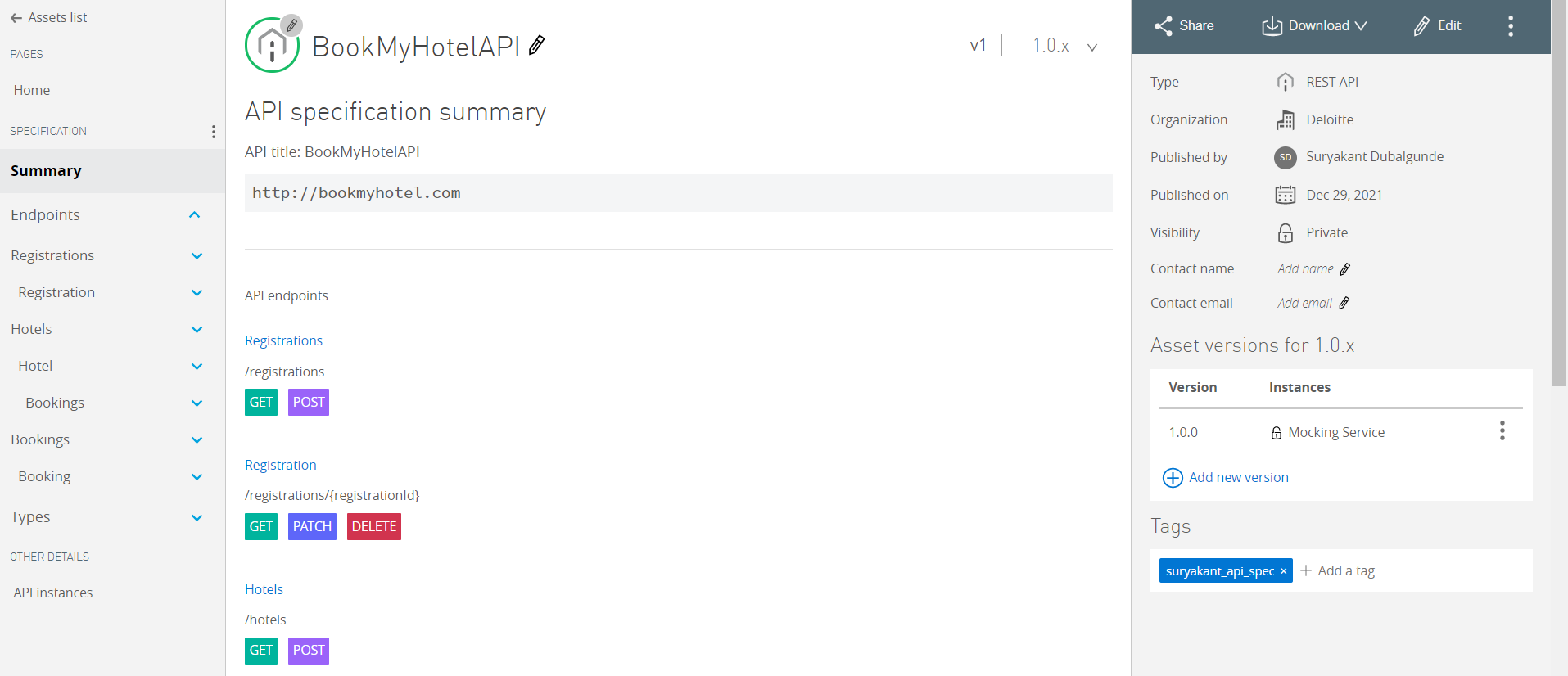
There u will see all listd version select your version to be updated n click on change

After change version you will see the updated asset exchane dependency with asset name n new version name.to add the reference of newly version in raml file go to files ->exchange\_modules->ErrorDatatype\_Fragment file->click on 3 dots select copy pth n paste that copied path in raml wherever applicable

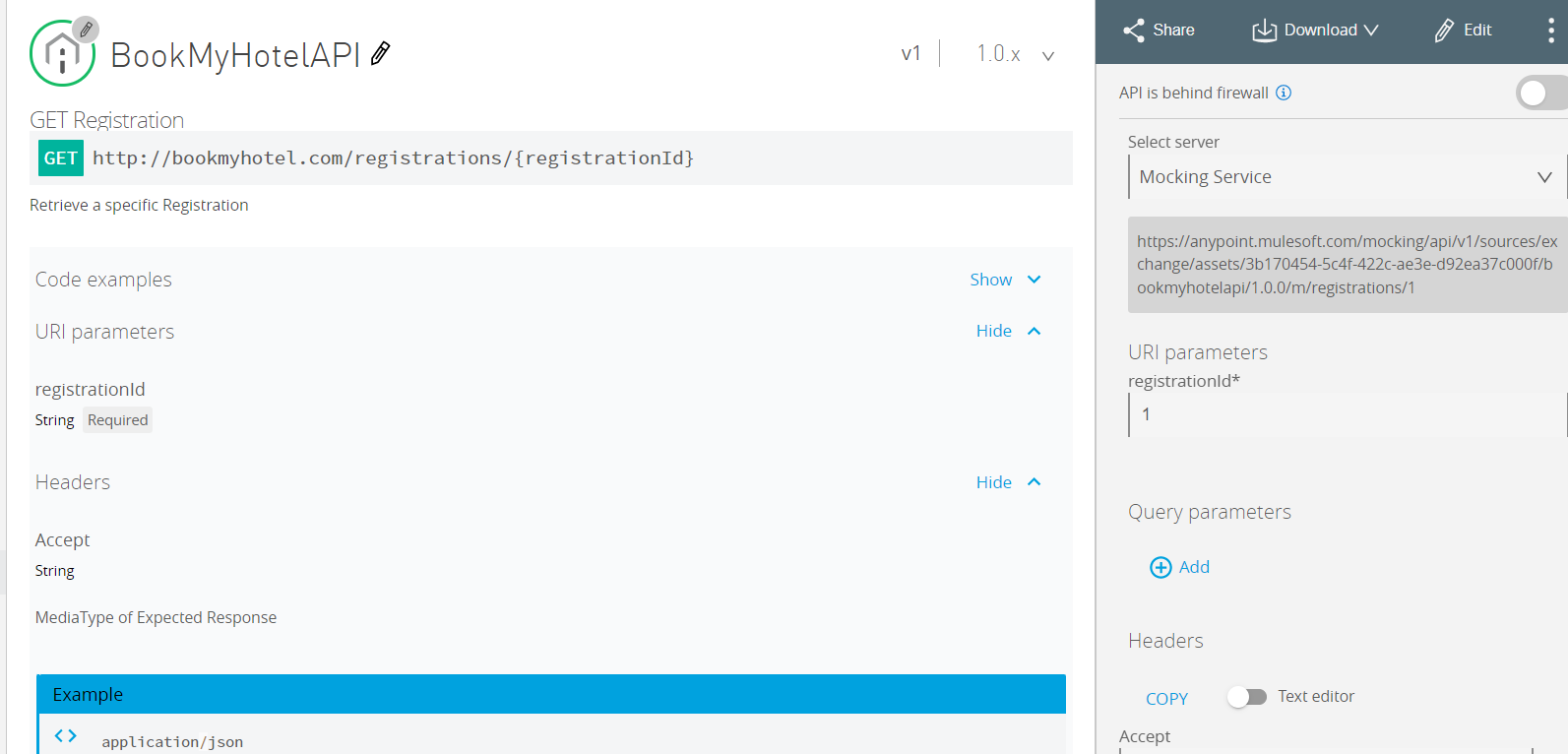
* **Publish API Specification**
* We can publish the API Spec in same way we did for publishing API Fragment

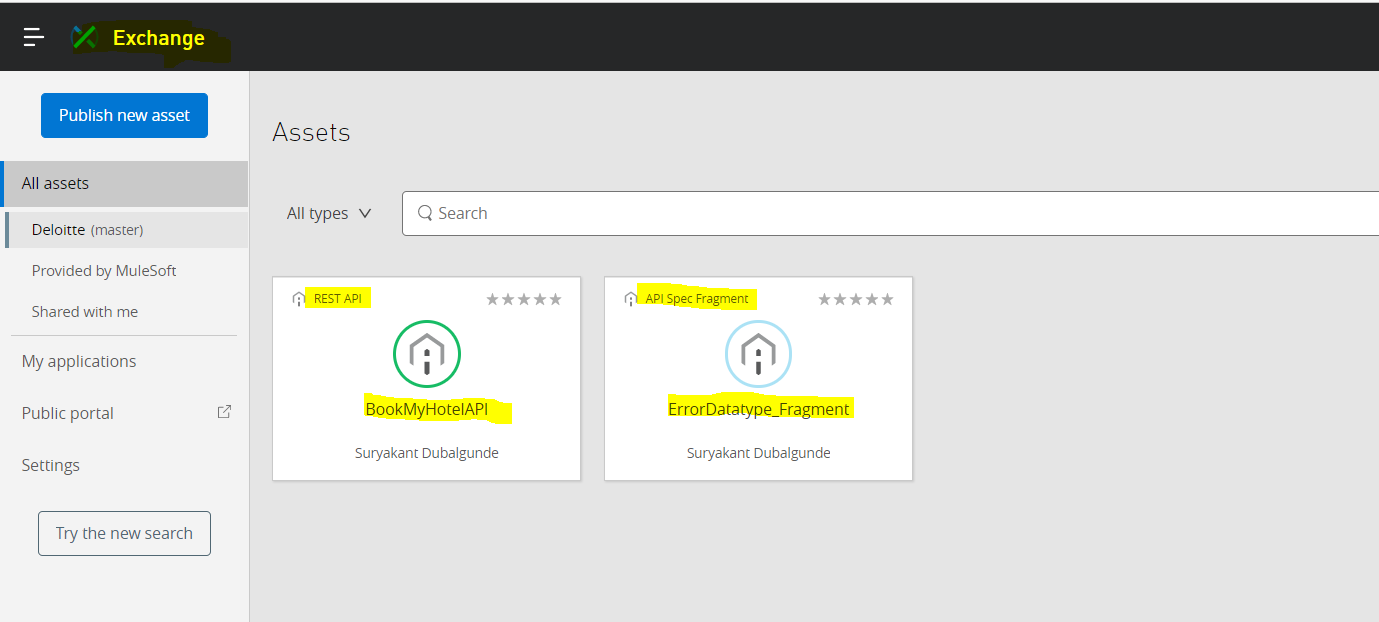




Once API portal is created you can test the API endpoints





* Share API within Organisation

-We can share our developed API

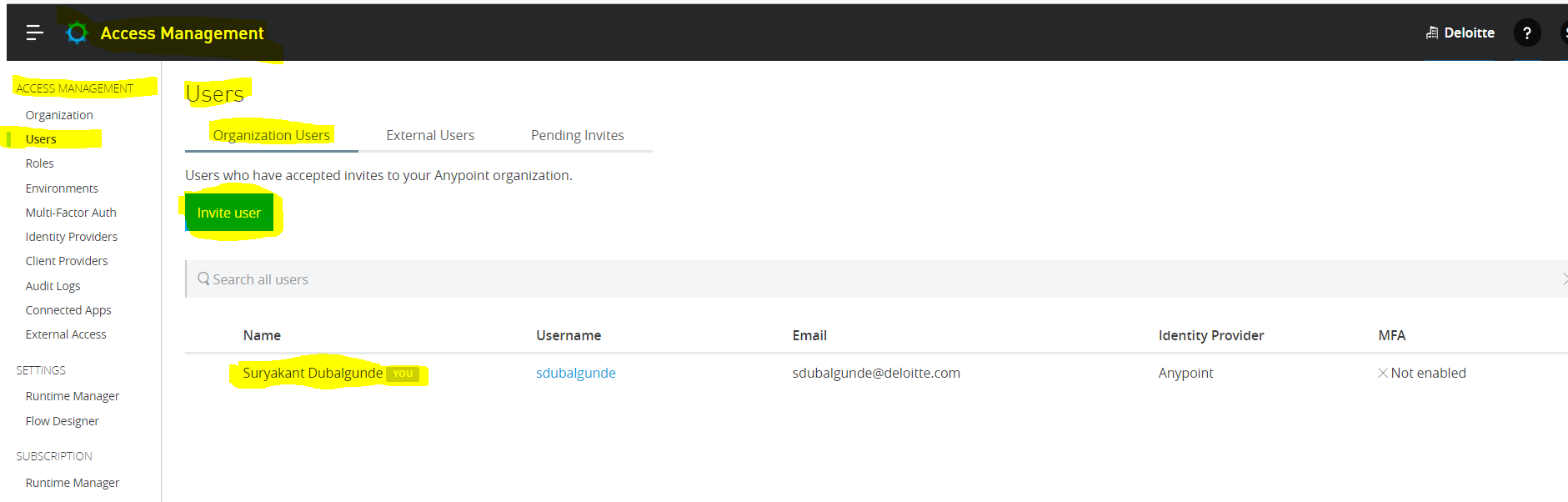
1.Another user within our org

2. our entire Org

3.external org.

-Sharing an api within org means sharing with a user/users from same org or sharing the API with everyone at organization.

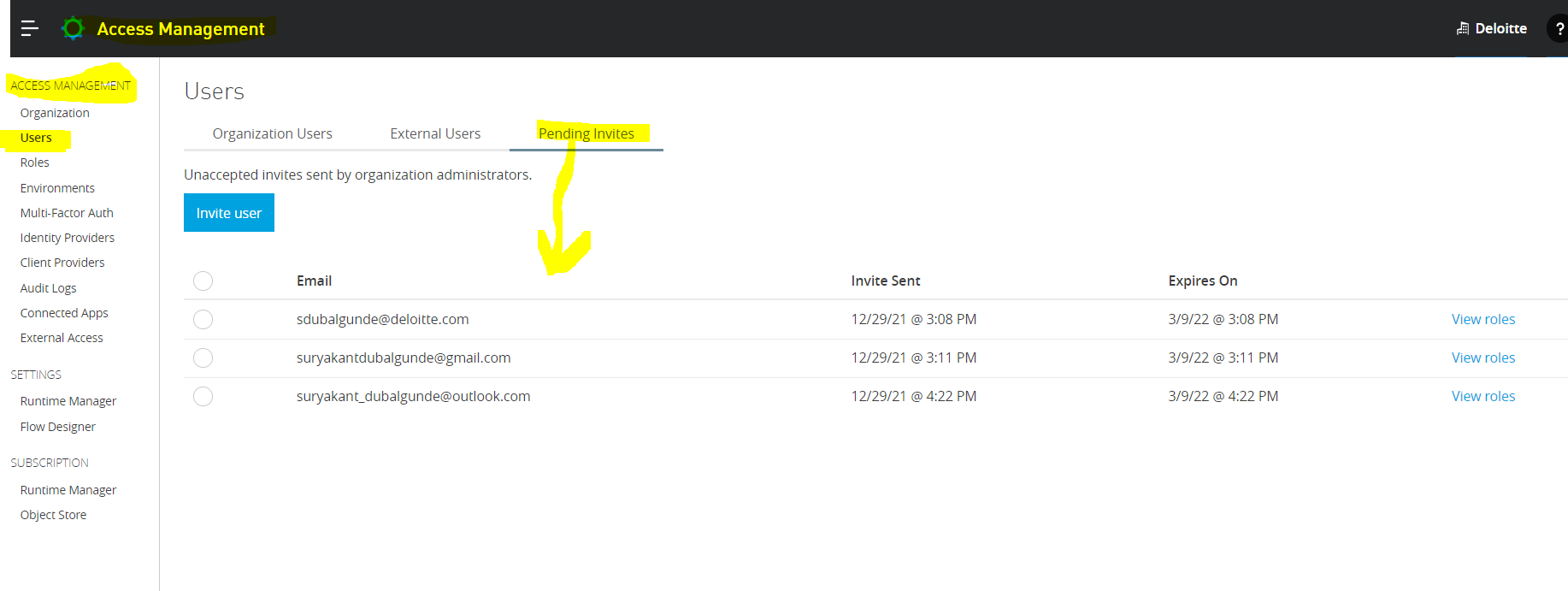
We can see users belongs to our organization are mentioned here



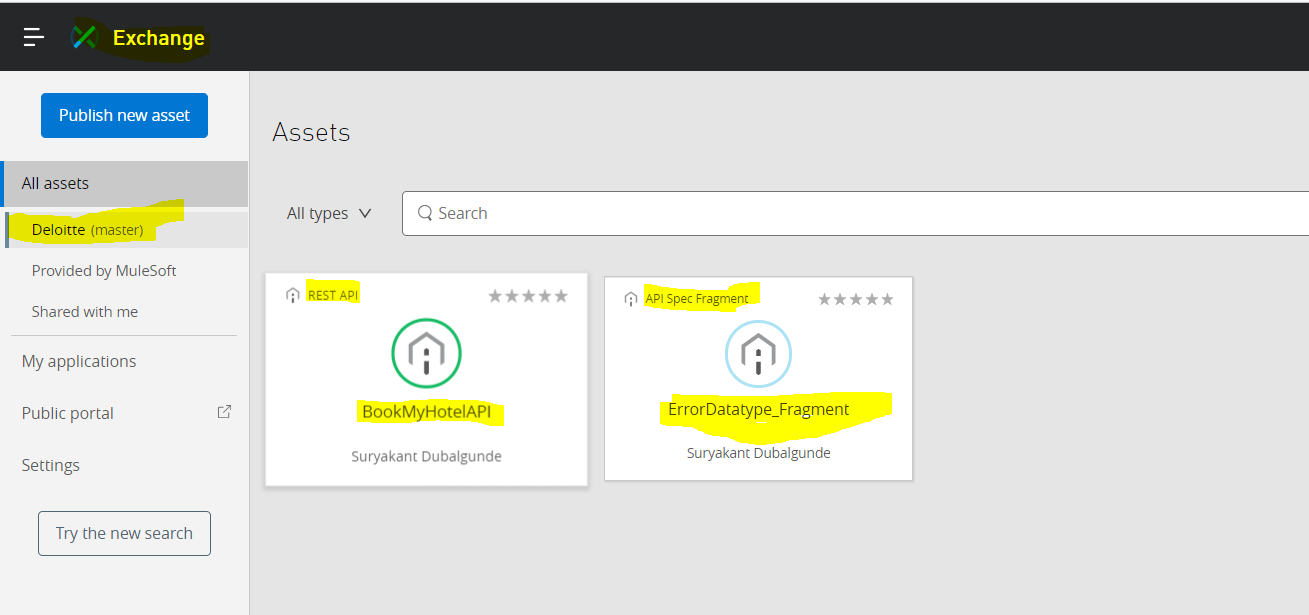
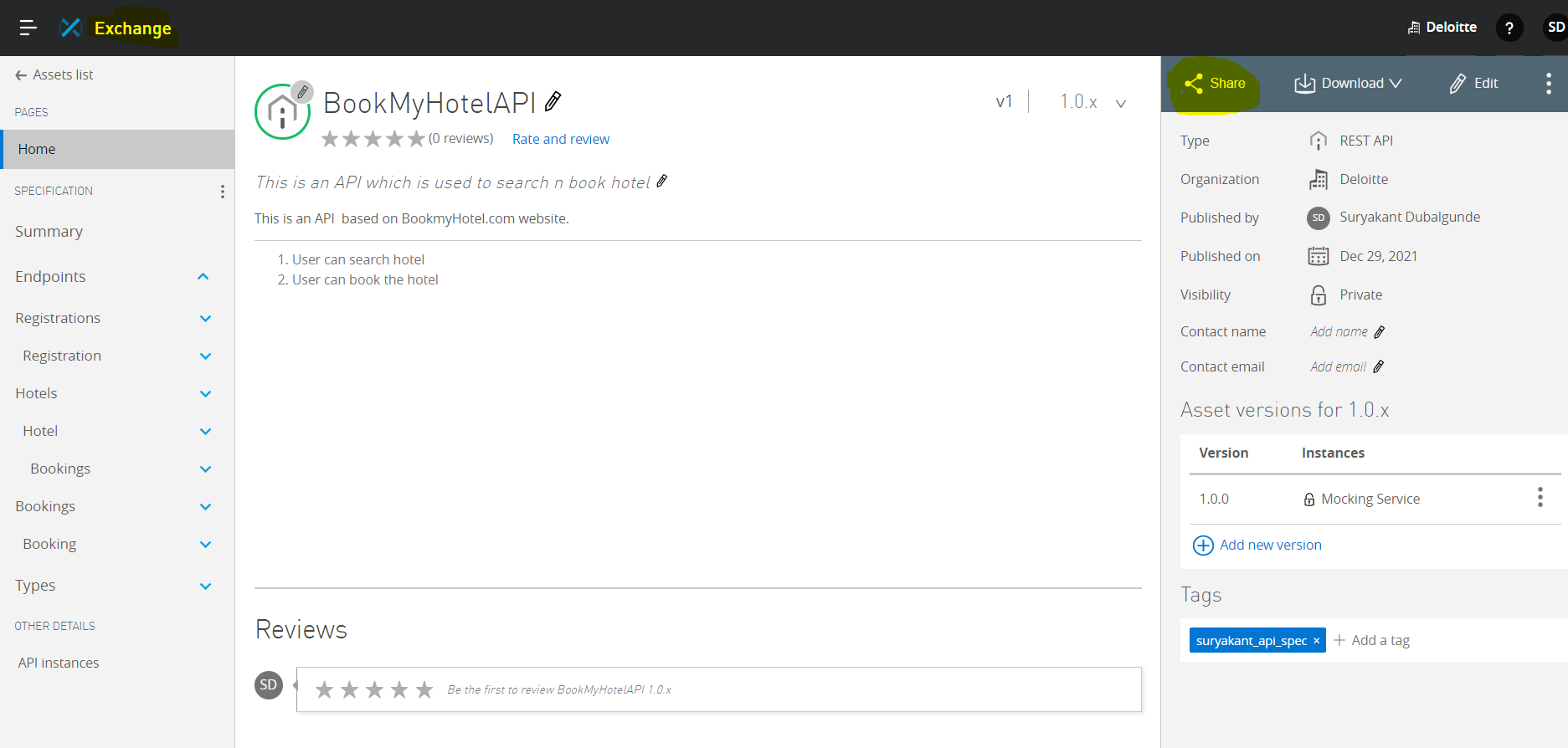
Currently no other user within my org is listed here so I cant share an api with a specific user but however I can share the api with my entire org so that in future anyone who joins my org can get the specified access to APIs.

I have sent the Request to join my organization to 3 users listed below:so what it will do is will send a mail to all the user mail with a link to join my org .Once particular users clicks on the link then he has to enter his cred(if user is already having an account with anypoint platform) or create a new uname n pwd for the Anypoint platform.

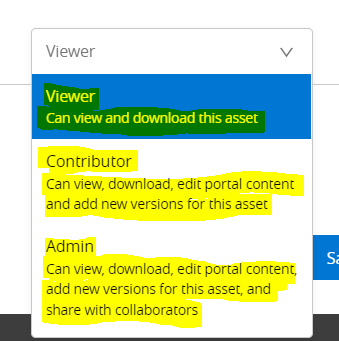
-if user is having already account n enters his cred it will promp that to join ----- org you need to delete the org with which you created an anypoint platform account.on entering delete ---orgname n clicking on ok your old org will be deleted n you will be part of that new org.then your name will come into the other users org users list.



To share the API within an org:

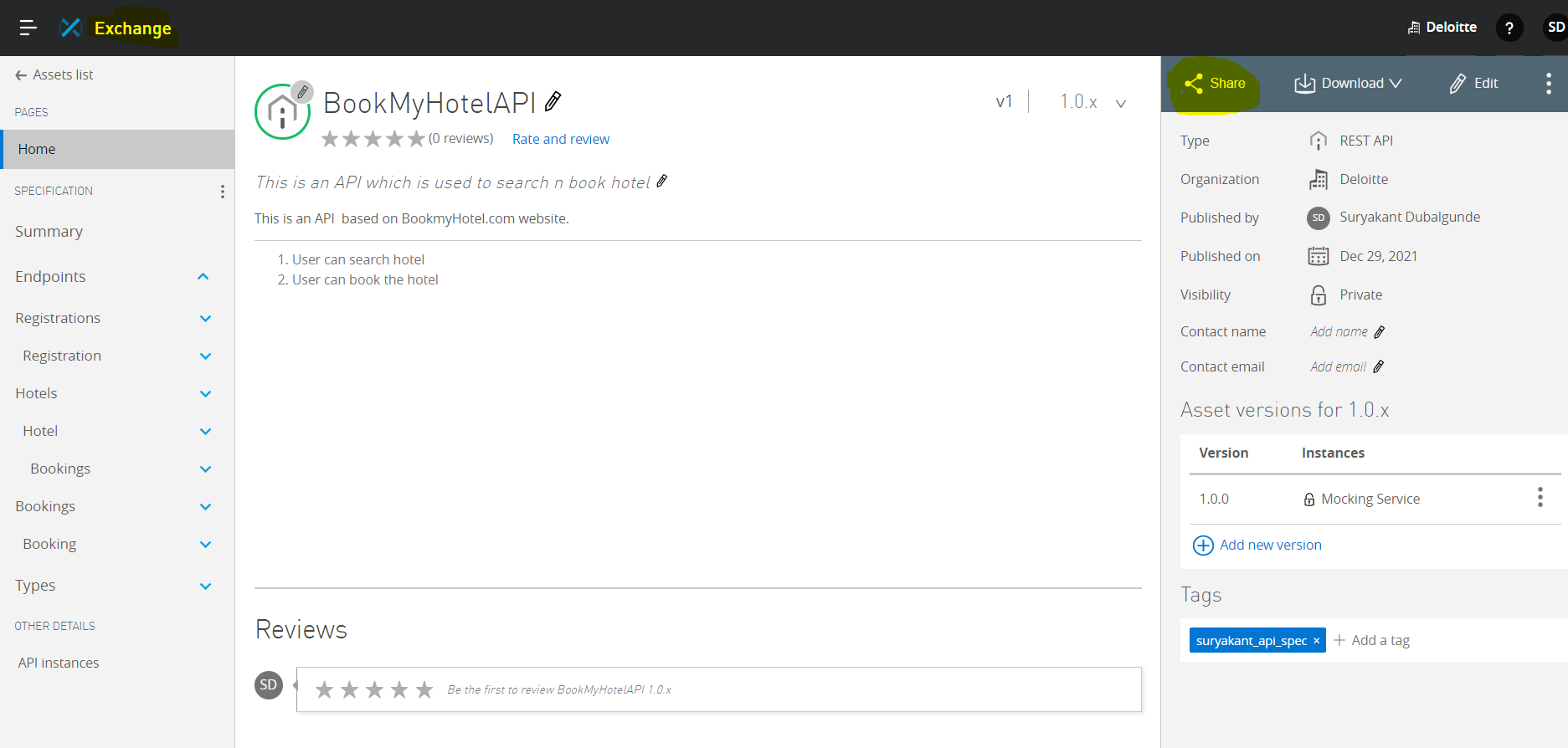
Go to Exchange->select n open an API which you want to share->click on share button.   

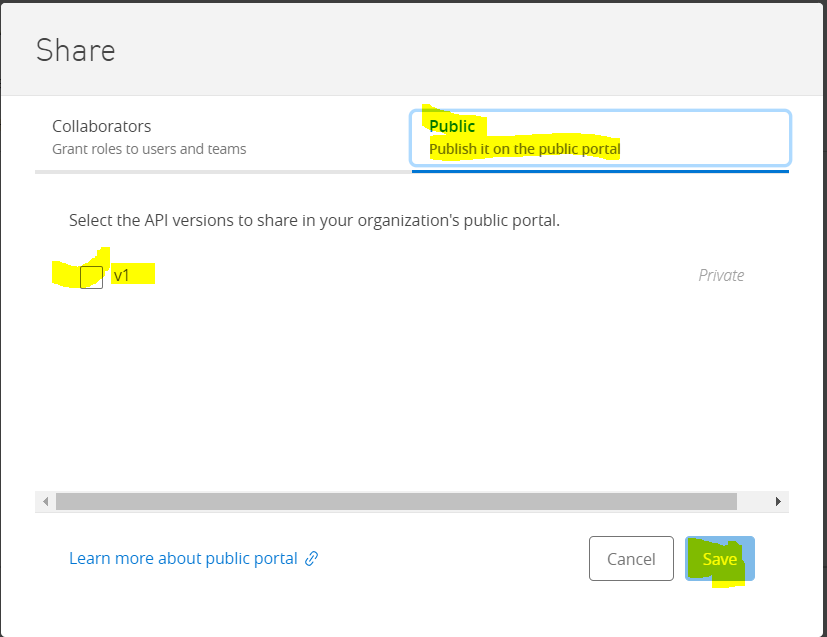
Share option within org provides three types of access level:-



* Share API on public portal(external org)

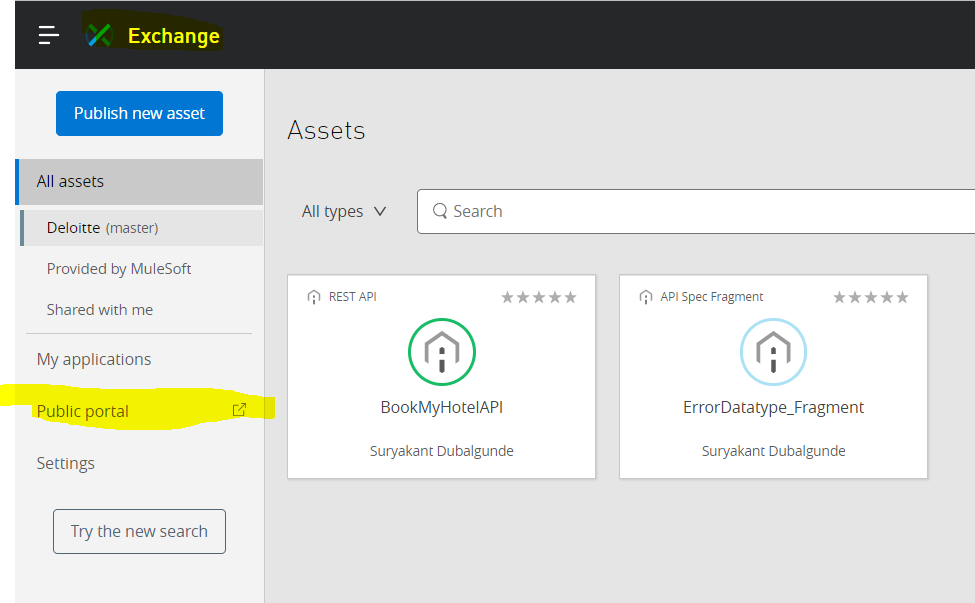
-Exchange provides an option to share an api with external users(who are not part of our Anypoint Organisation) called public portal.With this external usets can see the api n test the api but he can’t modify the api because while sharing an api with public option we don’t get an option to select the access level role like viewer,contributor,admin.by default external users can get viewers access that is download n view the api.



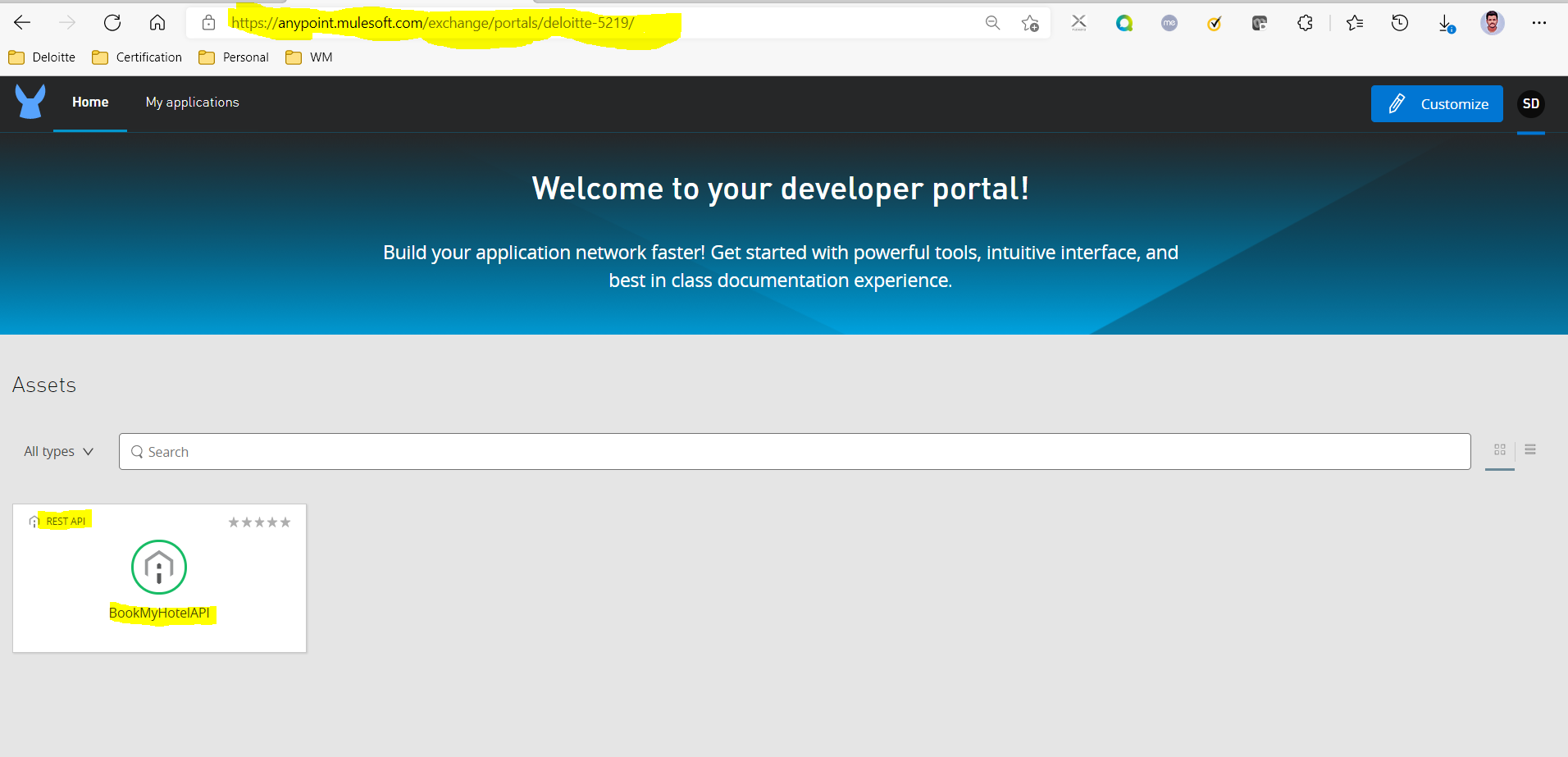


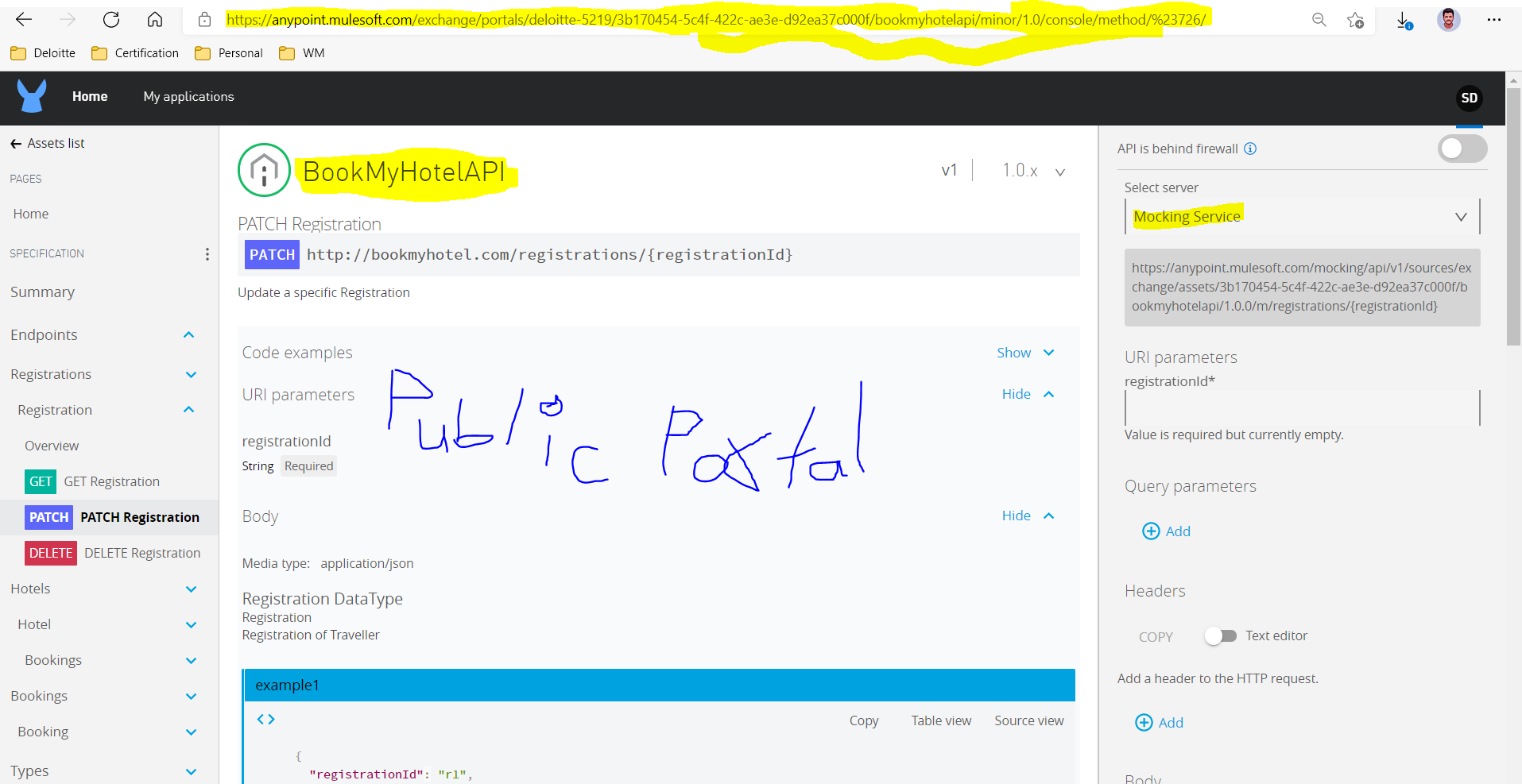
Here we only get to select the version of api which we want to hare on public portal.

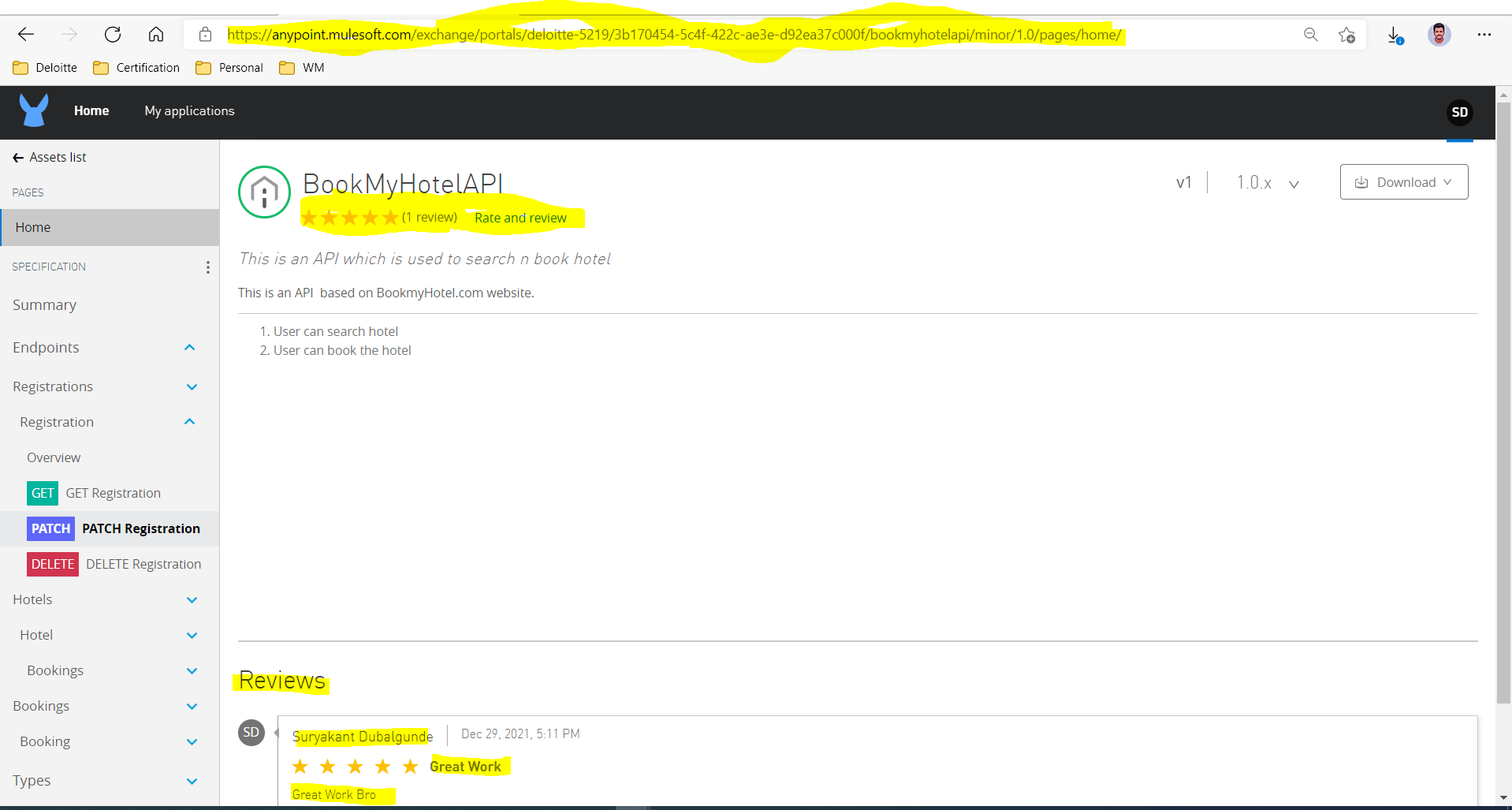
To check it on whether api is published on public portal,go to exchange->click on public portal link

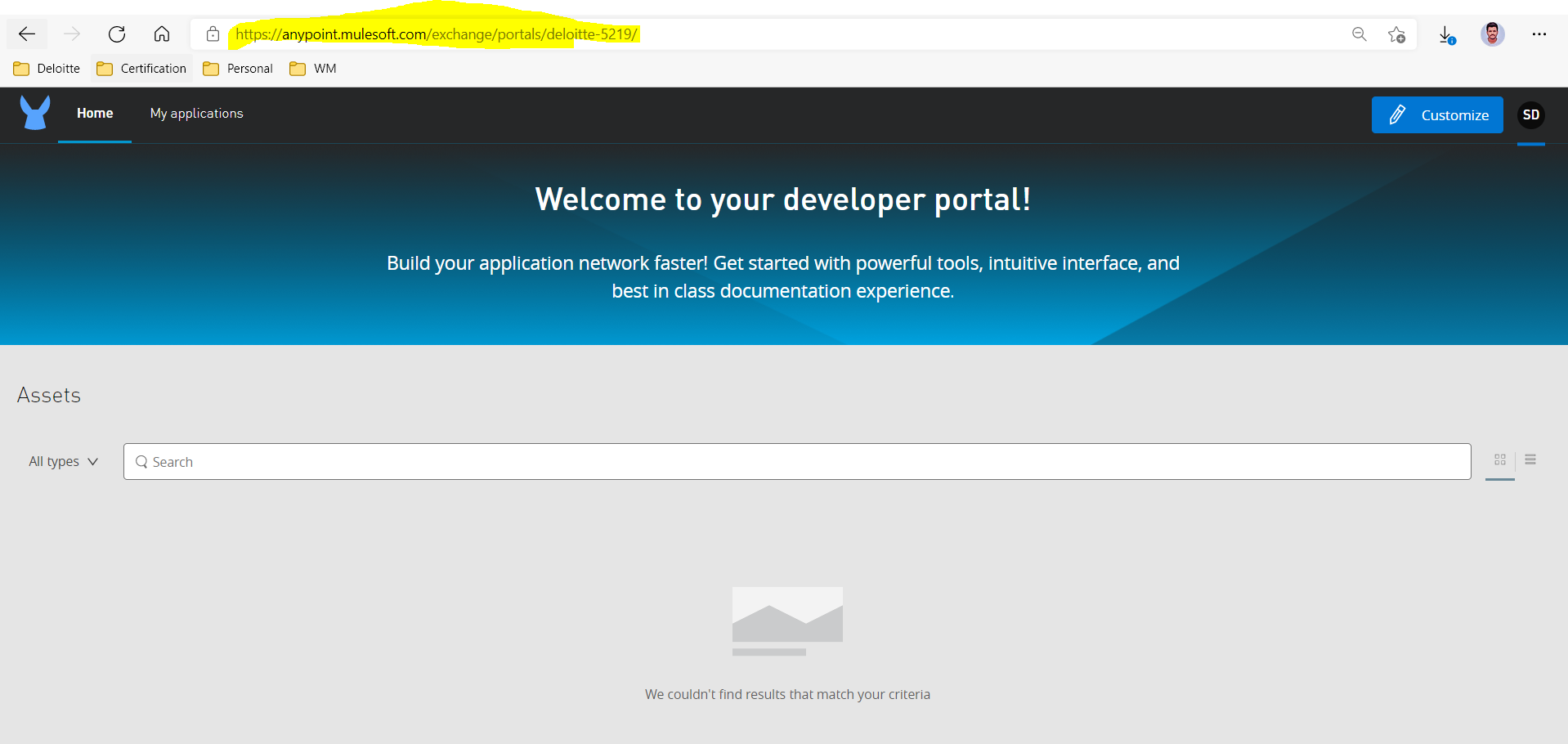


After clicking on public portal,you will see an API Listed ,you can test n you can provide the feedback in terms of rating n review for the api







To disable the API’s Public Portal access,we can go back to exchange->select n open the api->click on share link->go to public in share option n deselect the version n save.then you will see on public portal particular api is not listed overthere. 

* **API Notebook**

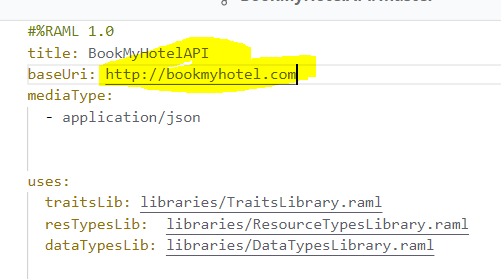
-API Notebook is web based tool which uses RAML Specifications to generate the client for API n calls API endpoints on mocking or real server.

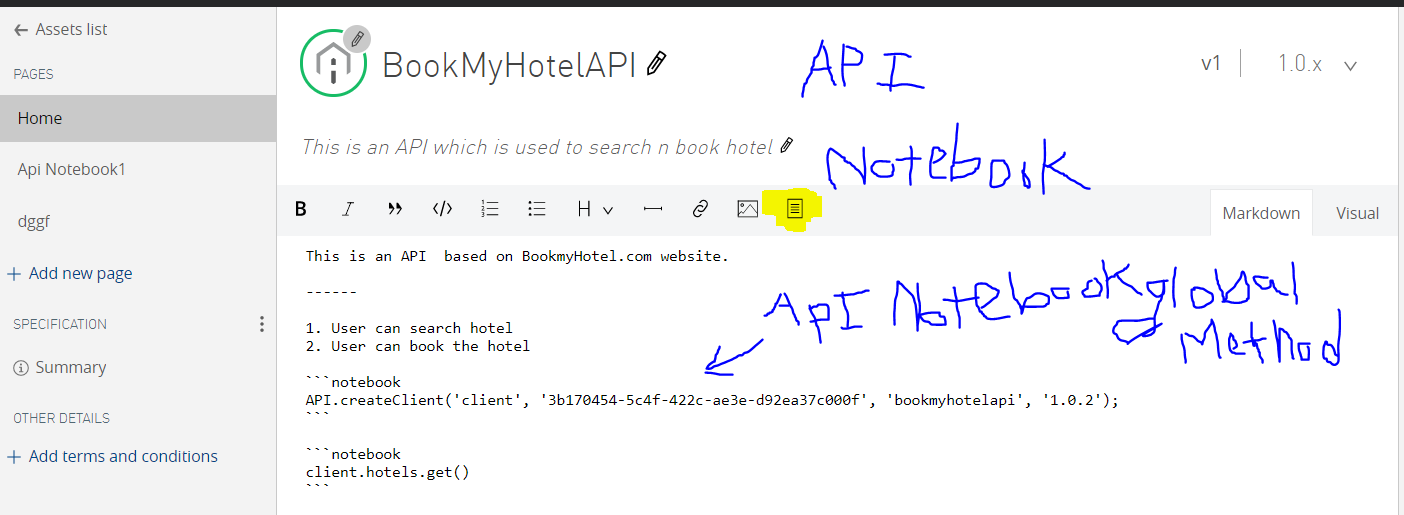
-Uses Javascript syntax

- API Notebook is another way of testing the API.

-API Notebook uses API Specification for testing n uses baseuri stated in the API Specification.

So currently we have stated <http://bookmyhotel.com> as baseuri it will try to call that uri but since that url is not hosted (realone)we will get an failure





* **Versioning**

-We can version the API at the time of publishing becoz in development n review phase there will be so many changes will happen.

API Version Asset Version

V1 1.0.0

V1 1.0.1

V1 1.0.2

V2 2.0.0

-We change the version of api when

1.If changes breaks communication between provider n consumer eg:We need to delete a resource or a method in resource

2. if need change in representation format eg:earlier api was providing application/json but now needs application/xml.

-We don’t do any change in Version of API in case of:

1. any tech change in backend system or data model.

2. no Api version change in case of new resource or method addition

-We can deprecate the api version of older version that is we notify the user that this version of api is not recommended to use.

-Each API has API Version n Each Asset version has asset version.

-For each api version,ultiple asset version can be created.

API Version Asset Version

V1 1.0.0

V1 1.0.1

V1 1.0.2

V2 2.0.0

V2 2.0.1

-Asset Version

Exchange asset version follows semantic versioning.

Semantic version refers to model of major.minor.patch

Format: major.minor.patch

For eg: 1.0.3

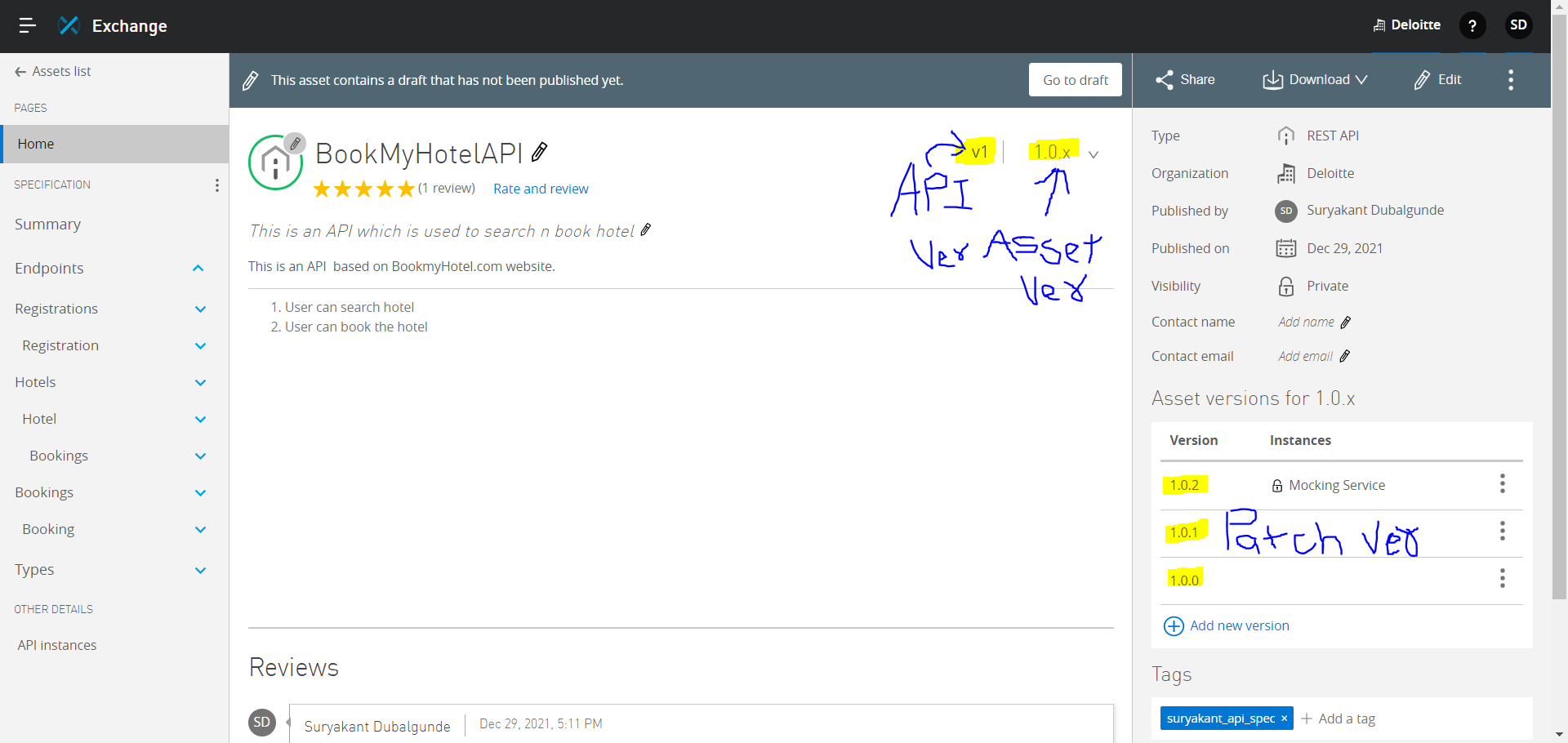
Where 1=major 0=minor 3=patch

Major version :refers to change in Api structure

Minor version:some new changes that doesnot break the api

Eg: adding optional method

Patch version:- bug fix

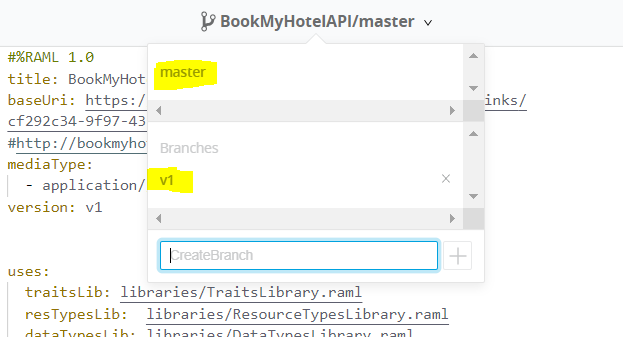


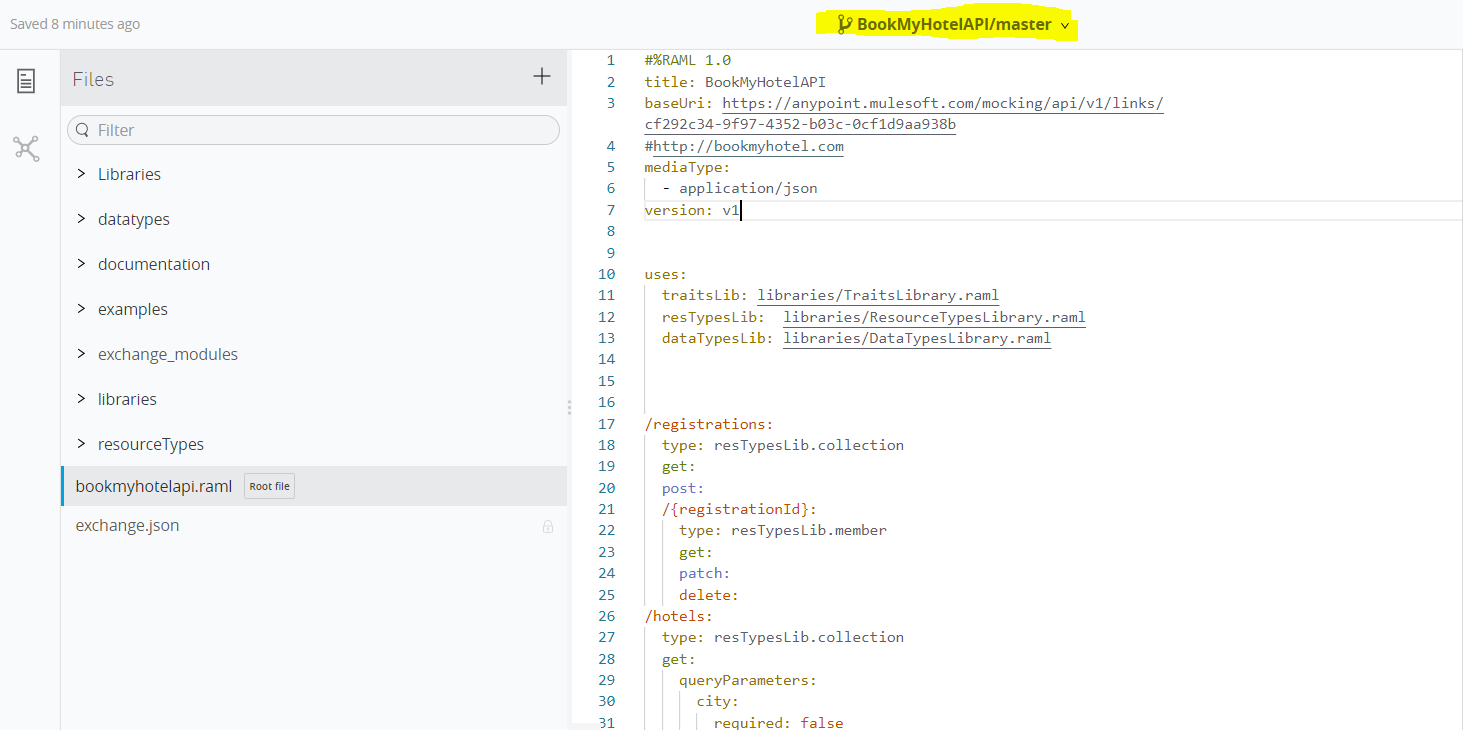
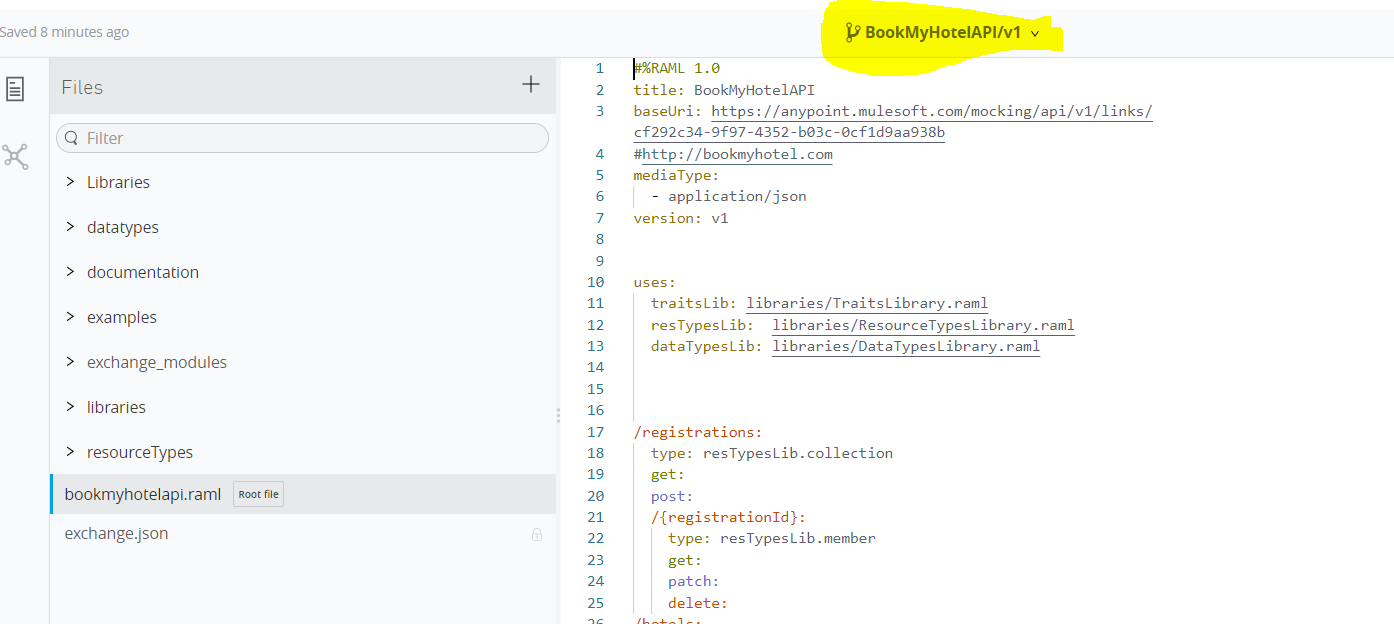
In Asset Versions for 1.0.x it shows only patch versions.

Minor version can be shown in asset ver like 1.0.x, 1.1.x

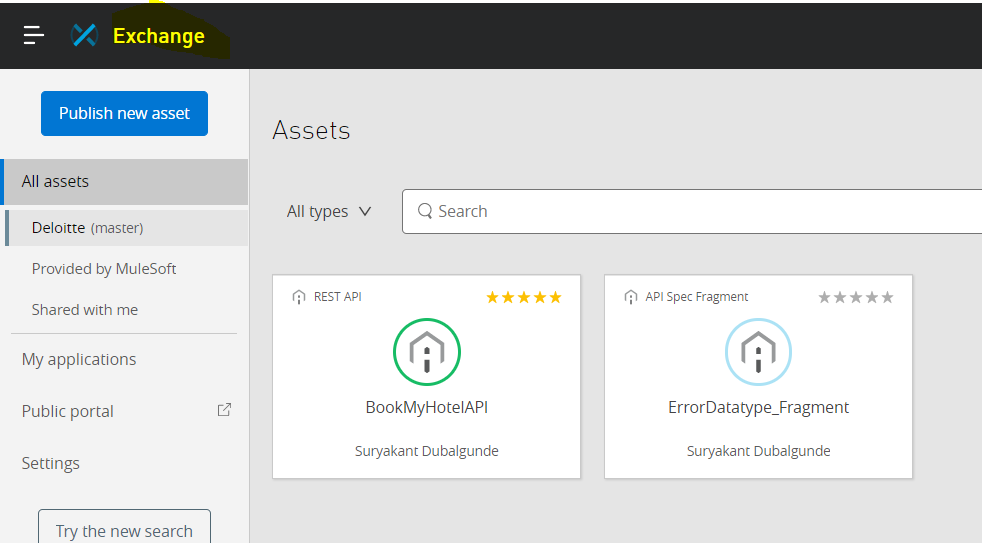
Major version can be shown in asset ver where 1.x, 2.x 3.x

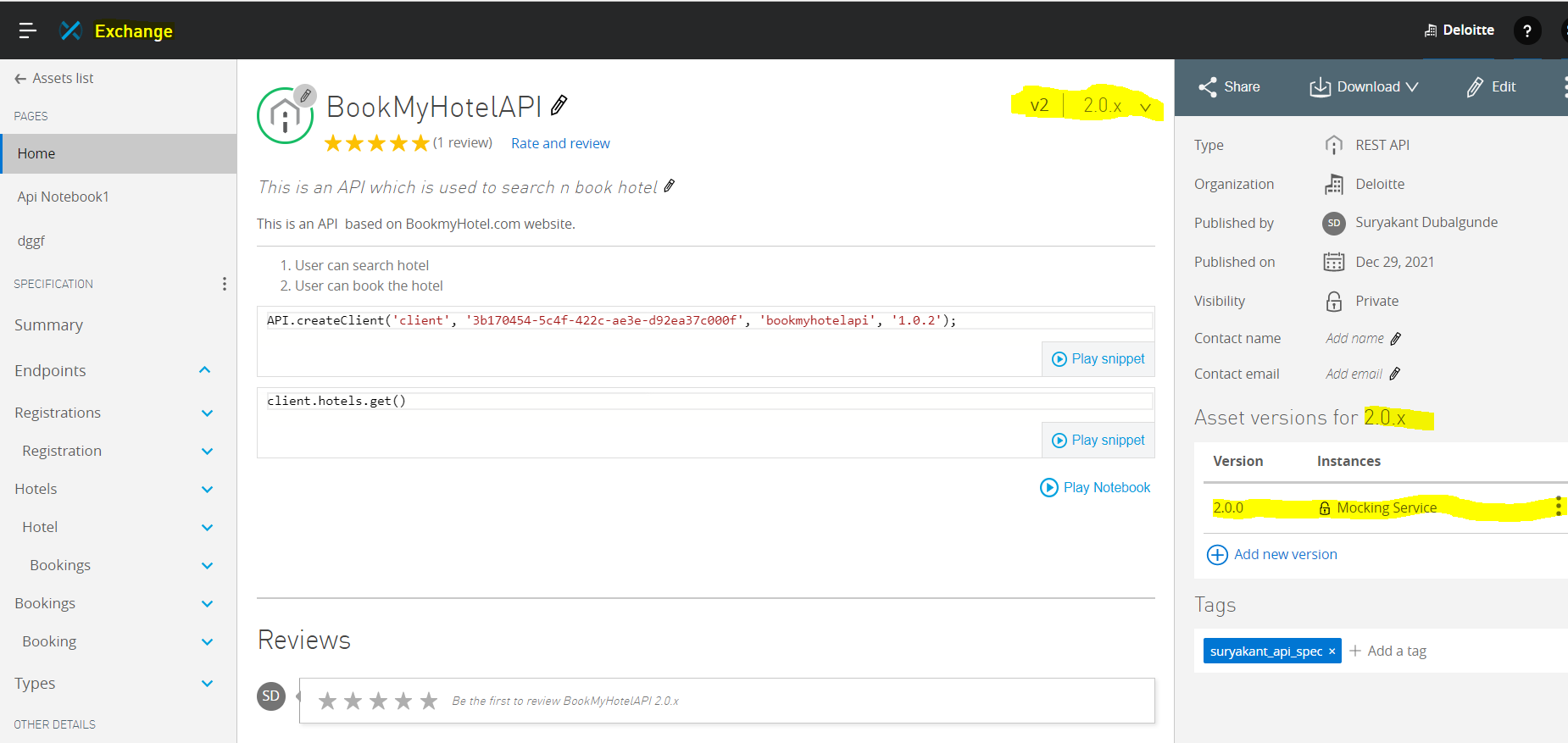
* While changing the API from v1 to v2 version we need to use branching concept like below

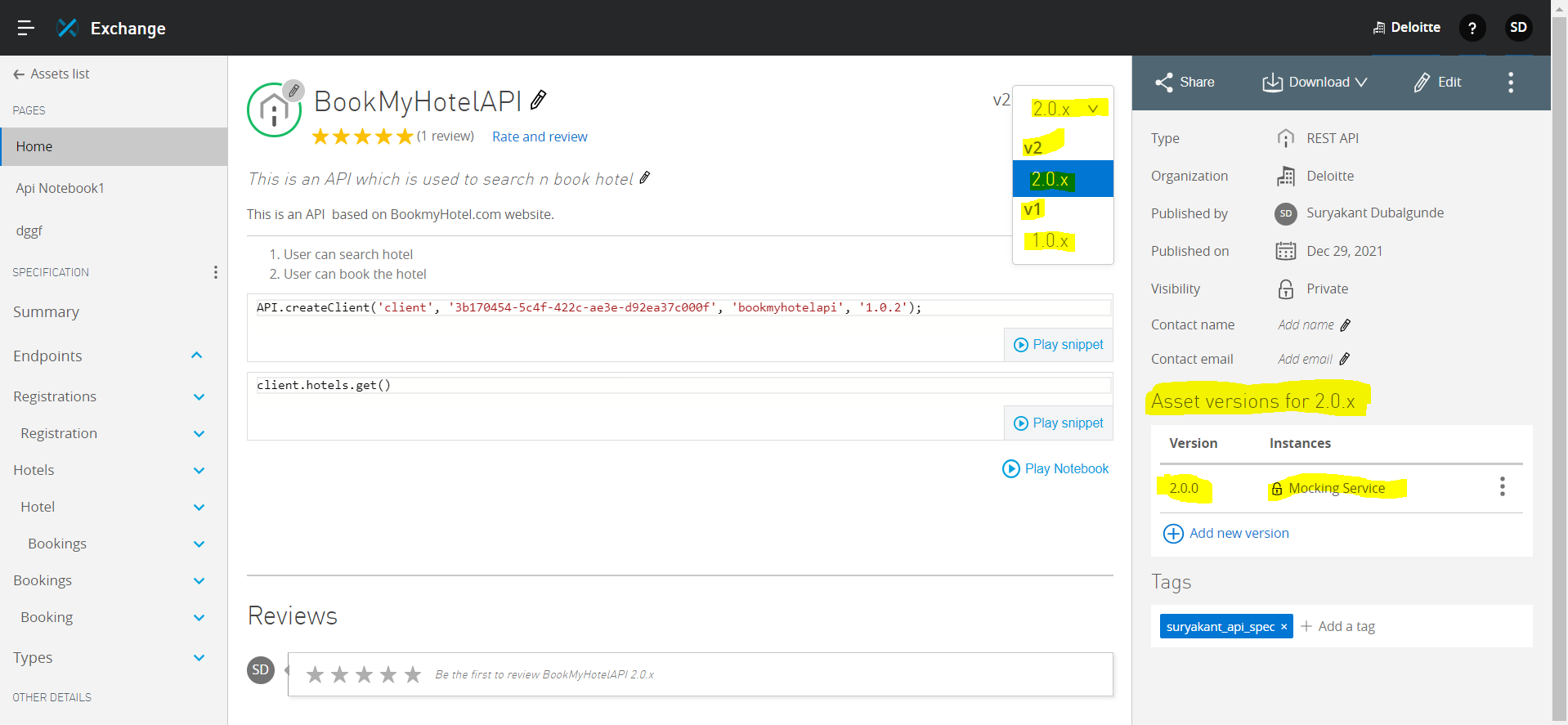
In API Designer,We are doing all raml activities by default in projectname/master branch.So to create v2 version in master branch we need to create a v1 branch where all master branch files will be copied automatically.Then it means like we have v1 n master branch where same code is there so now we can make changes in master branch n will specify version: v2 to create v2 

Master n v1 branch after creating v1 branch(will have same code)  

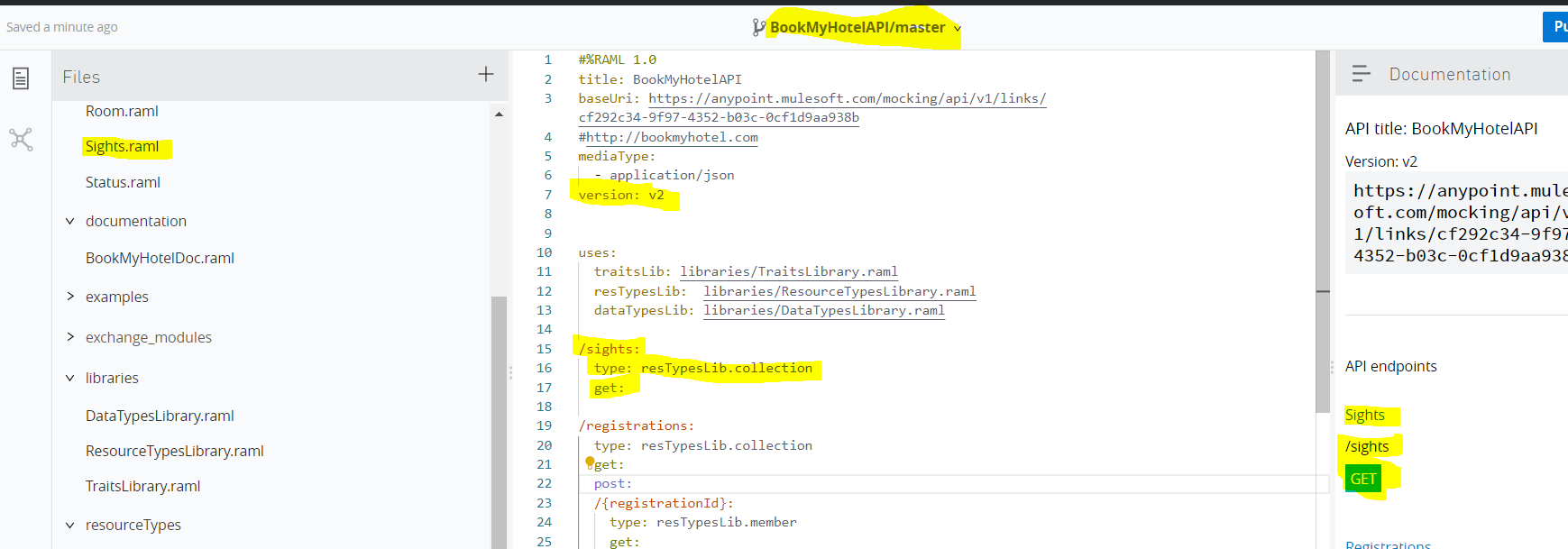
So now we can change version of master branch in root raml file as version: v2 n make required changes in files where applicable 

After publishing the api in exchange , 

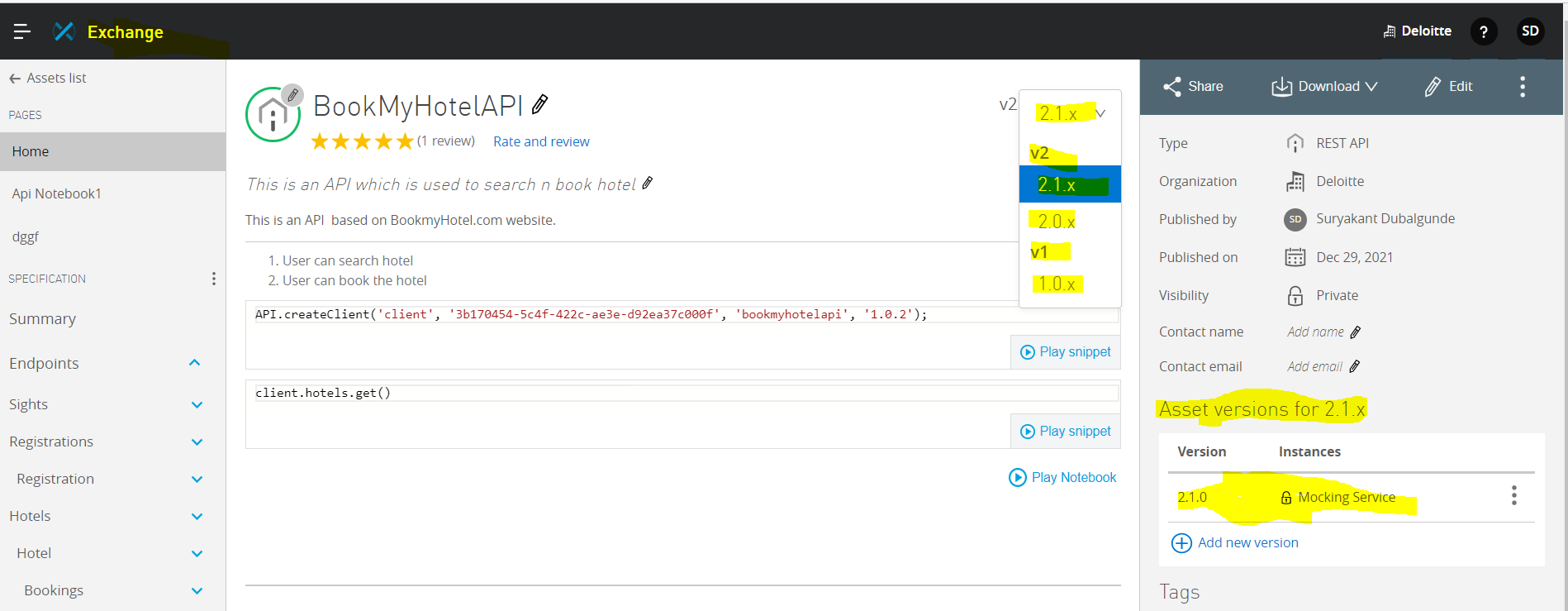




Now will change the minor version by adding/sights resource in Root Raml file…



After publishing the Minor changes within exchange

s

* **Security Schemes**

-Security Schemes are used to secure the access to API Endpoints

-Raml 1.0 supports below built in security Scheme types

1.Basic Authentication

2.Digest Authentication

3. OAuth 1.0/2.0

4.Pass-Though

5.x-other

-For using security scheme in RAML,We specify below

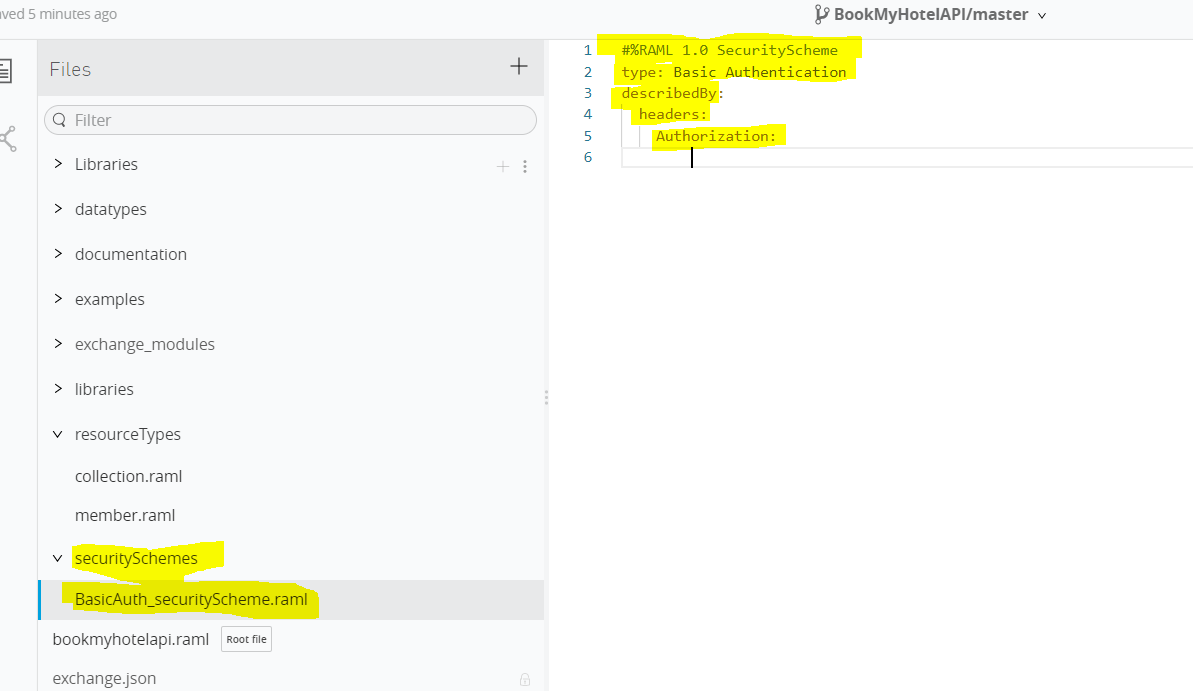
type

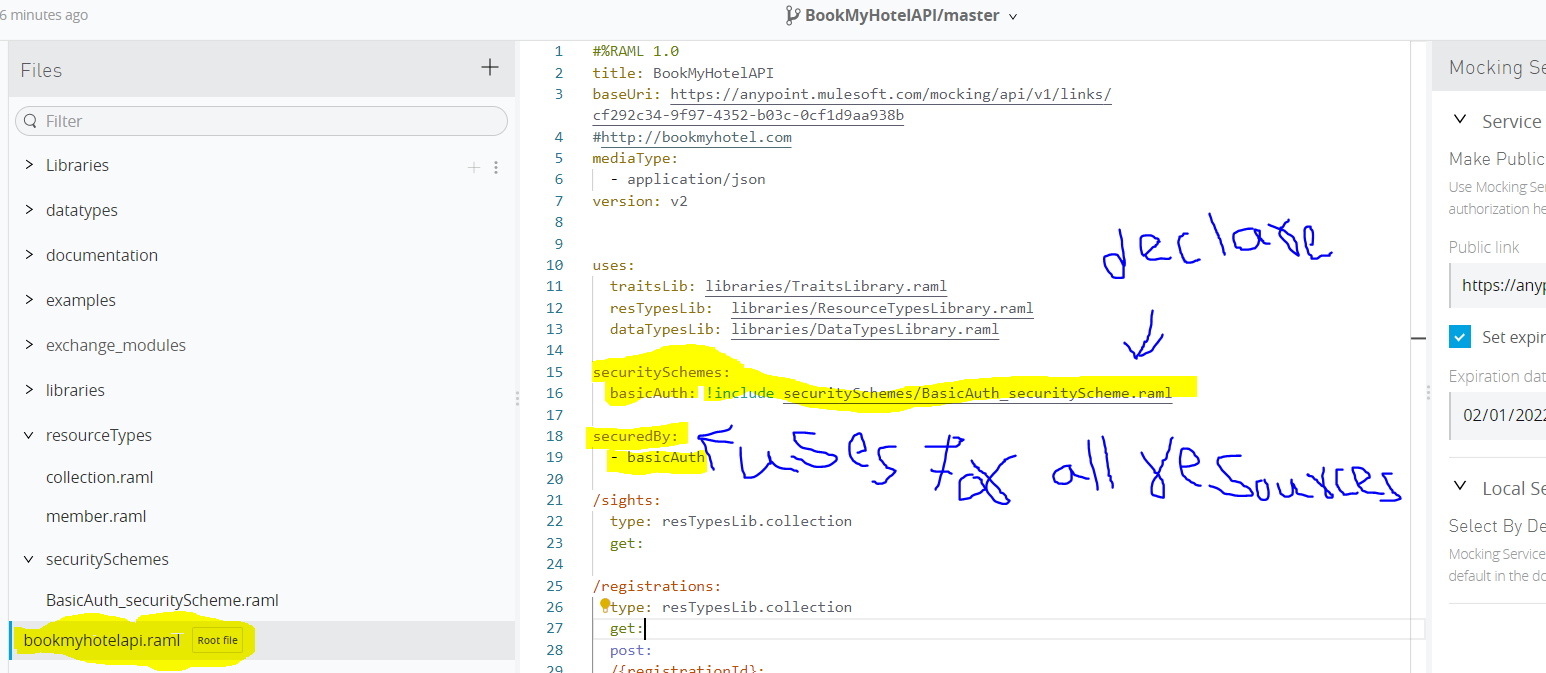
describedBy

settings

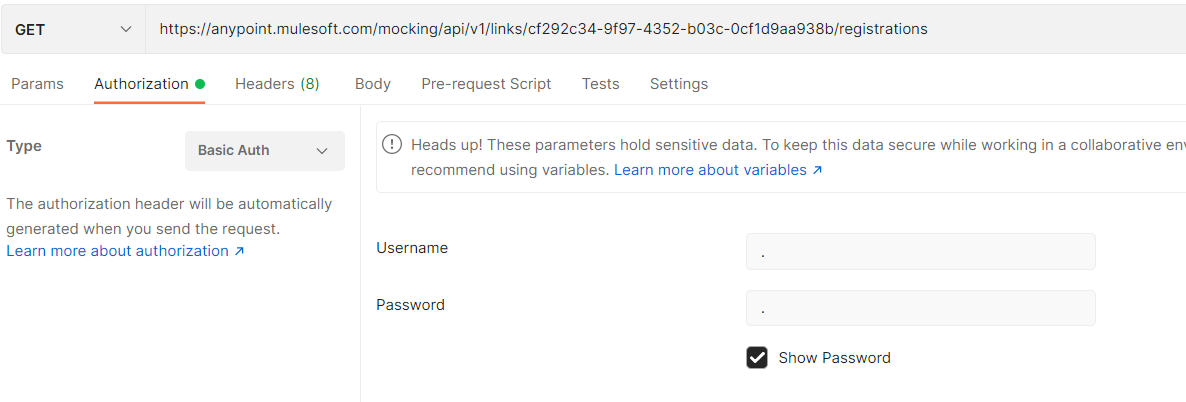
**Basic Authentication:**

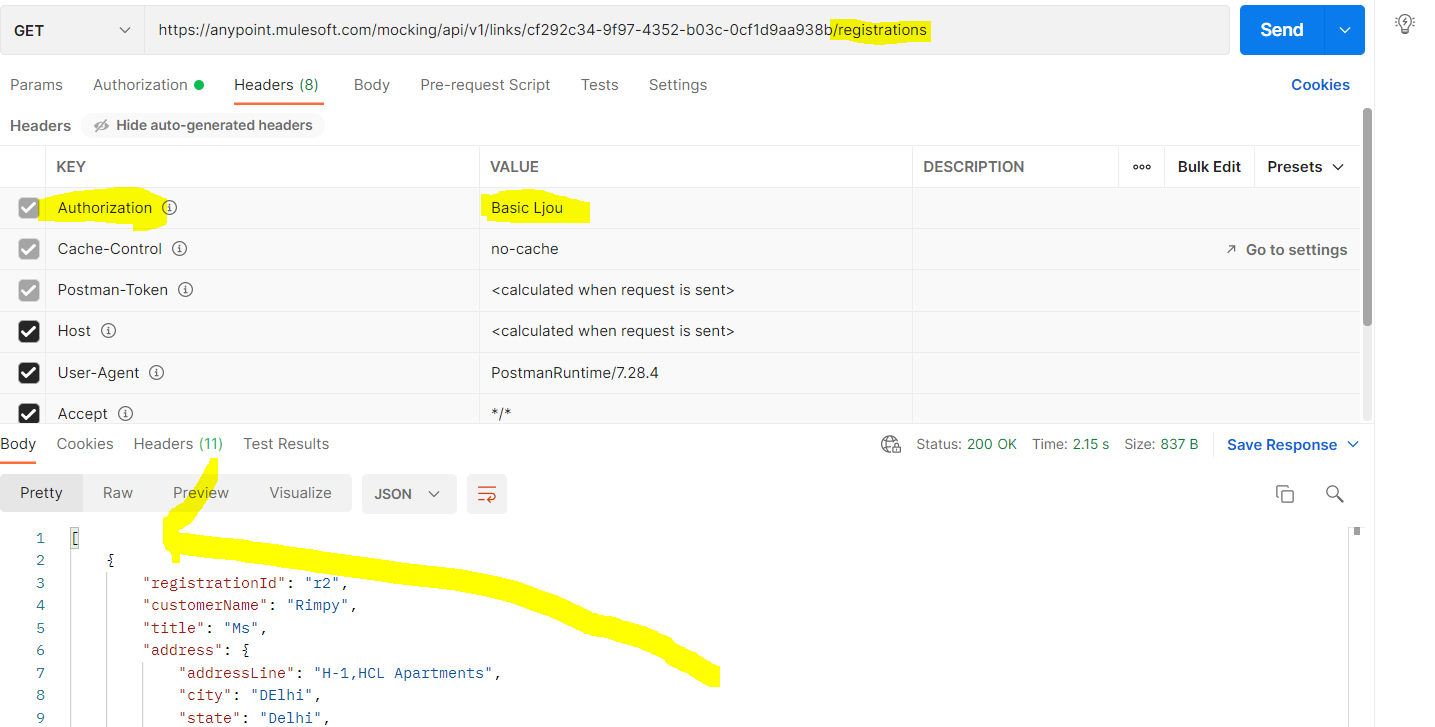
-Basic Authentication involves authentication of user by using username n password.

-Behind the scenes, username n pwd are saved in Authentication Header called Authorization: Basic BMsxd456 where uname n pwd are encoded in base 64 format. 

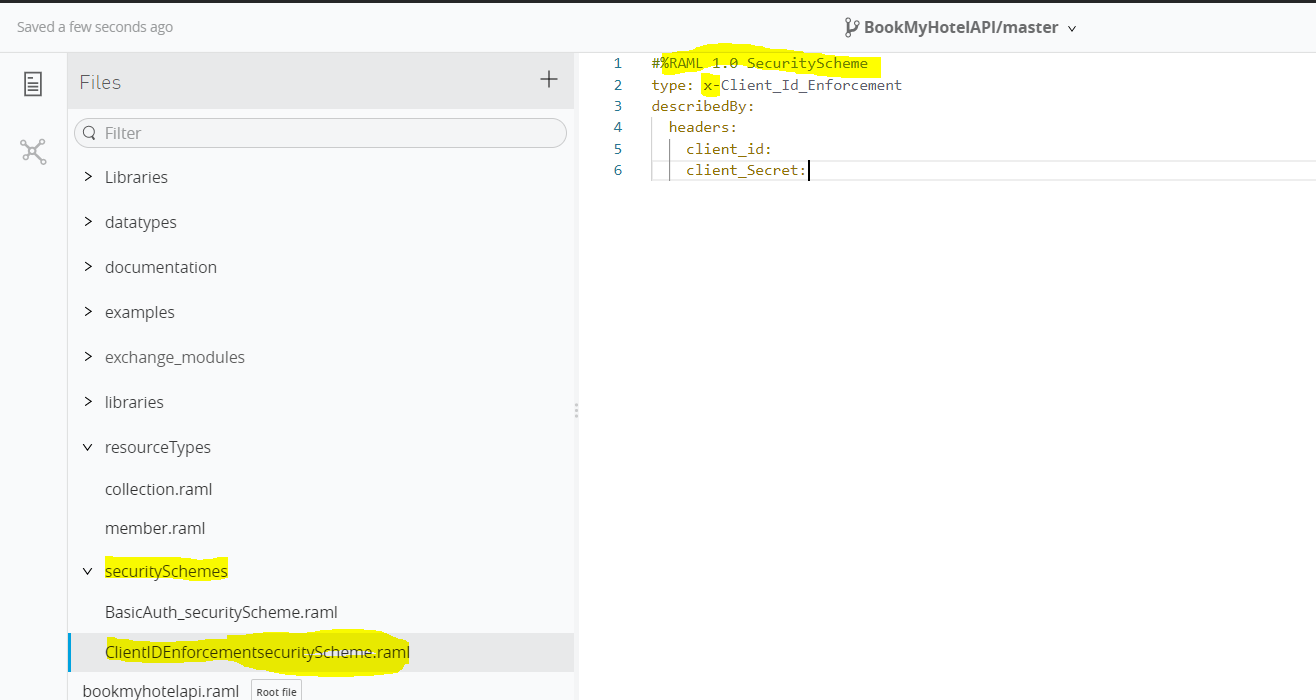


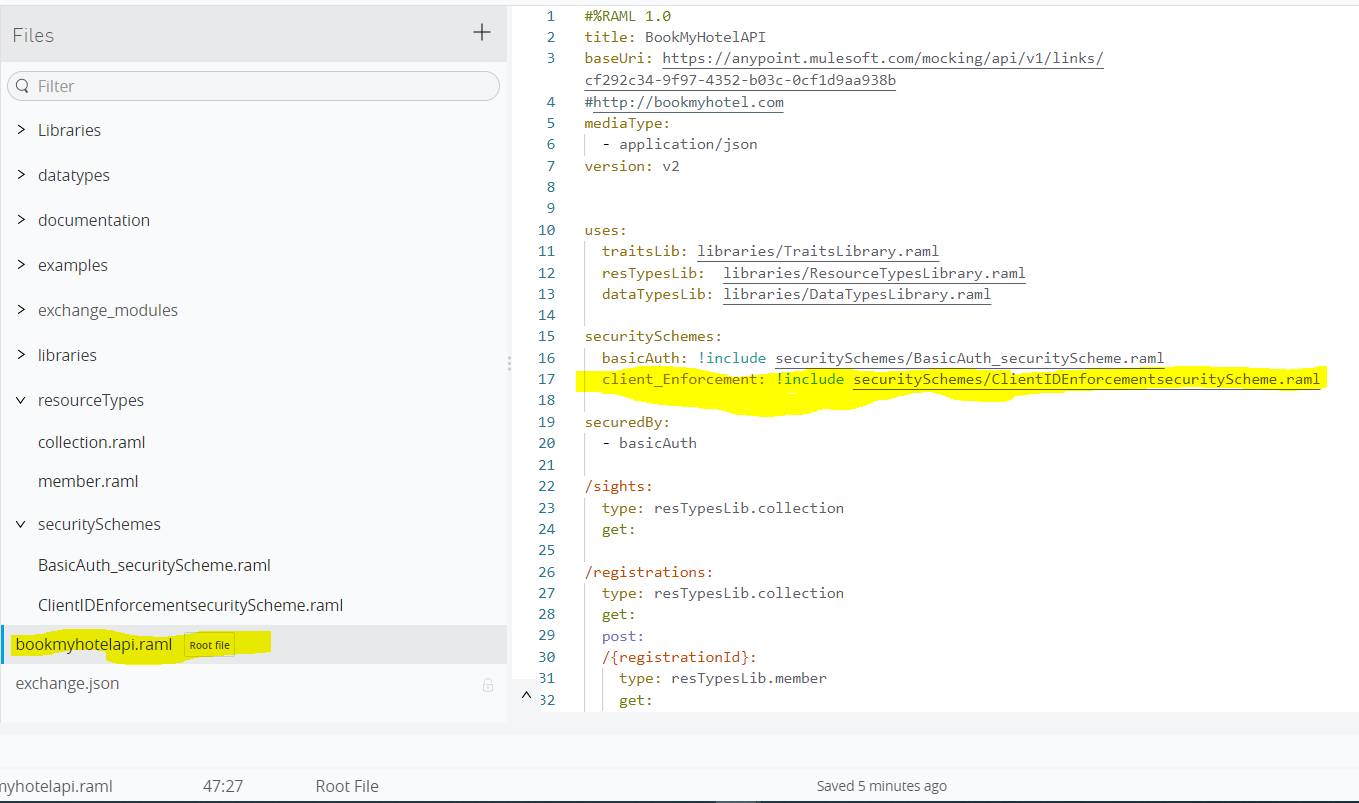
While testing with this security scheme if we don’t provide any value for uname n pwd in mocking service then it will provide error like authorization error is required.

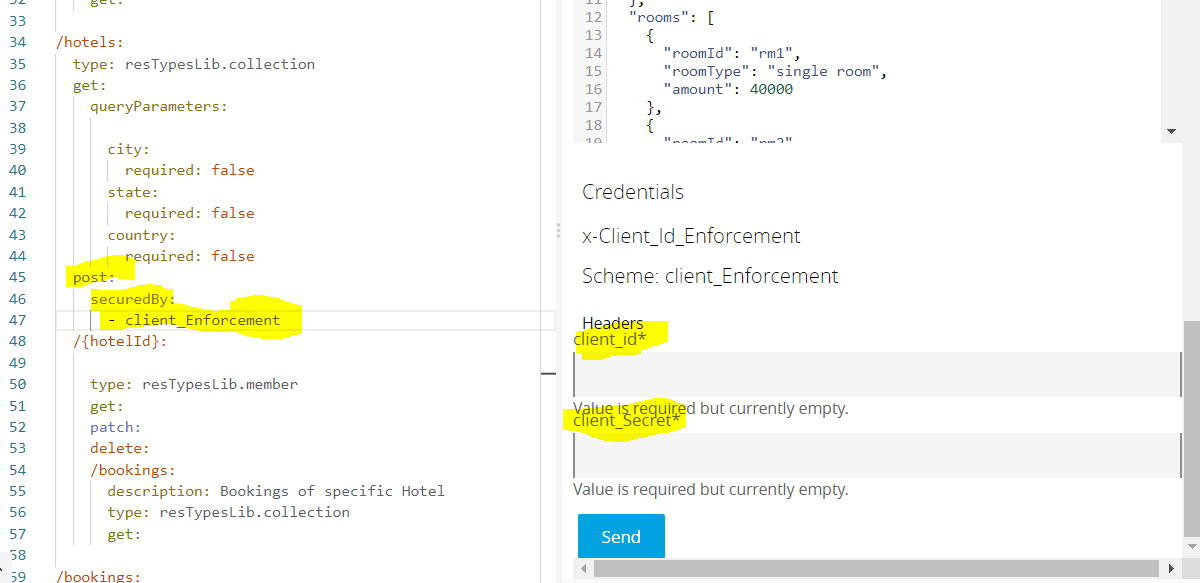
To test this in postman,we have to copy the public mocking link n then paste that link in postman request with get method by appending /resource\_name to copied url, then click send will get error like authorization header is missing so to solve it we have to go to authorization panel n from dropdown we have to select basic auth n then provide uname n pwd value n then we can click on send button now will get response n in header we can find Authorization Header with Basic xxxxxx value. 



* **Creating custom policy with x-**







Any

Scalar Types

* string
* null
* number<-Integer
* date-only
* time-only
* datetime
* Boolean
* File
* Datetime-only

Union

(combination of any datatype like object,string,boolean etc)

Array

(collection of same type.eg:-string array or number array)

Json Schema

XSD Schema

object

Response-Header

Content-Type

Request-Header

Accept

GET

Request-Header

Content-Type

Response-Header

Location

Request-Header

Content-Type

Response-Header

None

Response-Header

Content-Type

Request-Header

Accept

PATCH

POST

DELETE