

REST API Notes

# Date: 12 Jan 2023 (Old 23/08/2020)

End Point OR BaseURL - where API s hosted on the server.

Methods are to communicate with the Rest API.

Get - uses Query Params

Post - user form Params and requests body/body payload to send data.

Put - Replaces all current representations of the target resource with the uploaded content. It uses Query Params and Payload

Delete - Removes all current representations of the target resource given by the URI. It uses Query Params and Payload

Resources - BaseURL/resourceNameORCollectionName

Path Params: to point to the specific resource within the collection. Such as ID. Starts with / slash after resourceName.

Query Params: to sort the collection. Starts with the ‘?’ mark. For e.g. BaseURL/resourseName?key=qa123&place\_is=222

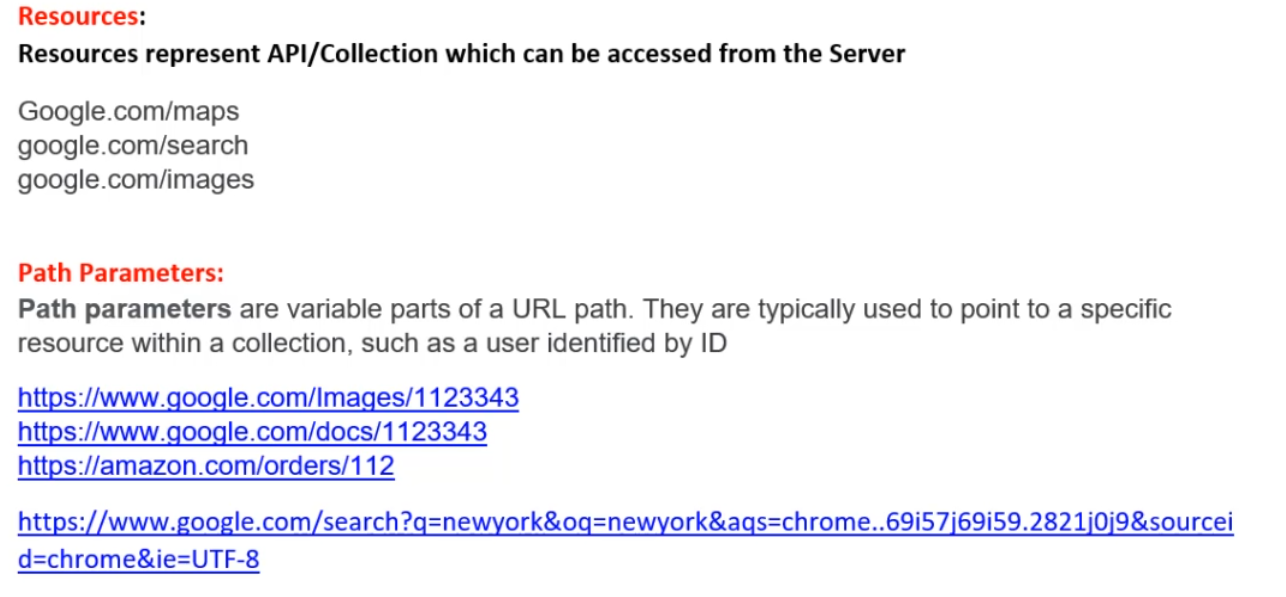
Headers/Cookies: Metadata associated with API request and Response. It is also an additional detail for API to process our request. For e.g Content-Type - application/json

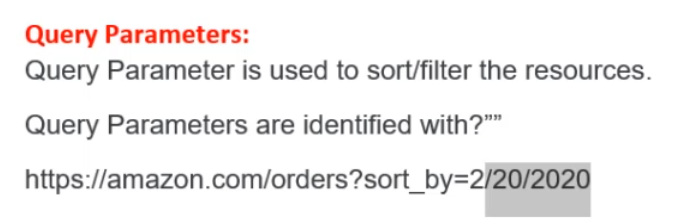
Response: it usually has Response body, headers, cookies

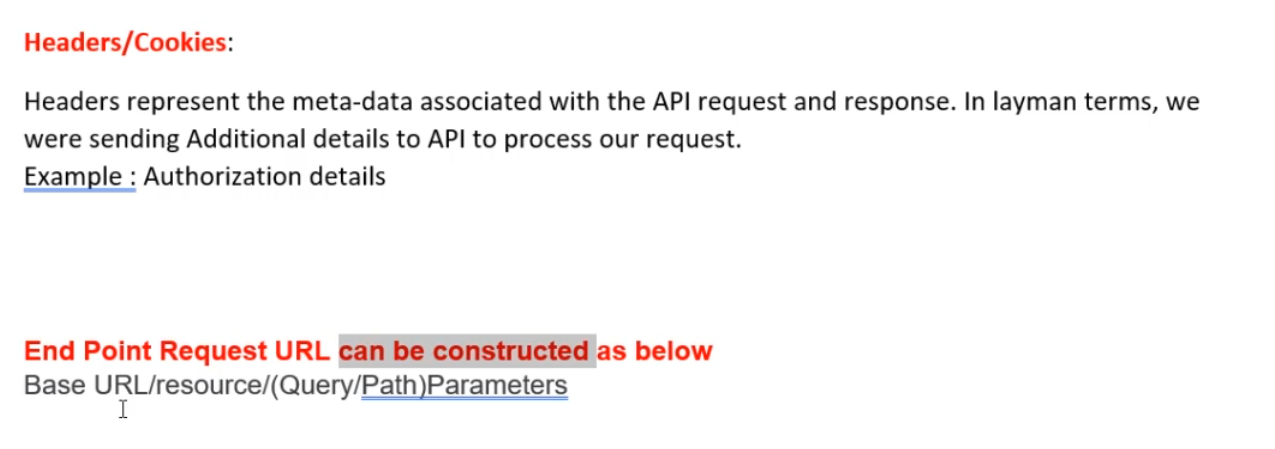
Status 200 Ok

Time: 123ms

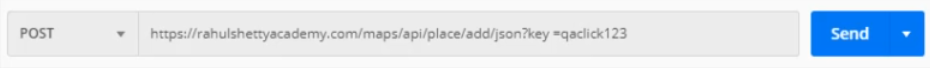
Size 120 B

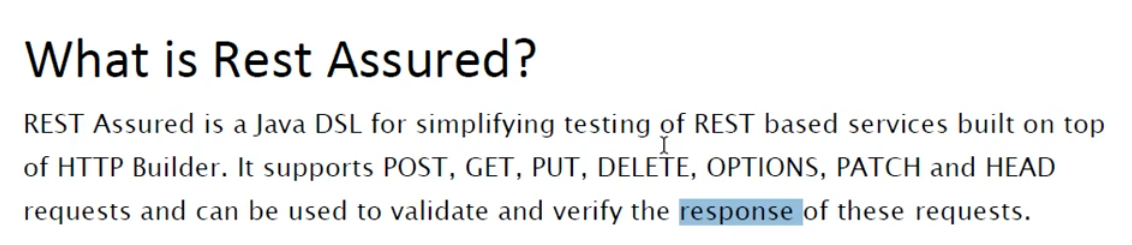












**Rest Assured Automation Setup:**

1. Download Java from the below link http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

2. Setup Environmental variables Path

* JAVA\_HOME - JDK PATH
* PATH - JDK/BIN PATH

3. Download Eclipse from the below link https://eclipse.org/downloads/

4. Download Rest Assured Jars from the below link https://github.com/rest-assured/rest-assured/wiki/Downloads

5. Set up Java Project in Eclipse and configure jars in the build path

**Download the following latest versioned Jars and add them to the Java build path.**



**Required Jars/Dependencies**

* Rest Assured
* JSON Path\*
* XML Path\*
* JSON Schema Validator\*
* JUNIT/TestNG
* Jackson Databind, Jackson Annotation and Jackson Core
* Gson

# RestAssured works in the format:

1. RestAssured.baseURI
2. .given()
   1. .header(“key”, ”value”)
   2. .queryParam(“key”, ”value”)
   3. .body(“<actualPayloadAsString>”)
      1. OR .body(“<CallTostaticPayloadReturningMethod>”)
3. .when()
   1. .post(“resourceName”) OR similarly get() OR delete() OR put() OR update()
4. .then()
5. .Log().all() // to log all the request or response specifications
6. ..assertThat()
   1. .statusCode(200)
   2. .body(“key”, eqaulTo(“value”))
   3. .header(“key”, ”value”)

In the main test Java class do the following

* Import the class as

import io.restassured.RestAssured;

* In the main method set the base URI as

RestAssured.baseURI = "<https://rahulshettyacademy.com>";

* Start with mentioning the given details such as query params, headers, and body as
  + given() // to use this given() method we have to static import the following RestAssured class — **import static io.restassured.RestAssured.\*;**
  + .log().all() // we can user Log all to print request details on console
  + .queryParam("key", "qaclick123")
  + .header("Content-Type", "application/json")
  + .body("<**actualStringpayload**>")
* Without ending the code continue with the when() and request resource and method details as
  + .when()
  + .post("/maps/api/place/add/json")
* After that use the then() to assert items from the response body details as
  + .then()
  + .log().all() // we can user Log all to print response details on console
  + .assertThat()
  + .statusCode(200)
  + .body("scope", equalTo("APP"))
  + // we have used a Hamcrest Matchers class method called equalTo() to compare any of the response body field and value.
  + // to use this equalTo method **we have to static import this class — import static org.hamcrest.Matchers.\*;**
  + .header("server", "Apache/2.4.41 (Ubuntu)");
  + // to compare the RESPONSE Headers **we can directly use the required header Key and value** as string in the .header() method unlike equalTo()**;**
  + We can also extract the response as a string and save it to a String variable.
    - **String resp** = given()......
    - .when()……….
    - .then()………..**extract().response().asString();**

## Parse the JSON response body using the JsonPath class

* + In order to Parse the JSON response we can use the JsonPath class.
    - Import the class as **import io.restassured.path.json.JsonPath;**
    - Pass the response string variable to the JsonPath object as
      * JsonPath jsp = new JsonPath(resp);
      * String placeId= jsp.getString("place\_id");
      * System.out.println("The placeID is = "+ placeId);
    - We can also create a static method in a ReusableMethods.java class.
      * JsonPath jsobj2 = ReusableMethods.responseStringToJsonObj(getResponse);
      * String actualAddress = jsobj2.getString("address");
      * System.out.println(actualAddress);
    - We can add TestNG dependency in the POM.xml and to the project and use the assertion to compare the variable we fetch from the response. For E.g.
      * Assert.assertEquals(newAddress, actualAddress );

Chanpet 5 video 3 parsing json over

# **Advance Payload Creation Strategies**

## Dynamic Build Json payload with external data input

* 1. Test class body method will call payload with user defined parameters E.g. .body(Payloads.addBookPayload("nk", "260103"))
  2. Payloads class is having parameterised method returning String Payload E.g. *public static* String addBookPayload(String isbn, String aisle){.....}

## Parameterize 1 test with multiple data sets/payloads

* 1. Using TestNG DataProvider annotation with the single or multidimensional array. The test will run as many times as many data sets you have For e.g.

### DataProvider use Example

*@DataProvider*(name = "myBooksDataProvider")

*public* Object[][] getDataMethod() {

*return new* Object[][]{{"nk1", "260111"}, {"nk2", "260112"}, {"nk3", "260113"}, {"nk4", "260114"}};

}

### @Test use Example

*@Test*(dataProvider = "myBooksDataProvider")

*public void* parameterizedJsonPayloadTest(String isbn, String isle) {

RestAssured.baseURI = "http://216.10.245.166";

String addBookResponse =

given().log().all()

.header("Content-Type", "application/json")

.body(Payloads.addBookPayload(isbn, isle))

## Send Static Json Payload files directly into method

### Direct string in the body E.g.

.body("{\n" +

"\"place\_id\":\""+addedPlace\_id+"\",\n" +

"\"address\":\""+newAddress+"\",\n" +

"\"key\":\"qaclick123\"\n" +

"}\n")

### Calling static payload method in the .body(), from Payloads class

.body(files.Payloads.addPlacePayload())

### Create JSON file on your computer and call the path in the .body() method

#### Create a file path reading static method

*public static* String generateStringFromResource(String path) *throws* IOException{

*return new* String(Files.readAllBytes(Paths.get(path)));

}

#### Call it in the .body() method

given().log().all()

.header("Content-Type", "application/json")

.body(generateStringFromResource("C:\\Users\\Hp\\Desktop\\AddBookJsonFile.json"))

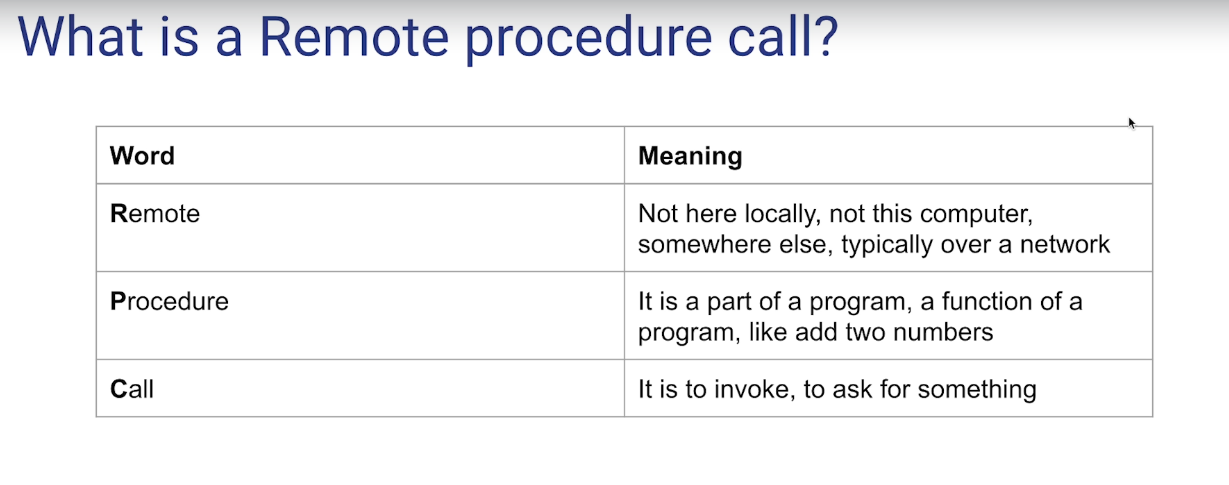
### Build JSON payload using excel and hashmap

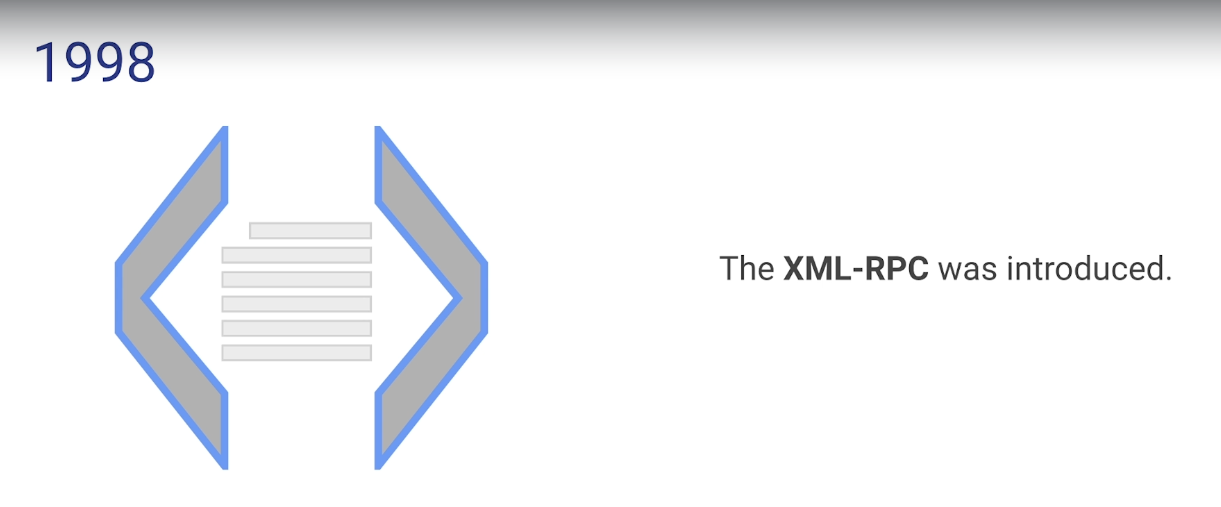
### Build Pojo classes for JSON payload

# 

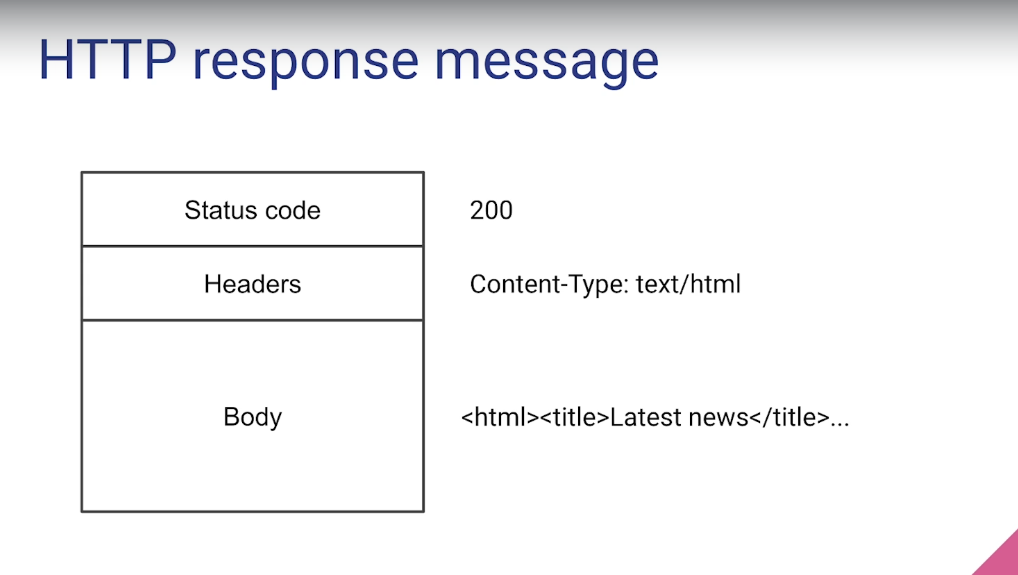
# 

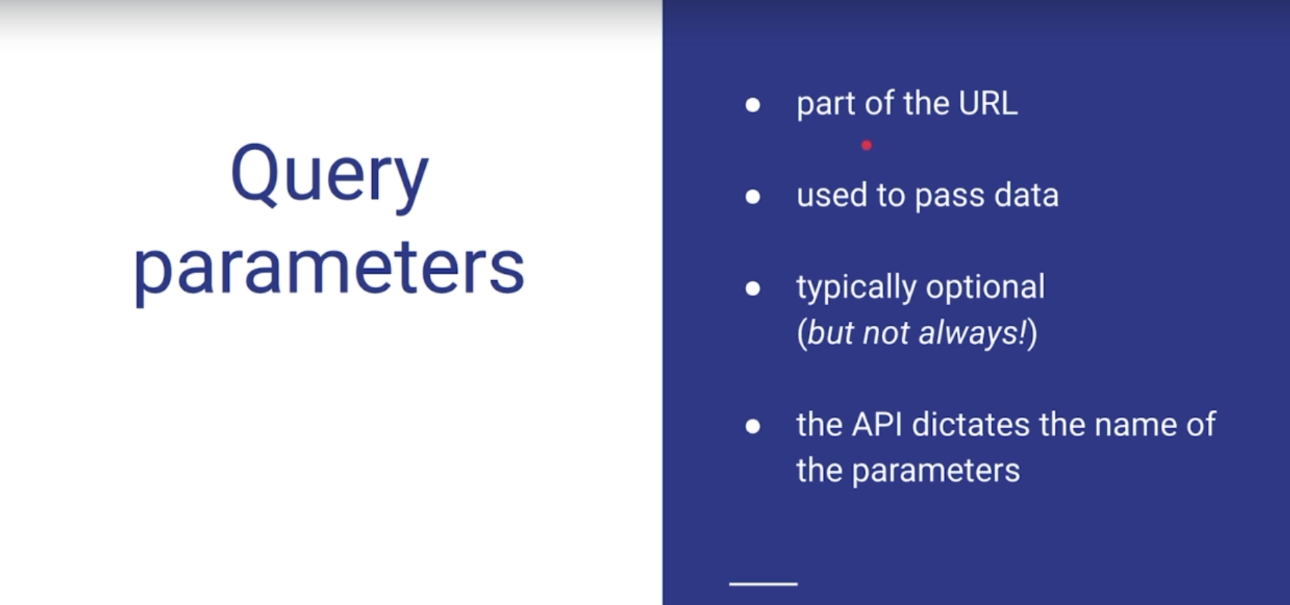
# Remote procedure call

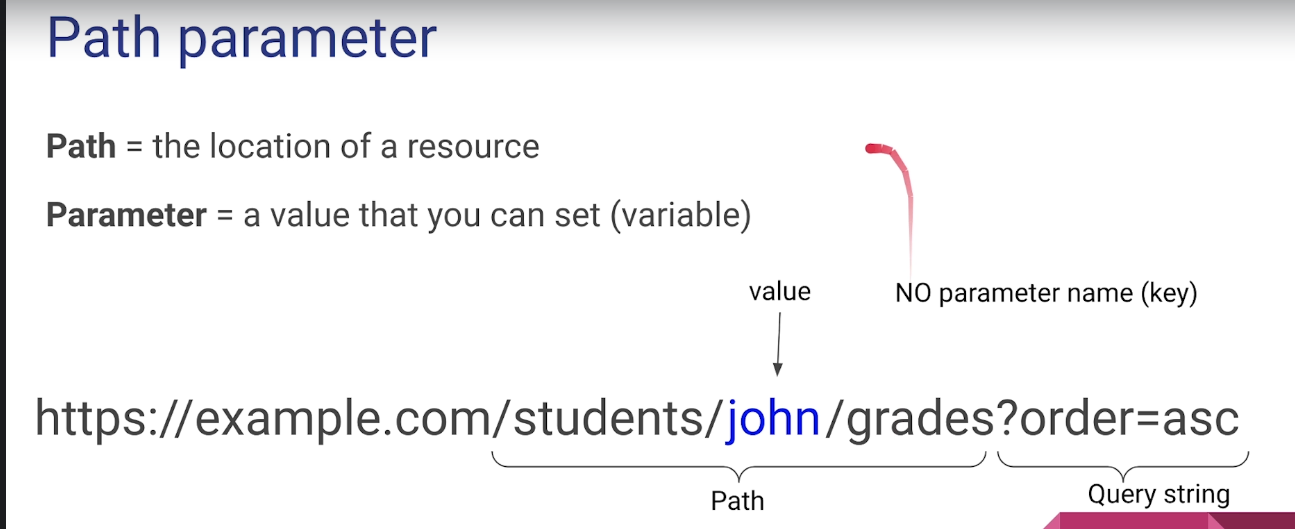


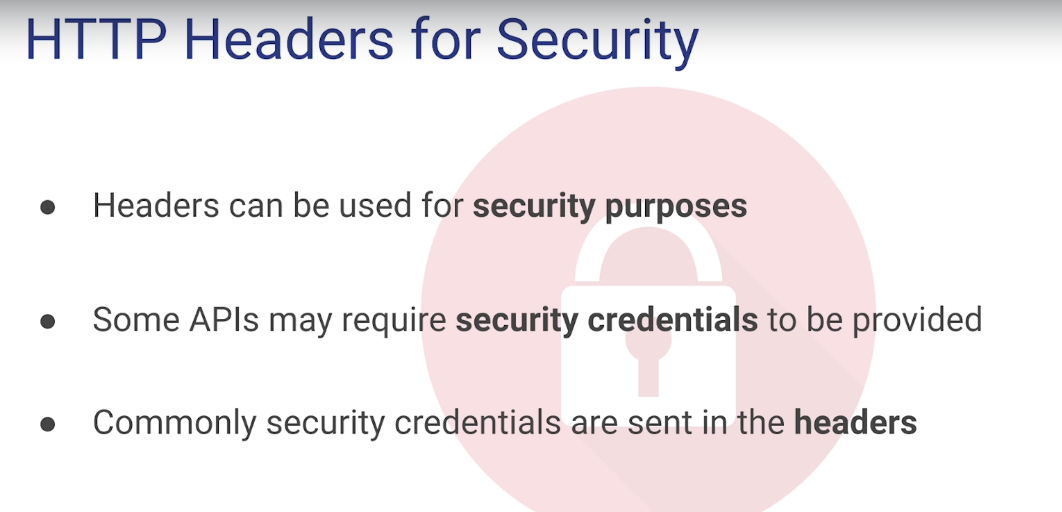


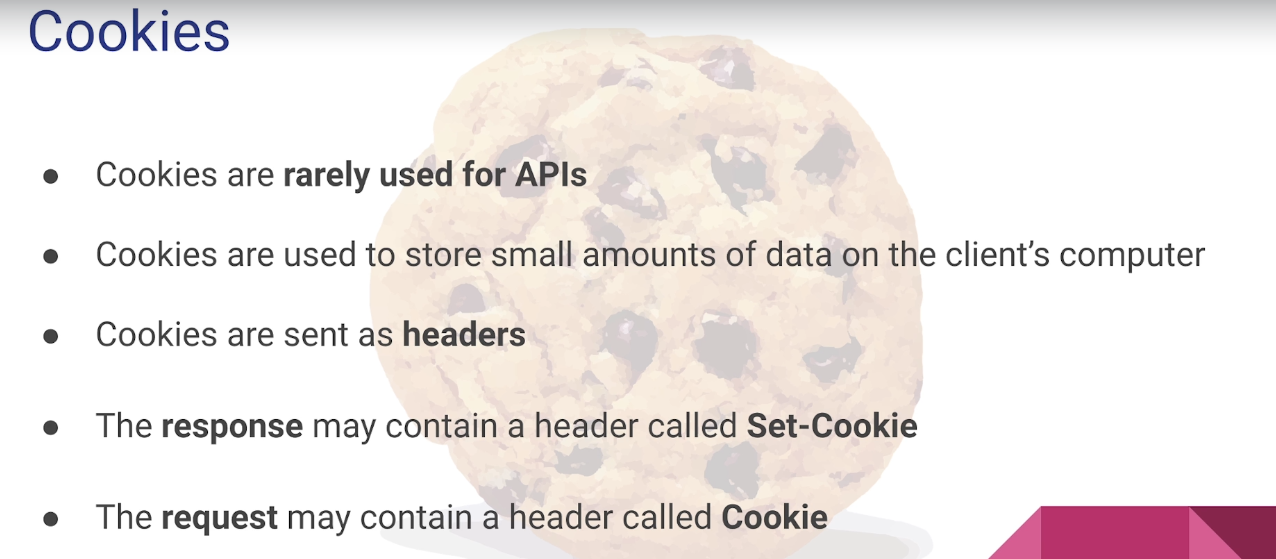
## 

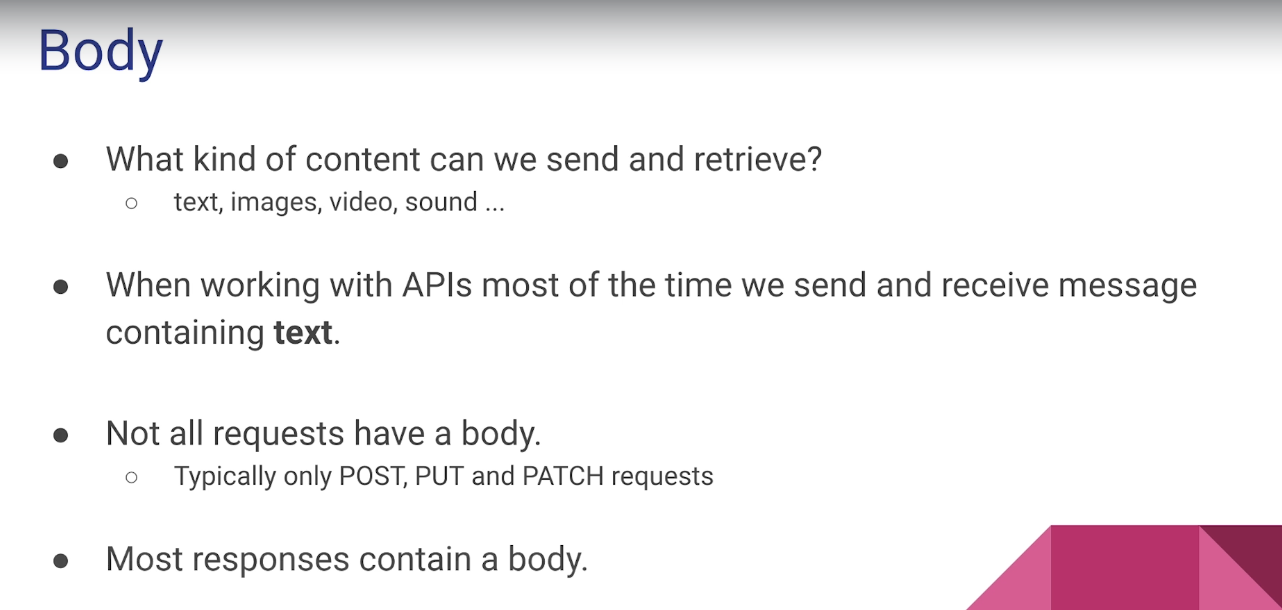


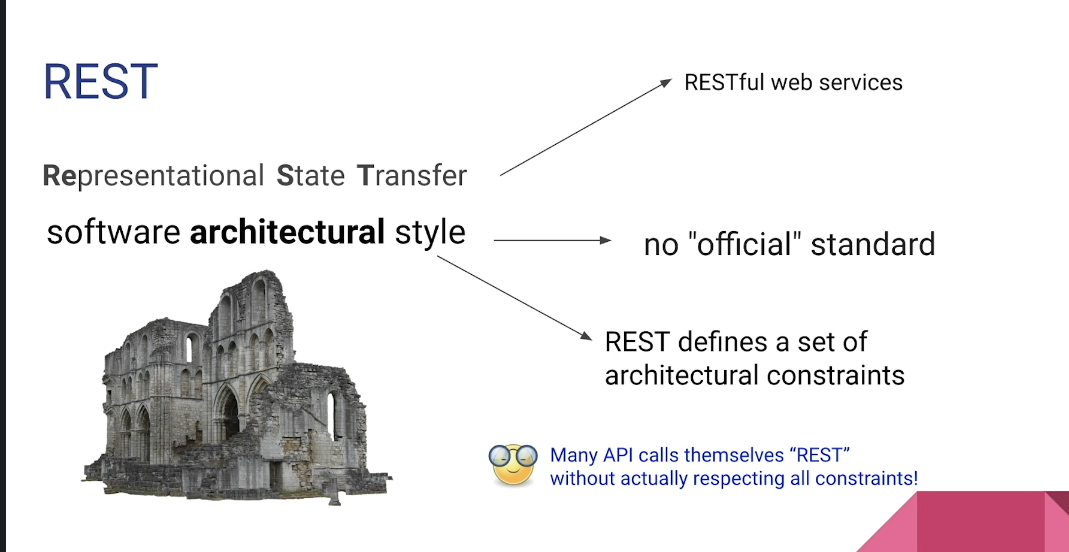


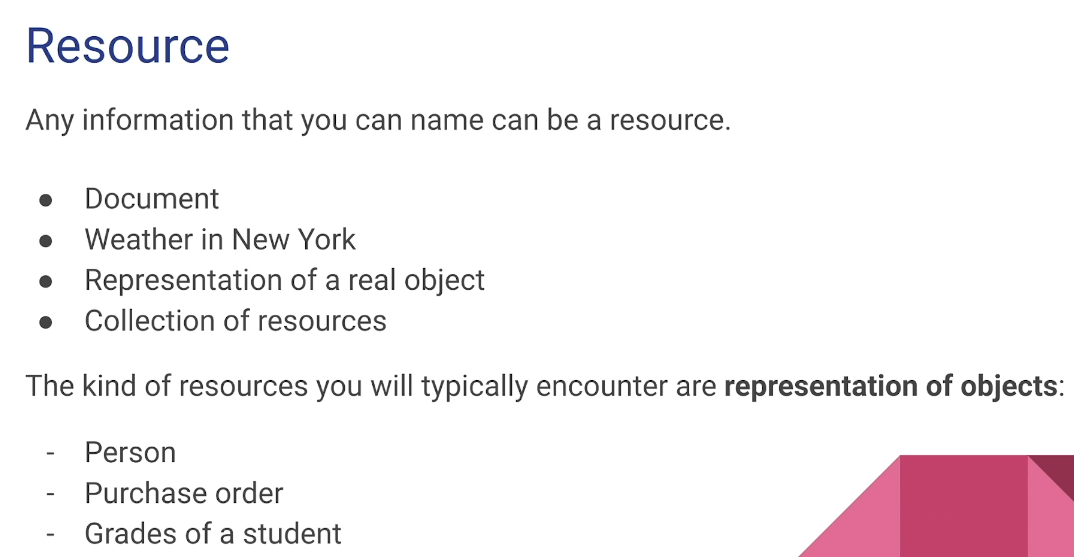


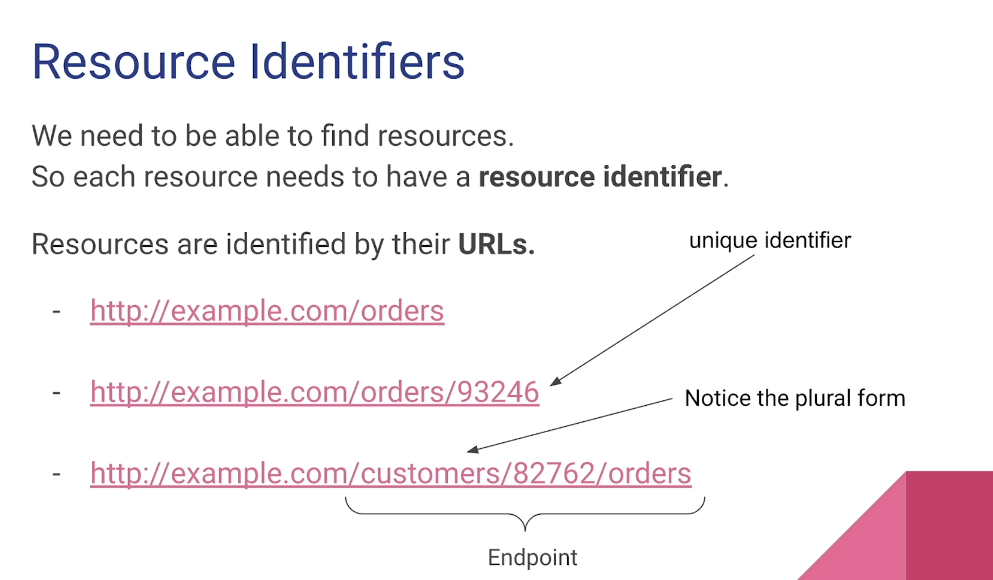






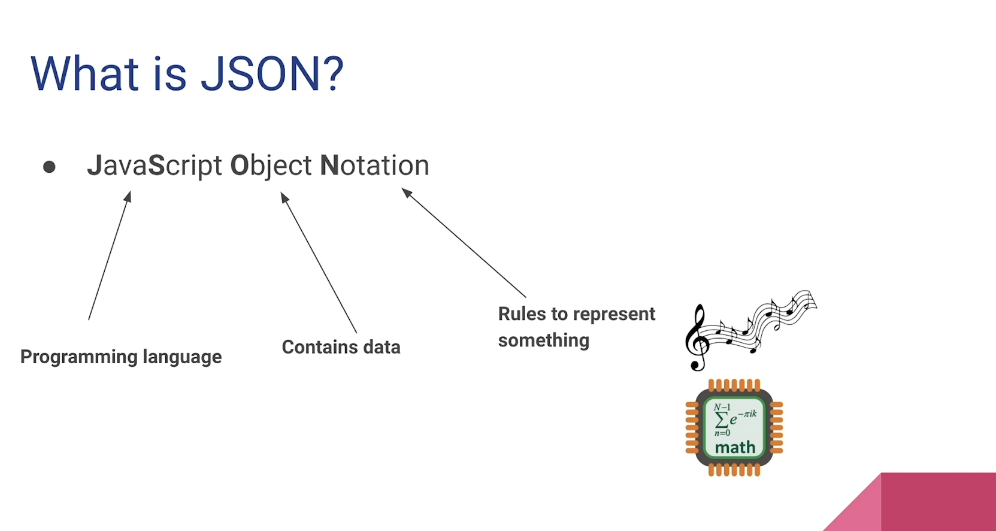


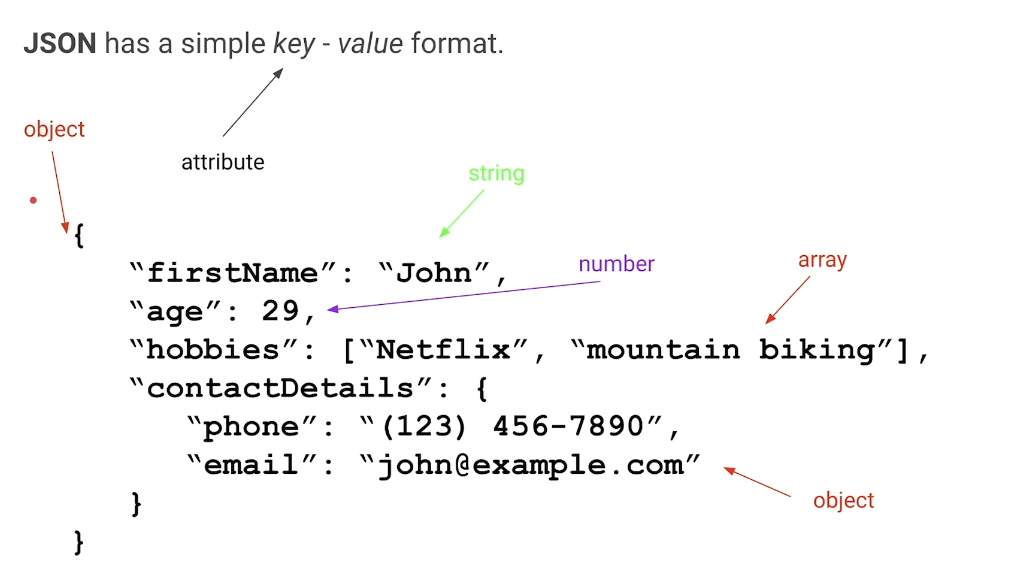


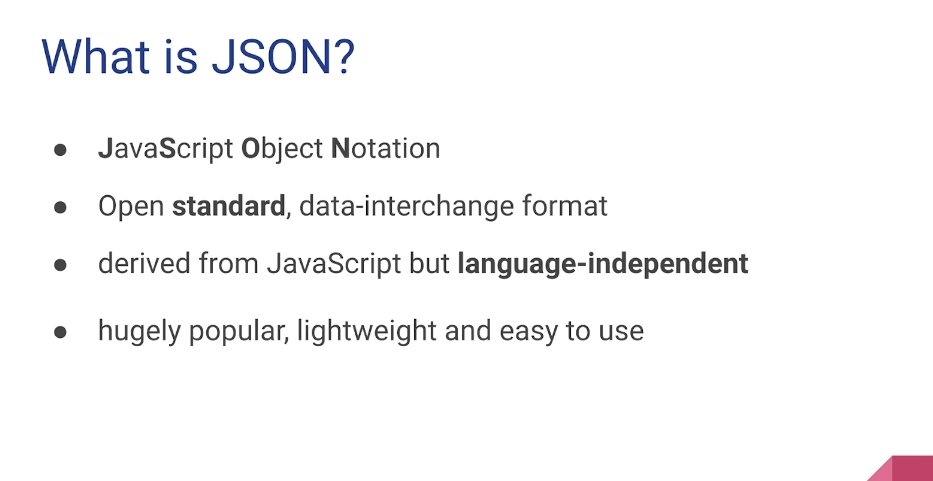


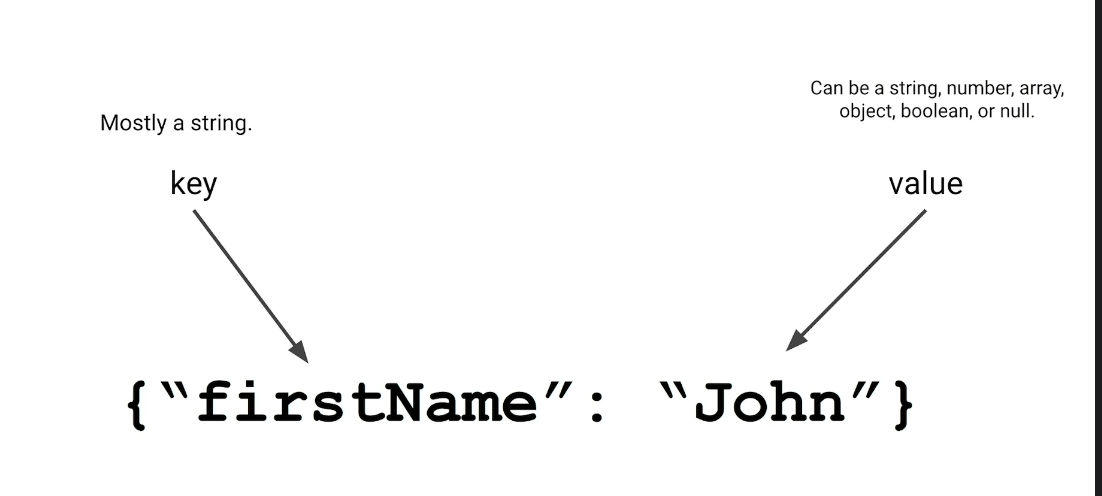
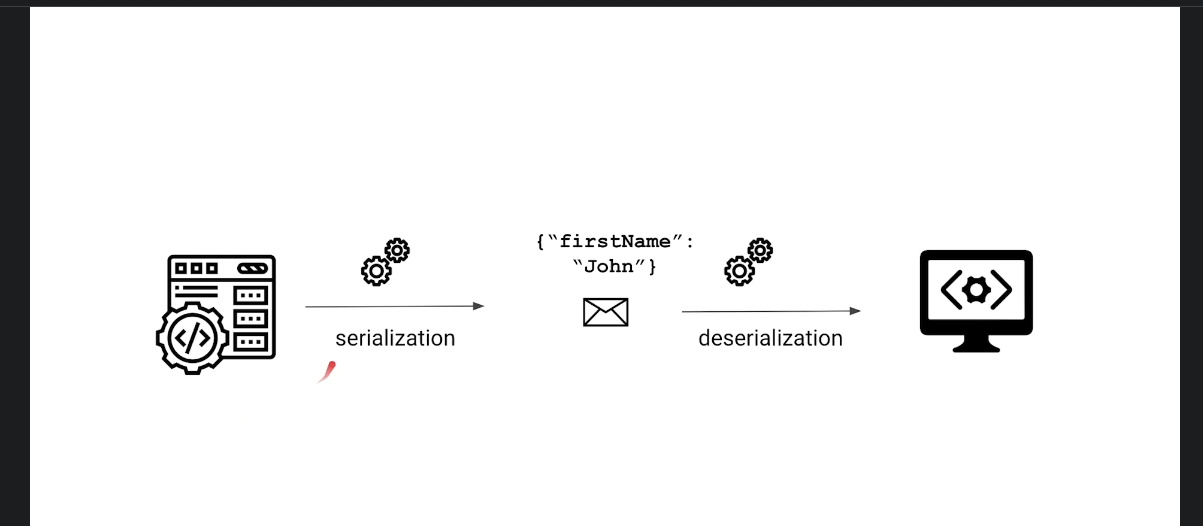


API refers to the whole set of protocols that allows communication between two **systems** while an **endpoint** is a URL that enables the API to gain access to resources on a server.

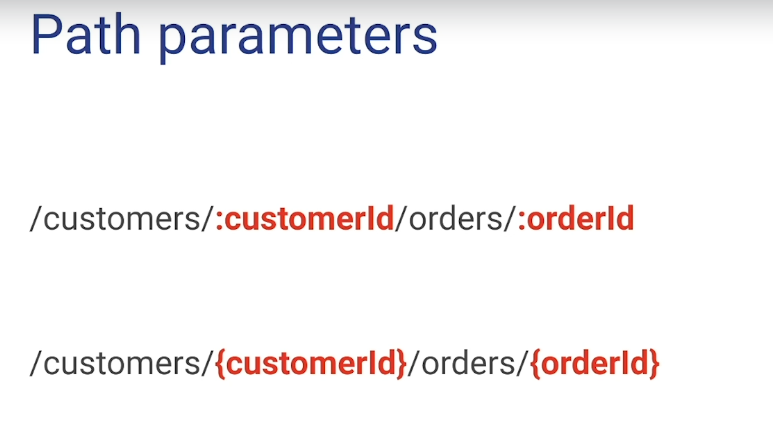


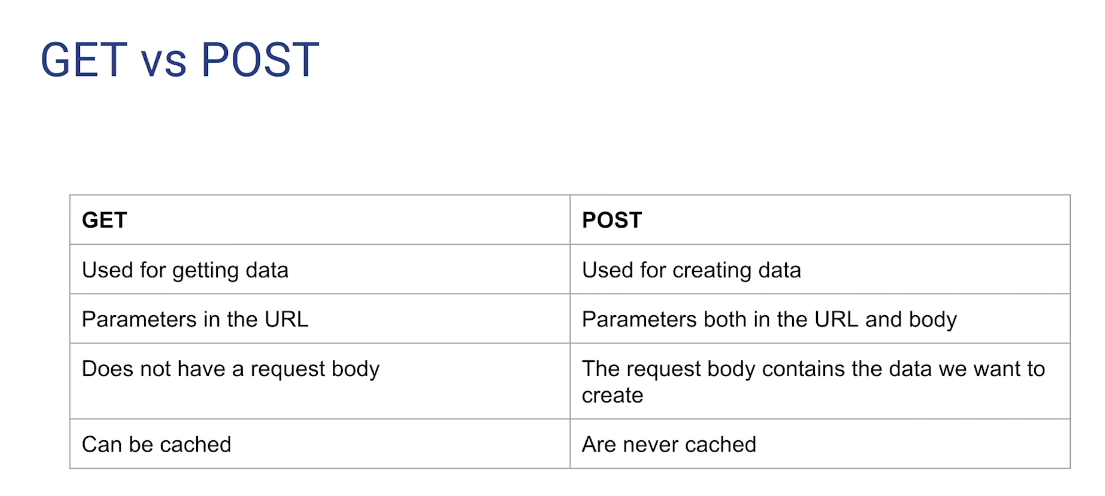


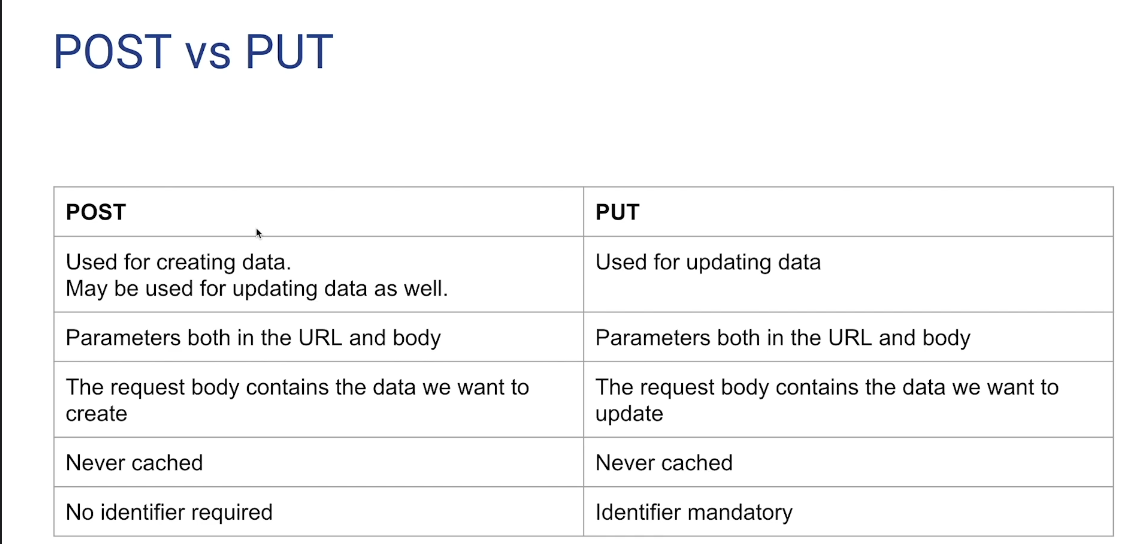


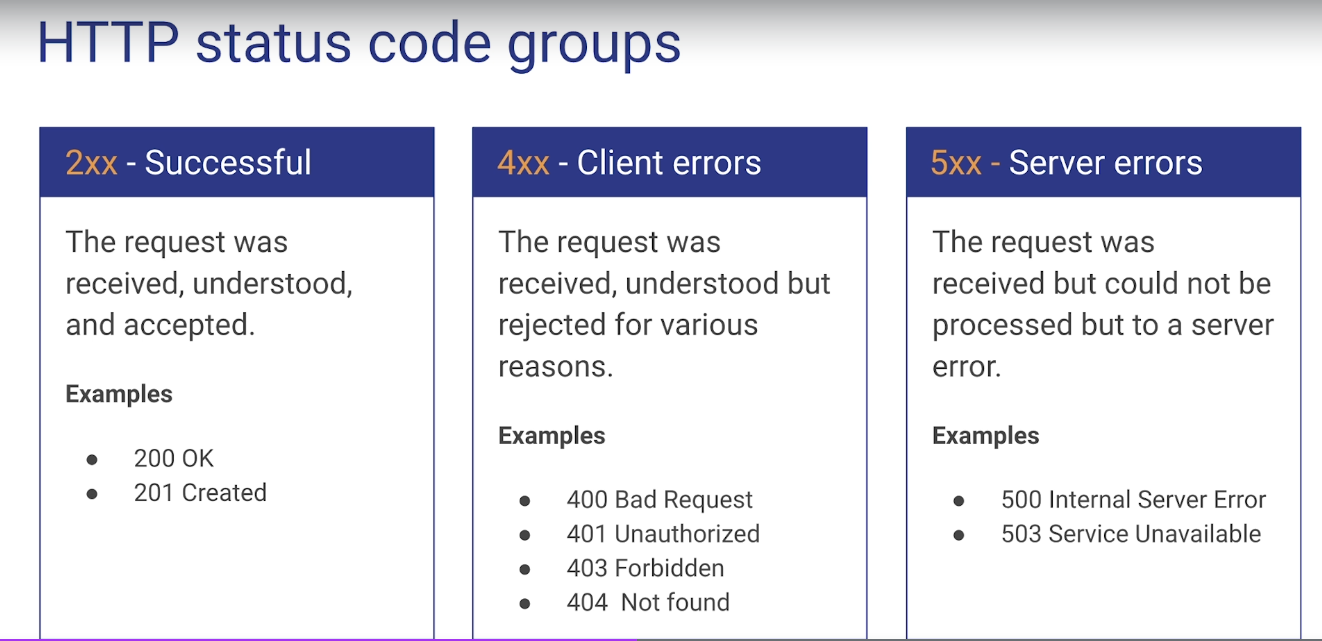
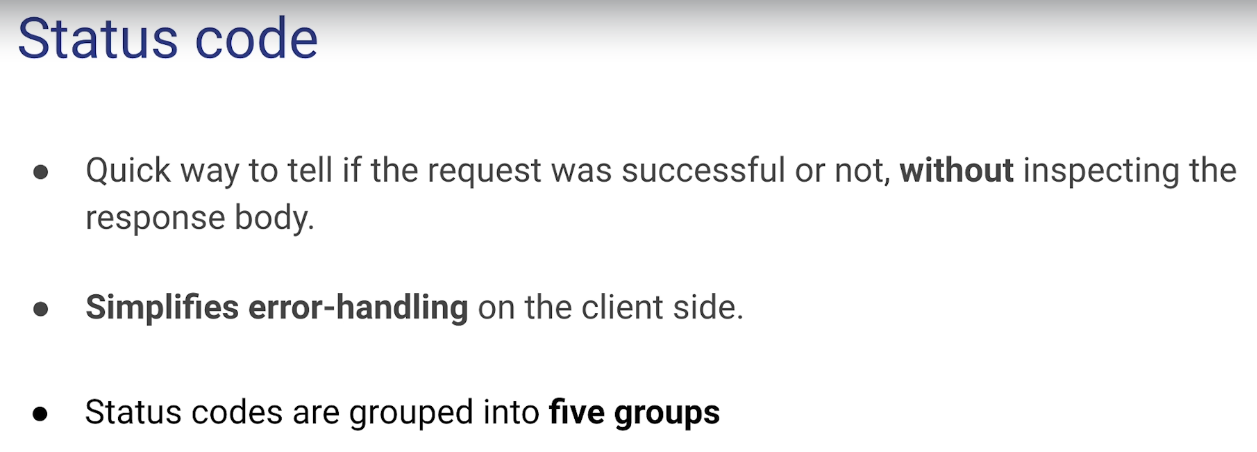
 

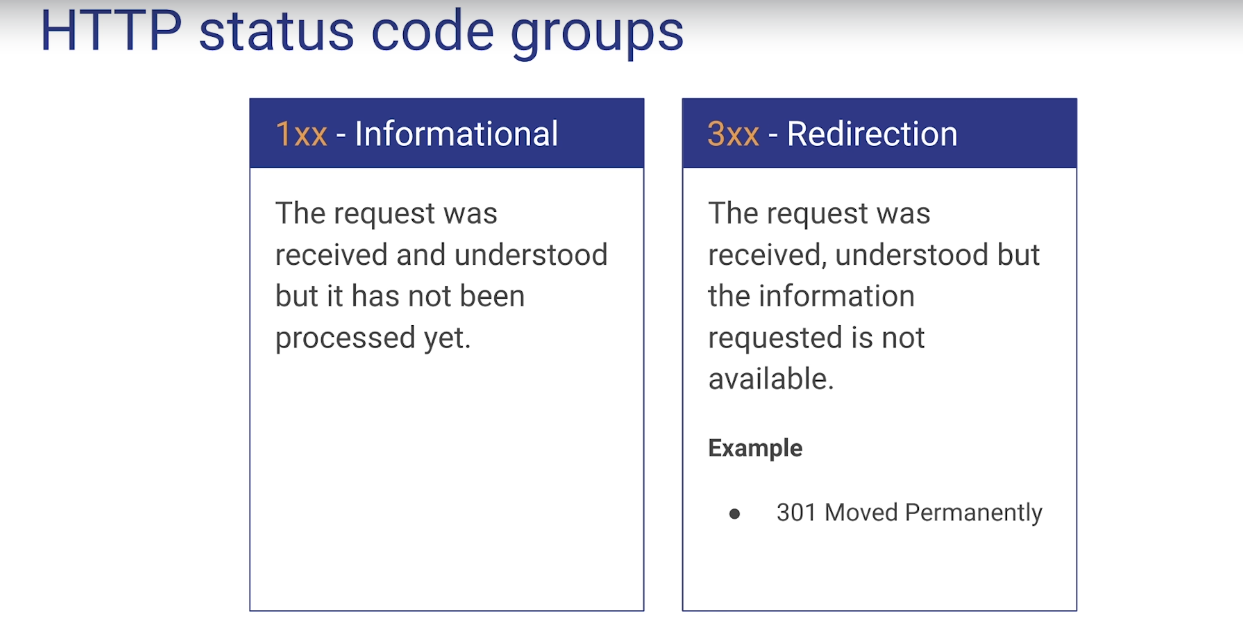














# Sending your first request

## You can make requests to APIs in Postman. An API request allows you to retrieve data from a data source, or to send data. APIs run on web servers, and expose endpoints to support the operations client applications use to provide their functionality.

## Each API request uses an HTTP method. The most common methods are GET, POST, PATCH, PUT, and DELETE.

## GET methods retrieve data from an API.

## POST sends new data to an API.

## PATCH and PUT methods update existing data.

## DELETE removes existing data.

## In Postman you can make API requests and examine the responses without using a terminal or writing any code. When you create a request and click Send, the API response appears inside the Postman user interface.

## 

## **Introduction to REST API and where it is used in Project Architecture**

* End point OR Base URL: Address where API is hosted on the server.
* CRUD operations are performed using different API
* HTTP methods to communicate with Rest API are GET, POST, PUT, DELETE.
* C - POST
* R - GET
* U - PUT
* D - DELETE
* **GET** - retrieve information from the server. No payload / body required. While using GET request it should only extract data and should have no other effect on the data.

How to send input data in GET?

**Using query Parameters.**

* **POST-** create something on the server. Post http method. It is used to send data to the server. For Example customer information, file upload etc using html forms.

How to send input data in POST?

**Using Form Parameters/Body payload.**

* **PUT-** Replaces the current representation of the target resource with the uploaded content or in other words it replaces the existing. It is to update the information on the server.
* **DELETE -** Removes all current representation of the target resource.
* *Notes Download from lecture*

**What is ‘Resources’:**

* **Resources represent API/Collection which can be accessed from the Server**
* Google.com/maps (it is Base\_url/resource)
* google.com/search
* google.com/images

**Path Parameters:**

***Path parameters*** are variable parts of a URL path. They are typically used to point to a specific resource within a collection, such as a user identified by ID

* <https://www.google.com/Images/1123343> (it is Base\_url/resource/Path\_parameters)
* <https://www.google.com/docs/1123343>
* <https://amazon.com/orders/112>
* <https://www.google.com/search?q=newyork&oq=newyork&aqs=chrome..69i57j0l7.2501j0j7&sourceid=chrome&ie=UTF-8>

**Query Parameters:**

* Query Parameter is used to sort/filter the resources.
* Query Parameters are identified with?””
* <https://amazon.com/orders?sort_by=2/20/2020> (it is Base\_url/resource?query\_parameters)

**End Point Request URL can be constructed as below**

* **Base URL/resource/(Query/Path)Parameters**

**Sending parameters**

You can send path and query parameters with your requests using the URL field and the Params tab.

* Query parameters are appended to the end of the request URL, following ? and listed in key value pairs, separated by & using the following syntax: ?id=1&type=new
* Path parameters form part of the request URL, and are referenced using placeholders preceded by : as in the following example: /customer/:id

**Sending body data**

You will need to send body data with requests whenever you need to add or update structured data. For example, if you're sending a request to add a new customer to a database, you might include the customer details in JSON. Typically you will use body data with PUT, POST, and PATCH requests.

* The Body tab in Postman allows you to specify the data you need to send with a request. You can send various different types of body data to suit your API.

**Headers/Cookies**:

* Headers represent the meta-data associated with the API request and response. In layman terms, we were sending Additional details to API to process our request.
* Example : Authorization details

### 9. Introduction to POSTMAN and Google Maps API's

* Download and install postman.
* Have the API Contract:.
  + **Base URL**: [https://rahulshettyacademy.com](https://rahulshettyacademy.com/)
  + **Resource**: /maps/api/place/add/json
  + **Query Parameters**: key =qaclick123
  + **Http Method**: POST
  + **Sample Body** :
  + {
  + <Details you want to send/post/add>
  + }
* Create a Collection/folder let say RSA\_Maps
* ‘Add request’ to this folder and name it as AddPlace.
* In the Right side fill details given in Contract:
  + Enter request URL.:Base URL/resource?Query\_Parameters
  + We can add a Query parameter in the Params and it will concatenate to the Base URL automatically.
  + If we are adding information via the API (as per contract) set the request Type as POST.
  + Select raw > JSON under the Body tab. Paste the Sample Body given in the Contract.
  + In Headers tab ‘Content-type’ will automatically set to ‘application/json’
* Click Send to send the request and check the response as per the contract.
  + Use the place\_id in further API to get the information.

### 14. Introduction on Test API's

## What is Rest Assured?

* **What is REST :** Representational state transfer (REST) is a de-facto standard for a software architecture for interactive applications that typically use multiple Web services. ... In a RESTful Web service, requests made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format.
* **What is REST assured** - REST Assured is a Java DSL (domain-specific language) for simplifying testing of REST based services built on top of HTTP Builder. It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests.
  + 1. Download Java
    2. Setup Environment variable Path
    3. Download install Eclipse > Create basics project.
    4. Download RestAssured Jars from <https://github.com/rest-assured/rest-assured/wiki/Downloads>
    5. Download all zip files.
    6. Add external jars with all the 6or7 zip folders.
* Works on three principles/methods:
  + given() : all input details (Base URI)
  + .when() : Submit the API using (Params > Headers > Body > Resource AND HTTP request Get/Post/Put…)
  + .then() : you will validate the response (assertthat().statusCode(200))
  + .log().all() method is used with given and then methods to get the response logs in the console otherwise not.
* Make sure you import 2 packages:
  + import io.restassured.RestAssured;
  + import static io.restassured.RestAssured.\*;

Example code:

**RestAssured.baseURI = "https://rahulshettyacademy.com";**

**given()**

**.log().all()**

**.queryParam("key", "qaclick123")**

**.header("Content-Type","application/json")**

**.body(AddPlacePayload.AddPlacePayLoadText())**

**.when()**

**.post("maps/api/place/add/json")**

**.then()**

**.log().all()**

**.assertThat()**

**.statusCode(200);**

## 

## Science

* import static org.hamcrest.Matchers.\*; this package is used to get auto suggestion on typing **equalTo** while validating the response body items for example:
  + .then()
  + .log().all()
  + .assertThat()
  + .statusCode(200);
  + **.body("scope", equalTo("APP"))**
  + **.header("server", "Apache/2.4.18 (Ubuntu)");**
* **How to validate response body level or Headers. Let's say, we need to validate an attribute of the response body:**
  + .then()
  + .log().all()
  + .assertThat()
  + .statusCode(200)
  + **.body("scope", equalTo("APP"))**
  + **.header("server", "Apache/2.4.18 (Ubuntu)")**
* **How to retrieve/extract the response body into 1 string so that you can re-use any attribute from the response body:**
  + **Syntax:**
  + **String response =**
  + given()
  + .log().all() // to Log the request in the console
  + .queryParam("key", "qaclick123")
  + .header("Content-Type","application/json")
  + .body(AddPlacePayload.AddPlacePayLoadText())
  + .when()
  + .post("maps/api/place/add/json")
  + .then()
  + .log().all() // to Log the response in the console
  + .assertThat()
  + .statusCode(200)
  + .body("scope", equalTo("APP"))
  + .header("server", "Apache/2.4.18 (Ubuntu)")**.extract().response().asString();**
  + **System.out.println(response); // this will print the response extracted into a string named as response**
* **JsonPath class object takes string as input and converts to Json and also helps in Parsing Json. Parsed values are stored in the object and methods of that object can be used to find values from the response/converted Json.**
  + **JsonPath js = new JsonPath(response);**
  + **String placeId = js.getString("place\_id");**
  + **System.out.println("I got the place\_id as = " + placeId);**
* **Update Place and validate response json attribute names ‘msg’ and its value. I have also re-used the place id coming from above code as a string:**
  + **given()**
  + .log().all()
  + .queryParam("key", "qaclick123")
  + .header("Content-Type","application/json")
  + .body("{\r\n" +
  + "\"place\_id\":\""**+placeId+**"\",\r\n" +
  + "\"address\":\"New address with Summer walk, USA\",\r\n" +
  + "\"key\":\"qaclick123\"\r\n" +
  + "}\r\n" +
  + "")
  + **.when()**
  + .put("/maps/api/place/update/json")
  + **.then()**
  + .log().all()
  + .assertThat() //below mentioning the required validations
  + **.statusCode(200)**
  + **.body("msg", equalTo("Address successfully updated"));**
* **Get the Place and validate response json attribute names ‘address’ and its value. I have also re-used the place id coming from above code as a string:**
  + //Get Place
  + String getPlaceresponse =
  + given()
  + .log().all()
  + **.queryParam("key", "qaclick123")**
  + **.queryParam("place\_id", placeId)**
  + .when()
  + .get("/maps/api/place/get/json")
  + .then()
  + .log().all()
  + .assertThat()
  + .statusCode(200)
  + .extract().response().asString();
* **Using the JsonPath class object to parse the string and getting the attribute named address value.**
* **JsonPath** js1 = new JsonPath(**getPlaceresponse**);
* String actualAddress = js1.getString("**address**");
* System.out.println("Actual address is now = " + actualAddress);
* **//Now use testNG jars, assertion to compare the newAddress and actualAddress**
* **Assert.assertEquals(actualAddress, newAddress);**

## Folder 6. Diving in Depth-Automating REST API's

* **1. Understanding Structure of Complex Nested Json and its array notations.mp4**
  + Useful website to see nested JSON we can use Jsoneditoronline.org
  + Use file:///E:/24%20July%20Udemy%20courses/Rest%20API%20Testing%20(Automation)%20from%20Scratch%20-RestAssured%20Java/6.%20Diving%20in%20Depth-Automating%20REST%20API's/2.%20Json%20used%20in%20this%20Section%20with%20Queries%20to%20solve.html
* **[ ] brackets in json are reflecting an array.**
* **Lets mock/dummy a response.**

### 1. Print No of courses returned by API

JsonPath js1 = new JsonPath(DummyResponse.DummyResponseText());

int count = js1.getInt("courses.size()"); // getting number of courses in the response

System.out.println(count);

### 2.Print Purchase Amount

* + // Print purchase amount.
  + int p = js1.getInt("dashboard.purchaseAmount");
  + System.out.println(p);

### 3. Print Title of the first course

* + // Print 1st course titles and price.
  + String CTitle1 = js1.get("courses[0].title"); //getting course array's 0th index > title
  + System.out.println("courses[0].title is " + CTitle1);
  + String CTitle1price = js1.getString("courses[0].price"); //getting course array's 0th index > price
  + System.out.println("courses[0].price is " + CTitle1price);

### 4. Print All course titles and their respective Prices

* + //Print all course tiles and its prices
  + for (int i = 0; i < count; i++) {
  + String CTs = js1.get("courses[" + i + "].title");
  + System.out.print(CTs);
  + System.out.println(" "+js1.get("courses[" + i + "].price").toString()); }

### 5. Print no of copies sold by RPA Course

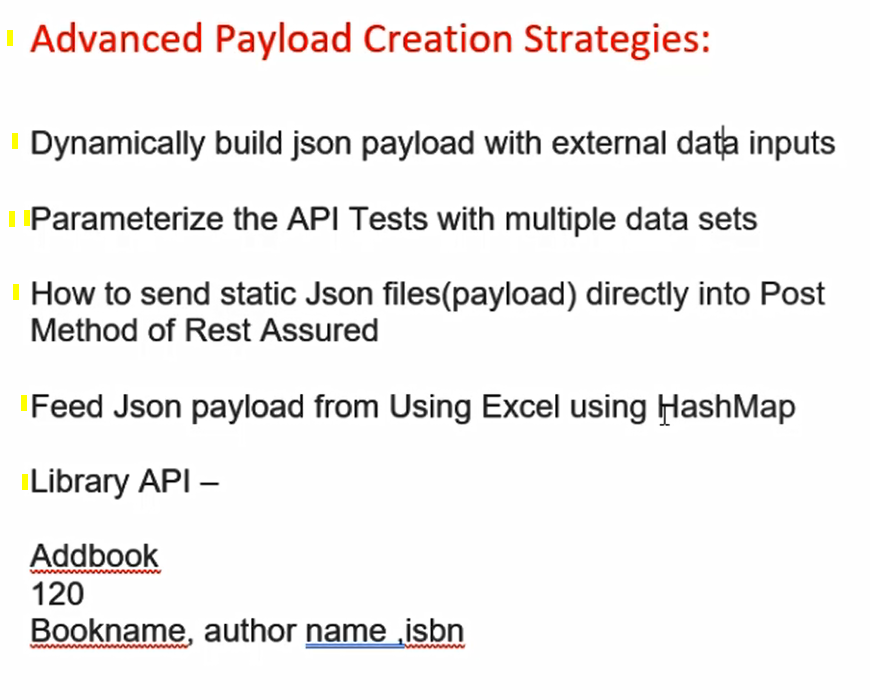
* + // Print RPA course titles and price and no of copies sold by RPA.
  + System.out.println("// Print RPA course titles and price and no of copies sold by RPA.");
  + for (int i = 0; i < count; i++) {
  + String CTs = js1.get("courses[" + i + "].title");
  + if (CTs.equalsIgnoreCase("RPA")) {
  + int CopyCount = js1.get("courses[" + i + "].copies");
  + System.out.println("course title is = " + CTs);
  + System.out.println("course title copies are = " + CopyCount);
  + System.out.println("RPA course price per unit is = "+ js1.get("courses[" + i + "].price").toString());
  + System.out.println("RPA course copies sold are = "+ CopyCount);
  + break; } }

### 6. Verify if Sum of all Course prices matches with Purchase Amount

* + System.out.println("// Print sum of courses and Validate.");
  + int total = 0;
  + for (int i = 0; i < count; i++) {
  + int Cprice = js1.getInt("courses[" + i + "].price");
  + int Ccopies = js1.getInt("courses[" + i + "].copies");
  + int amount = Cprice \* Ccopies;
  + System.out.println(amount);
  + total = total + amount;
  + }
  + System.out.println("Calculated total is = " + total);
  + int givenPurchaseAmount = js1.getInt("dashboard.purchaseAmount");
  + // Assert.assertEquals(total, givenPurchaseAmount); // OR
  + if(total==givenPurchaseAmount) {
  + System.out.println("total matched!!!"); }

| package Pack1;  import io.restassured.RestAssured;  import payloadFiles.DummyResponse;  import static io.restassured.RestAssured.\*;  import io.restassured.path.json.JsonPath;  import static org.hamcrest.Matchers.\*;  import org.testng.Assert;  public class Basic2 {  public static void main(String[] rgs) {  // Get API  /\*  \* String getPlaceresponse =  \*  \* given() .log().all() .queryParam("key", "qaclick123") .queryParam("place\_id",  \* placeId)  \*  \* .when() .get("/maps/api/place/get/json")  \*  \* .then() .log().all()  \* .assertThat().statusCode(200).extract().response().asString();  \*/  // Using JsonPath object to parse the string and getting the attribute named  // address value.  JsonPath js1 = new JsonPath(DummyResponse.DummyResponseText());  int count = js1.getInt("courses.size()"); // getting number of courses in the response  System.out.println(count);  // Print purchase amount.  int p = js1.getInt("dashboard.purchaseAmount");  System.out.println(p);  // Print course titles and price.  String CTitle1 = js1.get("courses[0].title");  System.out.println(CTitle1);  for (int i = 0; i < count; i++) {  String CTs = js1.get("courses[" + i + "].title");  System.out.println(CTs);  System.out.println(js1.get("courses[" + i + "].price").toString());  }  // Print RPA course titles and price and no of copies sold by RPA.  System.out.println("// Print RPA course titles and price and no of copies sold by RPA.");  for (int i = 0; i < count; i++) {  String CTs = js1.get("courses[" + i + "].title");  if (CTs.equalsIgnoreCase("RPA")) {  int CopyCount = js1.get("courses[" + i + "].copies");  System.out.println(CTs);  System.out.println(js1.get("courses[" + i + "].price").toString());  System.out.println(CopyCount);  break;  }  }  System.out.println("// Print sum of courses and Validate.");  int total = 0;  for (int i = 0; i < count; i++) {  int Cprice = js1.getInt("courses[" + i + "].price");  int Ccopies = js1.getInt("courses[" + i + "].copies");  int amount = Cprice \* Ccopies;  System.out.println(amount);  total = total + amount;  }  System.out.println("Calculated total is = " + total);  int givenPurchaseAmount = js1.getInt("dashboard.purchaseAmount");  // Assert.assertEquals(total, givenPurchaseAmount);  // OR    if(total==givenPurchaseAmount) {  System.out.println("total matched!!!");  }  }  } |
| --- |

## Folder 7. Handling Dynamic Json Payloads with Parameterization



### Advanced payload creation strategies:

* **Dynamically build json payload with external data inputs**
  + **Create a payload class and call the method in the test class.**

given()

.log().all()

.header("Content-Type", "application/json")

**.body(AddbookPayload.AddBookPayLoadText()) //payload without params**

//.body(AddbookPayload.AddBookPayLoadText("NK01isbn","NK01aisle"))

//above payload with 2 string params that will update the AddBookPayLoadText() method with the params

.when()

* + **Send payload attribute value from the test class to the payload class method.**

given()

.log().all()

.header("Content-Type", "application/json")

//.body(AddbookPayload.AddBookPayLoadText()) //payload without params

**.body(AddbookPayload.AddBookPayLoadText("NK01isbn","NK01aisle"))**

//above payload with 2 string params that will update the AddBookPayLoadText() method with the params

.when()

* + **Parameterize data - e.g 10 payloads**
    - Use data provider annotation with required array object and object name

@DataProvider(name = "getBookData")

public Object[][] getBookData() {

return new Object[][] { { "nayanisbn-1", "nkaisle-1" }, { "nayanisbn-2", "nkaisle-2" } }; }

* + - Use dataprovider object name in the @test annotation as

@Test(dataProvider = "getBookData")

public void addBook(String isbn, String aisle)

* + - The array elements will be sent to the test method
    - Test method will send the received elements as arguments to the dynamic payload class method.
    - The system will run the test with a given set of elements coming from a dataprovider.
* **Pass local file path as json payload**
  + Create a json file using notepad on your local drive. In that file paste the payload and save.
    - https://www.freeformatter.com/json-escape.html
  + Filename.json extension is a must.
  + Create a readAllBytes() method in the test class so that it can read the path and read inside the file to find the payload.

**Public static String generateStringFromResource(String path) throws IOException {**

**Return new String(Files.readAllBytes(Paths.get(path))): }**

* + Data provider not needed. No parameters needed to send to the payload class. No payload class needed.
  + Call the GenerateStringFromResource() method in the body() under given() method of the test class.

given()

.header(“content-Type”, “application/json”)

**.”body(GenerateStringFromResource(“E:\\nayan\\addbookdetails.json”))**

* + **Note**: Use this technique only if you have static payload otherwise please go for the previously mentioned dynamic technique.

Postman is an API client

To dev test share documentation for APIs

Create a postman id at getpostman.com

Download the windows app and install.

You can add the postman chrome extension as well.

# POSTMAN UI

Start a postman window app.

SOAP - simple object access protocol

RPC - Remote Procedure call

REST - Representational State Transfer

It is a architectural style / design pattern for web API

If any API follow these constraints

<https://scrolltest.com/2018/12/15/api-testing-tutorial/>

### 6. What is Collection | How to create Collection

It is a GROUP OF API requests, that are saved in a logical arrangement.

For a collection we can add:

* Authorization
* Pre-request Script
* Tests
* Common ‘Variable ‘

We can share the collection among workspaces and Run the collections and many more.

I can add folders in the collection.

### 7. How to run COLLECTIONS (Collection Runner)