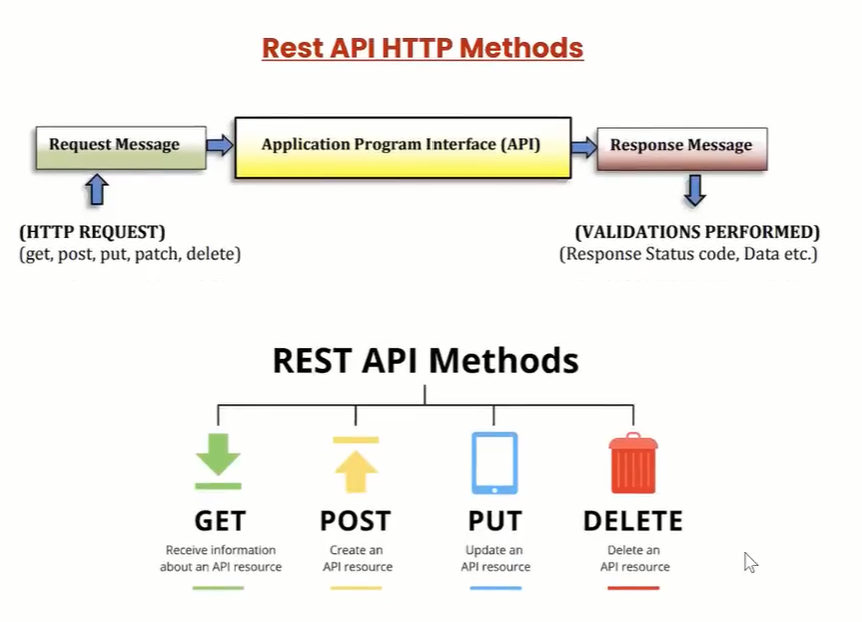
**UNIT – V**

**API Functional Testing & Automation:** REST Architecture-Understanding of REST Architecture and Richardson Maturity Model, API Testing, Usage of Postman, Automated testing Web Services (Amazon, Flip kart, GitHub etc.).

****

**R**epresentational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. **REST API** is a way of accessing web services in a simple and flexible way without having any processing.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It’s used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.

**Working:**A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON. But now JSON is the most popular format being used in Web Services. 



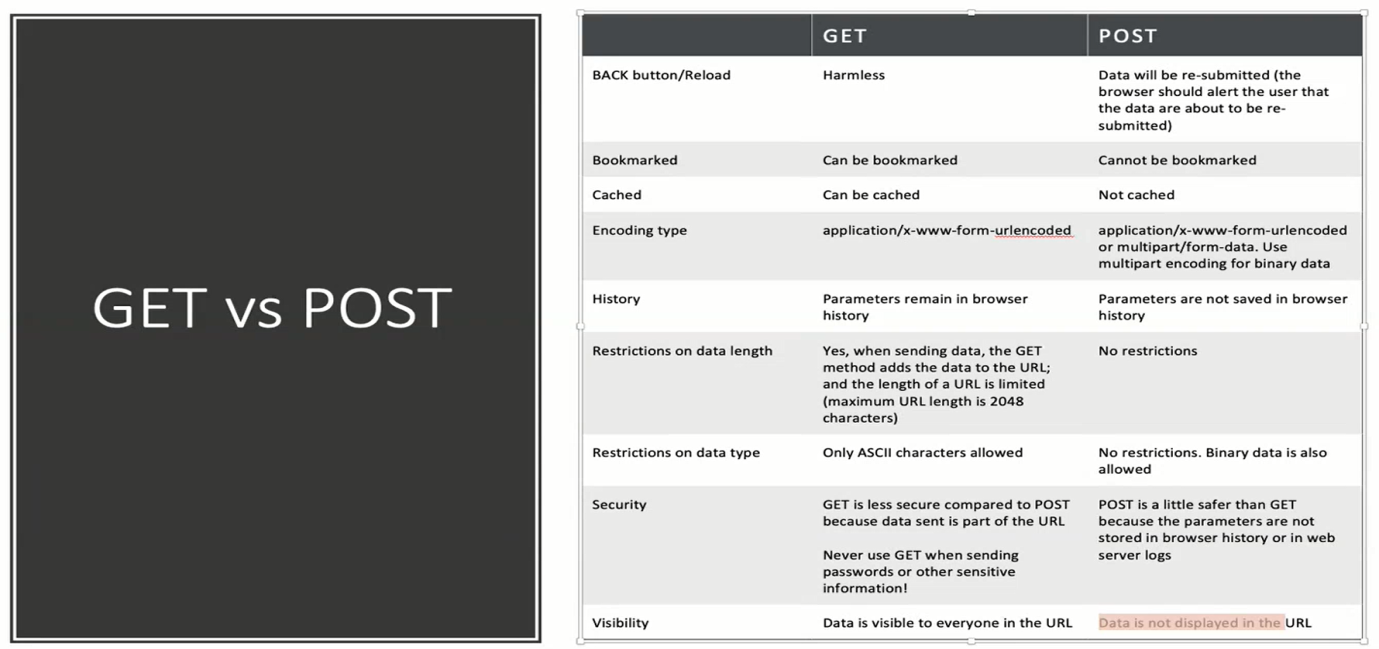
In **HTTP** there are five methods that are commonly used in a REST-based Architecture i.e., POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations respectively. There are other methods which are less frequently used like OPTIONS and HEAD.

* **GET:**The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).
* **POST:** The POST verb is most often utilized to **create** new resources. In particular, it’s used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.

***NOTE:****POST is neither safe nor idempotent.*

* **PUT:**It is used for **updating** the capabilities. However, PUT can also be used to **create** a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. PUT is not safe operation but it’s idempotent.
* **PATCH:**It is used to **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource. This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version. This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch. PATCH is neither safe nor idempotent.
* **DELETE:**It is used to **delete** a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.

**Idempotence:**An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same. Again, this only applies to the result, not the resource itself.

****

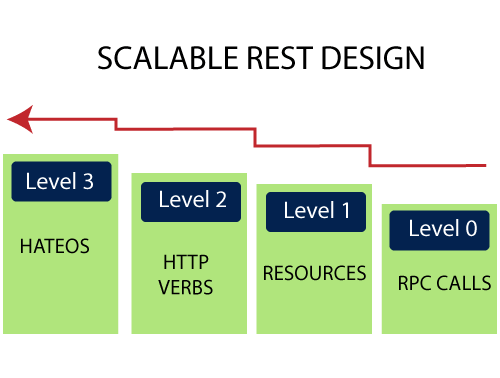
**Richardson Maturity Model**

**Richardson Maturity Model** grades API by their Restful maturity. It is proposed by **Leonard Richardson**. The Richardson maturity model is a way to grade your API according to the constraints of REST. It breaks down the principal element of the REST approach into **four** levels (0 to 3).

There are four levels:

* Level 0: The Swamp of POX
* Level 1: Resources
* Level 2: HTTP Verbs
* Level 3: Hypermedia Control

For example, a level higher is more Restful compared to one that is at a lower level. Only when an API reaches level 4, we consider it as a Restful API.



Level 0: The Swamp of POX

Level 0 is also known as POX (Plain Old XML). At level 0, HTTP is used only as a transport protocol that is used as a remote interaction. It does not take the advantages of HTTP like different HTTP methods, and HTTP cache. To get and post the data, we send a request to the same URI, and only the POST method may be used. These APIs use only one URI and one HTTP method called POST. In short, it exposes SOAP web services in the REST style.

**For example,** there can be many customers for a particular company. For all the different customers, we have only one endpoint. To do any of the operations like get, delete, update, we use the same POST method.

To get the data: http://localhost:8080/customer

To post the data: http://localhost:8080/customer

In the above two URIs, we have used the same URI and method to get and post the customers.

Level 1: Resources

When an API can distinguish between different resources, it might be at level 1. It uses multiple URIs. Where every URI is the entry point to a specific resource. It exposes resources with proper URI. Level 1 tackles complexity by breaking down huge service endpoints into multiple **different endpoints**. It also uses only one HTTP method POST for retrieving and creating data.

For example, if we want a list of specific products, we go through the URI http://localhost:8080/products. If we want a specific product, we go through the URI http://localhost:8080/products/mobile.

Remember the following points while building a URI:

* Use domain and subdomain to logically group or partition resources.
* Use **/** to indicate a hierarchical relationship.
* Use **,** and **;** to indicate non-hierarchical relationships.
* Use **-** and **\_** to improve the readability.
* Use **&** to separate parameters.
* Avoid including **file extensions**.

Level 2: HTTP Verbs

Level 2 indicates that an API must use the protocol properties to deal with scalability and failures. At level 2, correct **HTTP verbs** are used with each request. It suggests that in order to be truly RESTful, HTTP verbs must be used in API. For each of those requests, the correct HTTP response code is provided.

We don't use a single POST method for all requests. We use the **GET** method when we request a resource, and use the **DELETE** method when we want to delete a resource. Also, use the response codes of the application protocol.

For example, to get the customers, we send a request with the URI http://localhost:8080/customers, and the server sends proper response **200 OK**.

The following table shows the HTTP verbs and their usage:

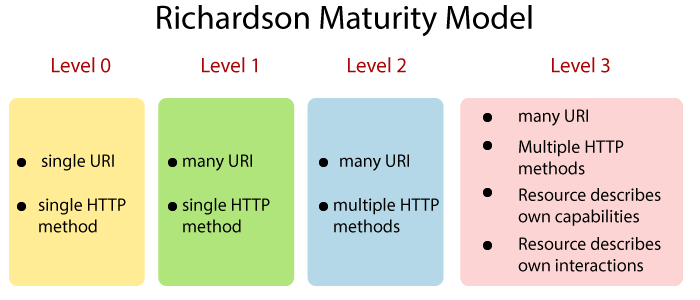
| **Verbs** | **Safety & Idempotency** | **Usage** |
| --- | --- | --- |
| GET | Y/Y | It retrieves the information. |
| POST | N/N | It is used to perform a variety of actions on the server, such as create a new resource, and update an existing resource, or making a mixture of changes to one or more resources. |
| DELETE | N/Y | It is used to delete a resource. |
| PUT | N/Y | It is used to update or replace an existing resource or to create a new resource with a URI specified by the client. |
| HEAD | Y/Y | It is used to retrieve the same headers as that of GET response but without any body in the response. |
| OPTIONS | Y/Y | It is used to find the list of HTTP methods supported by any resource or to ping the server. |
| TRACE | Y/Y | It is used for debugging, which echo's back headers that it has received. |

Level 3: Hypermedia Controls

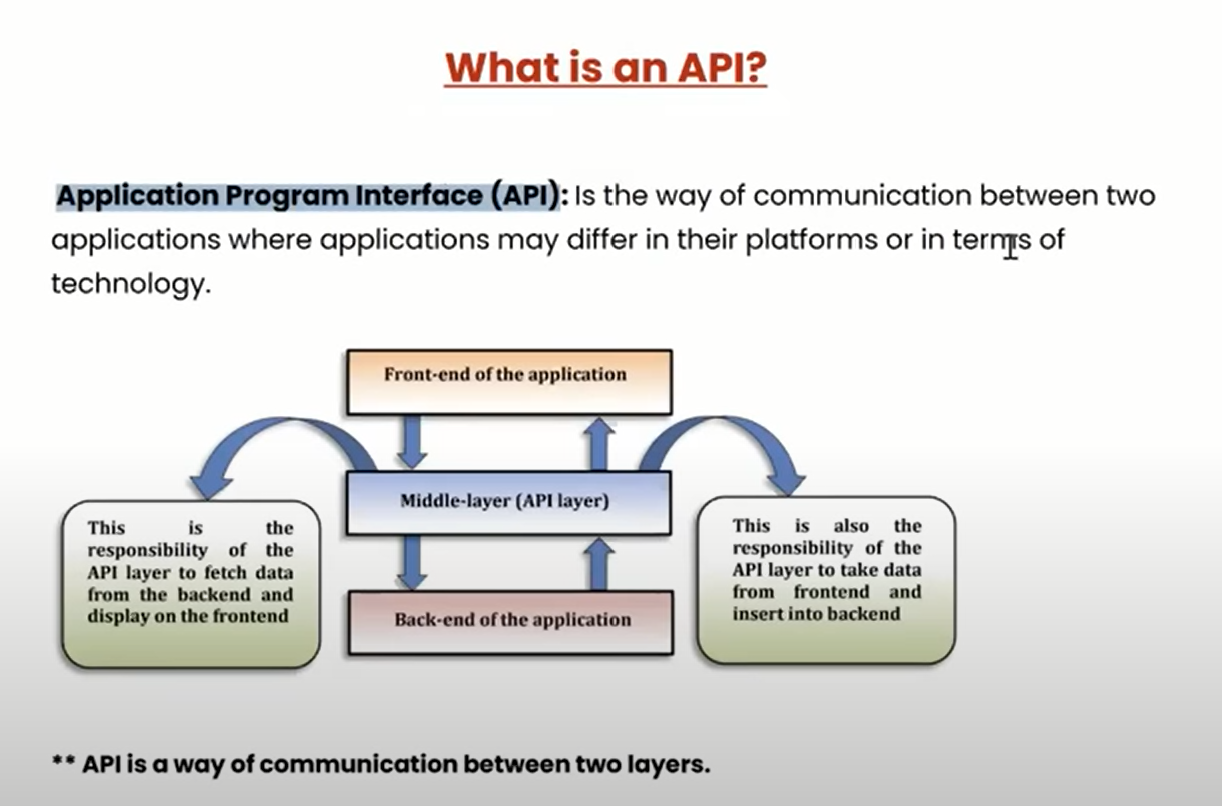
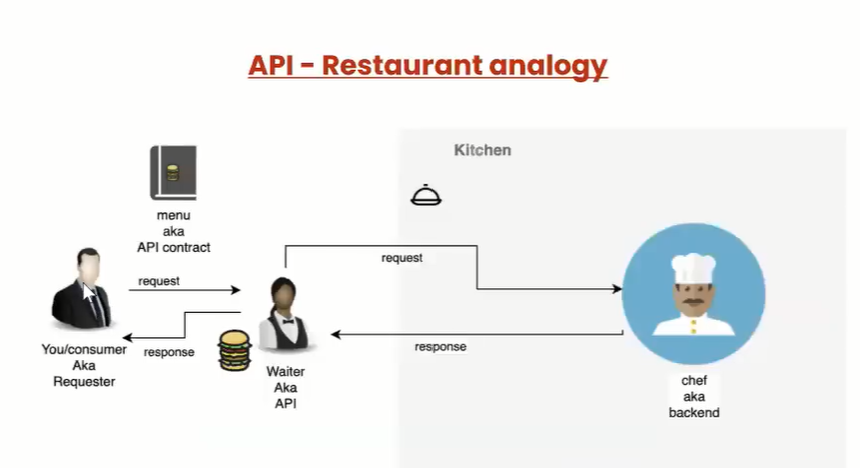
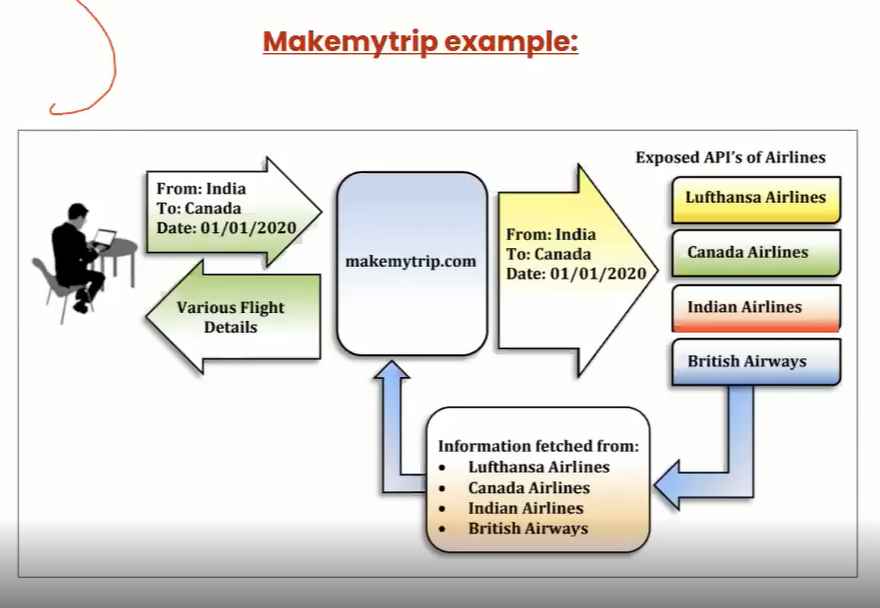
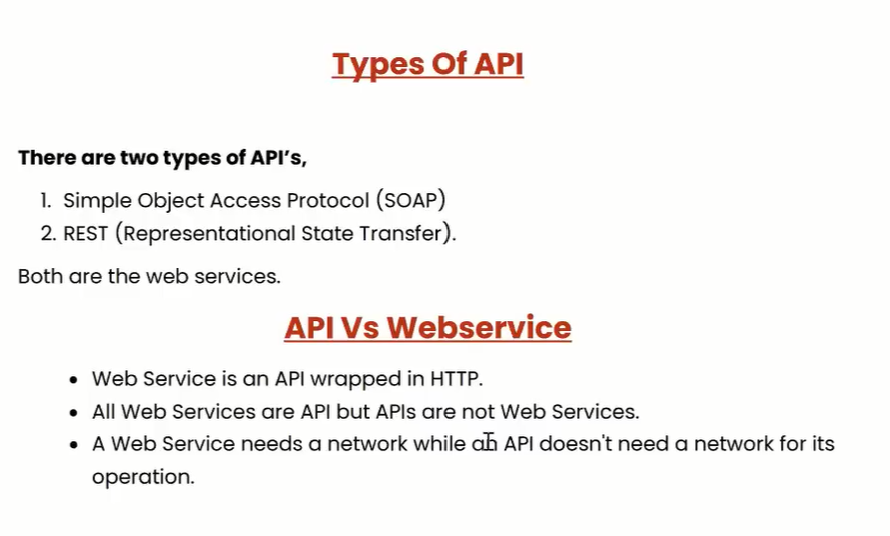
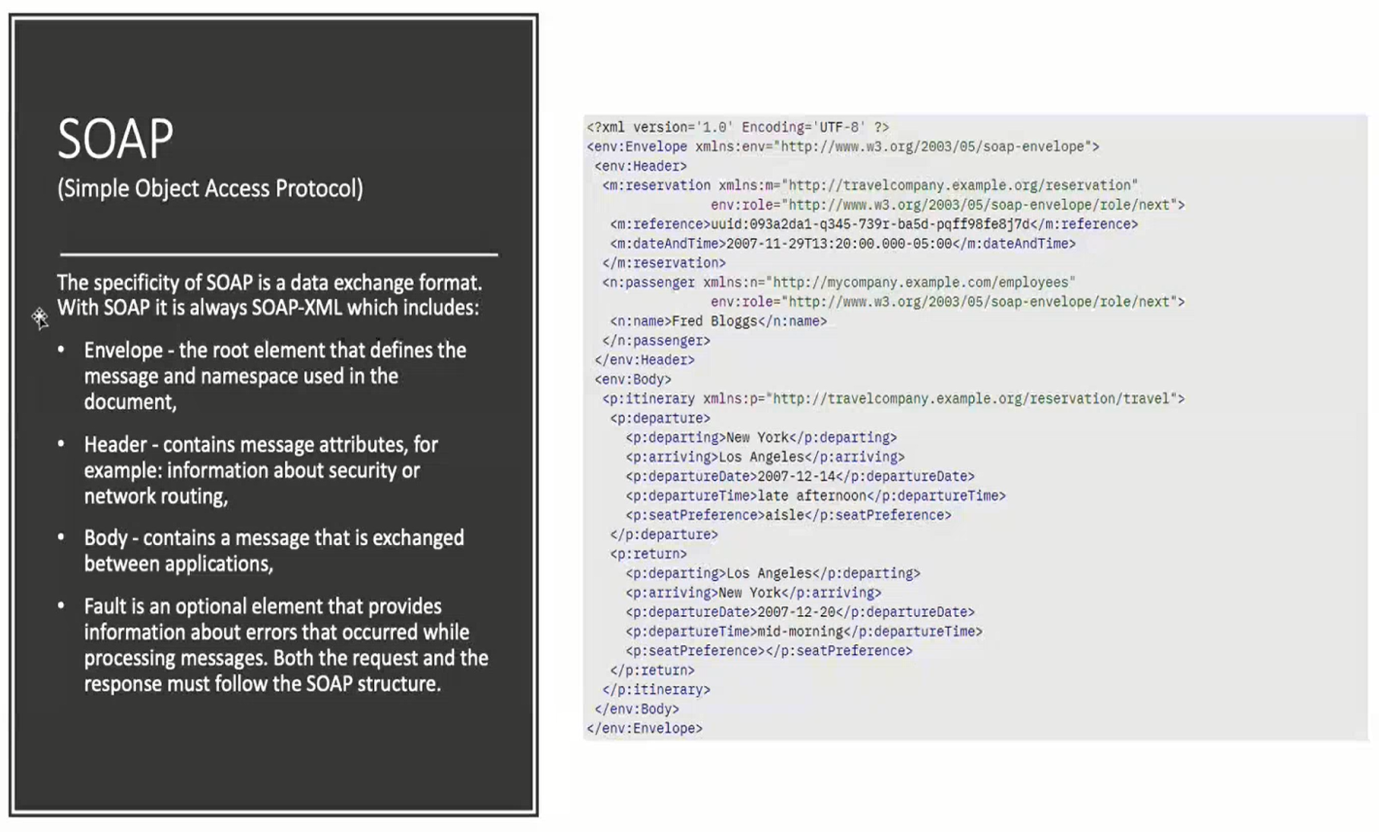
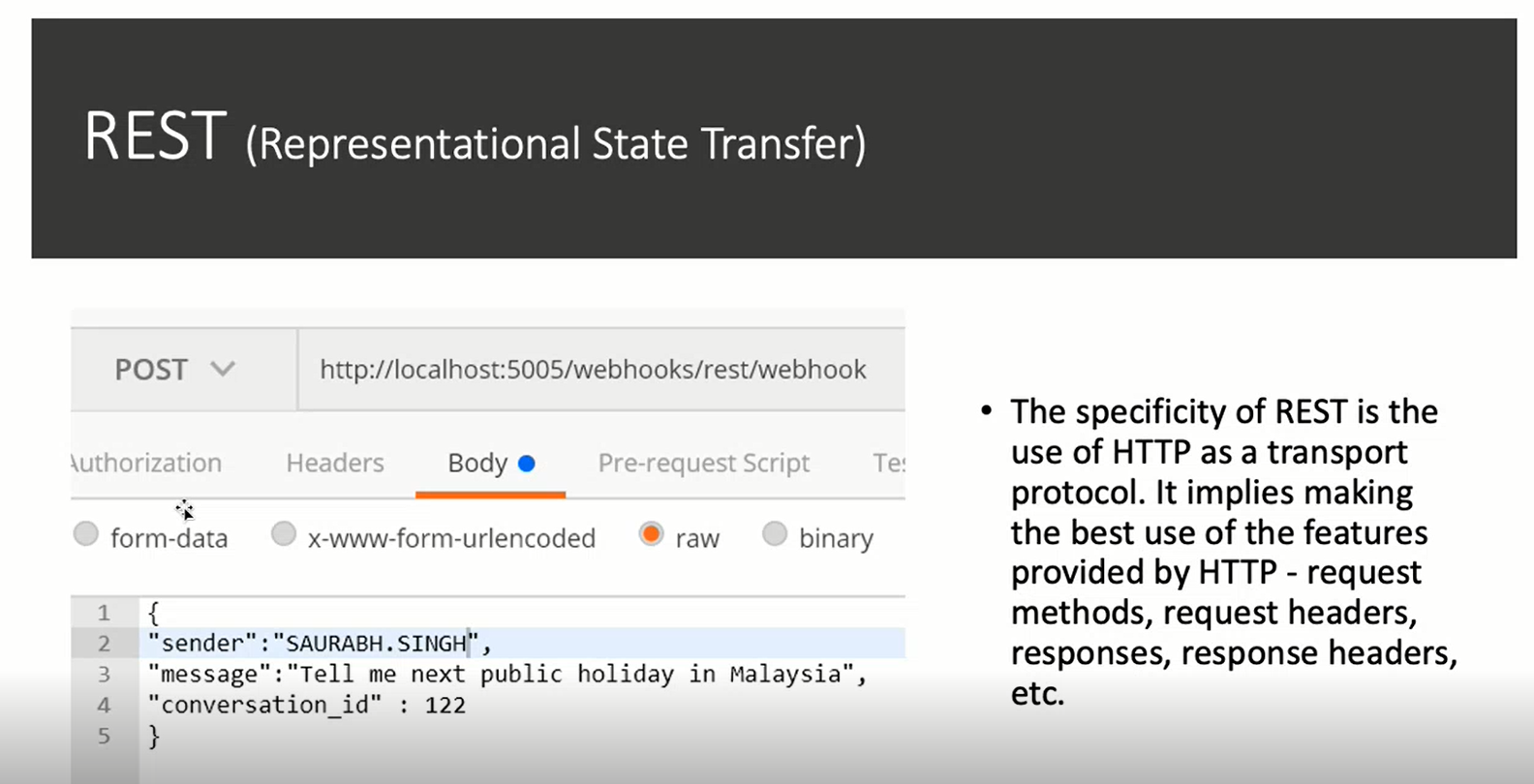
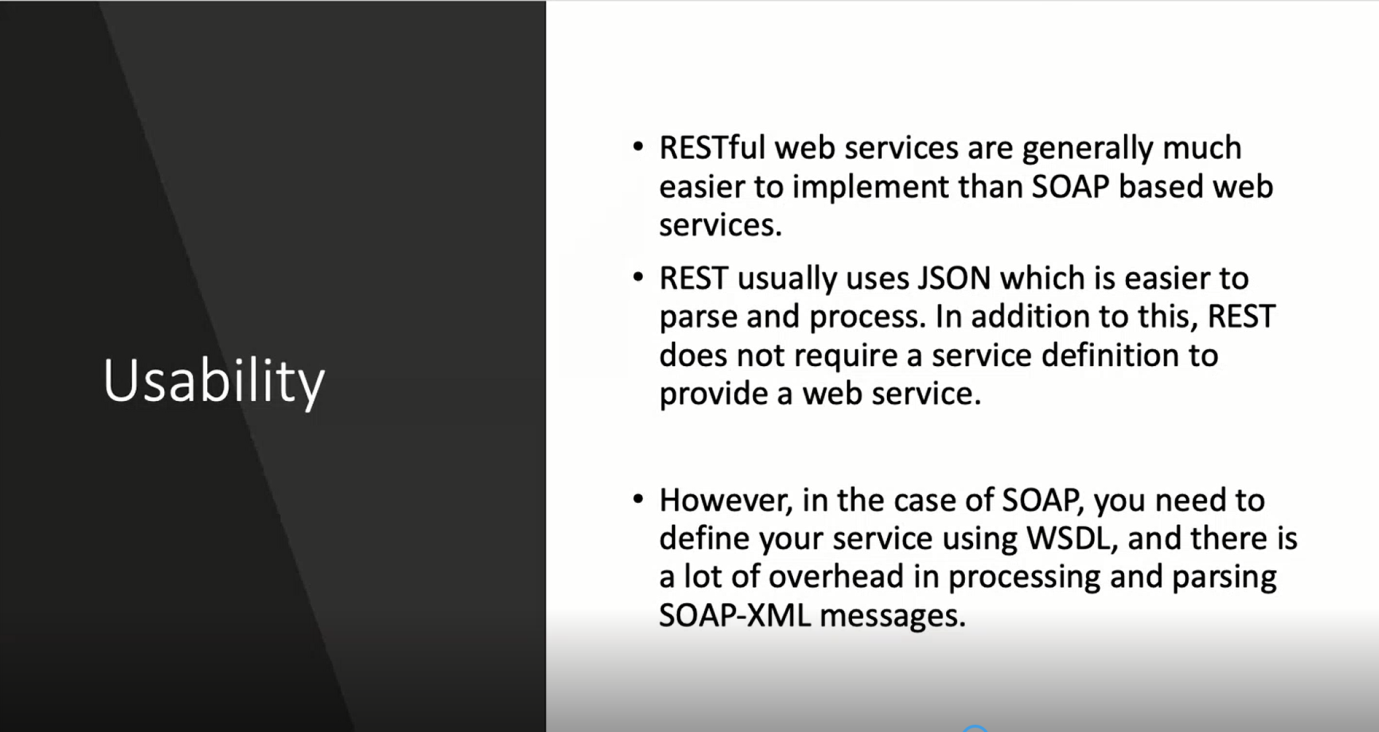
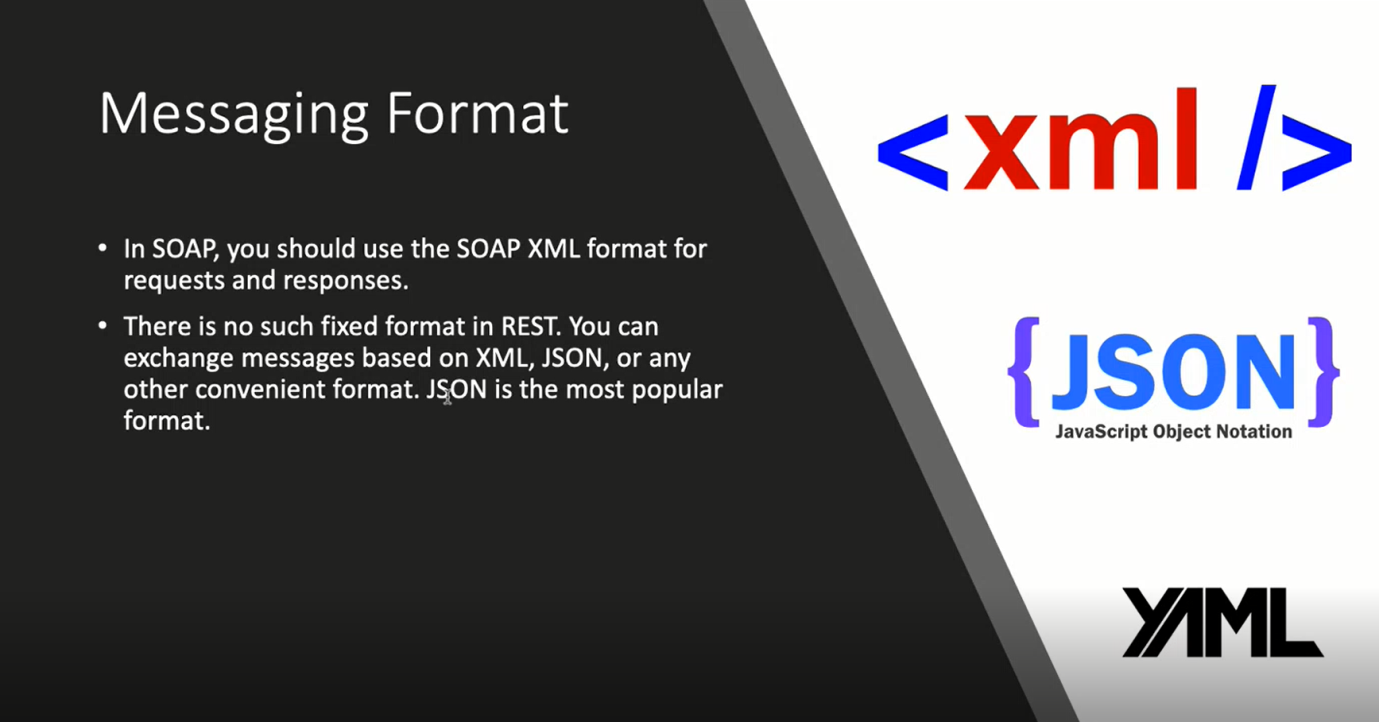
Level 3 is the highest level. It is the combination of level 2 and HATEOAS. It also provides support for HATEOAS. It is helpful in self-documentation.

For example, if we send a GET request for customers, we will get a response for customers in JSON format with self-documenting Hypermedia.

The following figure shows the overview of the model:



**What is API Testing?**

**** ****       

## What is Postman?

**Postman** is a scalable API testing tool that quickly integrates into CI/CD pipeline. API stands for Application Programming Interface which allows software applications to communicate with each other via API calls.

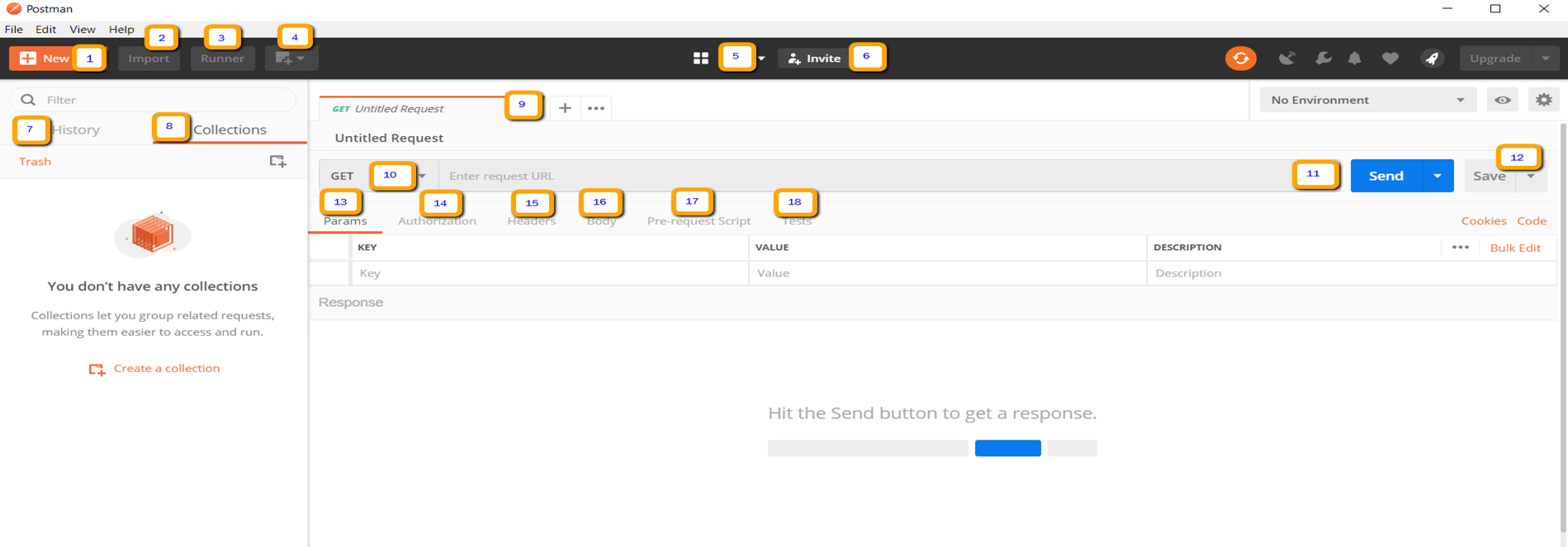
**Why Use Postman?**

With over 4 million users nowadays, Postman Software has become a tool of choice for the following reasons:

1. Accessibility – To use Postman tool, one would just need to log-in to their own accounts making it easy to access files anytime, anywhere as long as a Postman application is installed on the computer.
2. Use of Collections – Postman lets users create collections for their Postman API calls. Each collection can create subfolders and multiple requests. This helps in organizing your test suites.
3. Collaboration – Collections and environments can be imported or exported making it easy to share files. A direct link can also be used to share collections.
4. Creating Environments – Having multiple environments aids in less repetition of tests as one can use the same collection but for a different environment. This is where parameterization will take place which we will discuss in further lessons.
5. Creation of Tests – Test checkpoints such as verifying for successful HTTP response status can be added to each Postman API calls which help ensure [test coverage](https://www.guru99.com/test-coverage-in-software-testing.html).
6. Automation Testing – Through the use of the Collection Runner or Newman, tests can be run in multiple iterations saving time for repetitive tests.
7. Debugging – Postman console helps to check what data has been retrieved making it easy to debug tests.
8. [Continuous Integration](https://www.guru99.com/continuous-integration.html) – With its ability to support continuous integration, development practices are maintained.

**How to use Postman to execute APIs**

Below is the Postman Workspace. Let’s explore the step by step process on **How to use Postman** and different features of the Postman tool!



1. New – This is where you will create a new request, collection or environment.
2. Import – This is used to import a collection or environment. There are options such as import from file, folder, link or paste raw text.
3. Runner – Automation tests can be executed through the Collection Runner. This will be discussed further in the next lesson.
4. Open New – Open a new tab, Postman Window or Runner Window by clicking this button.
5. My Workspace – You can create a new workspace individually or as a team.
6. Invite – Collaborate on a workspace by inviting team members.
7. History – Past requests that you have sent will be displayed in History. This makes it easy to track actions that you have done.
8. Collections – Organize your test suite by creating collections. Each collection may have subfolders and multiple requests. A request or folder can also be duplicated as well.
9. Request tab – This displays the title of the request you are working on. By default, “Untitled Request” would be displayed for requests without titles.
10. HTTP Request – Clicking this would display a dropdown list of different requests such as GET, POST, COPY, DELETE, etc. In Postman API testing, the most commonly used requests are GET and POST.
11. Request URL – Also known as an endpoint, this is where you will identify the link to where the API will communicate with.
12. Save – If there are changes to a request, clicking save is a must so that new changes will not be lost or overwritten.
13. Params – This is where you will write parameters needed for a request such as key values.
14. Authorization – In order to access APIs, proper authorization is needed. It may be in the form of a username and password, bearer token, etc.
15. Headers – You can set headers such as content type JSON depending on the needs of the organization.
16. Body – This is where one can customize details in a request commonly used in POST request.
17. Pre-request Script – These are scripts that will be executed before the request. Usually, pre-request scripts for the setting environment are used to ensure that tests will be run in the correct environment.
18. Tests – These are scripts executed during the request. It is important to have tests as it sets up checkpoints to verify if response status is ok, retrieved data is as expected and other tests.

**Working with GET Requests**

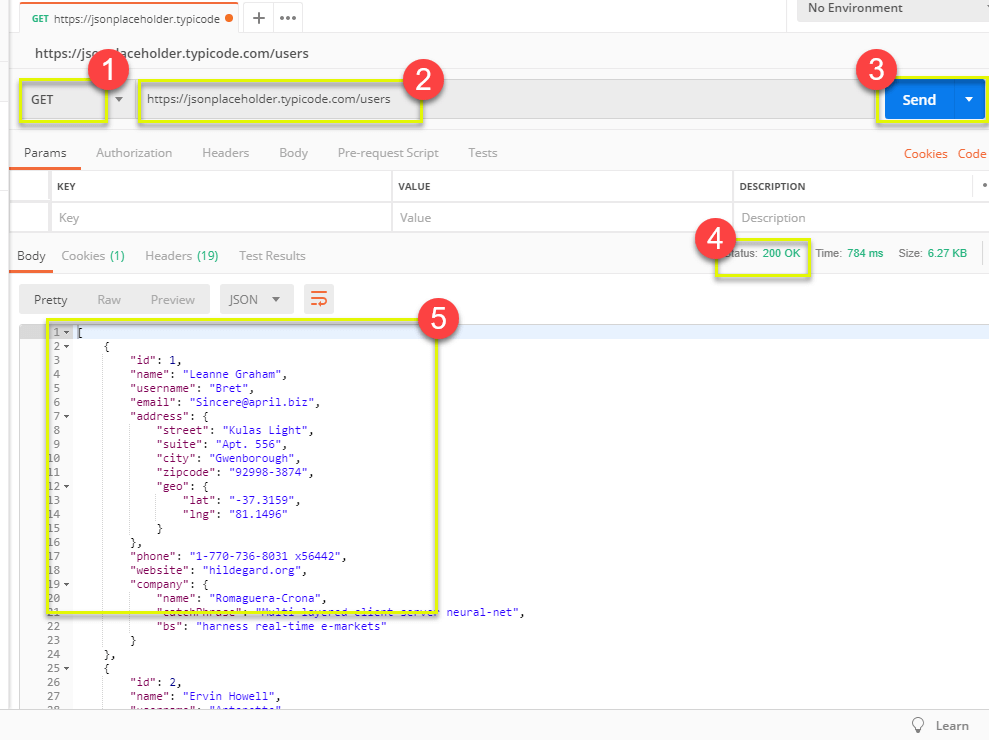
Get requests are used to retrieve information from the given URL. There will be no changes done to the endpoint.

We will use the following URL for all examples in this Postman tutorial

<https://jsonplaceholder.typicode.com/users>

In the workspace

1. Set your HTTP request to GET.
2. In the request URL field, input link
3. Click Send
4. You will see 200 OK Message
5. There should be 10 user results in the body which indicates that your test has run successfully.

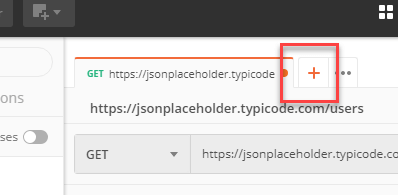


**\*Note:** There may be cases that Get Postman request may be unsuccessful. It can be due to an invalid request URL or authentication is needed.

**Working with POST Requests**

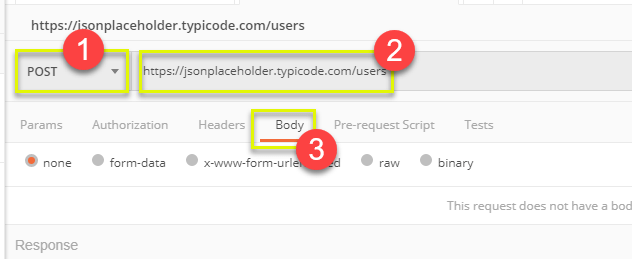
Post requests are different from Get request as there is data manipulation with the user adding data to the endpoint. Using the same data from the previous tutorial in Get request, let’s now add our own user.

**Step 1)** Click a new tab to create a new request.



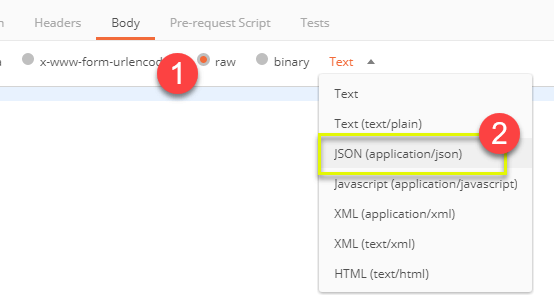
**Step 2)** In the new tab

1. Set your HTTP request to POST.
2. Input the same link in request url: <https://jsonplaceholder.typicode.com/users>
3. switch to the Body tab



**Step 3)** In Body,

1. Click raw
2. Select JSON



**Step 4)** Copy and paste just one user result from the previous get request like below. Ensure that the code has been copied correctly with paired curly braces and brackets. Change id to 11 and name to any desired name. You can also change other details like the address.

[

{

"id": 11,

"name": "Krishna Rungta",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

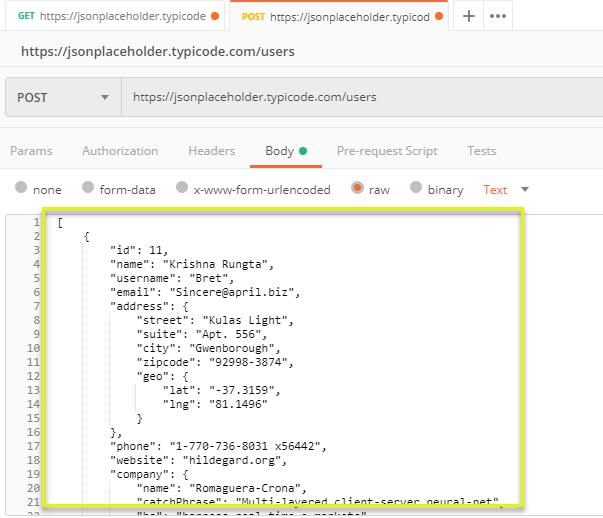
"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

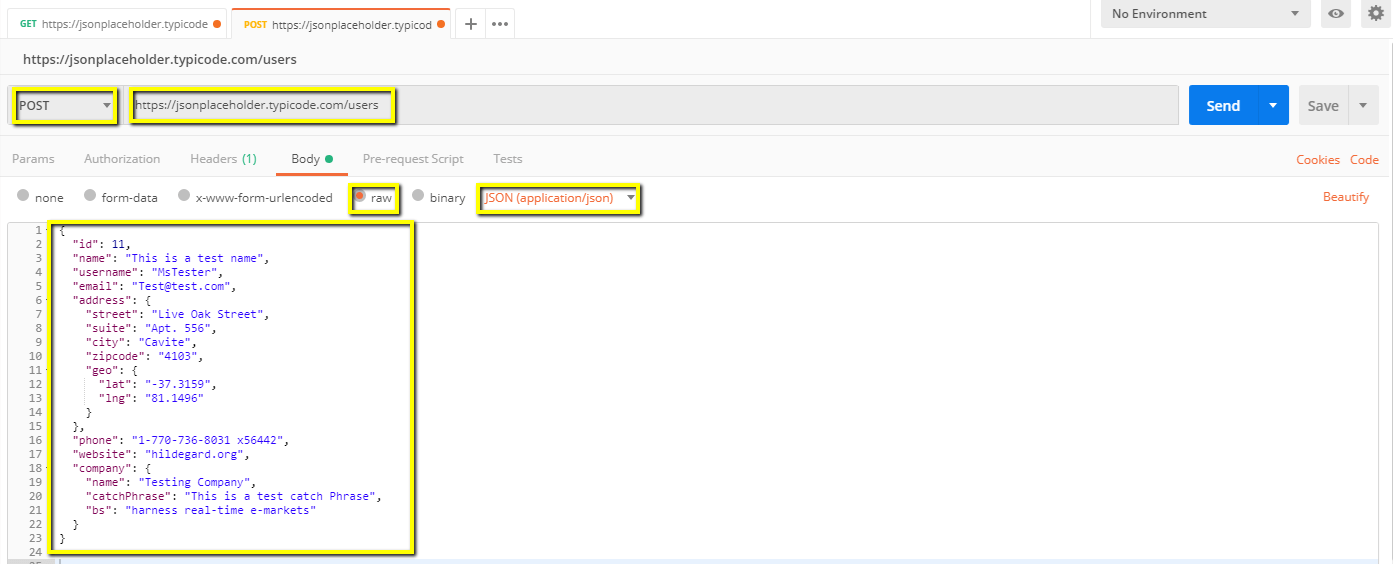
}

}

]

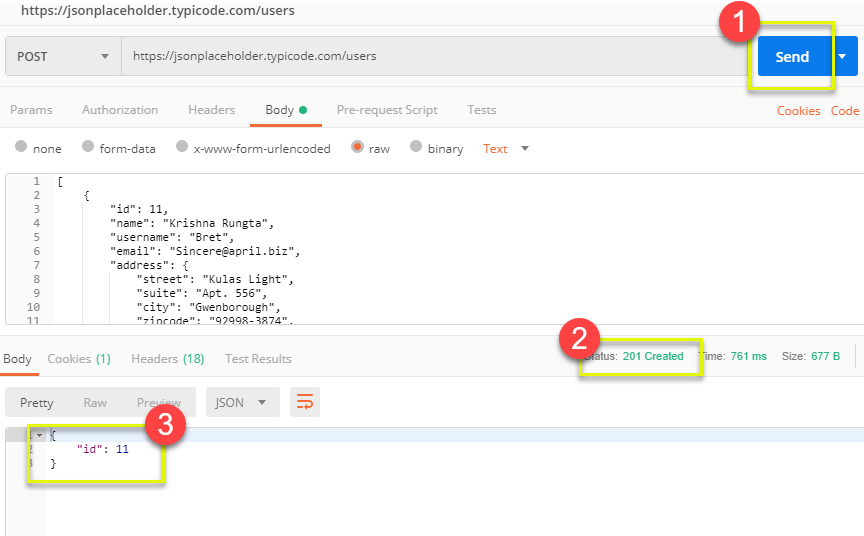


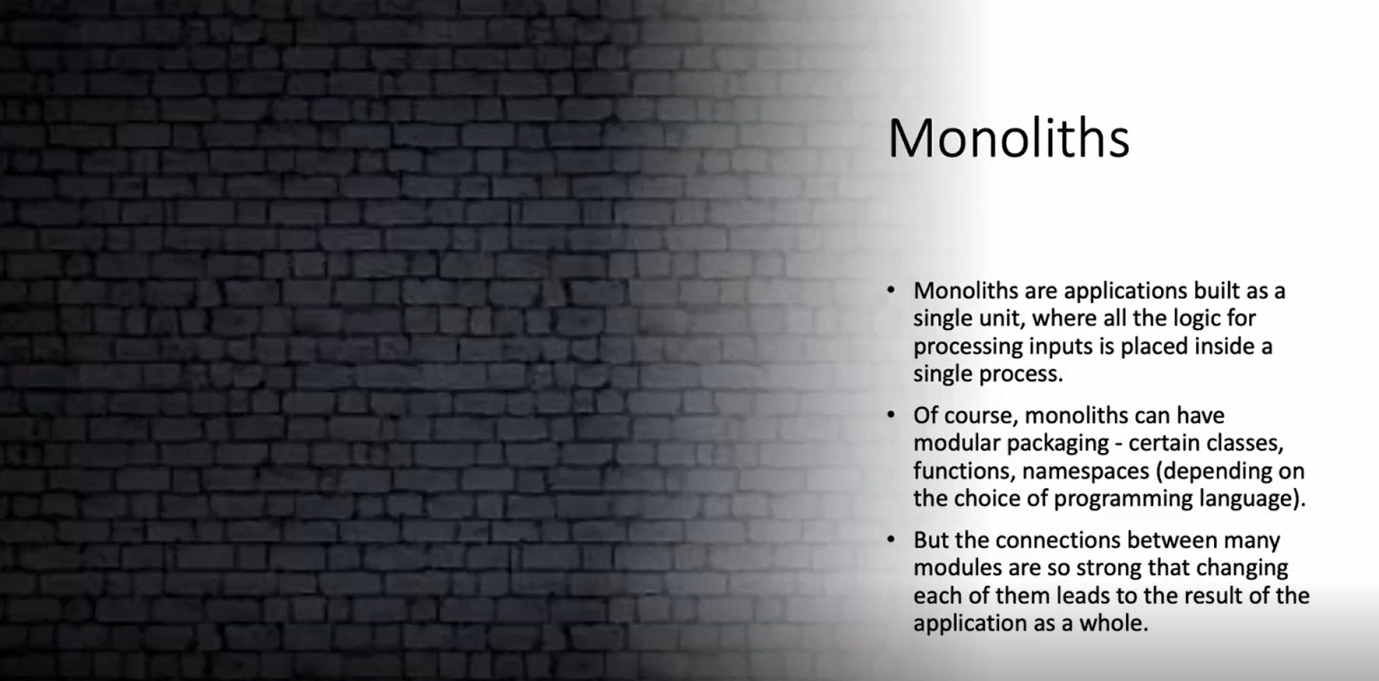
**\*Note:**Online Post request should have the correct format to ensure that requested data will be created. It is a good practice to use Get first to check the JSON format of the request. You can use tools like <https://jsonformatter.curiousconcept.com/>



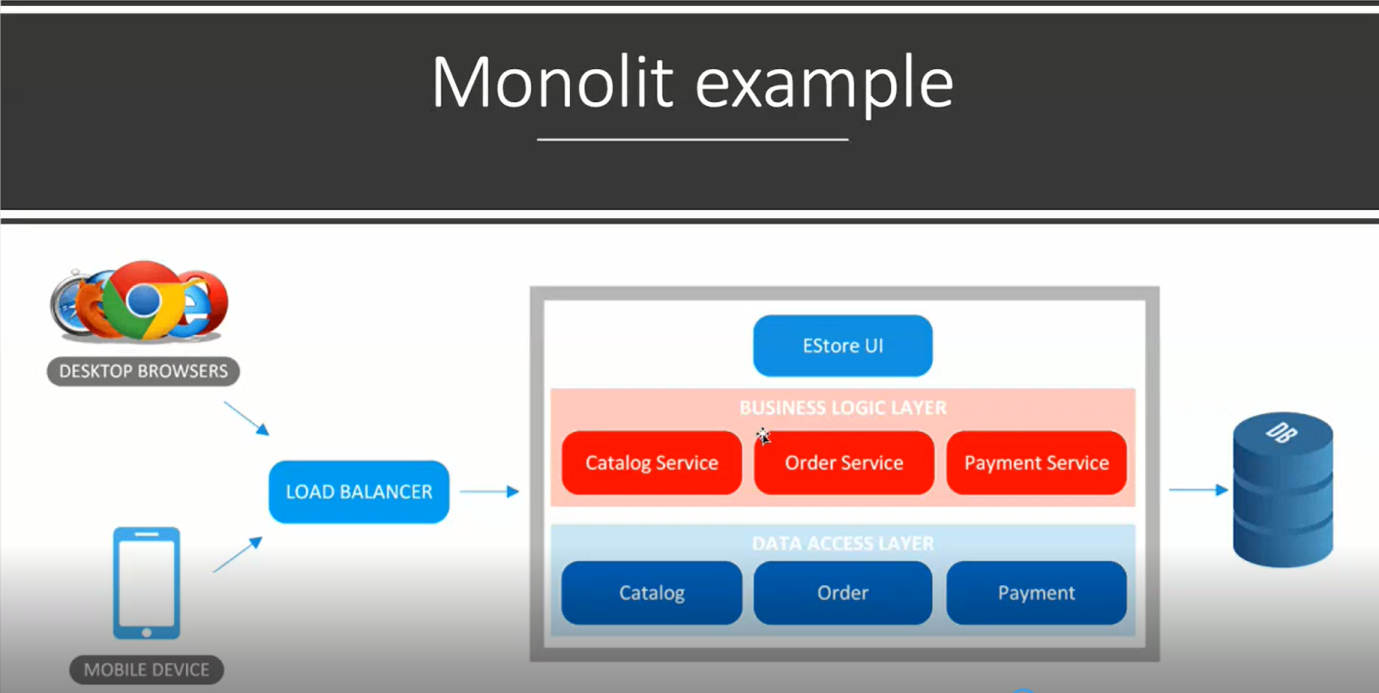
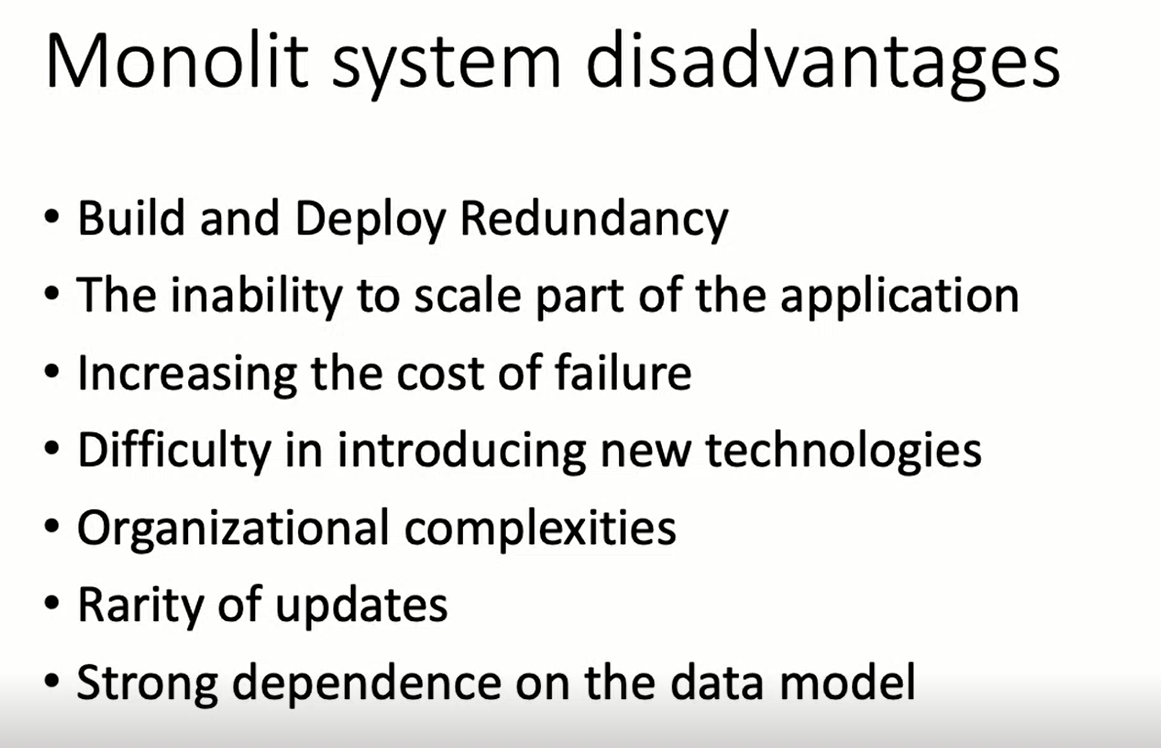
**Step 5)** Next,

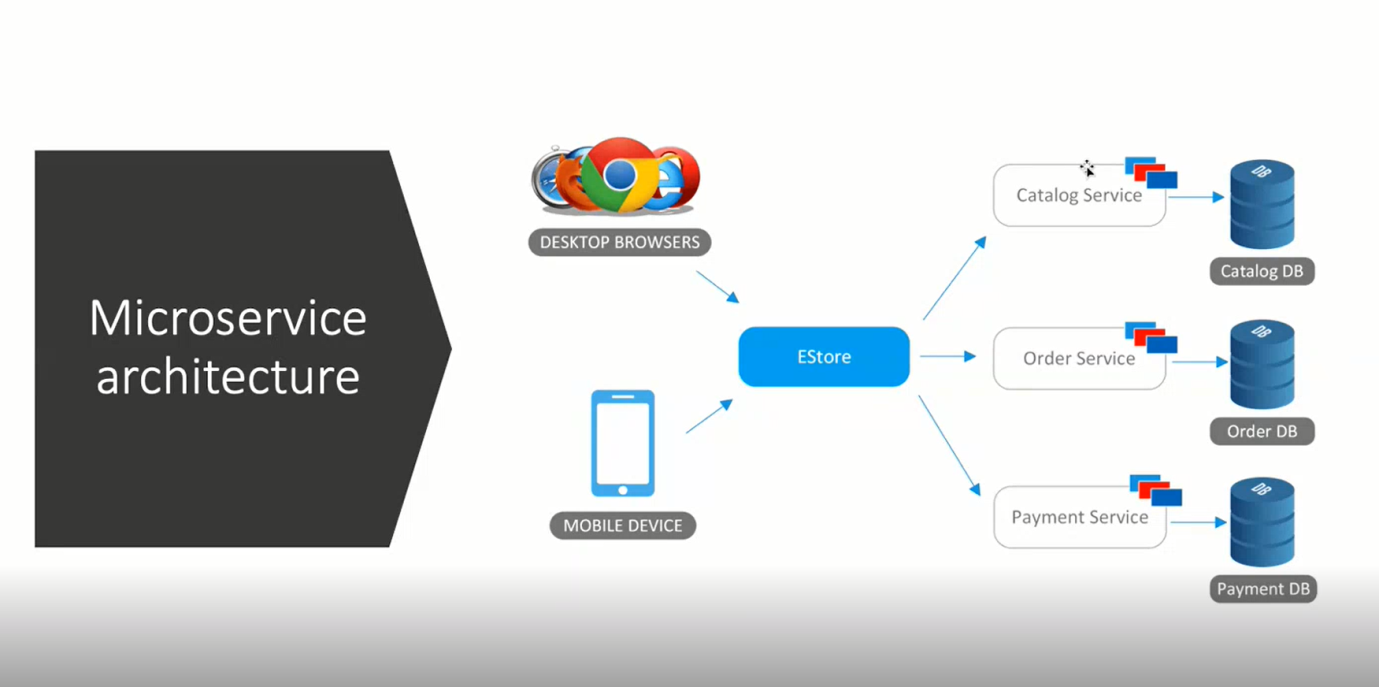
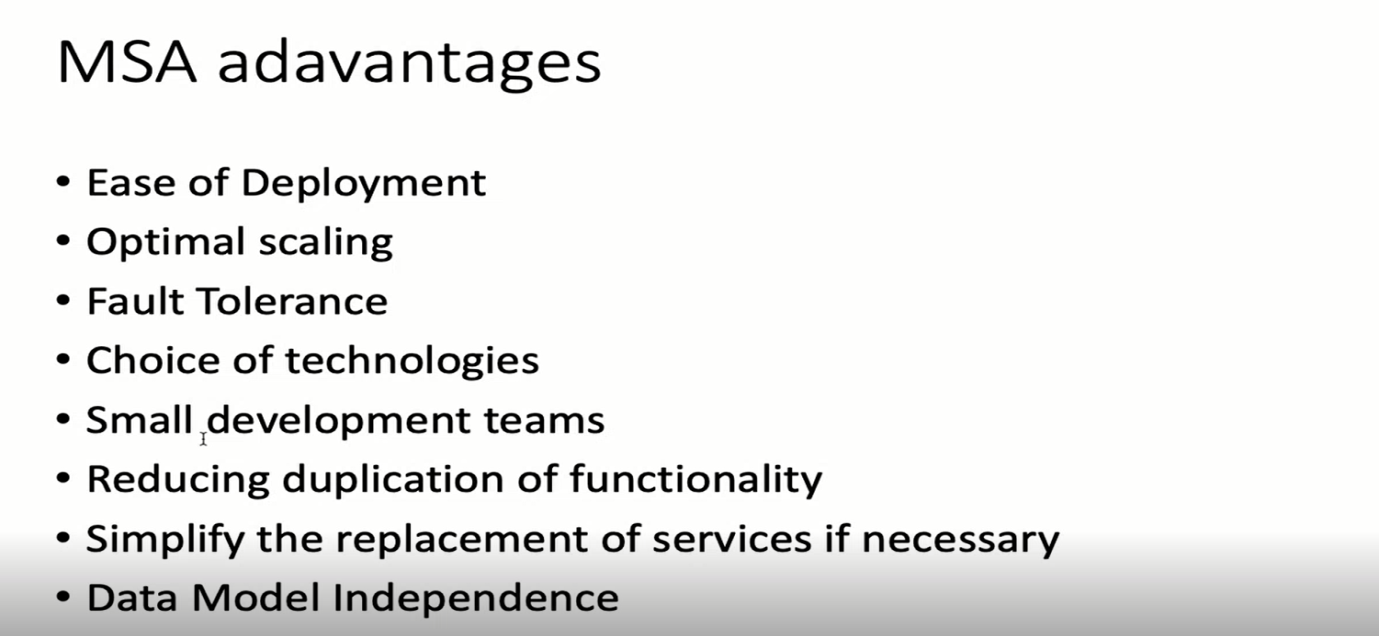
1. Click Send.
2. Status: 201 Created should be displayed
3. Posted data are showing up in the body.

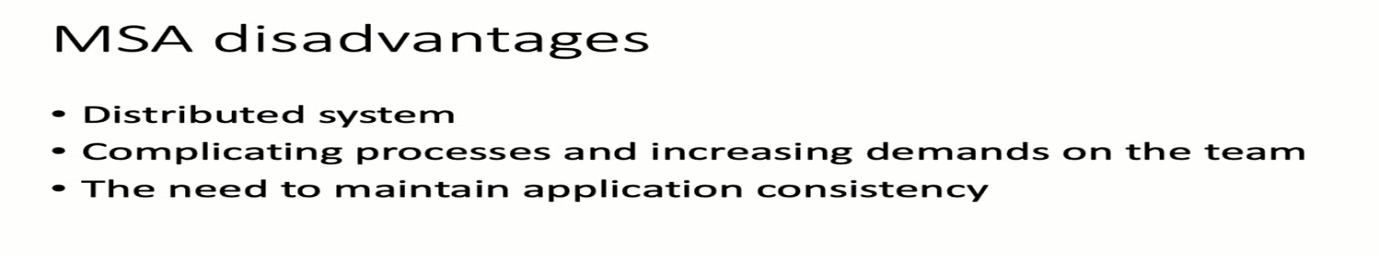




**Monolit example**

**** ****

**** **** 



**New pastebin**

