What is a client?

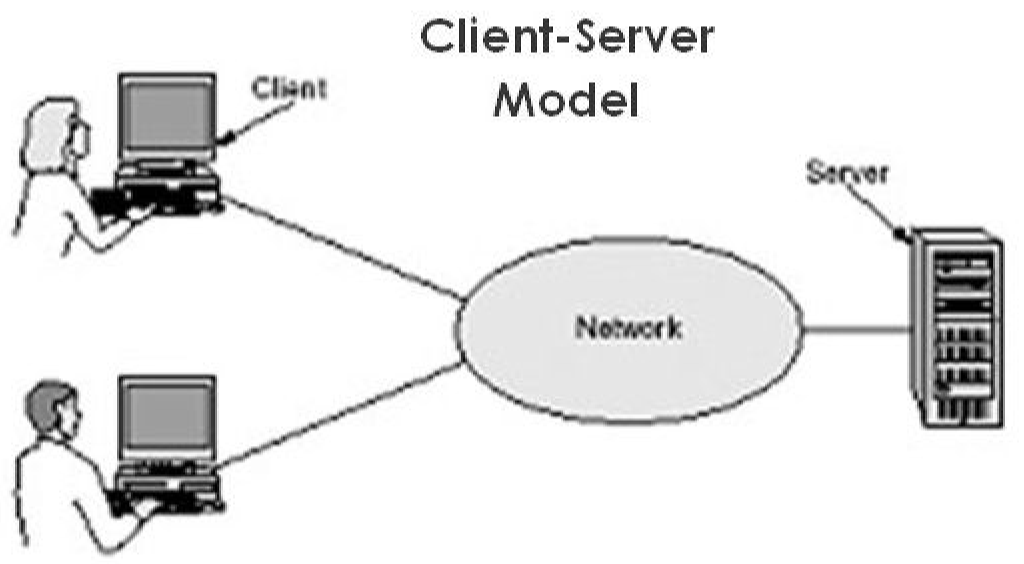
A client is a computer or a program that, as part of its operation, relies on sending a request to another program or a computer hardware or software that accesses a service made available by a server.

What is a server?

A server is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model.

What is a client-server model?

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system.



What are the types of servers?

There are two types of servers Physical and virtual servers.

Physical Servers?

A physical server is just as the name says, a server (physical computer) on which an Operating System, like Windows or Linux runs just as on any other computer.

- Much more expensive than a virtual server (VPS)

  Simply because of the resources needed to run and maintain a physical server, they are much more expensive.

- Harder to manage

  The physical servers are overall much harder to manage. This is especially true with restorations in cases of failures.

- Less scalable

  It is almost impossible to do a server upgrade without additional downtime. Also, it is worth noting that the future upgrades for a dedicated server should be taken into account when ordering the server. Otherwise, the upgrades may lead to ordering a completely new server. That will instead lead to an unplanned service migration and thus unplanned service downtime.

Virtual server?

A virtual server is a server that shares hardware and software resources with other operating systems (OS), versus dedicated servers. Because they are cost-effective and provide faster resource control, virtual servers are popular in Web hosting environments.

- Cheaper than a dedicated server

  The physical servers where the VMs are located can host hundreds of VMs. The resources are then divided among the VMs and so the VMs take very little resources on the parent host thus greatly reducing their price.

- Simplified management

  This is mainly the greatest advantage the VMs have over the physical servers. A VM is much easier to be managed than a physical server

- Simplified backup and restoration

  Where on each physical server a manifest needs to be made of its configuration, applications and what should or shouldn’t be backed up, for the VMs, backups from the whole VMs are made. When a failure occurs for whatever reason, these backups are ready to be restored immediately and a whole VM is restored instead. It is obvious that in such cases downtimes are greatly reduced.

- Scalable and flexible

  There is no downtime for performing resource (plan) upgrades with more RAM, CPU power, disk space etc.

- The perfect choice for hosting any web service

  Whether it is a small blog or a large social network with thousands of visitors per day, the VPS can be easily adjusted to match the load. If needed, more VPSes can quickly and easily be added into a cluster serving different aspects of the web service.

What is HTTP?

HTTP stand for Hypertext Transfer Protocol. HTTP is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.

HTTP Requests?

GET
The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
HEAD
The HEAD method asks for a response identical to that of a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
POST
The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI.
PUT
The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.
DELETE
The DELETE method deletes the specified resource.

What are the Parts of an HTTP request?

It consists of a request line, an optional header followed by an optional message body.It also contains the version of HTTP protocol.

Requests consists of the following elements:

- An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.
- The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here developer.mozilla.org), or the TCP port.
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request has been successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

What are HTTP Headers?

HTTP headers allow the client and the server to pass additional information with the request or the response. An HTTP header consists of its case-insensitive name followed by a colon ':', then by its value (without line breaks). Leading white space before the value is ignored.

Headers can be grouped according to their contexts:

- General header: Headers applying to both requests and responses but with no relation to the data eventually transmitted in the body.
- Request header: Headers containing more information about the resource to be fetched or about the client itself.
- Response header: Headers with additional information about the response, like its location or about the server itself (name and version etc.).
- Entity header: Headers containing more information about the body of the entity, like its content length or its MIME-type.

POSTMAN

So, postman basically is a HTTP client to test APIs, or say to get response of the request we send through the postman application.

So, for testing any API we just enter the URL we want response from and click the SEND button.

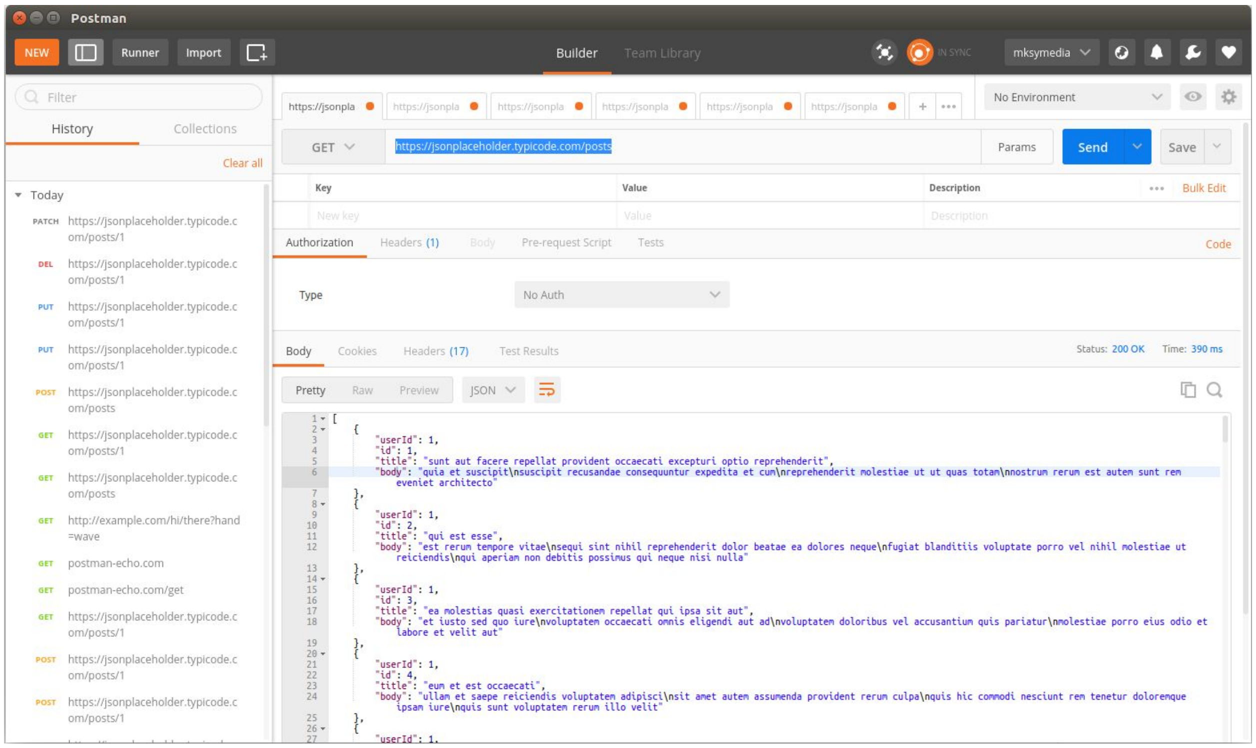
The response is then displayed on the screen below in the format XML,JSON,HTML etc.

It has a history tab towards the left which shows the history of the type of request sent.

I tried with GET https://jsonplaceholder.typicode.com/posts

And i got the response in the response viewer.

POSTMAN SCREENSHOTS



Postman

GET https://jsonplaceholder.typicode.com/posts

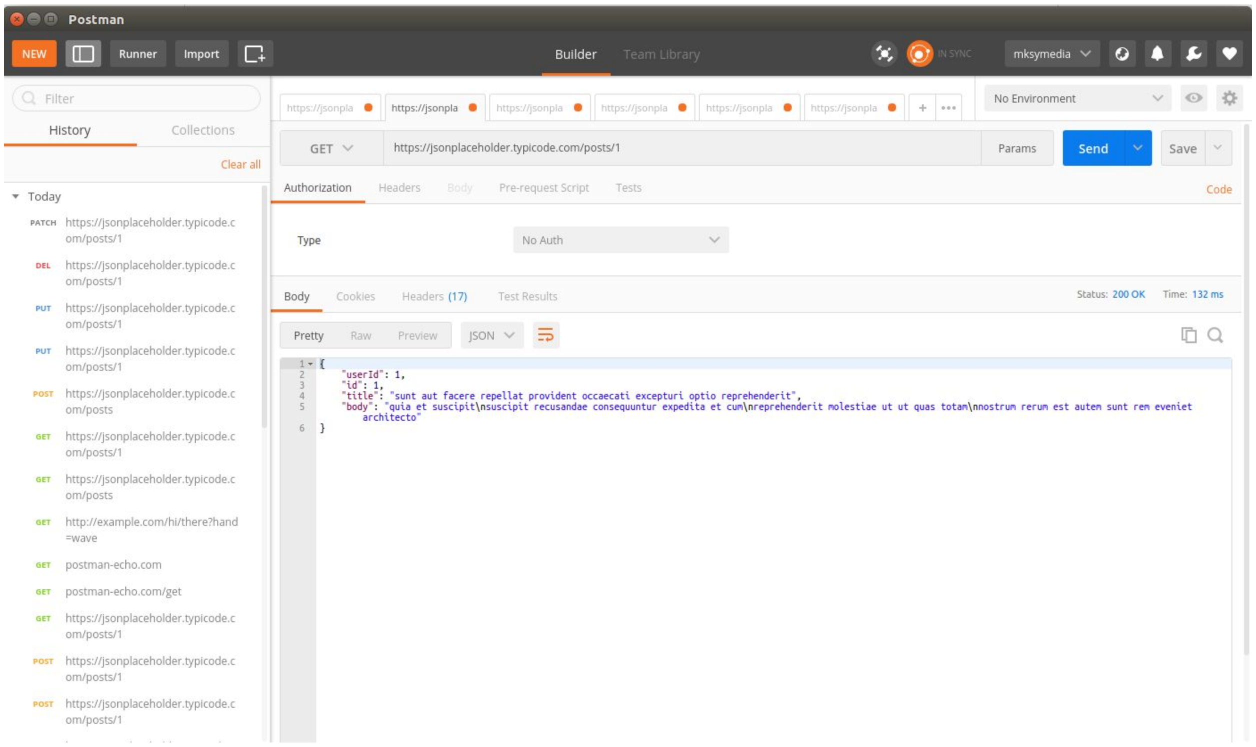
Authorization — Type: No Auth

History — Today:

```
PATCH https://jsonplaceholder.typicode.com/posts/1
DEL   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts
GET   https://jsonplaceholder.typicode.com/posts/1
GET   https://jsonplaceholder.typicode.com/posts
GET   http://example.com/hi/there?hand=wave
GET   postman-echo.com
GET   postman-echo.com/get
GET   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
```

Status: 200 OK Time: 390 ms

```
[
    {
        "userId": 1,
        "id": 1,
        "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
        "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
    },
    {
        "userId": 1,
        "id": 2,
        "title": "qui est esse",
        "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
    },
    {
        "userId": 1,
        "id": 3,
        "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
        "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut"
    },
    {
        "userId": 1,
        "id": 4,
        "title": "eum et est occaecati",
        "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit"
    },
    {
        "userId": 1,
```



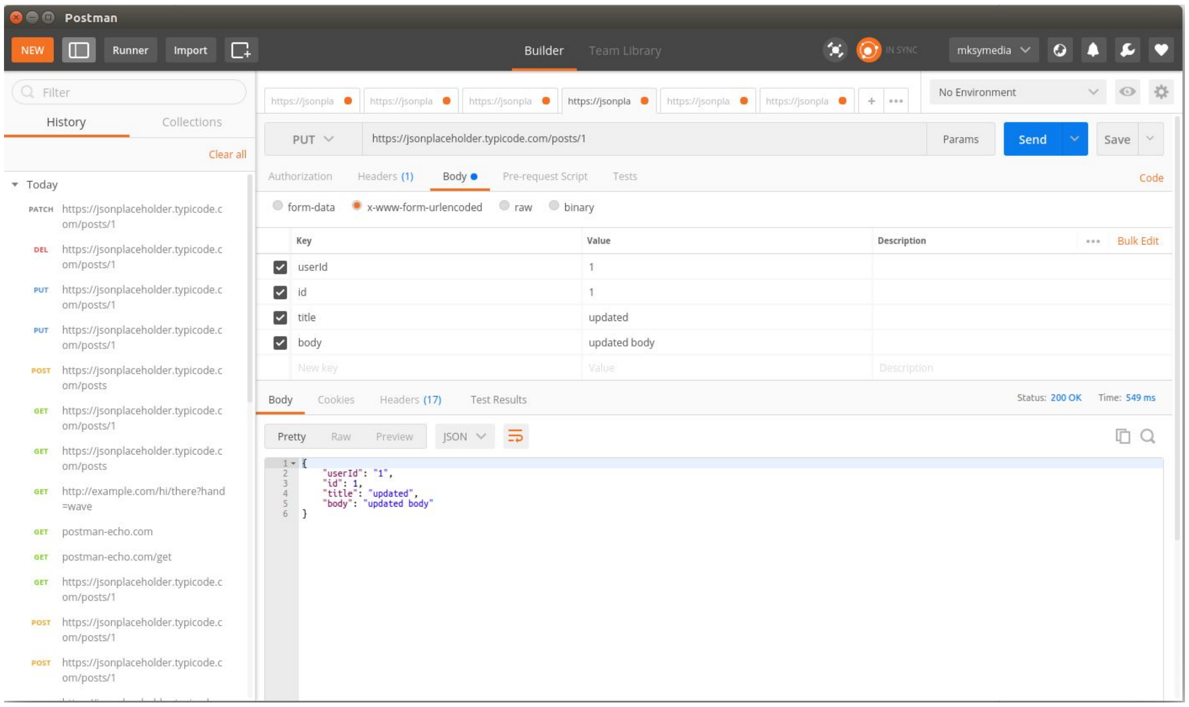
Postman

GET https://jsonplaceholder.typicode.com/posts/1

Authorization — Type: No Auth

Status: 200 OK Time: 132 ms

```
{
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
}
```



History (Today):

```
PATCH https://jsonplaceholder.typicode.com/posts/1
DEL   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts
GET   https://jsonplaceholder.typicode.com/posts/1
GET   https://jsonplaceholder.typicode.com/posts
GET   http://example.com/hi/there?hand=wave
GET   postman-echo.com
GET   postman-echo.com/get
GET   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
```

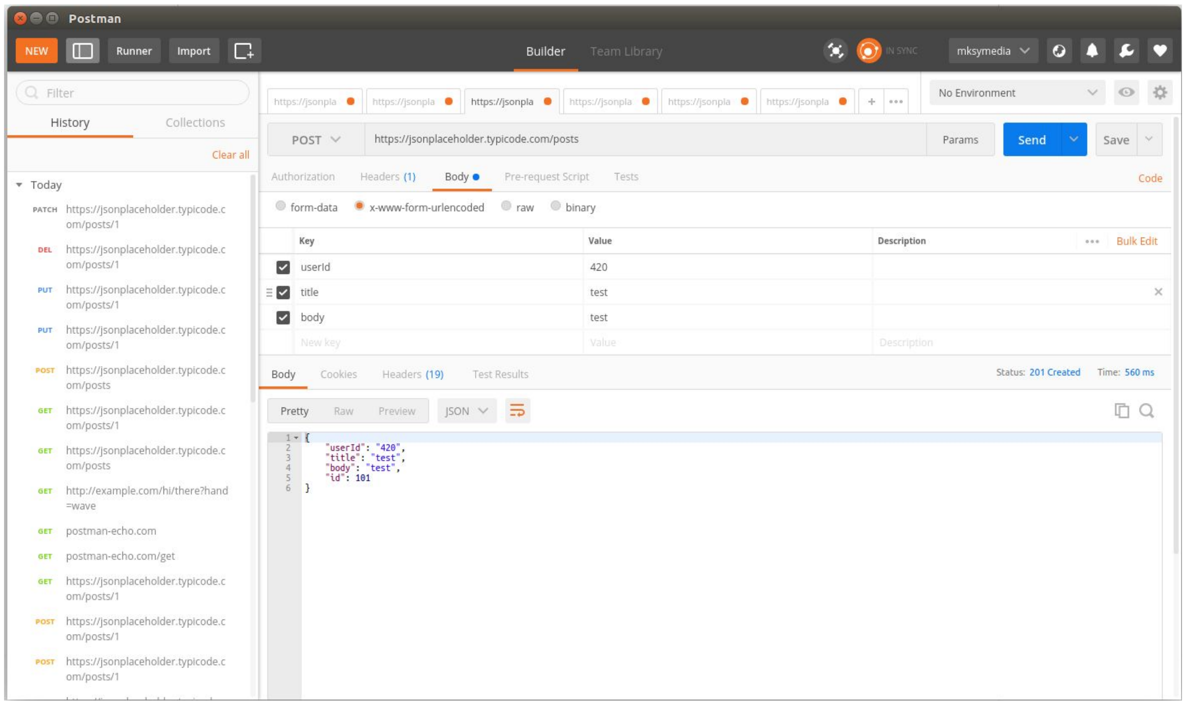
PUT https://jsonplaceholder.typicode.com/posts/1

Body: form-data [✓ x-www-form-urlencoded] raw binary

	Key	Value	Description
✓	userId	1	
✓	id	1	
✓	title	updated	
✓	body	updated body	
	New key	Value	Description

Status: 200 OK Time: 549 ms

```
{
    "userId": "1",
    "id": 1,
    "title": "updated",
    "body": "updated body"
}
```



History (Today):

```
PATCH https://jsonplaceholder.typicode.com/posts/1
DEL   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts
GET   https://jsonplaceholder.typicode.com/posts/1
GET   https://jsonplaceholder.typicode.com/posts
GET   http://example.com/hi/there?hand=wave
GET   postman-echo.com
GET   postman-echo.com/get
GET   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
```

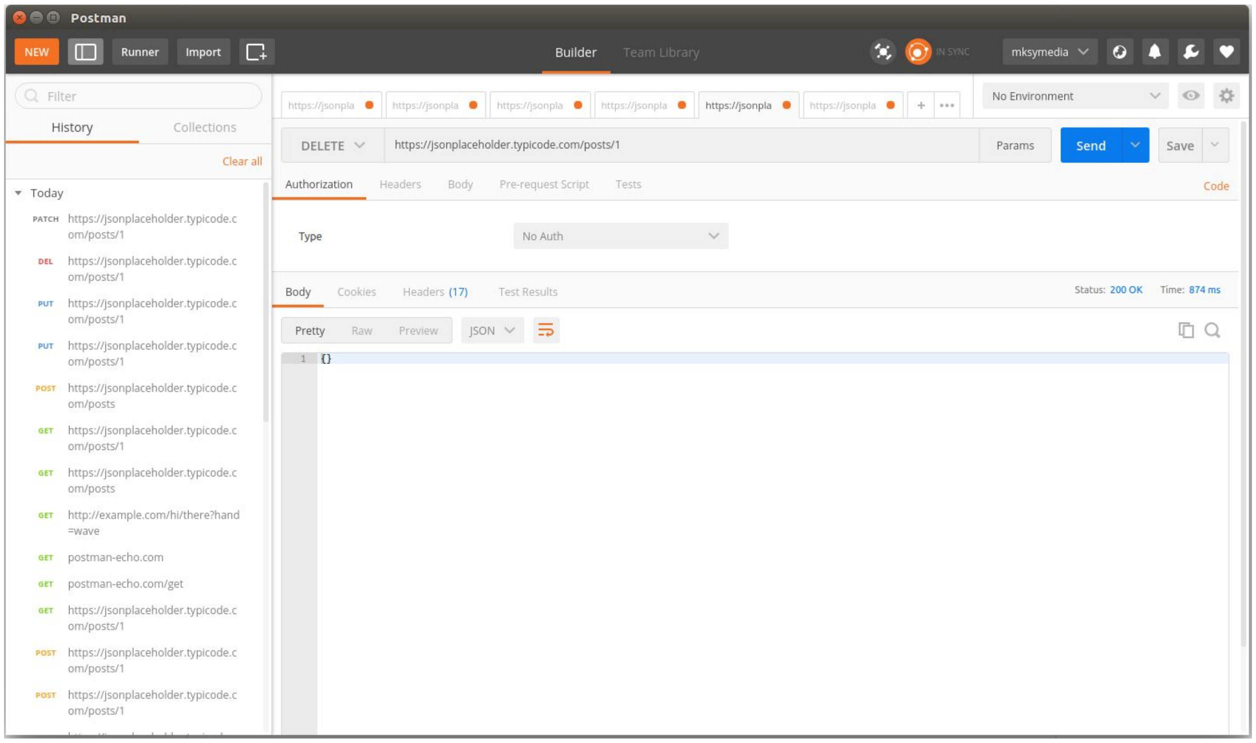
POST https://jsonplaceholder.typicode.com/posts

Body: form-data [✓ x-www-form-urlencoded] raw binary

	Key	Value	Description
✓	userId	420	
✓	title	test	
✓	body	test	
	New key	Value	Description

Status: 201 Created Time: 560 ms

```
{
    "userId": "420",
    "title": "test",
    "body": "test",
    "id": 101
}
```



```
PATCH https://jsonplaceholder.typicode.com/posts/1
DEL   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
PUT   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts
GET   https://jsonplaceholder.typicode.com/posts/1
GET   https://jsonplaceholder.typicode.com/posts
GET   http://example.com/hi/there?hand=wave
GET   postman-echo.com
GET   postman-echo.com/get
GET   https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
POST  https://jsonplaceholder.typicode.com/posts/1
```

```
DELETE https://jsonplaceholder.typicode.com/posts/1
Type: No Auth
Status: 200 OK   Time: 874 ms
{}
```

Postman has a nice UI, which makes it easy to add/remove parameters, changing things is more flexible. Postman also allows users to save requests etc, which cURL is not designed to do. In short, Postman allows a modern, simpler workflow.

cURL

The name stands for "Client URL". cURL is the most used command line tool for making API calls.

The commands are given with a word curl before them and it sends the same response as the postman does.

GET

```
Curl https://jsonplaceholder.typicode.com/posts
```

LIST 1

```
Curl https://jsonplaceholder.typicode.com/posts/3
```

HEADER AND RESPONSE

```
Curl -i https://jsonplaceholder.typicode.com/posts/3
```

Displays all the header attached to the 3rd post and also the 3rd post will be visible.

HEAD

```
Curl --head https://jsonplaceholder.typicode.com/posts/3
```

Displays the header related to 3rd post

DOWNLOAD

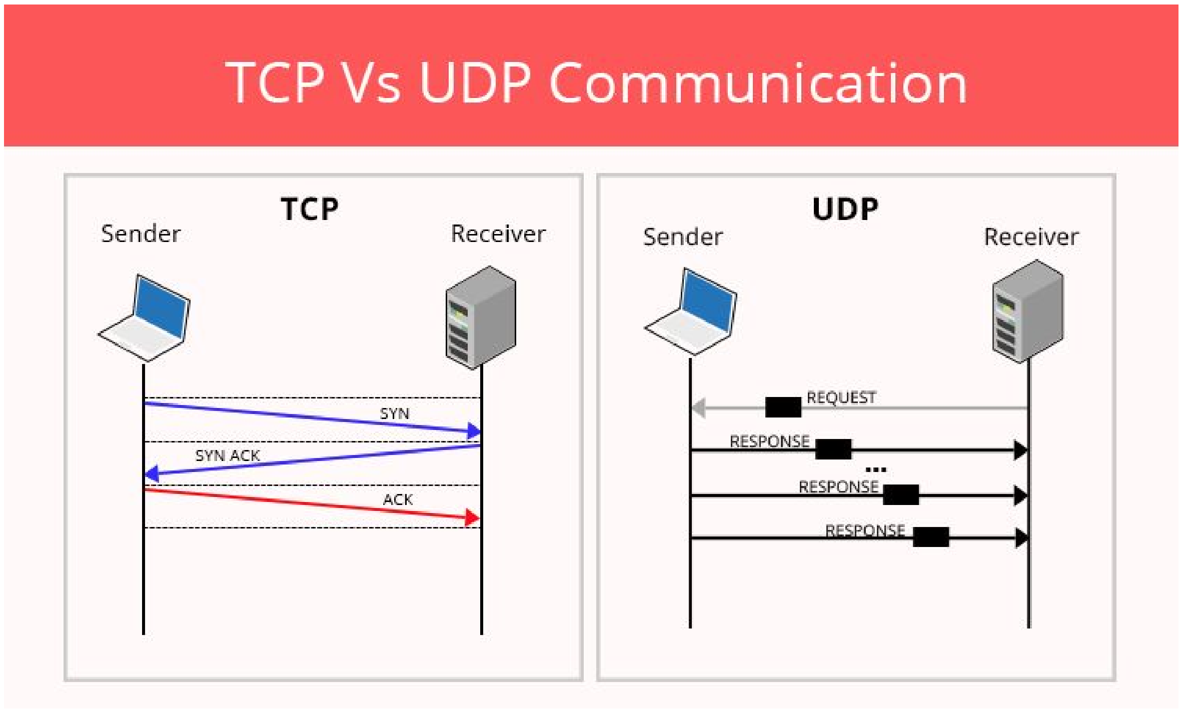
```
Curl -o test.txt https://jsonplaceholder.typicode.com/posts/3
```

It will save the content of post 3 in the file named as “test.txt”

Transmission Control Protocol(TCP)

TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and—because it is meant to provide error-free data transmission handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive.

So TCP is used whenever continuous flow of data is required. For example Emails



User Datagram Protocol(UDP)

UDP has small packet size. It is connectionless so we don't need to create and maintain a connection before sending the data. We have more control over data in UDP. It does not assure of the orderly transmission of data. It does not send any acknowledgment for the received packets.

Video streaming is an example.

Dictionary

Dictionaries are implemented as resizable hash tables. Compared to B-trees, this gives better performance for lookup (the most common operation by far) under most circumstances, and the implementation is simpler.

Dictionaries work by computing a hash code for each key stored in the dictionary using the hash() built-in function. The hash code varies widely depending on the key; for example, “Python” hashes to -539294296 while “python”, a string that differs by a single bit, hashes to 1142331976. The hash code is then used to calculate a location in an internal array where the value will be stored. Assuming that you’re storing keys that all have different hash values, this means that dictionaries take constant time – O(1), in computer science notation – to retrieve a key. It also means that no sorted order of the keys is maintained, and traversing the array as the `.keys()` and `.items()` do will output the dictionary’s content in some arbitrary jumbled order.

Every time a key is generated. The key is passed to a hash function. Every hash function has two parts a Hash code and a Compressor.

Hash code is an Integer number (random or nonrandom). In Java every Object has its own hash code. We will use the hash code generated by JVM in our hash function and to compress the hash code we modulo(%) the hash code by size of the hash table. So modulo operator is compressor in our implementation.

The entire process ensures that for any key, we get an integer position within the size of the Hash Table to insert the corresponding value.

So the process is simple, user gives a (key, value) pair set as input and based on the value generated by hash function an index is generated to where the value corresponding to the particular key is stored. So whenever we need to fetch a value corresponding to a key that is just O(1).