REST APIs with Python

What is an API ?

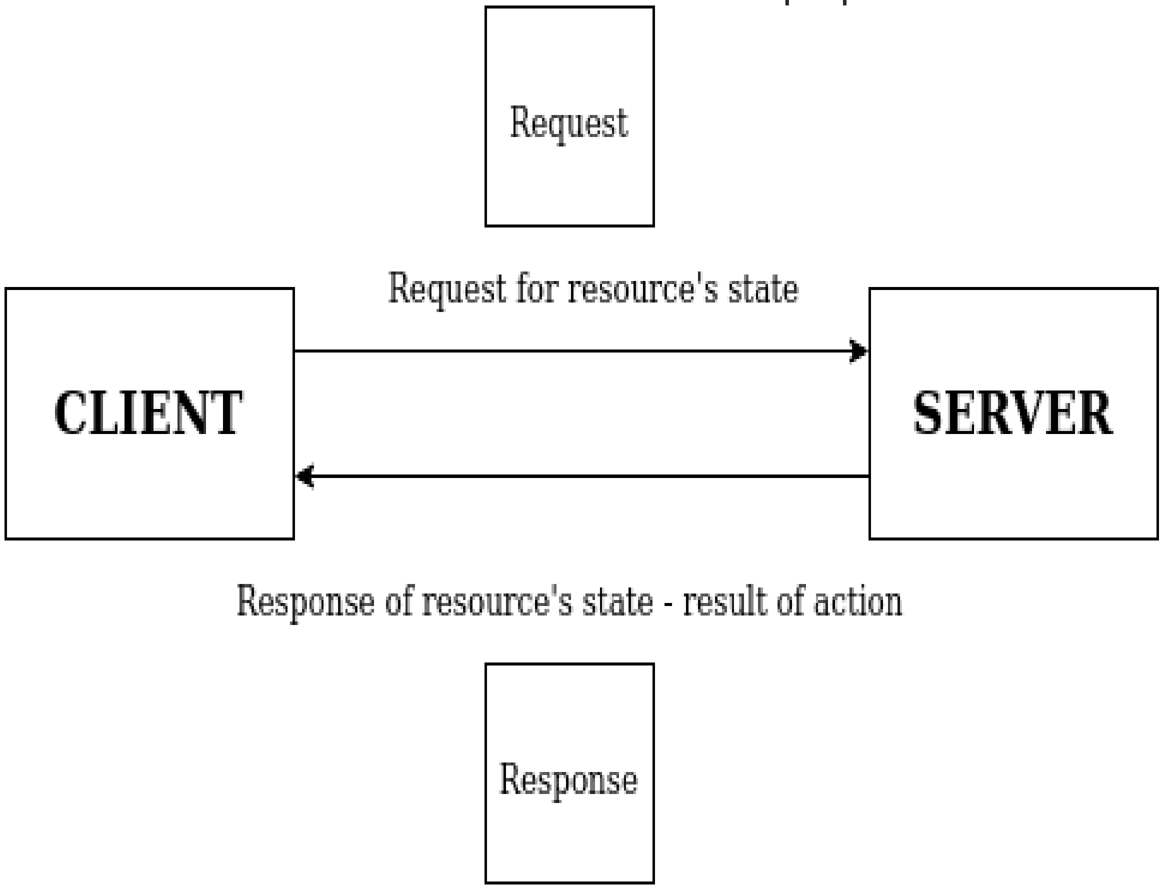
Application Programming Interface or just API is a way for two or more services to communicate with each other. In short it defines a contract, how two services communicate with each other using request and response.

What is a Web service ?

It is a strategy to make service of one application available to other applications over the network and it is platform independent . REST API is one type of web service .

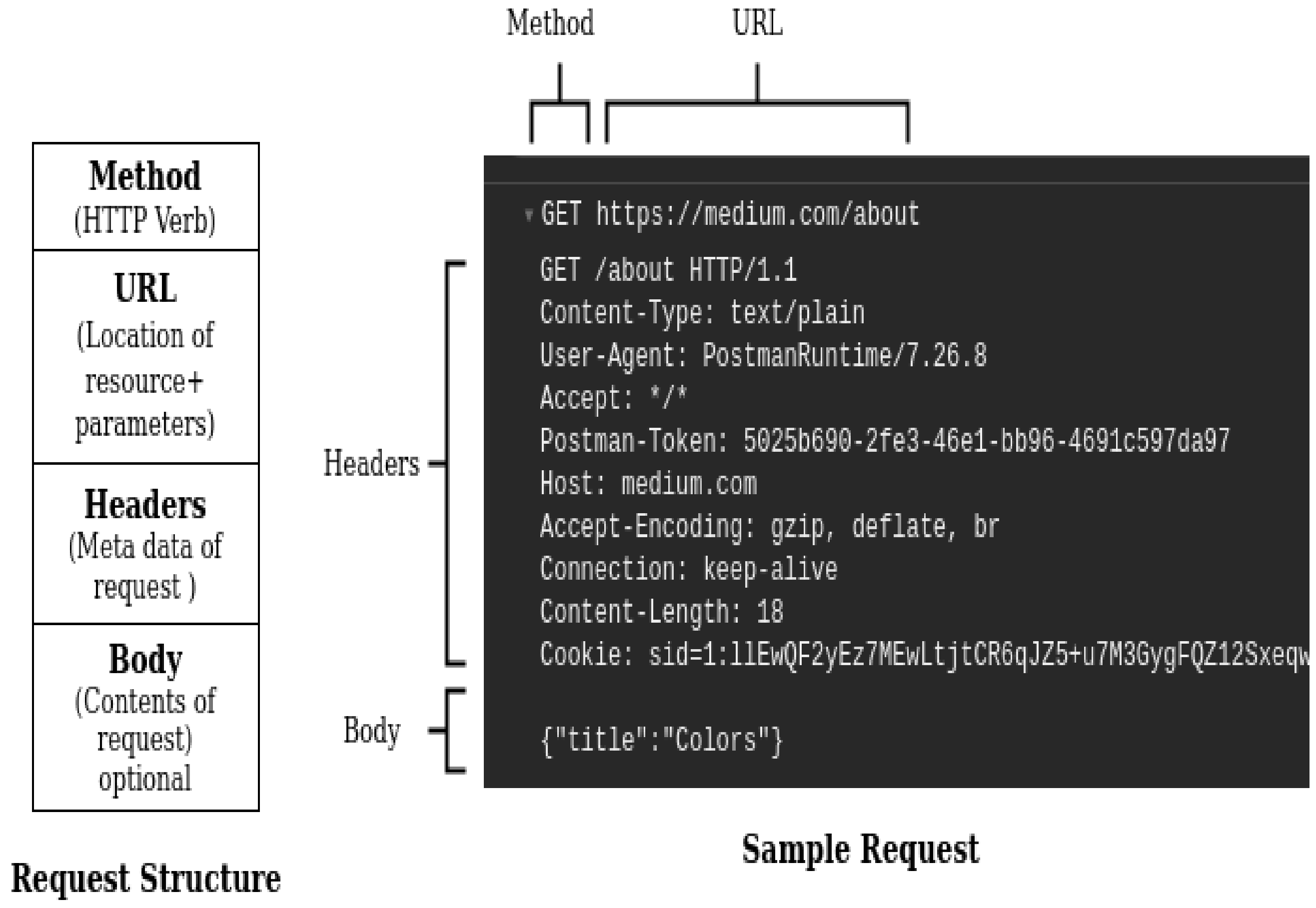
What is REST ?

- Representational State Transfer or just REST is an architectural style
- It revolves around resources (an entity) and clients accessing/modifying these resources by common interface using HTTP standard methods.
- Each resource is identified by URI (Uniform resource identifiers)
- Resource’s state is current resource properties


Request
Request for resource's state
CLIENT
SERVER
Response of resource's state - result of action
Response

REST Request and Response

Request Structure

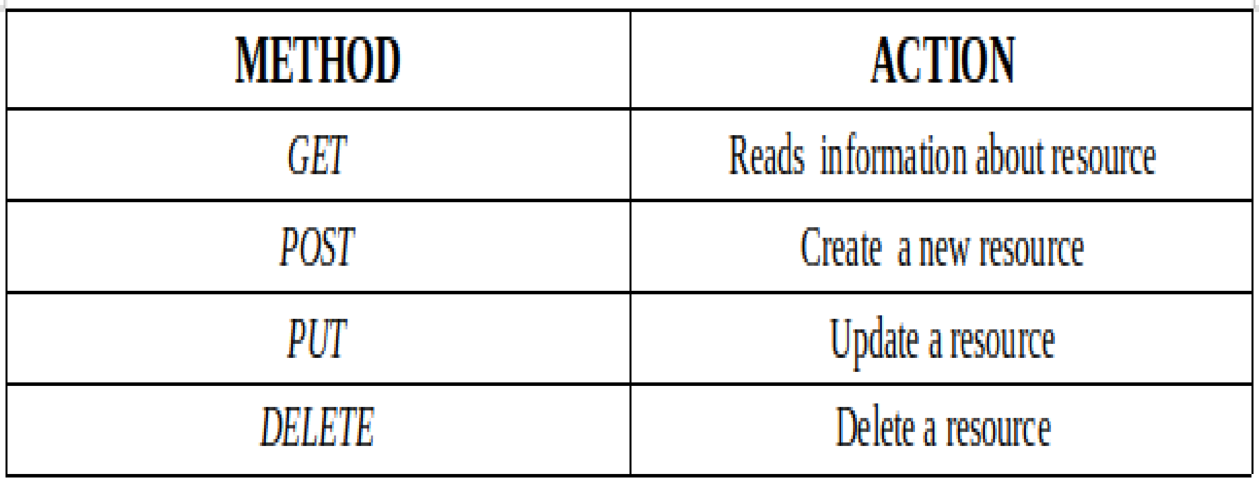


```
▾ GET https://medium.com/about

GET /about HTTP/1.1
Content-Type: text/plain
User-Agent: PostmanRuntime/7.26.8
Accept: */*
Postman-Token: 5025b690-2fe3-46e1-bb96-4691c597da97
Host: medium.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 18
Cookie: sid=1:1lEwQF2yEz7MEwLtjtCR6qJZ5+u7M3GygFQZ12Sxeqw

{"title":"Colors"}
```

Request structure and sample request

- *HTTP Verbs*

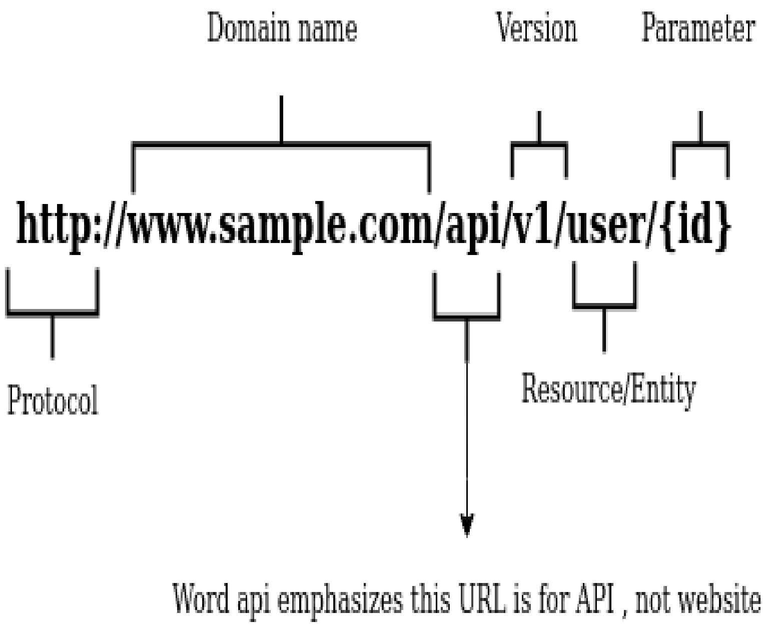
HTTP verbs are used to set the action to be performed. The most commonly used HTTP methods are :



METHOD	ACTION
GET	Reads information about resource
POST	Create a new resource
PUT	Update a resource
DELETE	Delete a resource

HTTP Methods

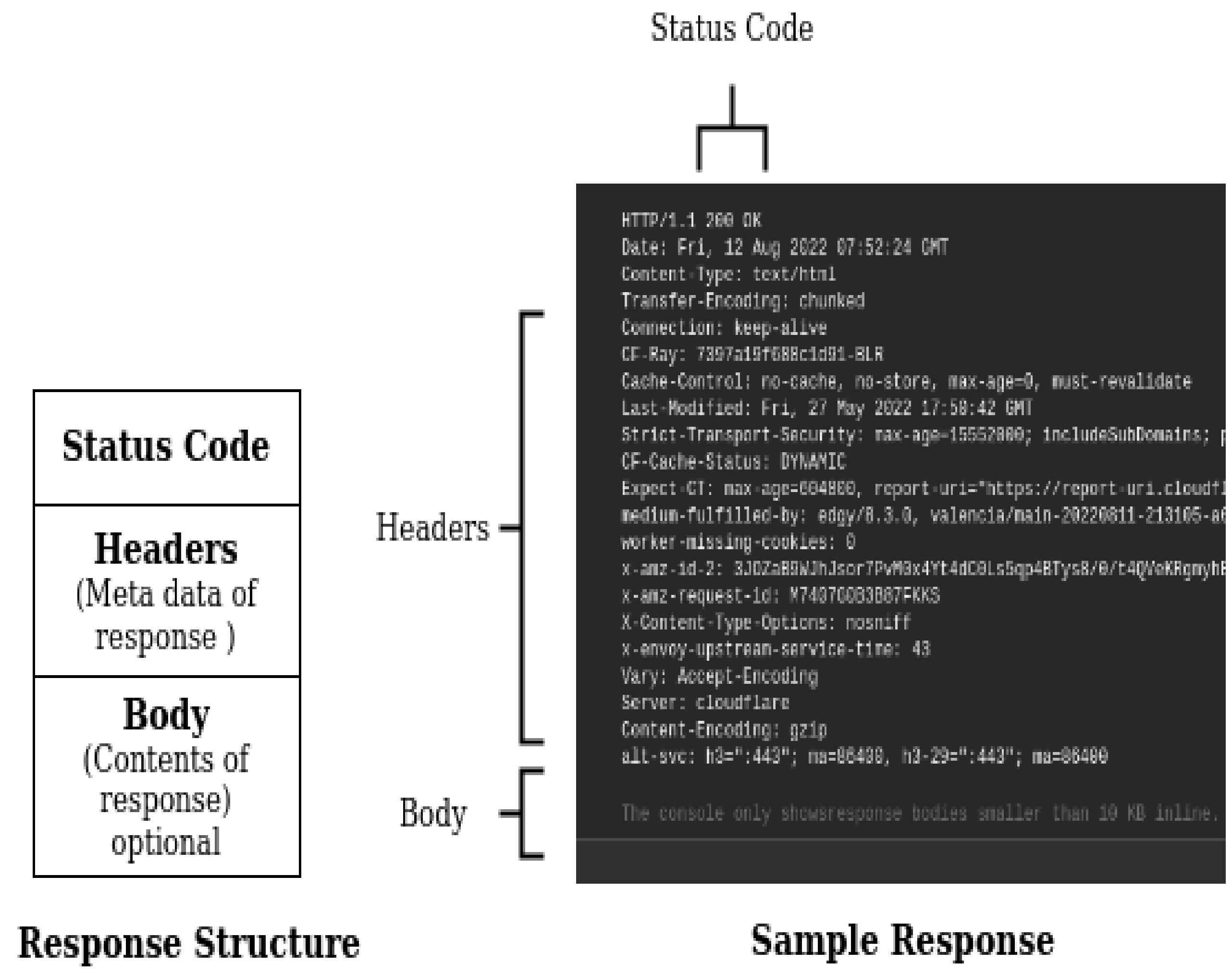
- *URL Structure*


Domain name
Version
Parameter
http://www.sample.com/api/v1/user/{id}
Protocol
Resource/Entity
Word api emphasizes this URL is for API , not website

URL Structure

Query parameters are used to query resources in GET methods. These parameters come at the end , that is after ‘?’ and are concatenated with ‘&’. Below is the example *http://www.sample.com/api/v1/stadium?country=india&state=karnataka*

Response Structure

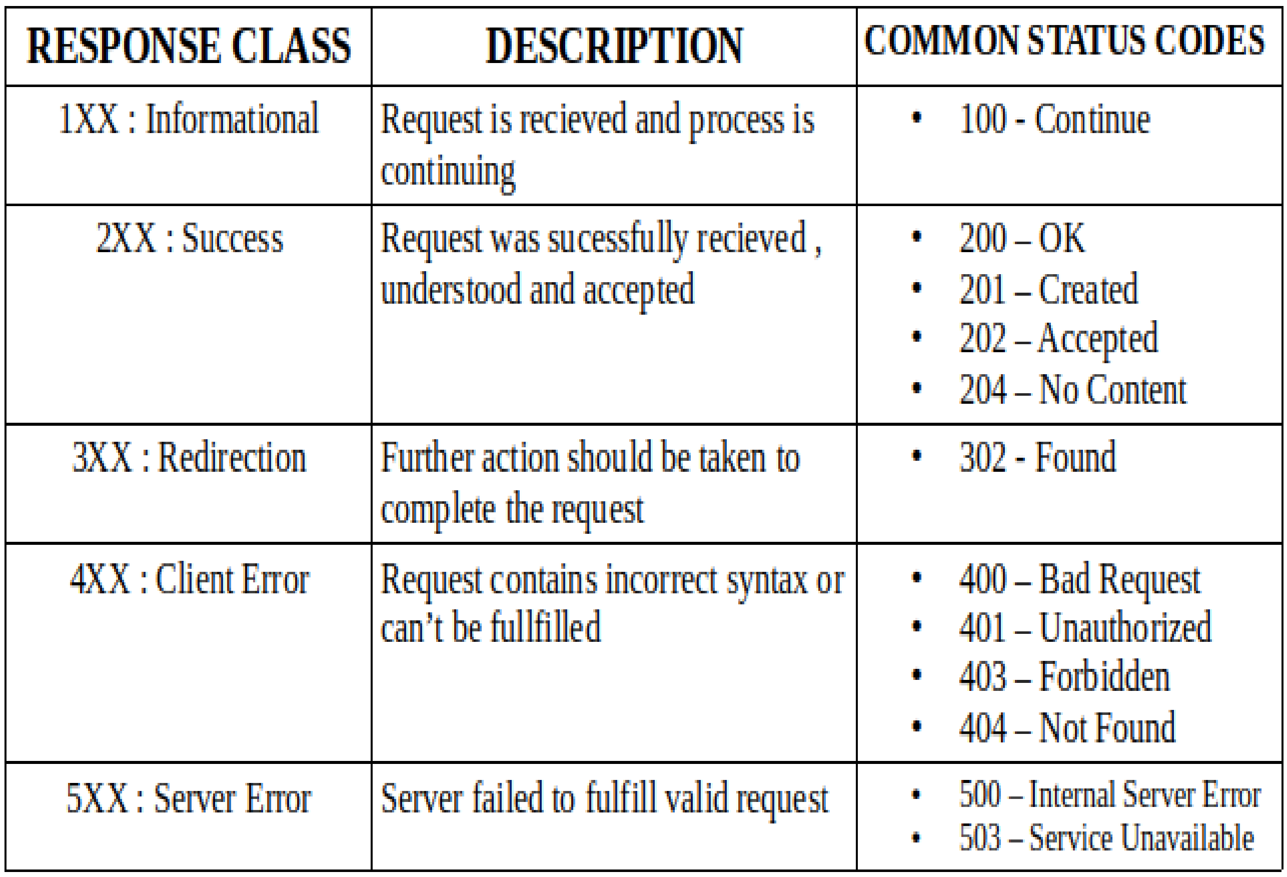


```
HTTP/1.1 200 OK
Date: Fri, 12 Aug 2022 07:52:24 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
CF-Ray: 7397a19f688c1d91-BLR
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Last-Modified: Fri, 27 May 2022 17:58:42 GMT
Strict-Transport-Security: max-age=15552000; includeSubDomains; p
CF-Cache-Status: DYNAMIC
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudf
medium-fulfilled-by: edgy/8.3.0, valencia/main-20220811-213105-a
worker-missing-cookies: 0
x-amz-id-2: 3JDZaB9WJhJsor7PvM0x4Yt4dC0Ls5qp4BTys8/0/t4QVeKRgmyhB
x-amz-request-id: M7407GGB3B87FKKS
X-Content-Type-Options: nosniff
x-envoy-upstream-service-time: 43
Vary: Accept-Encoding
Server: cloudflare
Content-Encoding: gzip
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400

The console only showsresponse bodies smaller than 10 KB inline.
```

Response structure and sample response

- *Status Codes*

The Status code notifies the status of the request . It is a 3 digit integer , where the first digit indicates the class of response.

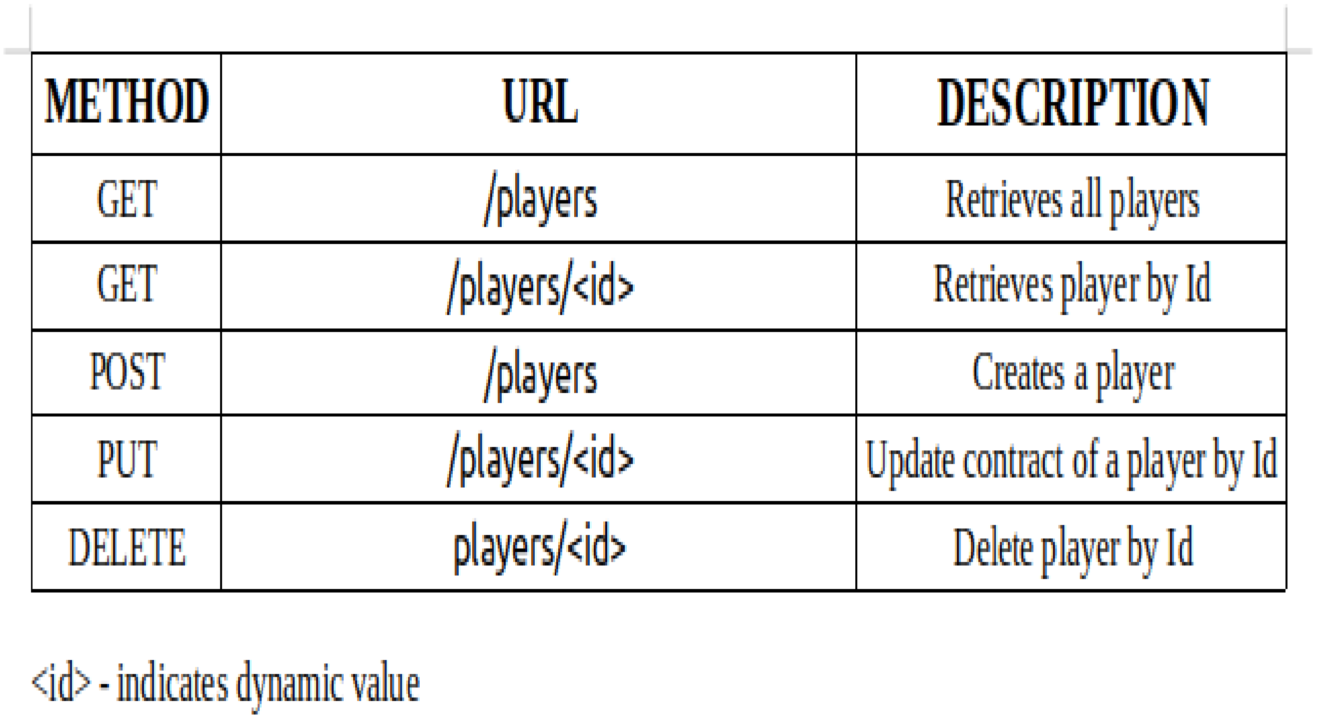


RESPONSE CLASS	DESCRIPTION	COMMON STATUS CODES
1XX : Informational	Request is recieved and process is continuing	• 100 - Continue
2XX : Success	Request was sucessfully recieved , understood and accepted	• 200 – OK • 201 – Created • 202 – Accepted • 204 – No Content
3XX : Redirection	Further action should be taken to complete the request	• 302 - Found
4XX : Client Error	Request contains incorrect syntax or can’t be fullfilled	• 400 – Bad Request • 401 – Unauthorized • 403 – Forbidden • 404 – Not Found
5XX : Server Error	Server failed to fulfill valid request	• 500 – Internal Server Error • 503 – Service Unavailable

Response Codes

Getting Started

We will create a simple project by building a few REST APIs listed below. Here we would perform CRUD operations on a *python list* instead of a database. This list has few dictionaries in it , where each dictionary represents a *player*. Each player dictionary has keys such as Jersey Number, Name, Age, Role and Contract.



METHOD	URL	DESCRIPTION
GET	/players	Retrieves all players
GET	/players/<id>	Retrieves player by Id
POST	/players	Creates a player
PUT	/players/<id>	Update contract of a player by Id
DELETE	players/<id>	Delete player by Id

<id> - indicates dynamic value

1. Installation

- Install the library [flask]
- Download [postman]

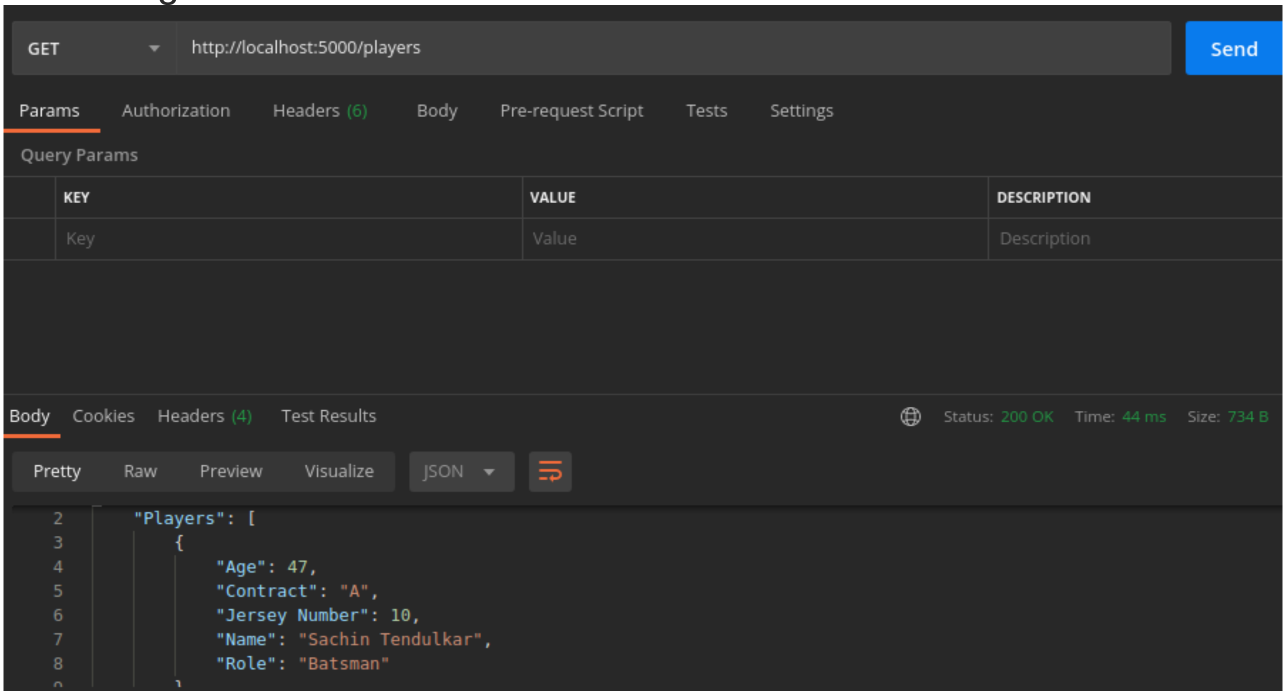
2. Code

- app.py

```
from flask import Flask, jsonify,request

app = Flask(__name__)
app.config['SECRET_KEY']='XXXXXXX'

players =[
          {"Jersey Number": 10,
           "Name" : "Sachin Tendulkar",
           "Age" : 47,
           "Role" : "Batsman",
           "Contract" : "A"
           },
          {"Jersey Number": 7,
           "Name" : "MS Dhoni",
           "Age" : 39,
           "Role" : "Wicket-Keeper",
           "Contract" : "A"
```

```
            },
            {"Jersey Number": 12,
            "Name" : "Yuvraj Singh ",
            "Age" : 37,
            "Role" : "Allrounder",
            "Contract" : "B"
            },
            {"Jersey Number": 28,
            "Name" : "Jasprit Bumrah",
            "Age" : 28,
            "Role" : "Bowler",
            "Contract" : "B"
            },
]

@app.route('/players',methods=["GET"])
def getPlayers():
    return jsonify({'Players':players})

@app.route('/players/<int:id>',methods=["GET"])
def getPlayersById(id):
    return jsonify({'Players':players[id]})

@app.route('/players',methods=["POST"])
def createPlayer():
    if request.method=='POST':
        temp ={}
        temp["Jersey Number"]=request.form['Jersey Number']
        temp["Name"]=request.form['Name']
        temp["Age"]=request.form['Age']
        temp["Role"]=request.form['Role']
        temp["Contract"]=request.form['Contract']
        players.append(temp)
        return jsonify({'Created':temp}),201

@app.route('/players/<int:id>',methods=["PUT"])
def updateContractById(id):
        temp = players[id]
        temp['Contract']= request.form['Contract']
        players[id]=temp
        return jsonify({'Updated':temp})

@app.route('/players/<int:id>',methods=["DELETE"])
def deleteById(id):
        players[id] = {}
        return jsonify({'Deleted':True})

if __name__ == '__main__':
        app.run(debug=True)
```

3. Testing


GET http://localhost:5000/players

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

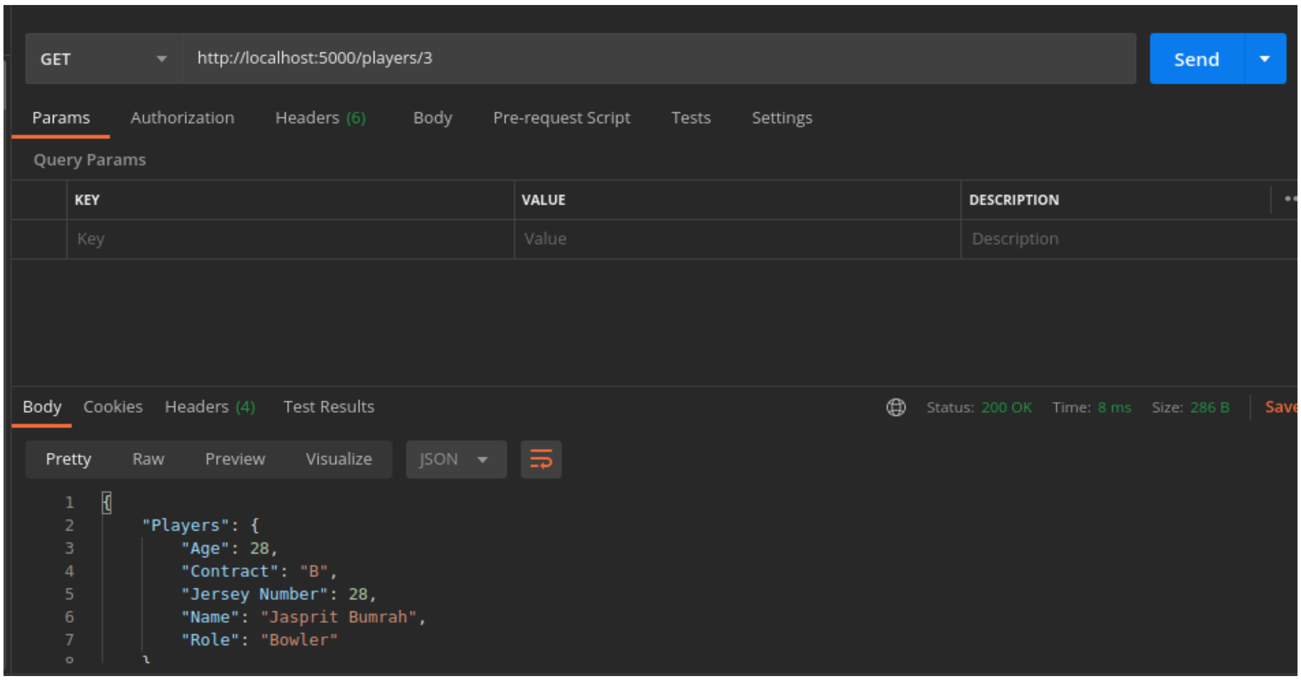
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body | Cookies | Headers (4) | Test Results — Status: 200 OK Time: 44 ms Size: 734 B

Pretty | Raw | Preview | Visualize | JSON

```
    "Players": [
        {
            "Age": 47,
            "Contract": "A",
            "Jersey Number": 10,
            "Name": "Sachin Tendulkar",
            "Role": "Batsman"
```

GET http://localhost:5000/players — Status 200


GET http://localhost:5000/players/3

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

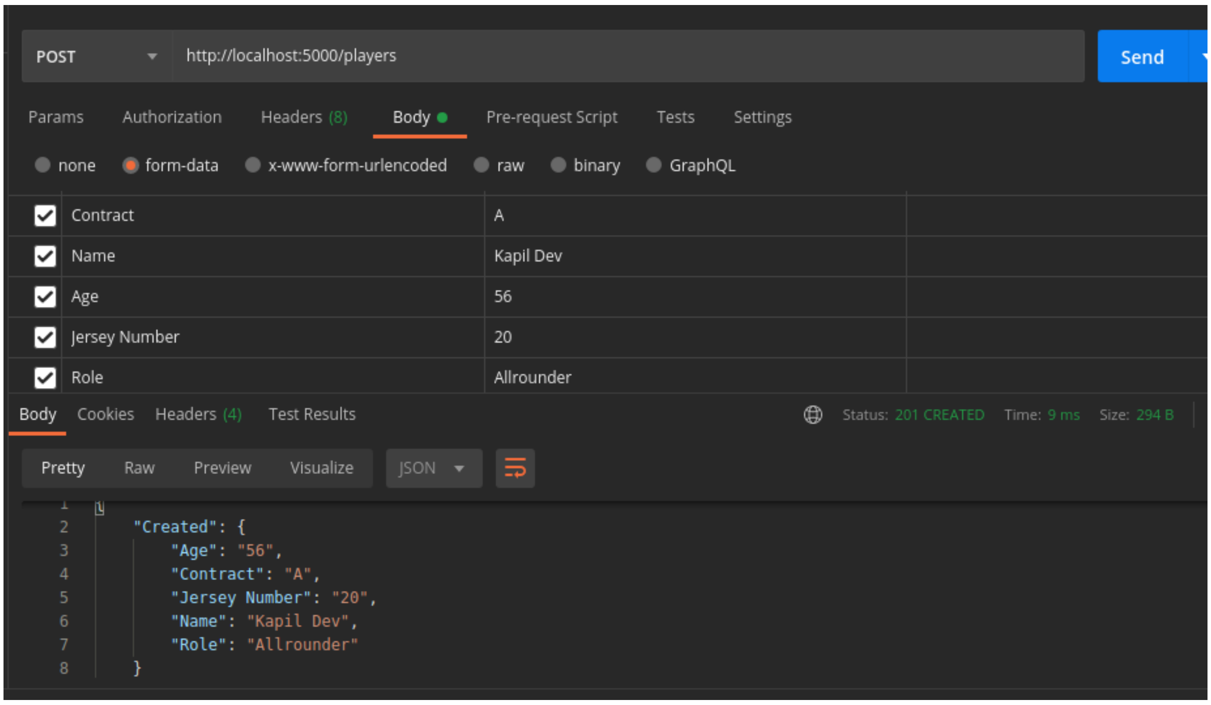
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body | Cookies | Headers (4) | Test Results — Status: 200 OK Time: 8 ms Size: 286 B

Pretty | Raw | Preview | Visualize | JSON

```
{
    "Players": {
        "Age": 28,
        "Contract": "B",
        "Jersey Number": 28,
        "Name": "Jasprit Bumrah",
        "Role": "Bowler"
```

GET http://localhost:5000/players/3 — Status 200



POST http://localhost:5000/players

Params Authorization Headers (8) Body Pre-request Script Tests Settings

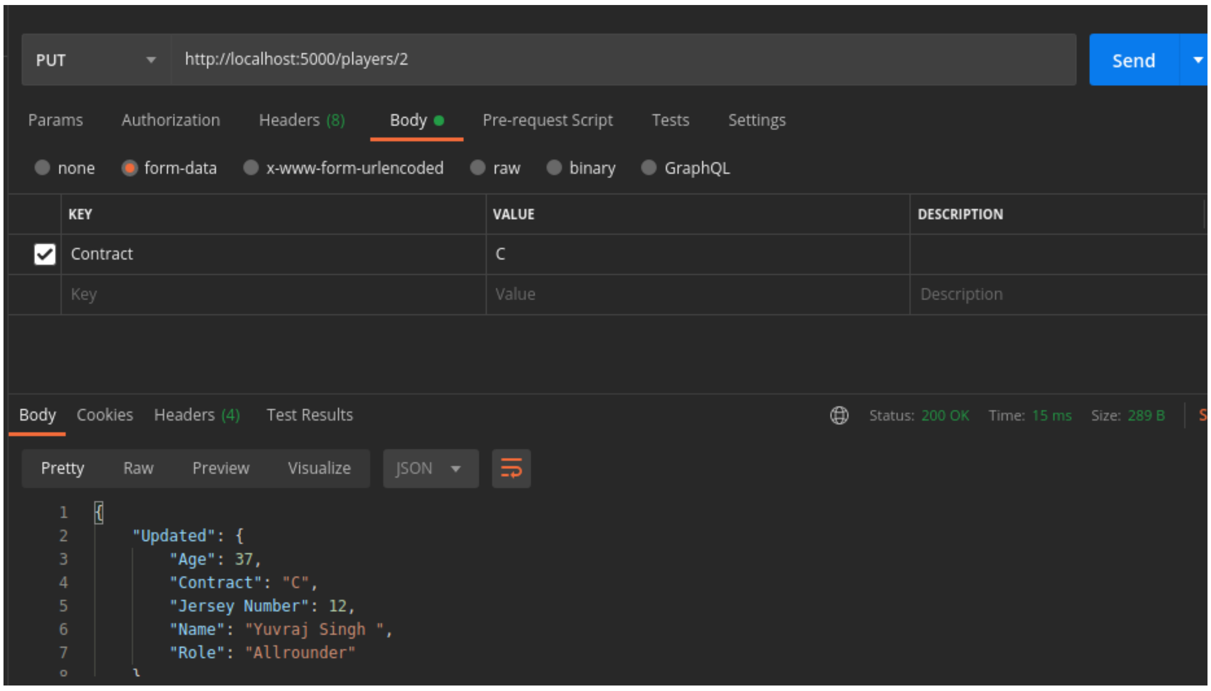
none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

✓	Contract	A
✓	Name	Kapil Dev
✓	Age	56
✓	Jersey Number	20
✓	Role	Allrounder

Body Cookies Headers (4) Test Results — Status: 201 CREATED Time: 9 ms Size: 294 B

```
{
    "Created": {
        "Age": "56",
        "Contract": "A",
        "Jersey Number": "20",
        "Name": "Kapil Dev",
        "Role": "Allrounder"
    }
```

POST http://localhost:5000/players — Status 201



PUT http://localhost:5000/players/2

Params Authorization Headers (8) Body Pre-request Script Tests Settings

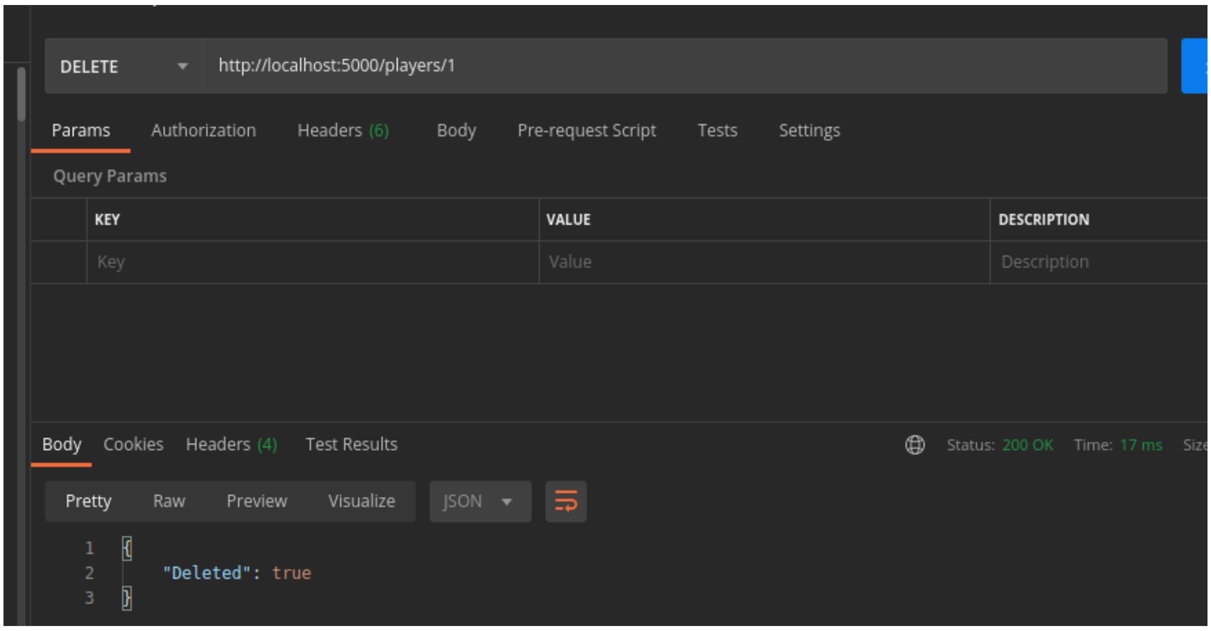
none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	DESCRIPTION
✓	Contract	C	
	Key	Value	Description

Body Cookies Headers (4) Test Results — Status: 200 OK Time: 15 ms Size: 289 B

```
{
    "Updated": {
        "Age": 37,
        "Contract": "C",
        "Jersey Number": 12,
        "Name": "Yuvraj Singh ",
        "Role": "Allrounder"
    }
```

PUT http://localhost:5000/players/2 — Status 200



```
{
    "Deleted": true
}
```

DELETE http://localhost:5000/players/1 — Status 200

Conclusion

In this story, we have seen the basics of REST architecture and created a simple project by building REST APIs using flask in three simple steps. Hope you have understood the basics of REST API’s.