**FLASK**

* Flask is a web framework that provides libraries to build lightweight web applications in python. It is developed by Armin Ronacher who leads an international group of python enthusiasts.
* It is based on WSGI toolkit and jinja2 template engine. Flask is considered as a micro framework.

WSGI=**web server gateway interface**

It is a standard for python web application development. It is considered as the specification for the universal interface between the web server and web application.

**Jinja2**

Jinja2 is a web template engine which combines a template with a certain data source to render the dynamic web pages.

Example:

**from** flask **import** Flask

app = Flask(\_\_name\_\_) #creating the Flask class object

@app.route('/') #decorator drfines the

**def** home():

**return** "hello, this is our first flask website";

**if** \_\_name\_\_ = ='\_\_main\_\_':

    app.run(debug = True)

The route() function of the Flask class defines the URL mapping of the associated function. The syntax is

**app.route(rule, options)**

It accepts the following parameters.

rule: It represents the URL binding with the function.

options: It represents the list of parameters to be associated with the rule object

run method of the Flask class is used to run the flask application on the local development server.

syntax

**app.run(host, port, debug, options)**

|  |  |  |
| --- | --- | --- |
| **SN** | **Option** | **Description** |
| 1 | host | The default hostname is 127.0.0.1, i.e. localhost. |
| 2 | port | The port number to which the server is listening to. The default port number is 5000. |
| 3 | debug | The default is false. It provides debug information if it is set to true. |
| 4 | options | It contains the information to be forwarded to the server. |

**APIs using GET, POST, PUT, DELETE request method**

**HTTP GET:**

* Use GET requests to retrieve resource representation/information only – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be safe methods.
* If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.
* For any given HTTP GET API, if the resource is found on the server, then it must return HTTP response code 200 (OK) – along with the response body, which is usually either XML or JSON content.
* In case resource is NOT found on server then it must return HTTP response code 404 (NOT FOUND).

Example request URIs

HTTP GET http://www.appdomain.com/users

HTTP GET http://www.appdomain.com/users?size=20&page=5

HTTP GET http://www.appdomain.com/users/123

HTTP GET http://www.appdomain.com/users/123/address

**HTTP POST**

* Use POST APIs to create new subordinate resources. e.g., a file is subordinate to a directory containing it or a row is subordinate to a database table
* If a resource has been created on the origin server, the response SHOULD be HTTP response code 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a location header.
* Many times, the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either HTTP response code 200 (OK) or 204 (No Content) is the appropriate response status.
* Responses to this method are not cacheable, unless the response includes appropriate cache control or expires header fields.

Example request URIs

HTTP POST http://www.appdomain.com/users

HTTP POST http://www.appdomain.com/users/123/accounts

**HTTP PUT**

* Use PUT APIs primarily to update existing resource .If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code 201 (Created) response and if an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.
* If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

Example request URIs

HTTP PUT http://www.appdomain.com/users/123

HTTP PUT http://www.appdomain.com/users/123/accounts/456

**HTTP DELETE**

* As the name applies, DELETE APIs are used to delete resources.
* A successful response of DELETE requests should be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.
* DELETE operations are idempotent. If a resource is deleted, it’s removed from the collection of resources.
* Repeatedly calling DELETE API on that resource will not change the outcome – however, calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed.
* If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to this method are not cacheable.

Example request URIs

HTTP DELETE <http://www.appdomain.com/users/123>

* HTTP DELETE http://www.appdomain.com/users/123/accounts/456