⇒ Till now, for every request send, the response msg and data only differed for successful & failed attempts. If the data requested for do not exist, or the operation requested for failed at server, only the returned response data differed.

But, based on data solely, the front-end could not figure out the proble or know its cause efficiently.
efficiently

⇒ For detecting what happened at the server end after a request, we need to send appropriate HTTP status code, alongwith response data.

⇒ There are standard HTTP status codes for each type of situations at Server end.

Ex:-

Suppose, in the project that we are making, suppose we get a id as path parameter, corresponding to which no post exists. Then, alongwith sending a failure message in response body as JSON, we will send/set the Response Status Code to 404

404 Status Code means the resource requested for, does not exist in the Server, i.e., Not Found.

⇒ For doing it in our code, we need to first import the following, to our script :-

response class :-

```
from fastapi import Response
```

This Response class will be required for handling response codes.

Using the Response class, to set status code :-

```
@app.get("/posts/{id}")
def get_post(id: int, response: Response):
    req_post = find_post(id)
    if (req_post == None):
        response.status_code = 404
        return {"post": None}
```

any custom made function to find the post

Thus, whatever status code we want to be for the Response, we need to assign that code to the status_code member variable of the Response class object. Once assigned, that will be the response code automatically we need not to do anything else.

We need to accept Response class object in the parameter of the path operation function.

Status class :-

- Instead of Hard coding the status codes directly as integer into the status_code variable, we can use the variables in the status class, that contain all the required codes.

- We need not remember the status codes, & whenever we will put . (dot) after the class, VS Code will suggest all the available status codes, & we can choose from that list.

Ex:-

```
from FastAPI import status
= = = :
    response.status_code = status.HTTP_
                           404_NOT_FOUND
```

HTTPException :-

- To handle the exceptional case, such as resource requested for does not exist, instead to assigning status code like this manually, we can directly just raise exception.

- HTTPException() takes two parameters (as keyword arguements) :-

status_code → The status code of the response.

detail → The error message that is to be sent as the response body.

The error message that will be sent as the response body.

Ex:-

```
from fastapi import HTTPException
= : = =

def get_post (id:int):
    post = find_post(id)
    if (post == None):
        raise HTTPException (status_code=404,
                             detail = "Error msg")
```

⇒ Changing / Setting Default status code for any Path Operation :-

⇒ Default status code for a path operation means when a request is made for it, that status code will be returned for its every response (if not changed using status_code variable).

⇒ Changing / Setting default status code for any

path operation can be done in the Decorator.

Ex:-

```
@app.post("/posts", status_code=201)
def create_post(post: Post):
    ....
```

This is to be added.

Deleting a Post/Entity :-

While deleting an entity, the default status code should be 204 : No Content.

When we set status code to 204, it is for No Content so in response, we cannot send any kind of data or msg — FastAPI restricts sending any data for 204 status code.

Thus, we just will send something like this :-

```
....
return Response(status_code=204)
```

2:10:00

Updating a Post/Entity :-

- For updating a post, the default status code is 200 : OK

- As convention, when we are updating an entity (post in this case), after updating, we must send the data back that is updated (i.e, the whole object data should be sent back in response)

- Here, too, we need to pass the id of the post (that is to be changed) as Path Parameter.
  Using the id, we need to fetch/find the index of that post first, and then overwrite the old data with new data.