

PLAYWRIGHT

API AUTOMATION

Scenario 1: Validate GET API returns the status code 200, and correct JSON data. Also handling the Authentication in order to capture the Access Token which in turn can be used for further requests to get the data info.

Steps:

1. Perform the login
2. Gather the Access Token
3. Use the Access Token as the Authentication to get the user details

Given Data:

Login URL: "https://dummyjson.com/auth/login"

Example credentials:

- username: emilys
- password: emilypass

GET URL: "https://dummyjson.com/auth/me"

Code:

```
#Scenario 1: Validate GET API returns the status code 200, and correct JSON data.
from playwright.sync_api import Playwright

def test_TC1(playwright:Playwright):

    API_Session = playwright.request.new_context()

    #Login: [POST]
    Login_API_Response = API_Session.post( url: "https://dummyjson.com/auth/login",
                                           data={"username":"emilys", "password":"emilyspass"},
                                           headers={"accept":"application/json"})

    print(f"POST Response: {Login_API_Response.json()}")

    Login_JSON = Login_API_Response.json()

    AccessToken = Login_JSON["accessToken"] #Capture the Access Token

    #Using the Access Token to get the data:
    GET_API_Response = API_Session.get( url: "https://dummyjson.com/auth/me",
                                        headers={"Authorization":AccessToken, "accept":"application/json"})

    assert GET_API_Response.status == 200
    print(f"GET Response: {GET_API_Response.json()}")

    GetData_JSON = GET_API_Response.json()

    assert GetData_JSON["email"] == "emily.johnson@x.dummyjson.com"
```

JSON Response in the Output:

```
PS D:\Python\PythonProject6\API_Automate> pytest -v -s test_scenario1.py
===== test session starts =====
platform win32 -- Python 3.14.0, pytest-9.0.1, pluggy-1.6.0 -- C:\Users\HP\AppData\Local\Programs\Python\Python314\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.14.0', 'Platform': 'Windows-11-10.0.26100-SP0', 'Packages': {'pytest': '9.0.1', 'pluggy': '1.6.0'}, 'Plugins': {'base-url': '2.1.0', 'bdd': '8.1.0', 'html': '4.1.1', 'metadata': '3.1.1', 'playwright': '0.7.2'}, 'JAVA_HOME': 'C:\\Program Files\\Java\\jdk-21', 'Base URL': ''}
rootdir: D:\\Python\\PythonProject6\\API_Automate
plugins: base-url-2.1.0, bdd-8.1.0, html-4.1.1, metadata-3.1.1, playwright-0.7.2
collected 1 item

test_scenario1.py::test_TC1 POST Response: {'accessToken': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Imh0dH8z0i8vZHVtbXlqc29uLmNvbS9pY29uL2VtaWx5cy8xMjg2ZmE1OTAsImV4cCI6MTc2ODY3NTE5MH0.WbF9sAHNgL0t7Ih80g6o2ZPrTacIBRnSowI9RIIPtQY', 'refreshToken': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Imh0dH8z0i8vZHVtbXlqc29uLmNvbS9pY29uL2VtaWx5cy8xMjg2ZmE1OTAsImV4cCI6MTc2ODY3NTE5MH0.EckSUQCZdq03gU2howZRC7CXvy5F3ZbP6wjes97hrw', 'id': 1, 'username': 'emilys', 'email': 'emily.johnson@x.dummyjson.com', 'firstName': 'Emily', 'lastName': 'Johnson', 'gender': 'female', 'image': 'https://dummyjson.com/icon/emilys/128'}
GET Response: {'id': 1, 'firstName': 'Emily', 'lastName': 'Johnson', 'maidenName': 'Smith', 'age': 29, 'gender': 'female', 'email': 'emily.johnson@x.dummyjson.com', 'phone': '+81 9 65-431-3024', 'username': 'emilys', 'password': 'emilyspass', 'birthDate': '1996-5-30', 'image': 'https://dummyjson.com/icon/emilys/128', 'bloodGroup': 'O-', 'height': 193.24, 'weight': 63.16, 'eyeColor': 'Green', 'hair': {'color': 'Brown', 'type': 'Curly'}, 'ip': '42.48.100.32', 'address': {'address': '626 Main Street', 'city': 'Phoenix', 'state': 'Mississippi', 'stateCode': 'MS', 'postalCode': '29112', 'coordinates': {'lat': -77.16213, 'lng': -92.084824}, 'country': 'United States'}, 'macAddress': '47:fa:41:18:ec:eb', 'university': 'University of Wisconsin-Madison', 'bank': {'cardExpire': '05/28', 'cardNumber': '3693233511855044', 'cardType': 'Diners Club International', 'currency': 'GBP', 'iban': 'GB74MH2UZLR9TRPHYNU8F8'}, 'company': {'department': 'Engineering', 'name': 'Dooley, Kozey and Cronin', 'title': 'Sales Manager', 'address': {'address': '263 Tenth Street', 'city': 'San Francisco', 'state': 'Wisconsin', 'stateCode': 'WI', 'postalCode': '37657', 'coordinates': {'lat': 71.814525, 'lng': -161.150263}, 'country': 'United States'}, 'ein': '977-175', 'ssn': '900-590-289', 'userAgent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.93 Safari/537.36', 'crypto': {'coin': 'Bitcoin', 'wallet': '0xb9fc2fe63b2a6c003f1c324c3bfa53259162181a', 'network': 'Ethereum (ERC20)'}, 'role': 'admin'}
PASSED

===== 1 passed in 1.64s =====
```

Scenario 2: Perform the POST request to validate the Response and Response Status Code

Step:

1. Send the Post Request
2. Validate the JSON Response
3. Validate the Status Code

Given Data: For a Single Set of Data

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

Payload:

```
{"title": "QA Automation", "body": "Testing POST method", "userId": 1}
```

Code:

```
def test_TC2(playwright:Playwright):

    API_Session = playwright.request.new_context()

    Post_response = API_Session.post(url="https://jsonplaceholder.typicode.com/posts",
                                     headers={"accept": "application/json"},
                                     data= {"title": "QA Automation", "body": "Testing POST method", "userId": 1})

    assert Post_response.status == 201
    print(f"POST Response: {Post_response.json()}")

    Response = Post_response.json()

    assert Response["title"] == "QA Automation"
```

JSON in Output:

```
===== 1 passed in 1.22s =====
PS D:\Python\PythonProject6\API_Automate> pytest -v -s test_scenario1.py::test_TC2
===== test session starts =====
platform win32 -- Python 3.14.0, pytest-9.0.1, pluggy-1.6.0 -- C:\Users\HP\AppData\Local\Programs\Python\Python314\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.14.0', 'Platform': 'Windows-11-10.0.26100-SP0', 'Packages': {'pytest': '9.0.1', 'pluggy': '1.6.0'}, 'Plugins': {'base-url': '2.1.0', 'bdd': '8.1.0', 'html': '4.1.1', 'metadata': '3.1.1', 'playwright': '0.7.2'}, 'JAVA_HOME': 'C:\\Program Files\\Java\\jdk-21', 'Base URL': ''}
rootdir: D:\Python\PythonProject6\API_Automate
plugins: base-url-2.1.0, bdd-8.1.0, html-4.1.1, metadata-3.1.1, playwright-0.7.2
collected 1 item

test_scenario1.py::test_TC2 POST Response: {'title': 'QA Automation', 'body': 'Testing POST method', 'userId': 1, 'id': 101}
PASSED
```

Given Data: For multi-set payload

POST: “https://dummyjson.com/products/add”

Payload:

```
{
  "products": [
    { "title": "Laptop", "price": 1200 },
    { "title": "Smartphone", "price": 800 },
    { "title": "Headphones", "price": 150 }
  ]
}
```

Code:

```
def test_TC3(playwright:Playwright):
    TC3_DataSet = [{"products": [{ "title": "Laptop", "price": 1200 }, { "title": "Mobile", "price": 600}]}
    API_Session = playwright.request.new_context()
    Post_Response = API_Session.post( url= "https://dummyjson.com/products/add",
                                     data = TC3_DataSet, headers={"accept":"application/json"})

    assert Post_Response.status == 201
    print(f"POST Response: {Post_Response.json()}")
```

If we had a response as:

```
response = {"products": [{ "title": "Laptop", "price": 1200 }, { "title": "Mobile", "price": 600}]}
```

Then, accessing the value “Mobile”:

```
response["products"][1]["title"]
```

Scenario 3: Asserting the invalid requests

Given Data:

Login URL: "https://dummyjson.com/auth/me"

Example credentials:

- username: TESTuser #invalid user name
- password: emilyspass

Code:

```
def test_TC4(playwright:Playwright):  
    API_Session = playwright.request.new_context()  
    GET_API_Response = API_Session.get( url="https://dummyjson.com/auth/me",  
                                       headers={"accept": "application/json"})  
  
    assert GET_API_Response.status == 401  
    print(f"GET Response: {GET_API_Response.json()}")
```

Output:

```
PS D:\Python\PythonProject6\API_Automate> pytest -v -s test_scenario1.py::test_TC4  
===== test session starts =====  
platform win32 -- Python 3.14.0, pytest-9.0.1, pluggy-1.6.0 -- C:\Users\HP\AppData\Local\Programs\Python\Python314\python.exe  
cachedir: .pytest_cache  
metadata: {'Python': '3.14.0', 'Platform': 'Windows-11-10.0.26100-SP0', 'Packages': {'pytest': '9.0.1', 'pluggy': '1.6.0'}, 'Plugins': {'base-url': '2.1.0', 'bdd': '8.1.0', 'html': '4.1.1', 'metadata': '3.1.1', 'playwright': '0.7.2'}, 'JAVA_HOME': 'C:\\Program Files\\Java\\jdk-21', 'Base URL': ''}  
rootdir: D:\Python\PythonProject6\API_Automate  
plugins: base-url-2.1.0, bdd-8.1.0, html-4.1.1, metadata-3.1.1, playwright-0.7.2  
collected 1 item  
  
test_scenario1.py::test_TC4 GET Response: {'message': 'Access Token is required'}  
PASSED  
  
===== 1 passed in 1.94s =====
```

Scenario 4: Automating the Chained API calls

Given Data:

Call List API

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

→ Extract the IDs of the users.

Use ID in Detail API

GET: <https://jsonplaceholder.typicode.com/posts?userId={{id}}>

→ Fetch all posts by the user with IDs captured.

Code:

```
def test_TC5(playwright:Playwright):
    API_Session = playwright.request.new_context()

    #Capture the User ID:

    UserList = API_Session.get(url="https://jsonplaceholder.typicode.com/users",
                               headers={"accept": "application/json"})

    UserList_JSON = UserList.json()
    print(f"User List: {UserList_JSON}")

    UserIDs = []
    for user in UserList_JSON:
        UserIDs.append(user["id"])

    print(f"UserIDs: {UserIDs}")

    # Using the IDs to fetch the Post details for each:

    for ID in UserIDs:
        response = API_Session.get(f"https://jsonplaceholder.typicode.com/posts?userId={ID}")

        print(f"Post by ID:{ID} is {response.json()}\n")
```

Output:

#Capturing the IDs

```
test_scenario1.py:test_TC5 User List: [{"id": 1, 'name': 'Leanne Graham', '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'}}}, {'id': 2, 'name': 'Ervin Howell', 'username': 'Antonette', 'email': 'Shanna@melissa.tv', 'address': {'street': 'Victor Plains', 'suite': 'Suite 879', 'city': 'Wisokyburgh', 'zipcode': '90566-7771', 'geo': {'lat': '-43.9509', 'lng': '-34.4618'}}, {'id': 3, 'name': 'Clementine Bauch', 'username': 'Samantha', 'email': 'Nathan@yesenia.net', 'address': {'street': 'Douglas Extension', 'suite': 'Suite 847', 'city': 'enizhaventown', 'zipcode': '59590-4157', 'geo': {'lat': '-68.6102', 'lng': '-47.0653'}}, {'id': 4, 'name': 'Patricia Lebsack', 'username': 'Karianne', 'email': 'Julianne@kory.org', 'address': {'street': 'Hoeger Mall', 'suite': 'Apt. 692', 'city': 'South Elvis', 'zipcode': '53919-4257', 'geo': {'lat': '29.4572', 'lng': '-164.2990'}}, {'id': 5, 'name': 'Chelsey Dietrich', 'username': 'Kamren', 'email': 'Lucio_Hettinger@annie.ca', 'address': {'street': 'Skiles Walks', 'suite': 'Suite 351', 'city': 'Rosemouth', 'zipcode': '33263', 'geo': {'lat': '-31.8129', 'lng': '62.5342'}}, {'id': 6, 'name': 'Mrs. Dennis Schulist', 'username': 'Leopoldo_Corkery', 'email': 'Karley_Dach@berlin.info', 'address': {'street': 'Norberto Crossing', 'suite': 'Apt. 950', 'city': 'South Christy', 'zipcode': '23505-1337', 'geo': {'lat': '-71.4197', 'lng': '71.7478'}}, {'id': 7, 'name': 'Kurtis Weissnat', 'username': 'Elwyn.Skiles', 'email': 'Telly.Hoeger@billy.biz', 'address': {'street': 'Rex Trail', 'suite': 'Suite 280', 'city': 'Howemouth', 'zipcode': '59804-1099', 'geo': {'lat': '24.8918', 'lng': '21.8984'}}, {'id': 8, 'name': 'Nicholas Runolfsson', 'username': 'Maxime_McCormick', 'email': 'Sherwood@rosamond.me', 'address': {'street': 'Ellsworth Summit', 'suite': 'Suite 729', 'city': 'Aliyaview', 'zipcode': '45169', 'geo': {'lat': '-14.3990', 'lng': '-120.7677'}}, {'id': 9, 'name': 'Jacynthe Leannon', 'username': 'Abernathy Group', 'address': {'street': 'Dayna Park', 'suite': 'Suite 449', 'city': 'Bartholomewbury', 'zipcode': '76495-2930', 'geo': {'lat': '24.6463', 'lng': '-168.8889'}}, {'id': 10, 'name': 'Katie Turnpike', 'username': 'Moriah.Stanton', 'email': 'Rev.Padberg@karina.biz', 'address': {'street': 'Kattie Turnpike', 'suite': 'Suite 198', 'city': 'Lebsackbury', 'zipcode': '31428-2261', 'geo': {'lat': '-38.2386', 'lng': '57.2232'}}, {'id': 11, 'name': 'Ambrose Leannon', 'username': 'Hoeger LLC', 'address': {'street': 'Centralized empowering task-force', 'suite': 'Suite 181', 'city': 'Target End-to-End Models', 'zipcode': '92998-3874', 'geo': {'lat': '-37.3159', 'lng': '81.1496'}}}]
```

#Post by IDs

```
Post by ID:1 is [{"userId": 1, 'id': 1, 'title': 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit', 'body': 'quia et suscipit\nsuscepit 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 ducimus unde\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\nquia sunt voluptatem rerum illo velit'}, {"userId": 1, 'id': 5, 'title': 'nesciunt quas odio', 'body': 'repudiandae veniam quaerat sunt sed\nnihil ut aut sunt\nvoluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque'}, {"userId": 1, 'id': 6, 'title': 'dolorem eum magni eos aperiam quia', 'body': 'ut aspernatur corporis harum nihil quis provident sequi\nmollitia nobis aliquid molestiae\nperspiciatis et ea nemo ab reprehenderit accusantium quas\nvoluptate dolores velit et doloremque molestiae'}, {"userId": 1, 'id': 7, 'title': 'magnam facilis autem', 'body': 'dolore placeat quibusdam ea quo vitae\nmagni quis enim qui quis quo nemo aut saepe\nquidem repellat excepturi ut quia\nsunt ut sed et eos\nsed quas'}, {"userId": 1, 'id': 8, 'title': 'dolorem dolore est ipsam', 'body': 'dignissimos aperiam dolorem qui eum\nfacilis quibusdam animi sint suscipit qui sint possimus cum\nquaerat magni maiores excepturi\nnispsam ut commodi dolor voluptatum modi aut vitae'}, {"userId": 1, 'id': 9, 'title': 'nesciunt iure omnis dolorem tempora et accusantium', 'body': 'consectetur animi nesciunt iure dolore\nnec nam quia ad\nveniam autem ut quam aut nobis\net est aut quod aut provident voluptas autem voluptas'}, {"userId": 1, 'id': 10, 'title': 'optio molestiae id quia eum', 'body': 'quo et expedita modi cum officia vel magni\ndoloribus qui repudiandae\nvero nisi sit\nquos veniam quod sed accusamus veritatis error'}]
```

```
Post by ID:2 is [{"userId": 2, 'id': 11, 'title': 'et ea vero quia laudantium autem', 'body': 'delectus reiciendis molestiae occaecati non minima eveniet qui voluptatibus\naccusamus in eum beatae sit\nvel qui neque voluptates ut commodi qui incidunt\nut animi commodi'}, {"userId": 2, 'id': 12, 'title': 'in quibusdam tempore odit est dolorem', 'body': 'itaque id aut magnam\npraesentium quia et ea odit et ea voluptas et\nsapiente quia nihil amet occaecati quia id voluptatem\nincidunt ea est distinctio odio'}, {"userId": 2, 'id': 13, 'title': 'dolorum ut in voluptas mollitia et saepe qui animi', 'body': 'aut dicta possimus sint mollitia voluptas commodi quo doloremque\nniste corrupti reiciendis voluptatem eius rerum\nsint cumque quod eligendi laborum\nminima\nperferendis recusandae assumenda consetetur porro architecto ipsum ipsam'}, {"userId": 2, 'id': 14, 'title': 'voluptatem eligendi optima', 'body': 'fuga et accusamus dolorum perferendis illo voluptas\nnon doloremque neque facere\nad qui dolorum molestiae beatae\nsed aut voluptas totam sit illum'}, {"userId": 2, 'id': 15, 'title': 'eveniet quod temporibus', 'body': 'reprehenderit quos placeat\nvelit minima officia dolores impedit repudiandae molestiae nam\nvoluptas recusandae quis delectus\nnon officiis harum fugiat vitae'}, {"userId": 2, 'id': 16, 'title': 'sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio', 'body': 'suscipit nam nisi quo aperiam aut\nnam periores eos fugit maiores voluptatibus\nquia\nvoluptatem quis ullam qui in alias quia est\nconsequatur magni mollitia accusamus ea nisi voluptate dicta'}, {"userId": 2, 'id': 17, 'title': 'fugit voluptas sed molestias voluptatem provident', 'body': 'eos voluptas et aut odit natus earum\naspernatur fuga molestiae ullam\ndeserunt ratione qui eos\nqui nihil ratione nemo velit ut aut id quo'}, {"userId": 2, 'id': 18, 'title': 'voluptate et itaque vero tempora molestiae', 'body': 'eveniet quo quis\nlaborum totam consequatur non dolor\nut et est repudiandae\nnest voluptatem vel debitis et magnam'}, {"userId": 2, 'id': 19, 'title': 'adipisci placeat illum aut reiciendis qui', 'body': 'illum quis cupiditate provident sit magnam\nnea sed aut omnis\nveniam maiores ullam consequatur atque\nadipisci quo iste expedita sit quos voluptas'}, {"userId": 2, 'id': 20, 'title': 'doloribus ad provident suscipit at', 'body': 'qui consequuntur ducimus possimus quisquam amet similique\nsuscipit porro ipsam amet\nneos veritatis officiis exercitationem vel fugit aut necessitatibus totam\nnon nam sit rerum consequatur expedita quidem cumque explicabo'}]
```

Scenario 5: Checking the Response Time

Code:

```
def test_TC6(playwright:Playwright):
    API_Session = playwright.request.new_context()

    start_time = time.time() #Records the Start time
    API_Response = API_Session.get( url="https://jsonplaceholder.typicode.com/users",
                                    headers={"accept": "application/json"})

    end_time = time.time() #Records the End time

    Total_time_ms = (end_time - start_time)*1000 #Convert to ms
    Total_time_s = (end_time - start_time)

    print(f"Response_Time in seconds: {Total_time_s}, Response_Time in milli-seconds: {Total_time_ms}")

    #conversion to 2-decimal places:
    print(f"Response_Time: {round(Total_time_ms,2)}")
```

Output:

```
===== 1 passed in 2.04s =====
PS D:\Python\PythonProject6\API_Automate> pytest -v -s test_scenario1.py::test_TC6
===== test session starts =====
platform win32 -- Python 3.14.0, pytest-9.0.1, pluggy-1.6.0 -- C:\Users\HP\AppData\Local\Programs\Python\Python314\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.14.0', 'Platform': 'Windows-11-10.0.26100-SP0', 'Packages': {'pytest': '9.0.1', 'pluggy': '1.6.0'}, 'Plugins': {'base-url': '2.1.0', 'bdd': '8.1.0', 'html': '4.1.1', 'metadata': '3.1.1', 'playwright': '0.7.2'}, 'JAVA_HOME': 'C:\\Program Files\\Java\\jdk-21', 'Base URL': ''}
rootdir: D:\\Python\\PythonProject6\\API_Automate
plugins: base-url-2.1.0, bdd-8.1.0, html-4.1.1, metadata-3.1.1, playwright-0.7.2
collected 1 item

test_scenario1.py::test_TC6 Response_Time in seconds: 0.9729018211364746, Response_Time in milli-seconds: 972.9018211364746
Response_Time: 972.9
PASSED
===== 1 passed in 2.19s =====
```


Scenario 6: Perform a Basic CRUD Operation

Given Data:

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

- ➔ Returns the List

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

- ➔ Create a new resource
- ➔ Payload: {"title": "foo", "body": "bar", "userId": 1}
- ➔ Returns created post with an ID

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

- ➔ Update the resource for the ID
- ➔ Data: {"id": 1, "title": "updated title", "body": "updated body", "userId": 1}
- ➔ Returns the updated post

Delete: <https://jsonplaceholder.typicode.com/posts/1>

- ➔ Delete the resource
- ➔ Returns an empty response

Code:

```
def test_TC7(playwright:Playwright):
    API_Session = playwright.request.new_context()

    #GET Data:
    GET_API_Response = API_Session.get(url: "https://jsonplaceholder.typicode.com/posts",
                                       headers={"accept": "application/json"})

    print(f"GET Response: {GET_API_Response.json()}\n")

    #POST Updates:
    POST_DATA = {"title": "foo", "body": "bar", "userId": 1}
    POST_API_Response = API_Session.post(url: "https://jsonplaceholder.typicode.com/posts",
                                       headers={"accept": "application/json"},
                                       data= POST_DATA)

    print(f"POST Response: {POST_API_Response.json()}\n")

    ## Hitting the GET again to check the Updated data is present in the response:
    GET_API_Response_2 = API_Session.get(url: "https://jsonplaceholder.typicode.com/posts",
                                       headers={"accept": "application/json"})

    ID_list = [] #Checking the new ID added

    for ID in GET_API_Response_2.json():
        ID_list.append(ID["id"])

    print(f"ID_list: {ID_list}\n")
```

```
## Hitting the GET again to check the Updated data is present in the response:
GET_API_Response_2 = API_Session.get(url: "https://jsonplaceholder.typicode.com/posts",
                                       headers={"accept": "application/json"})

ID_list = [] #Checking the new ID added

for ID in GET_API_Response_2.json():
    ID_list.append(ID["id"])

print(f"ID_list: {ID_list}\n")

#Updating the Existing data:
PUT_Data = {"id": 1, "title": "updated title", "body": "updated body", "userId": 1}
PUT_API_Response = API_Session.put(url: "https://jsonplaceholder.typicode.com/posts/1",
                                   headers={"accept": "application/json"},
                                   data=PUT_Data)

print(f"PUT Response: {PUT_API_Response.json()}\n")

#Delete the Data:
Delete_API_Resonse = API_Session.delete(url: "https://jsonplaceholder.typicode.com/posts/1",
                                       headers={"accept": "application/json"})

print(f"DELETE Response: {Delete_API_Resonse.json()}\n")
```