## 10. Requests Library with GraphQL

Working with GraphQL using Python's requests library is straightforward because GraphQL APIs typically accept POST requests with a JSON body that contains the query or mutation string — meaning GraphQL queries must be wrapped in JSON format.

### Basic Steps to Use GraphQL with requests

1. Import required libraries
2. Define the GraphQL query or mutation
3. Wrap it in a JSON payload
4. Send it using requests.post()
5. Parse the JSON response

### JSON Converter

➔ https://datafetcher.com/graphql-json-body-converter

**test_graphql_queries.py**

```
import json
import pytest
import requests
class TestGraphQLQueryTests:
  BASE_URL = "https://hasura.io/learn/graphql"
  AUTH_TOKEN = "Bearer eyJhbGciOiJSUzI1NiIsInR5c...."  # Replace with your full token
  HEADERS = {
     "Authorization": AUTH_TOKEN,
     "Content-Type": "application/json"
  }
```

**Fetch Users and Their Todos**

```
  @pytest.mark.run(order=1)
  def test_fetch_users_and_todos(self):
     query = {
       "query": "{ users { name todos { title } }}"
     }
     response = requests.post(self.BASE_URL, headers=self.HEADERS, json=query)
     assert response.status_code == 200
     data = response.json()
     print("\nResposne\n", json.dumps(data, indent=4))
     assert "data" in data
     assert len(data["data"]["users"]) > 0
     assert data["data"]["users"][0]["name"] is not None
     assert isinstance(data["data"]["users"][0]["todos"], list)
     print("Users fetched successfully")
```

**Fetch Limited Todos**

```
  @pytest.mark.run(order=2)
  def test_fetch_limited_todos(self):
        query = {
            "query": "query { todos(limit: 5) { id title } }"
        }
        response = requests.post(self.BASE_URL, headers=self.HEADERS, json=query)
        assert response.status_code == 200
        data = response.json()
        print("\nResposne\n", json.dumps(data, indent=4))
        todos = data["data"]["todos"]
        assert len(todos) <= 5
        assert todos[0]["id"] is not None
        assert todos[0]["title"] is not None
        print("Limited todos fetched successfully")
```

**Fetch Users with Recent Todos**

```
    @pytest.mark.run(order=3)
    def test_fetch_users_with_recent_todos(self):
        query = {
            "query": "query { users(limit: 2) { id name todos(order_by: {created_at: desc}, limit: 5) { id title
} } }"
        }
        response = requests.post(self.BASE_URL, headers=self.HEADERS, json={"query":
query["query"]})
        assert response.status_code == 200
        data = response.json()
        print("\nResposne\n", json.dumps(data, indent=4))
        users = data["data"]["users"]
        assert len(users) == 2
        assert users[0]["name"] is not None
        assert isinstance(users[0]["todos"], list)
        print("Recent todos fetched successfully")
```

**Fetch Todos Using Variables**

```
    @pytest.mark.run(order=4)
    def test_fetch_todos_with_variables(self):
        query = {
            "query": "query ($limit: Int!) { todos(limit: $limit) { id title }}",
            "variables": {
                "limit": 5
            }
        }
        response = requests.post(self.BASE_URL, headers=self.HEADERS, json=query)
        assert response.status_code == 200
        todos = response.json()["data"]["todos"]
        print("\ntodos\n", json.dumps(todos, indent=4))
        assert len(todos) == 5
        assert todos[0]["id"] is not None
        assert todos[0]["title"] is not None
        print("Todos fetched with variables successfully")
```

**Fetch Public Todos with Filter(Where Clause)**

```
    @pytest.mark.run(order=5)
    def test_5_fetch_public_todos(self):
        query = {
            "query": "{ todos(where: {is_public: {_eq: true}}) { title is_public is_completed }}"
        }
        response = requests.post(self.BASE_URL, headers=self.HEADERS, json={"query":
query["query"]})
        assert response.status_code == 200
        todos = response.json()["data"]["todos"]
        print("\ntodos\n", json.dumps(todos, indent=4))
        assert len(todos) > 0
        assert todos[0]["is_public"] is True
        assert todos[0]["is_completed"] is not None
        print("Public todos validated successfully")
```

**Command to execute**

➔ pytest -s -v -p no:warnings ./test_graphql_queries.py

**test_graphql_mutations.py**

```
import json
import pytest
import requests
class TestGraphQLMutationTests:
    BASE_URL = "https://hasura.io/learn/graphql"
    AUTH_TOKEN = "Bearer eyJhbGciOiJSUzI1NiIsInR5cC"  # Replace with your full token
    HEADERS = {
        "Authorization": AUTH_TOKEN,
        "Content-Type": "application/json"
    }
    inserted_todo_id = None
```

**Insert Todo**

```
    @pytest.mark.run(order=1)
    def test_insert_todo(self):
        insert_mutation = {
            "query": "mutation { insert_todos(objects: [{title: \"sdet\"}]) { affected_rows returning { id
created_at title } }}"
        }
    response = requests.post(self.BASE_URL, headers=self.HEADERS, json=insert_mutation)
    assert response.status_code == 200
    json_data = response.json()
    print("\nInsert Mutation Response:\n", json.dumps(json_data, indent=4))
    todo = json_data["data"]["insert_todos"]["returning"][0]
    self.__class__.inserted_todo_id = todo["id"]
    assert todo["title"] == "sdet"
    print("ToDo inserted with title:", todo["title"])
```

<u>Update Todo</u>

```
@pytest.mark.run(order=2)
def test_update_todo(self):
    assert self.__class__.inserted_todo_id is not None, "Insert must run first"
    update_mutation = {
        "query": f"""
            mutation {{
              update_todos(
                where: {{id: {{_eq: {self.inserted_todo_id}}}}},
                _set: {{title: "sdetqa", is_completed: true}}
              ) {{
                affected_rows
                returning {{
                  id
                  title
                  is_completed
                }}
              }}
            }}
        """
    }
    response = requests.post(self.BASE_URL, headers=self.HEADERS, json=update_mutation)
    assert response.status_code == 200
    json_data = response.json()
    print("\nUpdate Mutation Response:\n", json.dumps(json_data, indent=4))
    updated = json_data["data"]["update_todos"]["returning"][0]
    assert updated["title"] == "sdetqa"
    assert updated["is_completed"] is True
    print("ToDo updated to title:", updated["title"])
```

<u>Delete Todo</u>

```
@pytest.mark.run(order=3)
def test_delete_todo(self):
    assert self.__class__.inserted_todo_id is not None, "Insert must run first"
    delete_mutation = {
        "query": f"""
            mutation {{
              delete_todos(where: {{id: {{_eq: {self.inserted_todo_id}}}}}) {{
                affected_rows
                returning {{
                  title
                }}
              }}
            }}
        """
    }
    response = requests.post(self.BASE_URL, headers=self.HEADERS, json=delete_mutation)
    assert response.status_code == 200
    json_data = response.json()
    print("\nDelete Mutation Response:\n", json.dumps(json_data, indent=4))
    affected_rows = json_data["data"]["delete_todos"]["affected_rows"]
    assert affected_rows == 1
    print("ToDo deleted successfully")
```

**Command to execute**

➔ **pytest -s -v -p no:warnings ./test_graphql_mutations.py**

**Note**

➔ **self:** refers to the current instance of the test class.

➔ **self.__class__:** gets the class of the current instance.

➔ **self.__class__.inserted_todo_id:** refers to a class variable called inserted_todo_id. This is shared across all instances of the class (i.e., common memory).