4. Types of Authentication in the requests Library

Types of Authentication in Requests Library

1. Basic or Basic - Preemptive Authentication
2. Digest Authentication
3. Bearer Token Authentication
4. API Key Authentication
5. OAuth 2.0 Authentication

test_Authentication.py

```
import requests
from requests.auth import HTTPDigestAuth
class TestAuthentication:
```

Basic or Basic-Preemptive Authentication

```
  def test_basic_auth(self):
     response = requests.get(
        "https://postman-echo.com/basic-auth",
        auth=("postman", "password")
     )
     assert response.status_code == 200
     assert response.json().get("authenticated") is True
     print(response.json())
```

Digest Authentication

```
  def test_digest_auth(self):
     response = requests.get(
        "https://postman-echo.com/digest-auth",
        auth=HTTPDigestAuth("postman", "password")
     )
     assert response.status_code == 200
     assert response.json().get("authenticated") is True
     print(response.json())
```

Bearer Token Authentication

```
  def test_bearer_token_auth(self):
     bearer_token = "ghp_wB9HWlzzxQU6DxCjvXhRKBoWKuguhW4UC"  # Replace with a valid token
     headers = {
        "Authorization": f"Bearer {bearer_token}"
     }
     response = requests.get(
        "https://api.github.com/user/repos",
        headers=headers
     )
     assert response.status_code == 200
     print(response.json())
```

API Key Authentication

```
  def test_api_key_auth(self):
      params = {
          "q": "Delhi",
          "appid": "fe9c5cddb7e01d747b4611c3fc9eaf2c"  # Replace with a valid API key
      }
      response = requests.get(
          "https://api.openweathermap.org/data/2.5/weather",
          params=params
      )
      assert response.status_code == 200
      print(response.json())
```

Command to Execute

➔ pytest -s -v Day4/test_Authentication.py

Note

➔ In RestAssured, there are two types of authentication: Preemptive and Reactive.

➔ In Python's `requests` library, there is no such distinction because it automatically behaves as preemptive when using basic authentication.

➔ **Preemptive Authentication:**

- Credentials are sent in the first request automatically.
- Saves an extra round trip.
- Python's requests behaves this way by default when using auth=("username", "password").

➔ **Reactive Authentication:**

- The client waits for a 401 Unauthorized from the server before resending the request with credentials.
- Requires manual implementation in Python if needed

test_reactive_auth.py

```
import requests
class TestReactiveAuth:
  def test_reactive_auth(self):
```

Step 1: First request without authentication

```
      response = requests.get("https://postman-echo.com/basic-auth")
```

Step 2: If server responds with 401, retry with credentials

```
      if response.status_code == 401:
          response = requests.get(
              "https://postman-echo.com/basic-auth",
              auth=("postman", "password")
          )
```

Step 3: Validate the authenticated response

```
      assert response.status_code == 200
      assert response.json().get("authenticated") is True
```

```
    print("Reactive Authentication:", response.json())
```

OAuth 2.0 Authentication

1. From the application (Manual process)
   a. Client ID
   b. Client Secrete
2. Send Post request for getting token
   a. POST https://accounts.spotify.com/api/token
      i. Client ID
      ii. Client Secrete
      iii. Token URL
      iv. Redirect URL
      v. Grant type
      vi. Authorization code
3. We will get token once POST request is succesfull.
4. Use Token to do API call (Get request).

test_oauth2_authentication.py

```
import pytest
import requests
access_token = None  # Global token
```

Fixture function to generate token (called once once per test session)

```
@pytest.fixture(scope="session", autouse=True)
def generate_token():
  global access_token
  client_id = "8d20043bc76d420987e4959b3b1f55d3"
  client_secret = "4ee22d9c8ca844a8a6373bf584b1956f"
  token_url = "https://accounts.spotify.com/api/token"
  headers = {
     "Content-Type": "application/x-www-form-urlencoded"
  }
  form_data = {
     "grant_type": "client_credentials",
     "client_id": client_id,
     "client_secret": client_secret
  }
  response = requests.post(token_url, data=form_data, headers=headers)
  assert response.status_code == 200, f"Token request failed: {response.text}"
  access_token = response.json()["access_token"]
  print("Generated Token:", access_token)
class TestOAuth2SpotifyAPI:
```

Test 1: Fetch specific artist details using token

```
  def test_get_artist_details(self):
    assert access_token, "Token not generated"
    headers = {
        "Authorization": f"Bearer {access_token}"
    }
    api_url = "https://api.spotify.com/v1/artists/2NoJ7NuNs9nyj8Thoh1kbu"
    response = requests.get(api_url, headers=headers)
    assert response.status_code == 200, f"API call failed: {response.text}"
    data = response.json()
    print("Artist name:", data.get("name"))
```

<u>**Test 2: Get Arijit Singh's top tracks**</u>

```
def test_get_arijit_singh_top_tracks(self):
    assert access_token, "Token not generated"
    headers = {
        "Authorization": f"Bearer {access_token}"
    }
```

<u>**Step 1: Search for Arijit Singh**</u>

```
    search_url = "https://api.spotify.com/v1/search"
    params = {
        "q": "Arijit Singh",
        "type": "artist",
        "limit": 1
    }
    response = requests.get(search_url, headers=headers, params=params)
    assert response.status_code == 200, f"Search failed: {response.text}"
    artist_data = response.json()
    artist_items = artist_data.get("artists", {}).get("items", [])
    assert artist_items, "Artist not found."
    artist_id = artist_items[0]["id"]
    print("Arijit Singh Artist ID:", artist_id)
```

<u>**Step 2: Get top tracks for the artist (India market)**</u>

```
    top_tracks_url = f"https://api.spotify.com/v1/artists/{artist_id}/top-tracks"
    response = requests.get(top_tracks_url, headers=headers, params={"market": "IN"})
    assert response.status_code == 200, f"Top tracks fetch failed: {response.text}"
    tracks = response.json().get("tracks", [])
    print(f"Top {len(tracks)} Songs by Arijit Singh:")
    for i, track in enumerate(tracks, start=1):
        print(f"{i}. {track['name']}")
```

<u>**Note**</u>

1. Python's requests library does not have formParam like RestAssured. But we can use the data parameter in requests.post() to send form data. This sends data as application/x-www-form-urlencoded (like HTML forms).

2. If we already have a token
    a. Use the token in the **Authorization header** (Bearer {token}).
3. For more advanced OAuth flows (e.g., OAuth 1.0a or OAuth 2.0 token generation) use **requests-oauthlib** instead of **requests** .It handles the token exchange and authorization processes automatically
    a. pip install requests-oauthlib

Fixture function to generate token using requests-oauthlib (called once once per test session)

```
@pytest.fixture(scope="session", autouse=True)
def generate_token():
  global access_token
  client_id = "8d20043bc76d420987e4959b3b1f55d3"
  client_secret = "4ee22d9c8ca844a8a6373bf584b1956f"
  token_url = "https://accounts.spotify.com/api/token"
                                    Step 1: Create a client for the Client Credentials Flow
  client = BackendApplicationClient(client_id=client_id)
                                    Step 2: Create OAuth2 session using the client
  oauth = OAuth2Session(client=client)
                                    Step 3: Fetch the token using client credentials
  token = oauth.fetch_token(
     token_url=token_url,
     client_id=client_id,
     client_secret=client_secret
  )
                                    Store token globally
  access_token = token.get("access_token")
  assert access_token is not None, "Failed to retrieve access token"
  print("Generated token:", access_token)
```