5. File Upload & Download, Handling Response Cookies & Headers

File Upload and Download

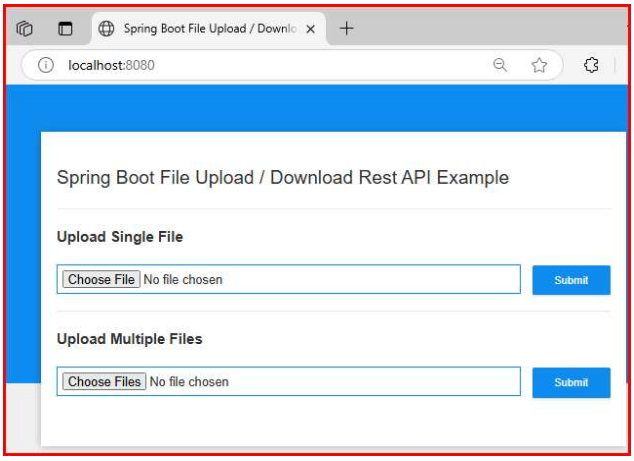
- ➔ When we upload a file (single or multiple) through a web UI, an API is triggered in the background.
- ➔ This API receives the file from the client and sends the request to the server. Then, the uploaded files are stored on the server or in the database.
- ➔ When we download a file, your browser asks the server for it. If the file is ready, the server says **"OK"** and sends the file. It also sends a **Content-Disposition message** that tells the browser, "This is a file to save," so the browser opens a save dialog or saves it automatically

Note

- ➔ It's hard to find public APIs for file uploads (single or multiple), so we will create our own API and test it using the python requests library.

Workout in Pycharm

- ➔ Create a Directory / Package. Keep the jar file in the same location after downloading from the link
  - ◆ https://github.com/Madhan-091296/spring-boot-file-upload-download-rest-api-master/blob/master/file-upload-RestAPI.jar
- ➔ Navigate to location in pycharm terminal and run below command.
  - ◆ **java -jar file-upload-RestAPI.jar**
- ➔ Open below URL on browser then check working fine.
  - ◆ http://localhost:8080/



Note

- ➔ **netstat -aon | findstr :8080** → To check if port 8080 is in use.
- ➔ **taskkill /PID <PID> /F** → To kill the process running on that port.
- ➔ **java -jar file-upload-RestAPI.jar** --server.port=8081 → To run the jar file on port 8081.

test_file_upload_download.py

```
import requests
import os
class TestFileUploadAndDownload:
  BASE_URL = "http://localhost:8080"
  FILE1 = "C:/Automation/automationFiles/Test1.txt"
  FILE2 = "C:/Automation/automationFiles/Test2.txt"
```

upload single file

```
def test_upload_single_file(self):
    with open(self.FILE1, 'rb') as file:
        files = {'file': file}
        response = requests.post(f"{self.BASE_URL}/uploadFile", files=files)
    assert response.status_code == 200, "Upload failed"
    response_json = response.json()
    assert response_json["fileName"] == "Test1.txt", "Wrong filename"
    print("Single File Upload Response:", response_json)
```

upload multiple files

```
def test_upload_multiple_files(self):
    with open(self.FILE1, 'rb') as f1, open(self.FILE2, 'rb') as f2:
        files = [('files', f1), ('files', f2)]
        response = requests.post(f"{self.BASE_URL}/uploadMultipleFiles", files=files)

    assert response.status_code == 200, "Multi upload failed"
    response_json = response.json()
    assert response_json[0]["fileName"] == "Test1.txt", "File1 mismatch"
    assert response_json[1]["fileName"] == "Test2.txt", "File2 mismatch"
    print("Multiple Files Upload Response:", response_json)
```

Download file

```
def test_download_file(self):
    filename = "Test1.txt"
    response = requests.get(f"{self.BASE_URL}/downloadFile/{filename}")
    assert response.status_code == 200, "Download failed"
    output_path = f"downloaded_{filename}"
    with open(output_path, 'wb') as f:
        f.write(response.content)
    assert os.path.exists(output_path), "File not saved"
    print(f"Downloaded: {output_path}")
```

Note

- ➔ When uploading a file using requests, open it in 'rb' (read binary) mode, not 'r'.This is because requests expects the file in bytes, not text.
- ➔ Using 'r' gives a string, which can cause upload errors or corrupt the file — especially for images, PDFs, or files with special characters.
- ➔ In requests, we don't need to set **{"Content-Type": "multipart/form-data"}** manually as like in RestAssured.It is set automatically when we use the files parameter.
- ➔ But if needed, we can set it explicitly as below:
  - ◆ **headers = {"Content-Type": "multipart/form-data"}**
  - ◆ **response = requests.post(url, files=files, headers=headers)**

Assignment - File Upload & Download

1. **File Upload**

a. Any type of file.

b. POST: https://the-internet.herokuapp.com/upload

2. **File Download**

a. GET : https://the-internet.herokuapp.com/download/myfile.txt

test_Cookies.py

```
import requests
class TestCookies:
  def test_cookies_in_response(self):
    url = "https://www.google.com/"
    response = requests.get(url)
    assert response.status_code == 200, "Status not 200"
                                Assert cookie "AEC" is present and not None
    assert "AEC" in response.cookies, "AEC not found"
    assert response.cookies.get("AEC") is not None, "AEC is None"
                                Extract a specific cookie
    cookie_value = response.cookies.get("AEC")
    print("Value of 'AEC' Cookie:", cookie_value)
                                Extract all cookies
    all_cookies = response.cookies.get_dict()
    print("All cookies:", all_cookies)
                                Printing cookies and their values using for loop
    for key, value in all_cookies.items():
      print(f"{key}: {value}")
    set_cookie_headers = response.headers.get('Set-Cookie')
    print("Set-Cookie:", set_cookie_headers)
```

Note

➔ Requests library does not provide detailed cookie info like expiry, secure etc. easily But we can access raw **Set-Cookie header(s)** if needed

test_Headers.py

```
import requests
class TestHeaders:
  def test_headers_in_response(self):
    url = "https://www.google.com/"
    response = requests.get(url)
                                Extract and print all the headers
    print("\nResponse Headers:")
    for key, value in response.headers.items():
      print(f"{key} ==> {value}")
                                Assertions
    assert response.status_code == 200, "Wrong status"
    assert "text/html" in response.headers.get("Content-Type", ""), "Wrong Content-Type"
```

```
assert response.headers.get("Content-Encoding") is not None, "Missing Content-Encoding"
assert response.headers.get("Content-Encoding") == "gzip", "Wrong Content-Encoding"
assert response.headers.get("X-Frame-Options") == "SAMEORIGIN", "Wrong X-Frame-Options"
assert response.headers.get("Server") == "gws", "Wrong Server"
```

Extract and Print specific header

```
header_value = response.headers.get("Content-Type")
print("\nValue of header (Content-Type):", header_value)
```