Day 5

📘 Report: Virtual Environment, Git Workflow, Selenium & API Testing Sessions

Session 1: Virtual Environment, Git Workflow, and Selenium Automation

This session focused on foundational development practices, including setting up Python virtual environments, working with Git, and introducing Selenium for web automation.

🔹 1. Virtual Environment Setup

The importance of virtual environments was emphasized to ensure dependency isolation across different Python projects.

- **Creation and Activation**
    - Created a virtual environment using:
    - python -m venv myenv
    - Activated the environment on Windows with:
    - myenv\Scripts\activate
- **Dependency Management**
    - Demonstrated how to install and manage libraries within the activated virtual environment.

🔹 2. Git Workflow Basics

Participants were introduced to core Git commands and their significance in project version control.

- **Commands Covered**
    - git init, git add, git commit, git push, and git pull
- **GitHub Integration**
    - Showed how to push a local project to GitHub and manage remote repositories for collaboration and backup.

🔹 3. Selenium Automation with Python

The session introduced browser automation using Selenium.

- **Installation**
    - Installed Selenium and Chrome WebDriver to enable browser-based testing.
- **Basic Automation Script**
    - Demonstrated a sample script:
    - from selenium import webdriver
    - driver = webdriver.Chrome()
    - driver.get("https://qxf2.com/selenium-tutorial-main")

- name = driver.find_element(by="id", value="name")
- name.send_keys("Qxf2")

- **Element Interaction**
  - Explained how to locate and interact with web elements using find_element and perform actions like entering text.

Session 2: API Testing with Postman and Python Requests

This session focused on understanding and implementing API testing using both **Postman** and the **Python requests module**.

🔹 **1. API Testing with Postman**

Participants explored the fundamentals of API testing through the Postman interface.

- **GET and POST Requests**
  - Demonstrated how to send requests, inspect responses, and validate data returned by APIs.

🔹 **2. API Testing Using Python Requests Module**

- **Making API Calls**
  - Executed API requests to a locally hosted server (http://127.0.0.1:5000/).
- **Sample GET Request**
- import requests
- response = requests.get("http://127.0.0.1:5000/cars", auth=("qxf2", "qxf2"))
- print(response.status_code)
- print(response.json())
- **Sample POST Request**
  - Sent data to the server to add a new car:
- response = my_session.post(url=base_url + 'cars/add', json={
- 'name': 'Gwagon',
- 'brand': 'Gwagon',
- 'price_range': '90-200lacs',
- 'car_type': 'sedan'
- }, auth=(username, password))
- **Response Validation with Assertions**
  - Validated the success of API calls using:

- assert response.status_code == 201, f"Expected status 201, got {response.status_code}"
- assert 'message' in response_content and 'successfully' in response_content['message'].lower(), "Car addition failed"
- Verified that the car list was updated post submission.

✅ Conclusion

The sessions provided a solid foundation in automation and API testing. Key takeaways included:

- Setting up and managing virtual environments for Python projects.
- Using Git and GitHub for effective version control.
- Automating browser tasks with Selenium.
- Performing robust API testing using both Postman and Python's requests library.

Participants engaged in hands-on exercises, reinforcing concepts through practical examples and real-time API validations using assertions.