**Web Automation and Testing with Selenium**

**Objectives:**

1. Gain proficiency in using Selenium for web automation and testing.
2. Learn to write Selenium test scripts to verify the functionality and reliability of web applications.
3. Understand how to interact with web elements, navigate through web pages, and perform various actions using Selenium.
4. Develop test scenarios for different routes and forms in a web application.
5. Use Selenium tests as a quality assurance tool to identify and prevent issues in web applications.

**Prerequisites:**

1. Basic knowledge of Python programming.
2. Familiarity with web technologies, HTML, and CSS.
3. Understanding of how to install Python libraries using pip.
4. Installation of the Selenium library (pip install selenium).
5. Familiarity with the basics of Flask web applications.

**Introduction:**

Testing is an essential aspect of web development. It ensures that web applications work as intended and helps prevent regressions. In this lab, you will explore web automation and testing with Selenium, a popular tool for automating browser interactions. You will work with a Flask web application and create Selenium test scripts to validate its functionality.

**What is Flask?**

Flask is a lightweight Python web framework that simplifies web application development. It allows you to build web applications quickly and efficiently. With Flask, you can define routes, create dynamic web pages, and handle HTTP requests easily.

A **route** is a URL pattern that defines how a web application should respond to specific HTTP requests. Routes are an essential part of defining the behavior of your web application. Each route is associated with a Python function that gets executed when a matching URL is requested by a client (e.g., a web browser).

**Creating a Basic Flask App**

1. **Install Flask**: First, make sure you have Flask installed. In your computers terminal or command prompt enter:

pip install Flask

1. **Create a Python File**: Create a Python file (e.g., app.py) in your project directory.

**3. Writing a Flask app:**

from flask import Flask #

This line creates an instance of the Flask application. The \_\_name\_\_ parameter is used to determine the root path for your application

app = Flask(\_\_name\_\_)

This line is a route decorator. It tells Flask to associate the function that follows it with the root URL ("/"). In this case, the hello function is associated with the root URL, so when you visit the root URL in your web browser, it will execute the hello function.

@app.route('/')

def hello():

return "Hello, Flask!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

**4: Run Flask app:**

Ensure that your Flask application is running.In the address bar of your web browser, type the following URL: <http://localhost:5000/> and you should see the Hello, Flask! Message appear in your browser.

A screenshot of a computer

Description automatically generated

**Assignment Setup:**

1. Software Provided: Download and unzip the code base to your respective IDE. It contains 2 python files, 1 for setting up a web server and creating a basic web application on your local machine, and a second program that uses Selenium to run tests on the web application, and 2 HTML files that should be stored in a ‘templates’ folder.
2. Install the required dependencies by navigating to the project directory in the terminal and entering: pip install -r requirements.txt

**Instructions:**

Then, import the necessary libraries at the beginning of your Python script:

from selenium import webdriver

**Step 1: Run web\_app.py**

**This code creates a basic web application using Flask and starts the Flask development server.**

A screen shot of a computer

Description automatically generated

**Step 2: Set Up the Selenium Web Driver**

Choose a web driver compatible with your preferred web browser (e.g., ChromeDriver, Firefox GeckoDriver). Download the appropriate driver in a location that is accessible to your script. Then, create a web driver instance:

python

# Specify the path to your web driver

driver = webdriver.Chrome(executable\_path='/path/to/chromedriver')

Replace webdriver.Chrome with webdriver.Firefox or the appropriate driver class if you're using a different browser.

**Step 3: Navigate to Different Routes**

A **route** is a URL pattern that defines how a web application should respond to specific HTTP requests. It specifies which function or view should be executed when a particular URL is accessed.

You can use the ‘**get’** method of the web driver to navigate to different routes in your Flask application:

A screenshot of a computer program

Description automatically generated

# Navigate to the search page

driver.get('http://localhost:5000/search')

# Navigate to the form page

driver.get('http://localhost:5000/form')

Make sure to adjust the URLs to match the actual address of your Flask application.

**Step 4: Interact with Web Elements**

A **web element** is an individual component or part of a web page that can be identified and interacted with using web automation tools and libraries like Selenium. Web elements can include things like buttons, text fields, links, checkboxes, radio buttons, dropdown menus, images, and any other visible or interactive element on a web page.

If you need to interact with web elements (e.g., filling out a form or clicking a button), you can use Selenium's methods.

In the provided code, there is a form.html file. It has a form, label, input, and button elements. Notice that each element has a name attribute. The name attribute can be used for targeting and scripting.

For example, to fill out a form and submit it:

python

# Find an input element by its name attribute and fill it with data

data\_input = driver.find\_element\_by\_name('data')

data\_input.send\_keys('Selenium Test Data')

# Find a submit button and click it

submit\_button = driver.find\_element\_by\_css\_selector('button[type="submit"]')

submit\_button.click()

In this example, we find an input element by its name attribute and send keys to it. Then, we locate a submit button by its CSS selector and click it.

**Step 5: Perform Assertions and Verifications**

You can use Selenium's capabilities to verify that the correct page is displayed after navigation or that certain elements are present. For example, you can use assertions to check the page title:

python

# Verify that the page title is as expected

assert 'Selenium Tutorial' in driver.title

**Step 6: Close the Web Driver**

After you have completed your interactions and verifications, it's essential to close the web driver to release resources:

python

driver.quit()

Remember to run this script after you've started your Flask application to ensure that the application is accessible. Make sure that your Flask application is running and reachable at the specified URLs before running the Selenium script. Additionally, you may need to adjust your script to match the specific structure and elements of your application.