Write-Up

Git-Hub Link:https://github.com/olomansaas/CapStoneProject_Final.git

#1 Rest-Assured:

- **Retrieve the list of all products in the store.**

1. HTTP GET request to the specified endpoint ("/get-products") on the base URI "http://localhost:9010".

2. It validates that the response has a status code of 200 and logs the details of the request and response.

- **Retrieve the list of all registered users.**

1. GET request to the "http://localhost:9010/get-users" endpoint, expects a response with a status code of 20, and logs all the details of the response.

2. It's likely part of an automated testing framework to ensure that the "/get-users" endpoint of the specified API.

- **Add the product.**

1. Use RestAssured to send an HTTP POST request to a specified endpoint ("/add-product") with a set of parameters representing a product.

2. The response is then validated to ensure it returns a status code of 200, and the details of the response are logged.

- **Delete the product.**

1. An HTTP DELETE request using RestAssured to delete a product with ID 101 from the API endpoint "http://localhost:9010/delete-product".

2. The code then asserts that the response status code should be 200 and logs the details of the request and response.

- **Update the product.**

1. RestAssured to make a PUT request to a REST API endpoint. It sets the base URI, base path, content type, request body (using the previously populated HashMap).

2. a response with a status code of 200. The log().all() is used to log the details of the response.

- **Update the product status.**

1. Use RestAssured to send a PUT request to update the status of a product, providing product information in the request body using a HashMap

2. The code assumes a successful update with a response status code of 200.

Conclusion:

A set of TestNG test methods for performing REST API testing using the RestAssured library. The tests cover various CRUD (Create, Read, Update, Delete) operations on a fictional product and user management system. The code demonstrates how to make HTTP requests, handle request parameters and bodies, and validate the response status codes. The test methods include scenarios for retrieving products and users, adding a new product, updating product information, updating product status, and deleting a product. The code serves as a foundation for testing the functionalities of a RESTful web service.

#2 Selenium Scripts using TestNG:

We have created this project using the Page Object Model. In this Project we have created 3 different java class in order to implement the criteria.

1. TestBase.java :

In this class we have defined the method to open the chrome Browser with the defined "Localhost:9010" website.

2. HomePage.java:

A. In this class we have defined the page object model and give the locator path in order to reach the following elements of the provided website.

B. This class encapsulates the web elements and actions related to a particular web page, promoting a modular and maintainable test code structure. The code appears to be interacting with a web page related to user authentication, cart management, and product search/filtering.

C. Elements Linked :

WebElement Declarations:

WebElement instances are declared for various elements on a web page using the @FindBy annotation. These elements include email, password, login button, add to cart button, home link, cart link, place order link, search input, filter button, filter option, and search button.

Constructor:

There is a constructor HomePage that takes a WebDriver parameter and initializes the web elements using PageFactory.initElements(driver, this).

Action Methods:

There are several methods representing actions that can be performed on the web page. These include entering email, entering a password, clicking on the login button, adding an item to the cart, navigating to the home page, navigating to the cart, navigating to the place order page, clicking on the filter button, selecting a filter option, entering a search query, and clicking on the search button.

Action Method Implementations:

Each action method is implemented using the corresponding WebElement and WebDriver methods. For example, the EnterEmail method clicks on the email input field and sends the keys "g***a.com," and the clickOnLogin method clicks on the login button.

Comments:

There are some comments within the code (e.g., "// Actions") that provide a brief description of the section they precede.

Print Statements:

There are print statements within the ClickOnFilter and ClickOnFilter_option methods, which can be helpful for debugging.

3. TestAll_Pages.java:

(Having The Following Pages)

- **Login page/Registration page:**

1. We have Registered the user and create a login method in this class .
2. Email and Password is entered in systematic order and then login button is pressed in order to enter the HomePage.
3. By the end of this method we have successfully entered on the Home Page.

- **Add product to cart page.**

1. A product Having id “Cart-102”has been added to the cart.
2. The page is then locatd to the Add to cart page in order to show its functionality.
3. By the end of this method we have returned to the home page .

- **Place order page**

1. We have moved to the cart page where we have the Place order butoon.
2. Then we have placethe order of the selected item .
3. Again we have redirected to the home page.

- **Search the product.**

1. We have clicked on the search Boiler plate and added the item ”Hamdard Safi Natural Blood Purifier Syrup”.
2. Then we pressed the search button and Then we redirect to the home page

- **Filter the product**

1. At last we have clicked on the filter butoon on the Home Page.
2. And we have provided the filter “Low To High”.

- **Close the Tab**

1.Finally we have provided the quit tab cmd to close the browser.

- **Additionally:**

Each test method includes explicit waits (Thread.sleep()) and scrolling actions using JavaScriptExecutor.

Conclusion:

In conclusion, this set of Selenium scripts uses TestNG to automate testing for the specified pages of a web application, covering login, registration, adding products to the cart, placing orders, searching for products, and filtering products. The test methods include actions and assertions relevant to each page's functionality.

We have to Regularly update the locators and actions as our application evolves.

#3 JMeter Scripts:

- **Objective:**

1. The purpose of this document is to outline the creation and usage of a performance testing module for a HealthCare web portal using JMeter and Also Perform Load testing.

System Requirements:

1.JMeter 5.1.1
2.Java Development Kit (JDK) Version 8 or higher
3.Access to the HealthCare web portal under test.

- **Methodology Followed:**

Test Plan Creation:

1.Open JMeter and create a new Test Plan.
2.Add a Thread Group to the Test Plan.

- **HTTP Requests:**

Add HTTP Request samplers for each function or endpoint you want to test. Configure the requests with appropriate parameters and headers.Consider using parameterization for dynamic values.

- **Config Elements**:

Various config elements are also used including:
1.HTTP Header manager
2.User Defined Variables
3.HTTP Request Defaults
And More……..

- **Listeners:**

Add Listeners to collect and analyze the test results.
Common listeners include in this project :

View Result Tree:
A JMeter listener that displays detailed information about each sample, including request and response data, allowing in-depth inspection of individual HTTP transactions.

Summary Report:
A JMeter listener that provides a summary of key performance metrics such as average response time, throughput, and error rates for all executed samplers in the test.

Aggregate Report:
A JMeter listener that presents aggregate performance statistics, including average response time, throughput, and error rates, for each sampler in a tabular format.

Graph Result:
A JMeter listener that visualizes performance metrics over time, allowing users to analyze trends and patterns in response times, throughput, and other key indicators through graphical representations.

Assertion Result:
A JMeter listener that reports the results of assertions applied during a test, indicating whether each assertion passed or failed for individual samples, helping identify deviations from expected behavior.

Conclusion:

The performance testing module provides a comprehensive framework for assessing the performance of a web portal using JMeter 5.1.1 and JDK 8. Regular execution of performance tests and analysis of results will ensure the continuous improvement of the web portal's performance.

#4 Cucumber-Gerkins Keyword For Api Testing:

1. Feature File:

The "Medicare Application Capstone Project" aims to provide a comprehensive and efficient API testing strategy for the Medicare application. This feature file outlines various scenarios to test different functionalities of the application's API.

Things That Our Feature File Will Cover:

1.Retrieve the list of all products in the store:

2.Retrieve the list of all Registered users:

3.Add the product:

4.Update the product:

5.Update the product status:

6.Delete the product:

Further-Eleboration:

This feature file outlines a series of scenarios for testing various functionalities of the Medicare application's API. Each scenario covers a specific action, such as retrieving product lists, registered users, adding products, updating product details and status, and deleting products. The purpose is to ensure the reliability and correctness of the API responses, thereby contributing to the overall quality and robustness of the Medicare application .

Cucumber-Step Defination:

- **Retrieve the list of all products in the store.**

1.HTTP GET request to the specified endpoint ("/get-products") on the base URI "http://localhost:9010".

2.It validates that the response has a status code of 200 and logs the details of the request and response.

- **Retrieve the list of all registered users.**

1.GET request to the "http://localhost:9010/get-users" endpoint, expects a response with a status code of 20, and logs all the details of the response.

2. It's likely part of an automated testing framework to ensure that the "/get-users" endpoint of the specified API.

- **Add the product.**

1.Use RestAssured to send an HTTP POST request to a specified endpoint ("/add-product") with a set of parameters representing a product.

2. The response is then validated to ensure it returns a status code of 200, and the details of the response are logged.

- **Delete the product.**

1.An HTTP DELETE request using RestAssured to delete a product with ID 101 from the API endpoint "http://localhost:9010/delete-product".

2. The code then asserts that the response status code should be 200 and logs the details of the request and response.

- **Update the product.**

1.Cucumber to make a PUT request to a API endpoint. It sets the base URI, base path, content type, request body (using the previously populated HashMap).

2. A response with a status code of 200. The log().all() is used to log the details of the response.

- **Update the product status.**

1.Use Cucumber to send a PUT request to update the status of a product, providing product information in the request body using a HashMap

2. The code assumes a successful update with a response status code of 200.

Conclusion:

In conclusion, the Java class "StepDefinationFile" contains Cucumber step definitions for testing a RESTful API that manages products. The steps include setting the base URI and path, executing GET, POST, PUT, and DELETE requests with various parameters, and validating the response status codes. The testing framework utilizes RestAssured for making HTTP requests and assertions. The scenarios cover basic CRUD operations, including adding, updating, deleting products, and updating the status code of a product. The code demonstrates a simple API testing framework using Cucumber and RestAssured.

#5 Postman-scripts to test:

Postman scripts to test the following API endpoints With Detailed Description:

RetrieveAll_Products:

Method: GET

URL: {{BaseUrl}}/get-products

Tests:

Ensure the status code is 200.

Check if the response body includes the code "101."

RetrieveAll_Users:

Method: GET

URL: {{BaseUrl}}/get-users

Test: Ensure the status code is 200.

AddThe_Product:

Method: POST

URL: {{BaseUrl}}/add-product

Body: JSON payload for adding a product.

UpdateThe_Product:

Method: PUT

URL: {{BaseUrl}}/update-product

Tests:

Ensure the status code is 200.

Check if the response body includes "Disprin+" and "101."

UpdateThe_ProductStatus:

Method: PUT

URL: {{BaseUrl}}/update-product-status

Tests:

Ensure the status code is 200.

Check if the response body includes "Disprin+ Status Updated Successfully."

DeleteThe_Product:

Method: DELETE

URL: {{BaseUrl}}/delete-product?id=101

Tests:

Ensure the status code is 200.

Check if the response body includes "Product with ID 101 Deleted Successfully."

Detailed Explanation:

1.RetrieveAll_Products: Tests the retrieval of products, checking both the status code and the presence of the code "101" in the response.

2. RetrieveAll_Users: Tests the retrieval of users, checking the status code.

3. AddThe_Product: Sends a POST request to add a product.

4. UpdateThe_Product: Tests the update of a product, checking the status code and the presence of "Disprin+" and "101" in the response.

5. UpdateThe_ProductStatus: Tests the update of a product's status, checking the status code and the presence of "Disprin+ Status Updated Successfully" in the response.

6. DeleteThe_Product: Tests the deletion of a product, checking the status code and the presence of "Product with ID 101 Deleted Successfully" in the response.

Conclusion:

This Postman project primarily focuses on testing various endpoints related to product and user management using different HTTP methods. The provided tests and scripts serve as a starting point and can be expanded based on the specific needs of the project.

==========**END**==========