**Writeup**

*Github:* ***https://github.com/ramyamogare/Capstoneproject***

**Selenium using TestNG**

**Project Structure and Components:**

**Page Object Model (POM):** The code follows the Page Object Model, a design pattern that organizes the codebase into separate classes, each representing a page of the application. This separation ensures maintainability and readability.

**BaseTest Class:** This class is responsible for setting up the WebDriver instance, launching the browser, maximizing the window, and navigating to the application's URL. It also contains the @AfterTest method to close the browser after the tests are executed.

**TestPage Class:** This class contains the main test method. It uses page objects to simulate user actions and interactions.

**Page Object Classes:** These classes represent different pages or components of the application. They contain WebElement references and methods to interact with those elements.

**Test Execution Flow:**

**Opening the Application:** The test begins by launching the browser, maximizing the window, and navigating to the application's URL using the BaseTest class.

**User Registration:** The test interacts with the RegisterPage page object to register a new user. This includes entering a name, email, and password, and then clicking the register button.

**User Login:** The test interacts with the LoginPage page object to log in using the registered email and password.

**Adding to Cart:** The test interacts with the AddToCartPage page object to add a product to the cart.

Verifying Success Message: After adding the product to the cart, the test verifies that the success message is displayed using assertions. It also clicks the "Home" link to proceed.

**Navigating to Cart and Placing Order:** The test navigates to the cart using the CartPage page object and clicks the "Place Order" button.

**Verifying Success Message Again:** Once the order is placed, the test verifies the success message again using assertions.

**Benefits and Advantages:**

Readability and Maintainability: The Page Object Model (POM) enhances code organization and readability. Page objects encapsulate the elements and methods related to each page, making maintenance and updates easier.

Modularity: By breaking down the test into smaller steps, the code becomes modular. Each page object represents a distinct component, promoting reusability across different tests.

User Simulation: The code simulates user actions like registration, login, and placing an order. This ensures that the application behaves as expected from the user's perspective.

Automated Verification: Assertions are used to automatically verify the correctness of certain interactions, such as success messages, reducing the need for manual verification.

Regression Testing: The automated tests can be rerun whenever changes are made to the application, ensuring that new developments do not break existing functionality.

Efficiency: Automated tests execute consistently and quickly, saving time compared to manual testing.

In conclusion, this Selenium code showcases how automation can streamline the testing process for buying sporty shoes through a web application. The Page Object Model enhances maintainability, while the test execution flow mirrors real user behavior, providing a reliable and efficient testing approach.

**Algorithm for Selenium using TestNG project**

Creating Selenium Scripts using TestNG:

1.Import necessary libraries for Selenium and TestNG.

2.Set up WebDriver (Chrome/Firefox) for browser automation.

3.Create separate TestNG test methods for the following pages:

Registration Page: Open the registration page URL. Automate entering user details and submitting the registration form.

Login Page: Open the login page URL. Automate entering valid/invalid credentials and clicking the login button.

Add To Cart Page: Open the product page URL. Automate selecting a product and adding it to the cart.

Cart Page: Open the cart/checkout page URL. Automate selecting an item and placing an order.

**Cucumber**

Automated testing is a crucial part of software quality assurance, and Cucumber, a Behaviour-Driven Development (BDD) testing framework, simplifies the process by allowing tests to be written in plain language. Here we'll dive into the details of the Cucumber code designed to automate the process of purchasing sporty shoes through a web application.

**Feature File:**

The foundation of the Cucumber test suite lies in the feature file. It's a text-based file that outlines the behaviour you want to test. In the context of buying sporty shoes, the feature file describes the steps users take to achieve their goal. Each scenario in the feature file consists of a sequence of steps that correspond to the actions a user would perform.

**Step Definitions:**

The Gherkin steps described in the feature file are mapped to Java methods in the step definition classes. These step definition classes contain the logic that interacts with the application, mirroring the steps described in the feature file.

**Setup Steps:** In the "Given" steps, the code sets up the initial state. For instance, the "User is on the Landing page" step is implemented by clicking on the "New User" button.

**Action Steps:** The "When" steps represent user actions like registration, login, adding to the cart, and placing an order. The corresponding Java methods interact with the application by entering information, clicking buttons, and navigating pages.

**Verification Steps:** The "Then" steps verify the expected outcomes of the actions. The code uses assertions to confirm that the expected messages or results match the actual outcomes on the application.

**Test Execution Flow:**

The test starts by navigating to the landing page and clicking the "New User" button.

The test then registers a new user by entering their details and clicking the register button. It simulates a user's action.

After registering, the test logs in using the registered email and password.

The user adds a product to the cart, mirroring the action a real user would take.

The test verifies that a success message is displayed after adding the product to the cart. Assertions ensure the correctness of this message.

The user proceeds to the cart, places an order, and again verifies the success message.

**Benefits and Advantages:**

Collaboration and Communication: Cucumber's plain language syntax fosters collaboration among teams by allowing business stakeholders, testers, and developers to understand and contribute to the testing process.

Test Coverage and Reliability: The scenarios outlined in the feature file cover various aspects of the application's functionality, ensuring comprehensive test coverage. This promotes a reliable application.

Readability: The feature file serves as living documentation that clearly articulates the application's expected behaviour.

Efficiency: Automation eliminates manual repetitive testing, saving time and effort.

Early Bug Detection: Automated tests catch issues early in the development cycle, preventing them from propagating further.

Regression Testing: Automated tests can be run repeatedly to ensure that new changes do not negatively impact existing functionality.

In summary, the Cucumber code provides an efficient and user-oriented approach to testing the process of buying sporty shoes. Through clear communication, effective automation, and comprehensive coverage, the code contributes to delivering a reliable and user-friendly web application.

**RestAssured**

Automated testing of APIs plays a crucial role in ensuring the reliability and functionality of web applications. RestAssured, a Java-based library, simplifies the process of testing RESTful APIs by providing an intuitive and expressive way to create and execute API requests. Here we'll delve into the details of the RestAssured code designed to automate the process of adding a product through a RESTful API.

**Code Structure and Components:**

Test Execution Order: The code is structured as a TestNG test class, which allows test methods to be executed in a specific order based on their priorities.

Test Method: The AddTheProduct() method is annotated with @Test and given a priority of 2, indicating its execution order.

API Request: The RestAssured library is used to send an HTTP POST request to the specified API endpoint. This request involves adding a product with specific details, such as ID, image URL, name, category, sizes, and price.

Assertions: After sending the request, assertions are used to validate the response. In this case, the expected status code is 201 (Created), indicating that the product was successfully added.

**Test Execution Flow:**

The test method begins by specifying the content type as JSON, indicating that the request payload and response are in JSON format.

A POST request is sent to the provided API endpoint using the .given().contentType("application/json") syntax. The request includes product details such as ID, image URL, name, category, sizes, and price.

The .when() section initiates the request by invoking the POST method.

The .then() section verifies the response. In this case, assertions are used to confirm that the response status code is 201 (Created). A log of the response is generated using .log().all().

The log provides detailed information about the request and response, aiding in debugging and troubleshooting.

**Algorithm**

**Automating API Endpoints using Rest-Assured:**

1. Create a maven repository in eclipse

2. Add the necessary dependencies in Pom.xml file

3. Crate 3 class file

1.Get1Demo

2. Get2Demo

3. Post demo

4. Write a java code for rest assured

5. Run the class file and get the outputs

6. Retrieve the list of all products using a GET request to /get-shoes endpoint.

7. Retrieve the list of all registered users using a GET request to /get-users endpoint.

8. Add a product using a POST request to /add-shoe endpoint with required parameters (id, image, name, category, sizes, price).

**JMeter**

JMeter is a powerful tool for load testing that allows you to simulate heavy user traffic on a web application to assess its performance and reliability. Here's how to create JMeter scripts for load testing:

1. **Download and Install JMeter:** If you haven't already, download and install Apache JMeter from the official website. JMeter is a Java-based application, so ensure you have Java installed as well.

2. **Create a New JMeter Test Plan:** Launch JMeter and start a new test plan. The test plan acts as a container for your load testing script. Right-click on "Test Plan" in the left panel and choose "Add > Thread Group."

3. **Add a Thread Group:** Thread Groups simulate virtual users or threads that perform actions concurrently. In the Thread Group settings, define the number of users (threads), the ramp-up period (how quickly users are added), and the loop count (how many times each user repeats the scenario).

4. **Configure Thread Group:** Under the Thread Group settings, you can specify the desired number of users, ramp-up time, and loop count based on your load testing goals.

5. **Add HTTP Request Samplers:** HTTP Request Samplers simulate user actions on the web application. Right-click on your Thread Group, choose "Add > Sampler > HTTP Request," and configure it by providing the URL of the homepage (e.g., <http://localhost:9010/>) and the product detail page (e.g., <http://localhost:9010/product-details>).

6. **Add Listeners for Analysis:** Listeners collect and display the load test results. Right-click on your Thread Group, choose "Add > Listener," and select appropriate listeners like "Summary Report" and "View Results Tree." These listeners show statistics and detailed response data.

7. **Run the Load Test:** Click the green "Start" button to run the load test. JMeter will simulate the specified number of users, and you can monitor real-time results in the listeners.

8. **Analyse Results:** Once the test is complete, review the results in the listeners. The "Summary Report" provides an overview of response times, throughput, and error rates. The "View Results Tree" shows individual requests and responses.

**Benefits and Advantages:**

Performance Assessment: Load testing helps identify performance bottlenecks, slow response times, and potential crashes under heavy traffic conditions.

Scalability Testing: By configuring different user loads, you can assess how well your application scales to handle increased traffic.

Realistic Simulation: JMeter simulates real user behaviour, allowing you to mimic various actions and interactions on your web application.

Early Issue Detection: Load testing uncovers issues before they impact actual users, enabling proactive optimization.

Data-Driven Decisions: Results help you make informed decisions about server resources, infrastructure, and potential optimizations.

Creating JMeter scripts for load testing involves setting up a test plan with a Thread Group, adding HTTP Request Samplers, and using listeners to analyse the test results. This process enables you to gauge your application's performance, identify areas for improvement, and ensure a seamless user experience even during peak loads.

**Algorithm**

**JMeter Scripts for Load Testing:**

1.Download and install JMeter if not already done.

2.Create a new JMeter Test Plan.

3.Add a Thread Group to simulate concurrent users.

4.Configure the Thread Group with the desired number of users, ramp-up period, and loop count.

Add HTTP Request Samplers for the homepage (http://localhost:9010/) and the product detail page (<http://localhost:9010/product-details>).

Add listeners (e.g., Summary Report, View Results Tree) to collect and analyse the load test results.

**Postman**

Postman is a versatile tool for testing APIs, allowing you to send requests, analyse responses, and validate API functionality. Here's how to create Postman scripts for API testing:

1. **Download and Install Postman:** If you haven't already, download and install Postman from the official website. Postman is available as a desktop application or as a web version.

2. **Create a New Postman Collection:** Open Postman and start a new collection. Collections help organize your API requests and tests logically. Click the "New" button and choose "Collection."

3. **Add Requests for Each API Endpoint:** Under your created collection, click the "Add Request" button. This is where you'll define each API endpoint you want to test.

4. **Retrieve the List of All Products:** For testing the endpoint that retrieves the list of all products, set the request type to GET and the URL to <http://localhost:9010/get-shoes>.

5. **Retrieve the List of All Registered Users:** Similarly, for testing the endpoint that retrieves the list of all registered users, set the request type to GET and the URL to <http://localhost:9010/get-users>.

6. For testing the endpoint that adds a product, set the request type to POST and the URL to [http://localhost: **Add a Product:** 9010/add-shoe](http://localhost:9010/add-shoe). In the request body, provide the required parameters like ID, image URL, name, category, sizes, and price.

7. **Send Requests and Analyse Responses:** Click the "Send" button for each request to send it to the specified API endpoint. Postman will display the response, including status code, headers, and response body.

**Benefits and Advantages:**

Ease of Use: Postman provides a user-friendly interface to create, send, and analyze API requests without requiring in-depth programming knowledge.

Efficiency: Rapidly test various API endpoints and methods, accelerating the testing process.

Response Validation: Postman allows you to validate response data using assertions. You can ensure that the expected response matches the actual one.

Automation Possibilities: Postman enables the creation of automated test scripts using its scripting capabilities.

Collaboration: Share collections with team members, facilitating collaboration in API testing efforts.

Creating Postman scripts for API testing involves setting up a collection, adding requests for each API endpoint, and sending requests to analyse responses. This process streamlines API testing, helps ensure endpoint functionality, and supports the efficient development of reliable web applications.

**Algorithm**

Postman Scripts for API Testing:

1.Download and install Postman if not already done.

2.Create a new Postman Collection.

3.Add a request for each API endpoint:

4.Retrieve the list of all products: Set the request type to GET and URL to <http://localhost:9010/get-shoes>.

5.Retrieve the list of all registered users: Set the request type to GET and URL to http://localhost:9010/getusers.

6.Add a product: Set the request type to POST and URL to http://localhost:9010/add-shoe, along with required parameters in the request body.