WRITEUP:

SELENIUM:

**Test Method:** The code defines a TestNG test method named buyShoes().

Page Object Model (POM): It follows the Page Object Model design pattern. It creates instances of different page classes (e.g., LandingPage, RegisterPage, LoginPage, AddtoCartPage, PlaceOrder) to interact with elements on each page.

**User Registration:**

It starts by clicking on the "New User Register" button on the landing page.

It then fills in registration information (name, email, and password) and clicks the "Register" button on the registration page.

After registration, it clicks the "Logout" button.

**User Login:**

It logs in using the registered email and password.

**Add to Cart:**

It scrolls the webpage using JavaScript to bring the "Add to Cart" button into view.

It waits for 3 seconds (using Thread.sleep) to allow time for the button to load.

It clicks the "Add to Cart" button.

**Place Order:**

It clicks on the "home" and "Cart" buttons.

It scrolls the webpage again using JavaScript.

It clicks the "Place Order" button.

**Assertions:**

It captures the success message displayed on the page.

It asserts that the actual success message matches the expected message, which is "Success!" using Assert.assertEquals().

This code essentially represents a test scenario where a user registers, logs in, adds an item to the cart, and places an order on a website. It's a common approach used in automated testing to ensure that a web application's functionality works as expected.

While it's not a traditional algorithm, it follows a procedural flow of actions to test the application's behavior.

**RESTASSURED:**

**GETProducts Class:**

This class contains a single test method named getServerResponse().

It sends an HTTP GET request to the endpoint http://localhost:9010/get-users.

It asserts that the response status code is 200 (OK).

It logs the entire response.

**GETShoes Class:**

Similar to the GETProducts class, this class contains a single test method named getServerResponse().

It sends an HTTP GET request to the endpoint http://localhost:9010/get-shoes.

It asserts that the response status code is 200 (OK).

It logs the entire response.

**POSTSportShoe Class:**

This class contains a single test method named getServerResponse().

It sends an HTTP POST request to the endpoint http://localhost:9010/add-shoe with query parameters (id, image, name, category, sizes, and price).

It asserts that the response status code is 200 (OK).

It logs the entire response.

Here's a high-level algorithm for each of the test methods:

For getServerResponse() methods in GETProducts and GETShoes classes:

Send an HTTP GET request to the specified endpoint.

Receive the response from the server.

Assert that the response status code is 200 (indicating a successful response).

Log the entire response, including headers and body.

For getServerResponse() method in POSTSportShoe class:

Construct an HTTP POST request with query parameters (id, image, name, category, sizes, price).

Send the HTTP POST request to the specified endpoint.

Receive the response from the server.

Assert that the response status code is 200 (indicating a successful response).

Log the entire response, including headers and body.

These test classes are designed to verify the functionality of the RESTful API endpoints by checking the response status codes and logging the responses. They do not perform any complex algorithmic operations but are important for testing the integration and correctness of API endpoints in your application.

POSTMAN:

Creating 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/get-users.

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

JMeter:

http://localhost:9010/

1.First Recorded the steps from registering to placing the order by using the blazemeter by using the above link

2.And then opened that recored. jmx in JMeter

3. Create a new JMeter Test Plan.

4.Add a Thread Group to simulate concurrent users.

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

6.Add the listener

7.Add the View Results Tree

8.click on Run

GITHUB REPOSITORY:

https://github.com/Meghanabattagani11/Simplilearn\_capstoneProject1.git