Write Up naveen n k

**Tester Details:**

* **Name:** NAVEEN N K
* **Submitting to:** Simplilearn Pvt
* **Company:** Mphasis Pvt
* **Project:** Sporty Shoes
* **GitHub URL:** [**https://github.com/nknaveengowda/Capstone-Project-Sporty-shoes**](https://github.com/nknaveengowda/Capstone-Project-Sporty-shoes)

1. **Create Selenium scripts using TestNG to test all the pages in the web app that will automate:**

**● Login page**

**● Registration Page**

**● Add Product to cart page.**

**● Place Order Page**

Step 1: Project Setup

1. **Create a Maven Project:**
   * Open your preferred IDE and create a new Maven project.
2. **Add Dependencies:**
   * Update the **pom.xml** file with dependencies for Selenium WebDriver and TestNG.
3. **Configure WebDriver:**
   * Download the WebDriver executable (e.g., ChromeDriver) and set the path in your script.

Step 2: Test Classes

1. **Create Test Classes:**
   * Create separate TestNG classes for each page to be tested, such as LoginPageTest, RegistrationPageTest, AddToCartPageTest, and PlaceOrderPageTest.

Step 3: Implement Test Methods

**3.1 Login Page Test (LoginPageTest.java):**

* In the **LoginPageTest** class:
  + Use **@BeforeClass** annotation to set up WebDriver and navigate to the login page.
  + Use **@Test** annotation for the actual test method.
  + Implement login test steps within the **@Test** method.
  + Use assertions (e.g., TestNG's **Assert** class) to verify expected outcomes.

**3.2 Registration Page Test (RegistrationPageTest.java):**

* Similar to the login page test, create a **RegistrationPageTest** class with setup and test methods specific to the registration page.

**3.3 Add Product to Cart Page Test (AddToCartPageTest.java):**

* Create an **AddToCartPageTest** class and implement the necessary test methods for adding a product to the cart.

**3.4 Place Order Page Test (PlaceOrderPageTest.java):**

* Similarly, create a **PlaceOrderPageTest** class and implement test methods for placing an order.

Step 4: Execute Tests

1. **Run TestNG Tests:**
   * Run the test classes as TestNG tests. You can do this by right-clicking on the test class and selecting "Run As" > "TestNG Test."
2. **Review Test Results:**
   * Check the test results in the TestNG console or the generated reports.
3. **Create JMeter scripts to do load testing of the homepage and the product detail page.**

Step 1: Download and Install JMeter

1. **Download JMeter:**
   * Visit the official Apache JMeter website and download the latest version of Apache JMeter.
2. **Install JMeter:**
   * Extract the downloaded ZIP file to your preferred location.
3. **Launch JMeter:**
   * Navigate to the **bin** directory and run the **jmeter.bat** (Windows) or **jmeter.sh** (Linux) executable.

Step 2: Create a New Test Plan

1. **Open JMeter:**
   * Launch JMeter, and you will see the JMeter GUI.
2. **Create a New Test Plan:**
   * Right-click on "Test Plan" in the left panel.
   * Choose "Add" > "Threads (Users)" > "Thread Group."
3. **Configure Thread Group:**
   * Set the number of threads (virtual users) and loop count based on your load testing requirements.

Step 3: Add HTTP Requests for Homepage

1. **Add HTTP Request Sampler:**
   * Right-click on the Thread Group.
   * Choose "Add" > "Sampler" > "HTTP Request."
2. **Configure HTTP Request for Homepage:**
   * Set the server name or IP address and path for the homepage.
   * Define any necessary parameters.

Step 4: Add HTTP Requests for Product Detail Page

1. **Add Another HTTP Request Sampler:**
   * Repeat the process to add an HTTP Request Sampler for the product detail page.
   * Configure the request with the appropriate server name, path, and parameters.

Step 5: Add Listeners for Monitoring

1. **Add Listeners:**
   * Right-click on the Thread Group.
   * Choose "Add" > "Listener" > Select appropriate listeners (e.g., View Results Tree, Summary Report, or Response Times Over Time).
2. **Configure Listeners:**
   * Configure the listeners to capture and display relevant performance metrics.

Step 6: Configure Test Execution

1. **Configure Test Execution:**
   * Set up test execution parameters such as ramp-up time, duration, and other relevant settings.
2. **Run the Test:**
   * Click the "Run" menu and choose "Start" to run the test plan.

Step 7: Analyze Results

1. **View Results:**
   * Once the test execution completes, review the results in the listeners.
2. **Create Postman scripts to test the following API endpoints:**

* **Retrieve the list of all products in the store.**
* **Retrieve the list of all registered users.**
* **Add the product.**
* **Delete the product.**
* **Update the product.**

Step 1: Download and Install Postman

1. **Download Postman:**
   * Visit the Postman website and download the Postman app suitable for your operating system.
2. **Install Postman:**
   * Follow the installation instructions to install Postman on your machine.

Step 2: Create a New Collection

1. **Open Postman:**
   * Launch the Postman app.
2. **Create a New Collection:**
   * Click on "New" to create a new collection.
   * Name the collection (e.g., "API Testing").

Step 3: Add Requests for API Endpoints

**3.1 Retrieve the List of All Products**

1. **Add a Request:**
   * Click on "New Request" within your collection.
   * Name the request (e.g., "Get All Products").
2. **Configure Request:**
   * Set the request type to "GET."
   * Enter the URL for retrieving the list of all products.

**3.2 Retrieve the List of All Registered Users**

1. **Add Another Request:**
   * Create a new request for retrieving the list of all registered users.
   * Name the request appropriately.
2. **Configure Request:**
   * Set the request type to "GET."
   * Enter the URL for retrieving the list of registered users.

**3.3 Add a Product**

1. **Add Another Request:**
   * Create a new request for adding a product.
   * Name the request appropriately.
2. **Configure Request:**
   * Set the request type to "POST."
   * Enter the URL for adding a product.
   * Define the request body with necessary parameters.

**3.4 Delete a Product**

1. **Add Another Request:**
   * Create a new request for deleting a product.
   * Name the request appropriately.
2. **Configure Request:**
   * Set the request type to "DELETE."
   * Enter the URL for deleting a product.
   * Include any necessary parameters (e.g., product ID).

**3.5 Update a Product**

1. **Add Another Request:**
   * Create a new request for updating a product.
   * Name the request appropriately.
2. **Configure Request:**
   * Set the request type to "PUT" or "PATCH" based on your API.
   * Enter the URL for updating a product.
   * Define the request body with updated information.

Step 4: Execute Requests and Review Responses

1. **Execute Requests:**
   * Click on the "Send" button to execute each request.
   * Observe the response details in the Postman interface.
2. **Review Responses:**
   * Check the response status codes, headers, and body content for each request.
   * Ensure that the responses align with expected outcomes.

Step 5: Save and Share the Collection

1. **Save Collection:**
   * Save the collection in Postman to reuse and share it later.
2. **Export Collection:**
   * Export the collection as a JSON file for sharing with team members or storing in version control.
3. **Automate the below API endpoints using Rest-Assured**

* **Retrieve the list of all products in the store.**
* **Retrieve the list of all registered users.**
* **Add the product.**
* **Delete the product.**
* **Update the product.**

Step 1: Set Up Your Project

1. **Create a New Maven Project:**
   * Open your preferred IDE (Eclipse, IntelliJ, etc.).
   * Create a new Maven project.
2. **Add Dependencies:**
   * In your **pom.xml** file, include dependencies for Rest-Assured, TestNG, and any other required libraries.

Step 2: Plan Your Test Cases

1. **Retrieve the List of All Products:**
   * Identify the API endpoint for retrieving the list of products.
   * Plan how to validate the response (e.g., check for a successful status code).
2. **Retrieve the List of All Registered Users:**
   * Identify the API endpoint for retrieving the list of registered users.
   * Plan how to validate the response.
3. **Add a Product:**
   * Identify the API endpoint for adding a product.
   * Plan how to send the request with necessary parameters.
   * Plan how to validate the response.
4. **Delete a Product:**
   * Identify the API endpoint for deleting a product.
   * Plan how to send the request with the product ID.
   * Plan how to validate the response.
5. **Update a Product:**
   * Identify the API endpoint for updating a product.
   * Plan how to send the request with the updated product information.
   * Plan how to validate the response.

Step 3: Create Test Classes

1. **Create a Test Class:**
   * Create a TestNG test class for API testing (e.g., **ApiTests**).
2. **Initialize Base URI:**
   * Use setup methods (e.g., **@BeforeClass**) to set up the base URI for your API.

Step 4: Implement Test Cases

1. **Retrieve the List of All Products:**
   * Implement a test method to send a GET request to the products endpoint.
   * Validate the response to ensure it meets the expected criteria.
2. **Retrieve the List of All Registered Users:**
   * Implement a test method to send a GET request to the users endpoint.
   * Validate the response.
3. **Add a Product:**
   * Implement a test method to send a POST request to add a product.
   * Include necessary parameters in the request.
   * Validate the response.
4. **Delete a Product:**
   * Implement a test method to send a DELETE request to delete a product.
   * Provide the product ID in the request.
   * Validate the response.
5. **Update a Product:**
   * Implement a test method to send a PUT or PATCH request to update a product.
   * Include the updated product information.
   * Validate the response.

Step 5: Execute Tests

1. **Run TestNG Tests:**
   * Execute the TestNG tests to run your API test cases.
2. **Setup Cucumber in Java Project and write Feature Files using Gherkin to test the API endpoints**

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

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

**● Add the product.**

**● Delete the product.**

**● Update the product.**

Step 1: Set Up Your Project

1. **Create a New Maven Project:**
   * Open your preferred Integrated Development Environment (IDE) (Eclipse, IntelliJ, etc.).
   * Create a new Maven project.
2. **Add Dependencies:**
   * In your **pom.xml** file, include dependencies for Cucumber, TestNG, and any other required libraries.

Step 2: Create Feature Files

1. **Create a "features" Directory:**
   * Inside the **src/test/resources** directory, create a new folder named "features."
2. **Write Feature Files:**
   * Within the "features" folder, create feature files using Gherkin syntax.
   * For each API endpoint, create a feature file with scenarios that describe the behavior of the API.

Step 3: Write Gherkin Scenarios

1. **Retrieve the List of All Products:**
   * Write a Gherkin scenario that describes the steps to retrieve the list of all products.
2. **Retrieve the List of All Registered Users:**
   * Write a Gherkin scenario for retrieving the list of all registered users.
3. **Add a Product:**
   * Write a Gherkin scenario for adding a product, including the necessary steps.
4. **Delete a Product:**
   * Write a Gherkin scenario for deleting a product, specifying the required steps.
5. **Update a Product:**
   * Write a Gherkin scenario for updating a product, detailing the steps involved.

Step 4: Implement Step Definitions

1. **Create Step Definitions:**
   * Create step definition classes for each feature file.
   * Map Gherkin steps to Java methods in these classes.

Step 5: Execute Cucumber Tests

1. **Run Cucumber Tests:**
   * Execute your Cucumber tests using the TestNG runner or a Cucumber runner class.
2. **Review Test Results:**
   * Check the test results to ensure that scenarios pass, and steps are executed successfully.

Step 6: Refine and Maintain

1. **Refine Step Definitions:**
   * Enhance and refine step definitions to align with the API behavior.
2. **Maintain Feature Files:**
   * Regularly update feature files to reflect changes in the API or requirements.