**Testing Framework Documentation for Sporty Shoes Application**

**1. Project Overview**

**Project Name**: Sporty Shoes Testing Framework
**Description**:
This project is a testing framework designed to validate the Sporty Shoes application's functionalities, performance, and reliability. It includes API testing, load testing, unit testing, and UI testing. The primary goal is to ensure that the application meets user expectations in terms of functionality and performance.

**2. Tester Details**

- **Tester Name**: Sunil Krishna Yelamanchili
- **Role**: Automation Tester
- **Date**: 05-11-2024

**3. Concepts Used in the Project**

- **Postman**: For API testing of the product and user endpoints.
- **JUnit**: For unit testing backend services to ensure code correctness.
- **JMeter**: For load testing to measure application performance under stress.
- **Selenium**: For UI testing to validate user workflows.
- **Spring Boot Security**: Utilized in the project's security configuration, which required handling authentication in testing.

**4. UI Testing with Selenium**

**Objective**: Verify that the web app's UI elements function correctly.

**Steps**:

1. Set up Selenium WebDriver and wrote scripts for key user actions like logging in and browsing products.
2. Executed tests to ensure each UI workflow operated as expected.

**Sample Test Case**:

- **Scenario**: Verify that a user can log in and view the product list.
- **Steps**:
    1. Open the login page.
    2. Enter valid credentials.

3. Assert that the product list page is displayed after login.

**5. Load Testing with JMeter**

**Objective**: Assess the performance of the homepage and product detail page under load.

**Setup**:

1. Created a JMeter test plan with HTTP requests to the homepage and product detail page.
2. Configured multiple threads to simulate concurrent users.
3. Added assertions to check response time, throughput, and error rate.

**Execution**:

- Ran the test in JMeter, gradually increasing user load.
- Monitored metrics such as response time and error rate to determine performance thresholds.

**Results**:

- Included JMeter report in the GitHub repository.
- Observations: Documented response times and throughput at various load levels.

**6. Unit Testing with JUnit**

**Objective**: Ensure that backend services and methods function as expected.

**Steps**:

1. Created unit tests for the ProductService, UserService, and PurchaseService.
2. Used assertions to validate expected vs. actual outcomes of methods such as getAllProducts(), getAllUsers(), and savePurchase().

**7. API Testing in Postman**

**Objective**: Validate API endpoints for retrieving products and users information.

**Setup and Test Execution Steps**:

1. **Setup Postman Collection**:
    - Created a new Postman collection named **SportyShoesAPITests**.
    - Added requests for:

        - GET /api/products to retrieve the list of all products.
        - GET /api/users to retrieve the list of all registered users.

2. **Add Authentication**:
    - Configured Basic Authentication with admin credentials for secure access to the /api/products endpoint.
3. **Assertions**:
    - **Response Code**: Check for a 200 OK status.
    - **Response Format**: Validate that the response is in JSON format.
    - **Required Fields**: Check for fields like productId, productName, and price in the product list response.
4. **Execution**:
    - Ran the collection in the Postman Collection Runner to validate responses.
    - Exported the collection as a JSON file and included it in the GitHub repository.

**8. GitHub Repository Documentation**

**Repository Structure**

- **/src**: Contains all source code and testing scripts.
- **/tests**: Contains Postman collections, JMeter files, and unit tests.
- **/docs**: Contains documentation files, reports, and screenshots.