**API Testing and Functional Testing - IT22332080**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testing Method | Test Description | API Used | Expected Response | Response Code |
| Positive API Testing | Verify GET request returns all products | GET /api/productsList | JSON response containing "products" | 200 |
| Negative API Testing | Accessing invalid endpoint | GET /api/invalidProducts | 404 Not Found | 404 |
| Negative API Testing | Sending unsupported POST request | POST /api/productsList | Method not allowed (ideally 405, but current test expects 200) | 200 / 405 |
| Positive API Testing +  **Performance Testing** | Validate response time is under 2 seconds | GET /api/productsList | Response time < 2000ms | 200 |
| Positive API Testing | Validate JSON structure contains product list | GET /api/productsList | JSON path "products" should exist | 200 |
| Negative API Testing | Validate POST method returns proper error and JSON fields | POST /api/productsList with JSON header | JSON with responseCode = 405, message = "This request method is not supported." | 405 |
| Mixed API Testing | Compare valid GET and invalid POST responses | GET and POST /api/productsList | GET returns products; POST should return method not allowed (currently expects 200) | 200 / 405 |
| Negative API Testing | Search without sending required parameter | POST /api/searchProduct | responseCode = 400, message = "Bad request, search\_product parameter is missing in POST request." | 200 (HTTP), 400 (logic) |

**UI Testing**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Name** | Test Description | Input Data | Expected Result | Actual Result | Pass/Fail | Testing Type | Tools/Technologies Used |
| Invalid Email Subscription Test | Verify that invalid email input is rejected by the form | 1235 | Form should not submit, browser blocks it | Form blocked | Pass | UI Testing | Selenium WebDriver, ChromeDriver, TestNG, Java |

This software testing project was developed using **Java** as the primary programming language and **Maven** for build automation and dependency management. For API testing, the project utilizes **REST Assured**, a powerful Java library that allows for seamless testing of RESTful web services through fluent syntax. Test cases for both API and UI components are structured and executed using **TestNG**, which provides annotations like @Test, @BeforeClass, and @AfterClass for test lifecycle management and reporting. The project also incorporates **Selenium WebDriver** for automated **UI testing**, enabling browser-based testing scenarios such as form validation and user interaction. Specifically, **ChromeDriver** is used as the interface between Selenium and the Google Chrome browser to simulate real-world user activity. All test executions are managed through the **Maven Surefire Plugin**, which automatically identifies and runs test classes, produces concise output reports, and supports redirecting test logs to files. This setup ensures a robust, modular, and fully automated testing framework for validating both the functionality and user interface of the application.

Intro for arunalu and migara-

This project uses **Java** for writing test logic and **Maven** to manage dependencies. **Selenium Java (4.31.0)** handles browser automation, while **Selenium DevTools v118** is used for advanced browser features like network monitoring. **TestNG (7.11.0)** structures and runs the tests. **WebDriverManager (6.1.0)** takes care of downloading the correct browser drivers. For logging, it uses **Log4j**, **Log4j2**, and **SLF4J (2.0.9)** to track and organize test output.

**Description: Login Functionality (testLoginUserWithCorrectCredentials)**

The **login test** validates that a user can successfully log in with correct credentials. It navigates to the login page, fills in a valid email and password, and verifies the "Logged in as" label with the correct username. This confirms that authentication and session handling are functioning as intended.

**Description: Login with Incorrect Credentials (testLoginUserWithIncorrectCredentials)**

This negative test checks if the system correctly handles invalid login attempts. The test tries to log in using the wrong credentials and verifies that the user is either not logged in or redirected properly. (Note: Your assertion currently checks for "Jane Doe" but may require adjustment depending on intended behavior.)

**Description: Register Functionality (testRegisterUser)**

The **registration test** automates the complete flow of creating a new user account on [automationexercise.com](https://automationexercise.com). It begins by verifying the visibility of the home page and navigates to the "Signup/Login" section. It then fills in unique user details such as name and email, followed by detailed account information (password, DOB, address, etc.). The test ensures that the "ACCOUNT CREATED!" confirmation appears and validates the user is logged in. Finally, the test verifies account deletion with the "ACCOUNT DELETED!" message. Screenshots are taken at each key step to aid in debugging and reporting.

**Description: Register with Existing Email (testRegisterUserEmailFailure)**

This negative test case checks the application's response when attempting to register with an already existing email. After entering the name and existing email and clicking the signup button, it waits for the error message "Email Address already exist!" and verifies that the user remains on the signup page without progressing further.

Arunalu- perfomace test

This project uses **Apache JMeter** programmatically via Java to perform **performance testing** on the https://automationexercise.com/api/createAccount API endpoint. It simulates multiple concurrent users (threads) sending POST requests with dynamically generated account data to evaluate server responsiveness under load. The setup involves configuring a **Test Plan**, **Thread Group**, **Loop Controller**, **HTTP Sampler**, **Header Manager**, and saving results to a .jtl file. Key technologies used include **JMeter’s core engine**, **HTTPSamplerProxy** for request handling, **HeaderManager** for setting content type, and **ResultCollector** for logging outcomes. This setup enables automated load testing without using the JMeter GUI, making it suitable for integration into CI/CD pipelines or custom test frameworks.