Approach

Assignment Task 1: Web Automation

I have chosen the website "https://www.saucedemo.com/" as the login website.

Automation Approach:
The automation approach prioritizes modularity, reusability, and robust reporting to ensure the framework is maintainable and scalable for real-world testing scenarios.
Key components are:

- **Page Object Model (POM):** It encapsulates page elements and actions to reduce code duplication and improve maintainability.
- **Data-Driven Testing:** It uses a database to fetch login credentials (**sqlite** database as per requirement).
- **Cross-Browser Support:** Configure the project to support multiple browser for running test cases with DriverFactory utility
- **Error Handling and Reporting:** Integrates ExtentReports for detailed, emailable HTML reports with screenshots for failed tests.
- **TestNG Framework:** Manages test execution, setup, and teardown, with dynamic test data via @DataProvider.

Tools and Technologies

- **Selenium WebDriver:** For browser automation.
- **Java:** Programming language for scripting.
- **TestNG:** Test framework for managing test cases and execution.
- **ExtentReports:** For generating emailable HTML reports.
- **Sqlite Database:** For storing test data (login credentials).
- **ConfigReader:** To manage configuration settings (e.g., browser, URL).
- **Maven**: For dependency management.

Dependencies	Version	Reasons
lombok	1.18.38	Required to fix "build failed" issue after integration of ExtentReports.
selenium-java	4.34.0	Provides WebDriver APIs for browser automation and web UI testing.
testng	7.11.0	Testing framework used to write and run structured test cases with annotations.
extentreports	5.1.2	Used to generate emailable HTML reports.

Dependencies	Version	Reasons
sqlite-jdbc	3.50.2.0	JDBC driver for SQLite database used for storing test data.
commons-io	2.19.0	Provides utility classes for file operations. Current use case involves FileUtils for copying screenshot files.

In order to verify the authentication process, I conducted a thorough set of login tests. My approach involved:

1) Valid and Invalid Credentials: Confirmed successful logins for authorized users and ensured the system correctly blocked invalid usernames or wrong passwords.
2) Edge Cases:
    I. Tested empty username and password fields to check for proper error responses.
    II. Security Checks: Ran tests against SQL injection and Cross-Site Scripting (XSS) attacks to validate the system's protection mechanisms.
    III. Input Validation: Evaluated the system's ability to handle special characters and excessively long usernames without issues.
    IV. Whitespace Treatment: Verified how usernames and passwords with spaces or whitespace characters were processed.

The complete test cases is provided in the table below

Test Case ID	Test Case Name	Username	Password	Expected Result
TC01	Valid user	standard_user	secret_sauce	TRUE
TC02	Locked out user	locked_out_user	secret_sauce	FALSE
TC03	Problem user valid login	problem_user	secret_sauce	TRUE
TC04	Performance glitch user	performance_glitch_user	secret_sauce	TRUE
TC05	Invalid user	invalid_user	secret_sauce	FALSE
TC06	Wrong password	standard_user	wrong_password	FALSE
TC07	Empty username	*(empty)*	secret_sauce	FALSE
TC08	Empty password	standard_user	*(empty)*	FALSE
TC09	Both fields empty	*(empty)*	*(empty)*	FALSE
TC10	SQL injection username	' OR 1=1--	any	FALSE
TC11	SQL injection password	standard_user	' OR '1'='1	FALSE
TC12	Special characters	special!@#$	pa$$word!@#	FALSE
TC13	Long username (50 'a's)	aaaaaaaaaaaaaaaaaaaaaaaaaaaaa......	secret_sauce	FALSE
TC14	XSS input	<script>alert(1)</script>	test	FALSE
TC15	Username with whitespace	standard_user	secret_sauce	FALSE
TC16	Password with whitespace	standard_user	secret_sauce	FALSE
TC17	Username all spaces		secret_sauce	FALSE
TC18	Password all spaces	standard_user		FALSE

Process Documentation

1) **BasePage.java**
    1) I created BasePage as the backbone of the Page Object Model for this project.
    2) I made the WebDriver static here so that all page objects can share the same driver instance
    3) I included simple, reusable methods like find(), set(), and click() to abstract away repetitive Selenium code.

2) **LoginPage.java**
    1) I identified and made the login page elements (username, password and login button) private as these elments are the properties belonging to LoginPage alone.
    2) **click** (), **setText**() and **find**() is reused in this page from the parent class BasePage to send data to element and perform user interactions.
    3) **isLoginSuccessful**() function is defined to check if the user has successfully logged into the the homepage(inventory.html) and checked with logic that the url contains “**inventory**”.
    4) There are two functions defined in this page in case dynamic fetch is needed for the usenames

3) **Base Test**
    1) I built this class to centralize the setup, teardown, and report management for all test classes.
    2) Integrated **ConfigReader** to read values like browser and baseUrl from “config.properties” file, for cross browser testing functionality(currently chrome and firefox integrated)
    3) Then also defined **DriverFactory** class to initialize driver based on the config file data
    4) **@BeforeClass** initializes WebDriver, page objects, and ExtentReports
    5) **@BeforeMethod** dynamically sets test names in reports, keeping execution logs organized and easy to trace.
    6) **@AfterClass** quits the browser and flushes reports, ensuring clean teardown and no resource leakage.
    7) I made **getReportName**() and **getDocumentTitle**() overridable so each test class can generate context-specific reports.

4) **LoginTest**
    1) This class holds the actual test for the login functionality in a data-driven way, fetching all test data directly from a database named ***testdb(The login.sqlite file is in the database directory that populates the data)***
    2) The use of TestNG’s @DataProvider with DBUtility.getLoginData() enables scalable, maintainable, and dynamically controlled test coverage.
    3) By adding a try-catch block inside the test, I ensured that crashes or unexpected errors are logged cleanly, and screenshots are captured using ScreenshotUtility for post-analysis.

5) **Utilities**
    - I created utility base to share the WebDriver instance
    - **setUtilityDriver**() fetches the initialized driver from BasePage, keeping everything aligned and avoiding driver duplication or conflicts.

1) ConfigReader
   I built this to centralize configuration management using **”config.properties”** file.

2) DBUtility
   1. This utility allows to run data-driven tests by pulling login credentials from a real database.

3) DriverFactory
   1. This utility class allows to create the WebDriver dynamicall**y.**

4) ExtentReportManager
   1. I centralized report management to handle initialization, logging, and flushing in one place.
   2. **Report filenames are dynamically** timestamped, ensuring unique test runs generate unique reports.
   3. I used thread-safe ThreadLocal storage for ExtentTest to ensure parallel-safe reporting.

5) ScreenshotUtility
   1. I added this to automatically capture screenshots on failure or crash and is used in the catch block of login test
   2. It saves screenshots inside reports/screenshots/.
   3. The relative path return ensures correct embedding in HTML reports, even when opened locally.
   4. I extended from Utilities to reuse the global driver without additional setup.

Final Result:

The final obtained results is provided in the table below. All the test passed without any errors.

Test Case ID	**Test Case Name**	**Username**	**Password**	**Expected Result**	**Actual Result**
TC01	Valid user	standard_user	secret_sauce	TRUE	TRUE
TC02	Locked out user	locked_out_user	secret_sauce	FALSE	FALSE
TC03	Problem user valid login	problem_user	secret_sauce	TRUE	TRUE
TC04	Performance glitch user	performance_glitch_user	secret_sauce	TRUE	TRUE
TC05	Invalid user	invalid_user	secret_sauce	FALSE	FALSE
TC06	Wrong password	standard_user	wrong_password	FALSE	FALSE
TC07	Empty username	*(empty)*	secret_sauce	FALSE	FALSE
TC08	Empty password	standard_user	*(empty)*	FALSE	FALSE
TC09	Both fields empty	*(empty)*	*(empty)*	FALSE	FALSE
TC10	SQL injection username	' OR 1=1--	any	FALSE	FALSE
TC11	SQL injection password	standard_user	' OR '1'='1	FALSE	FALSE
TC12	Special characters	special!@#$	pa$$word!@#	FALSE	FALSE
TC13	Long username (50 'a's)	aaaaaaaaaaaaaaaaaaaa........	secret_sauce	FALSE	FALSE
TC14	XSS input	<script>alert(1)</script>	test	FALSE	FALSE
TC15	Username with whitespace	standard_user	secret_sauce	FALSE	FALSE
TC16	Password with whitespace	standard_user	secret_sauce	FALSE	FALSE
TC17	Username all spaces		secret_sauce	FALSE	FALSE
TC18	Password all spaces	standard_user		FALSE	FALSE