**CAPSTONE PROJECT**

**AUTOMATION TESTING FOR LUMA WEBSITE**

**WEBSITE LINK**: https://magento.softwaretestingboard.com

**Introduction:**

The Automation Testing Capstone Project focused on the website marked as significant milestone in the participants' learning journey. The project provided an opportunity to apply knowledge and skills acquired during the automation testing training, incorporating Selenium, TestNG, Cucumber, Apache POI, a hybrid framework, and advanced reporting tools like Allure/Extent. The objective is to create a comprehensive automation testing solution, fostering practical experience and showcasing participants' abilities to potential employers**.**

**Problem Statement & Expectation:**

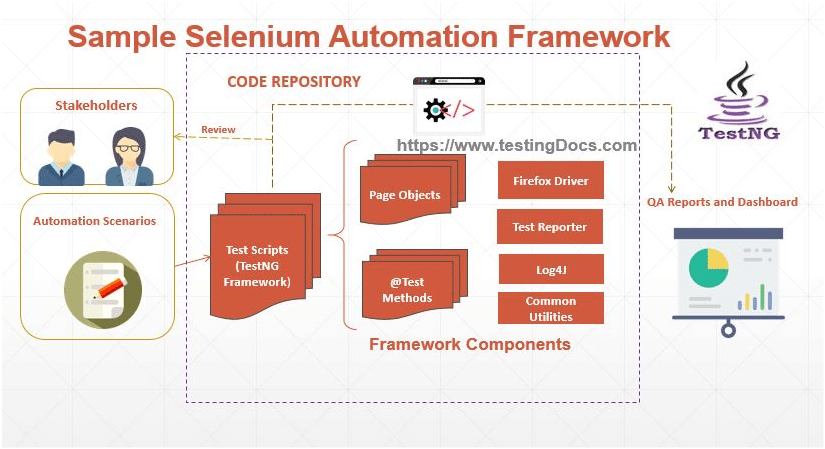
In an online shopping/pet portal, where participants are going to work on different projects. Where user can register, login and buy any of the available items. Users can search for the item and add to the cart. Update the cart with quantity or with another item if required and make the payment. Objective is to Automate these workflows using Selenium with TestNG, Page Object Model and getting the data from data sheet/properties file. There by, we are testing faster and accurate using automation technique.

**Goal**

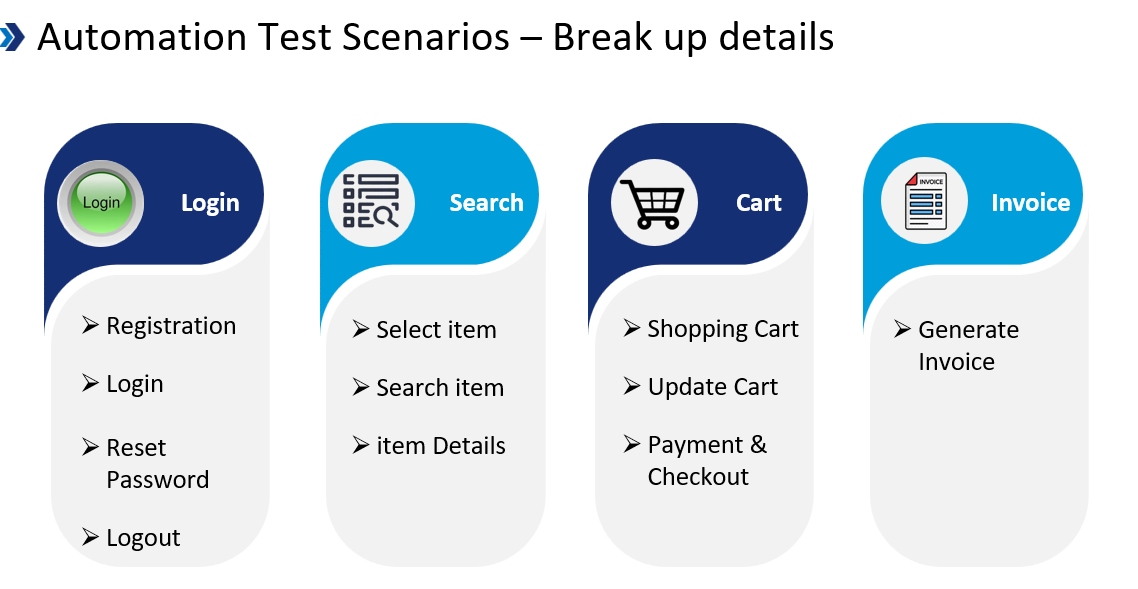
The goal of this project is to automate the end-to-end user workflows of the Magento Luma application, including user registration, login, product search, cart management, and the checkout process. It aims to develop a robust, scalable, and maintainable hybrid automation framework by leveraging Selenium WebDriver, TestNG, BDD-Cucumber, and Apache POI for data-driven testing. Additionally, the project focuses on generating advanced reports using Extent and Allure, ensuring cross-browser compatibility, and providing comprehensive documentation for easy execution and future maintenance.

**OUR JOURNEY WITH AUTOMATION:**

* Selenium
* Page Object Model
* Page Factory
* Cucumber BDD
* TestNG
* Allure Report
* Extent Report
* Java
* Apache POI
* Maven
* Git/Github
* Eclipse
* Jenkins



**TEST SCENARIO MODULES:**



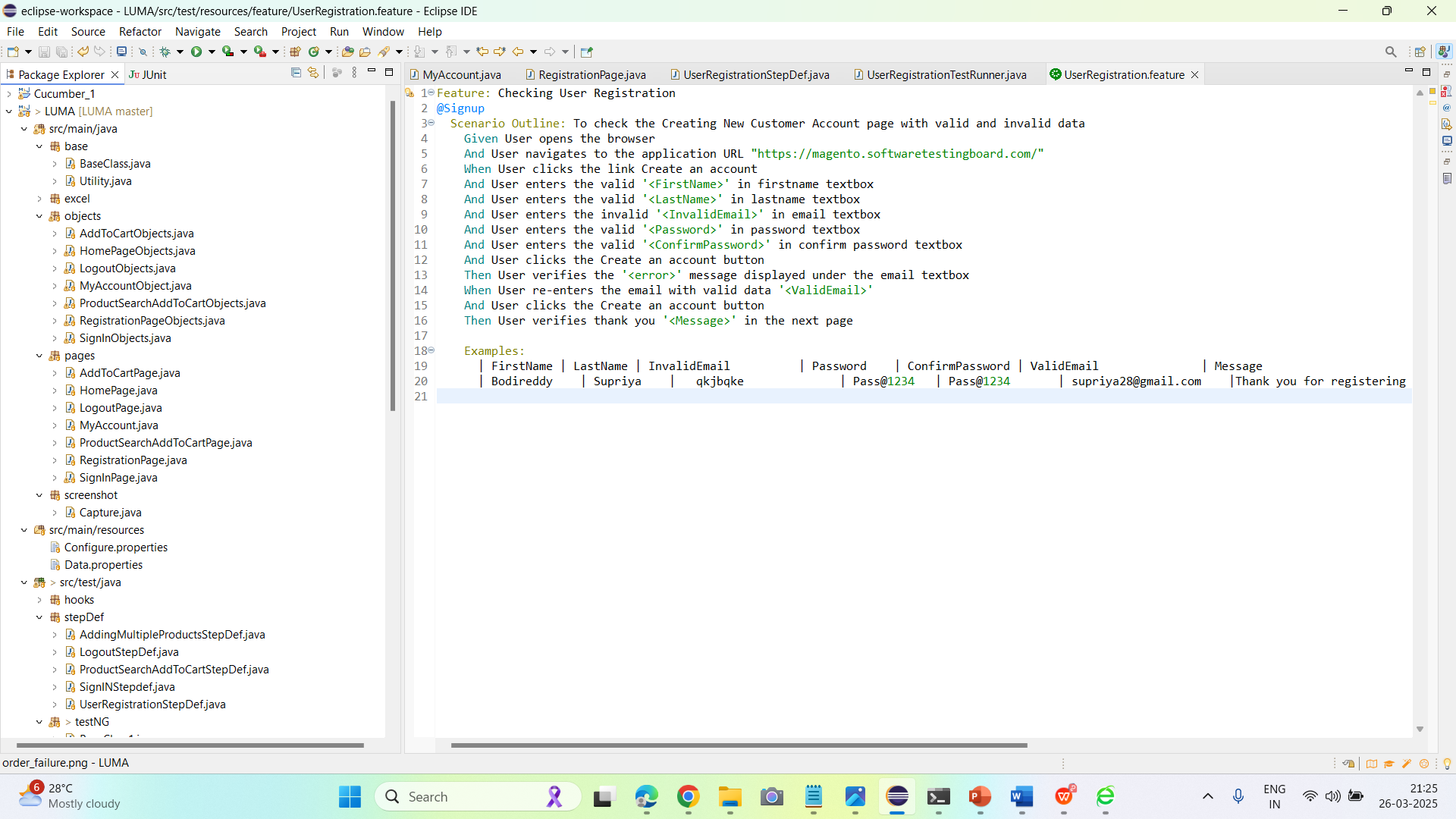
**Project Title: <Magento Luma>**

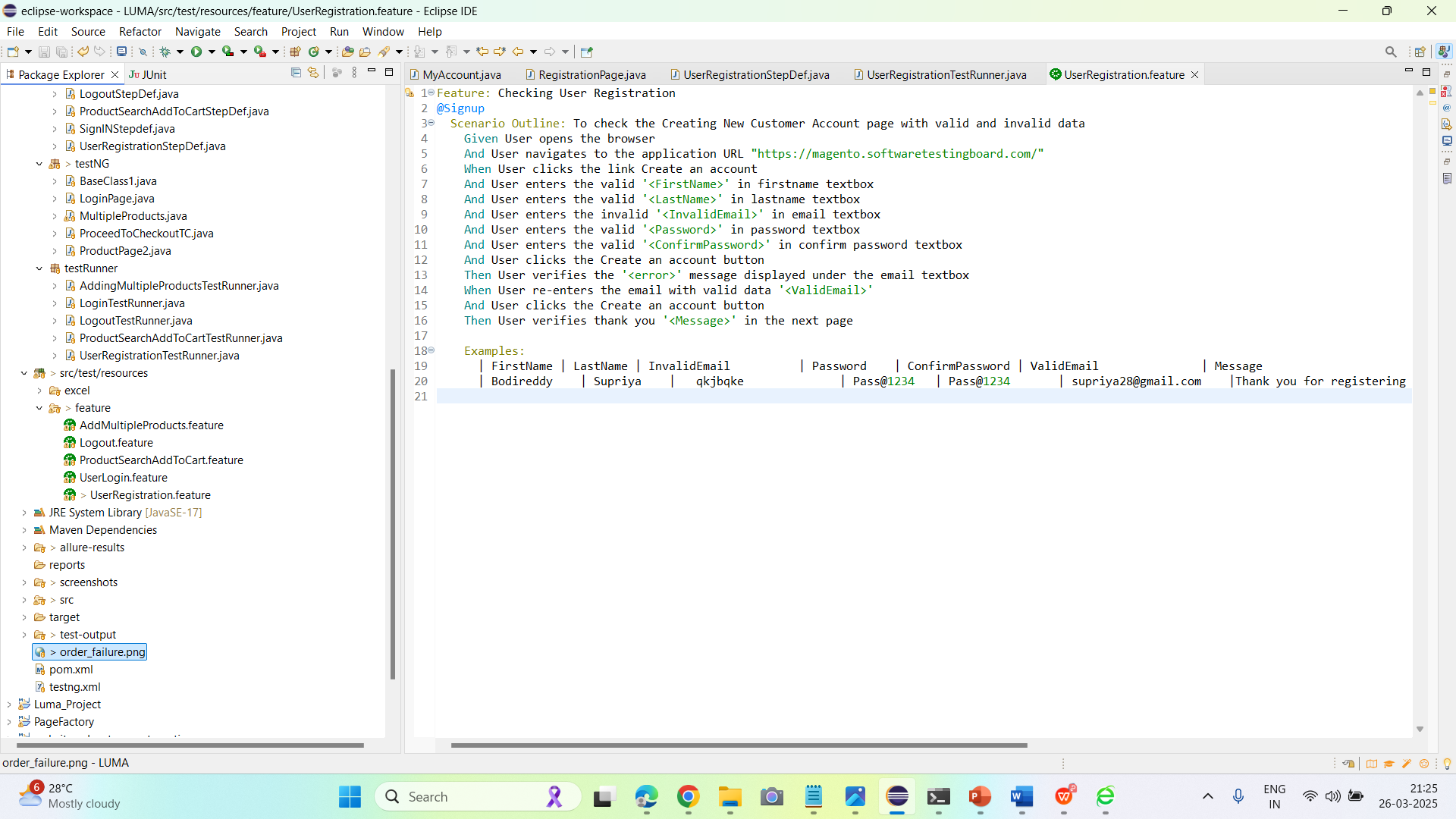
**Objective:** The central aim of this capstone project is to construct a durable and streamlined test automation framework for evaluating the E-commerce/learning store. The framework integrates Selenium, BDD-Cucumber, Apache POI, Hybrid methodology, reporting tools and basic API Testing.

**Test Flow:**

* Scenario 1: Registration
* Scenario 2: Login
* Scenario 3: Add to Cart
* Scenario 4: Adding Multiple Products To Cart
* Scenario 5: Checkout
* Scenario 6: Logout

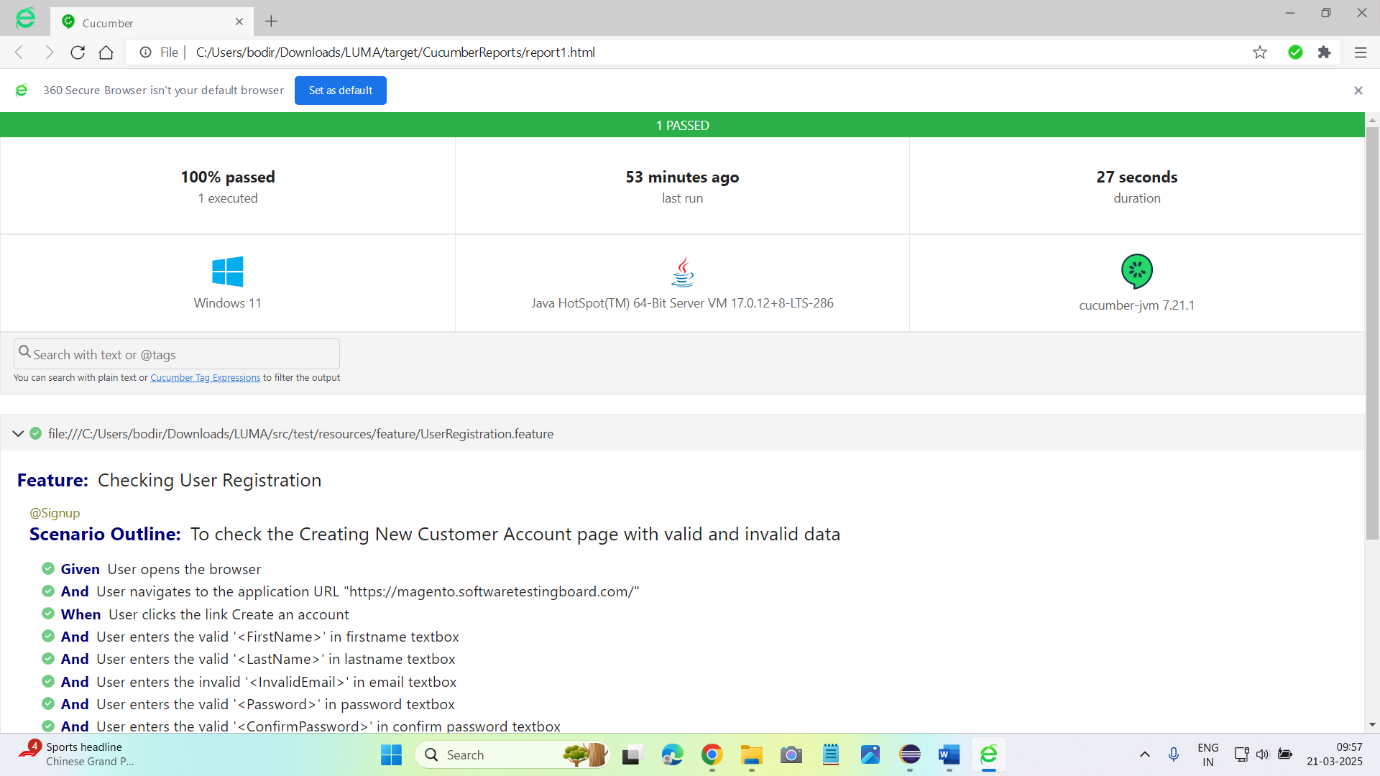
**Project Structure:**





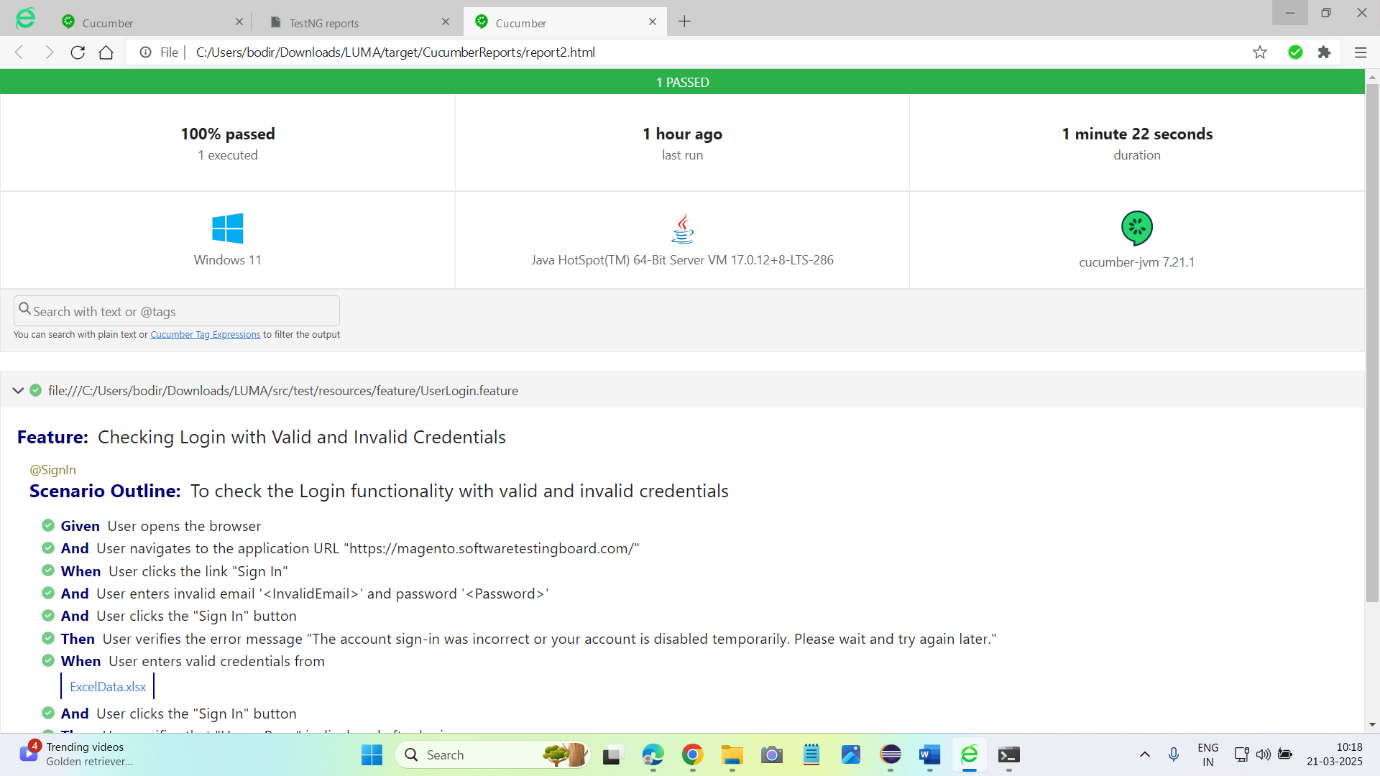
**Scenario 1:** Registration for Luma website with valid and invalid login using Cucumber BDD framework using POM, and generating allure reports.

**OUTPUT:** Cucumber Reports:

****

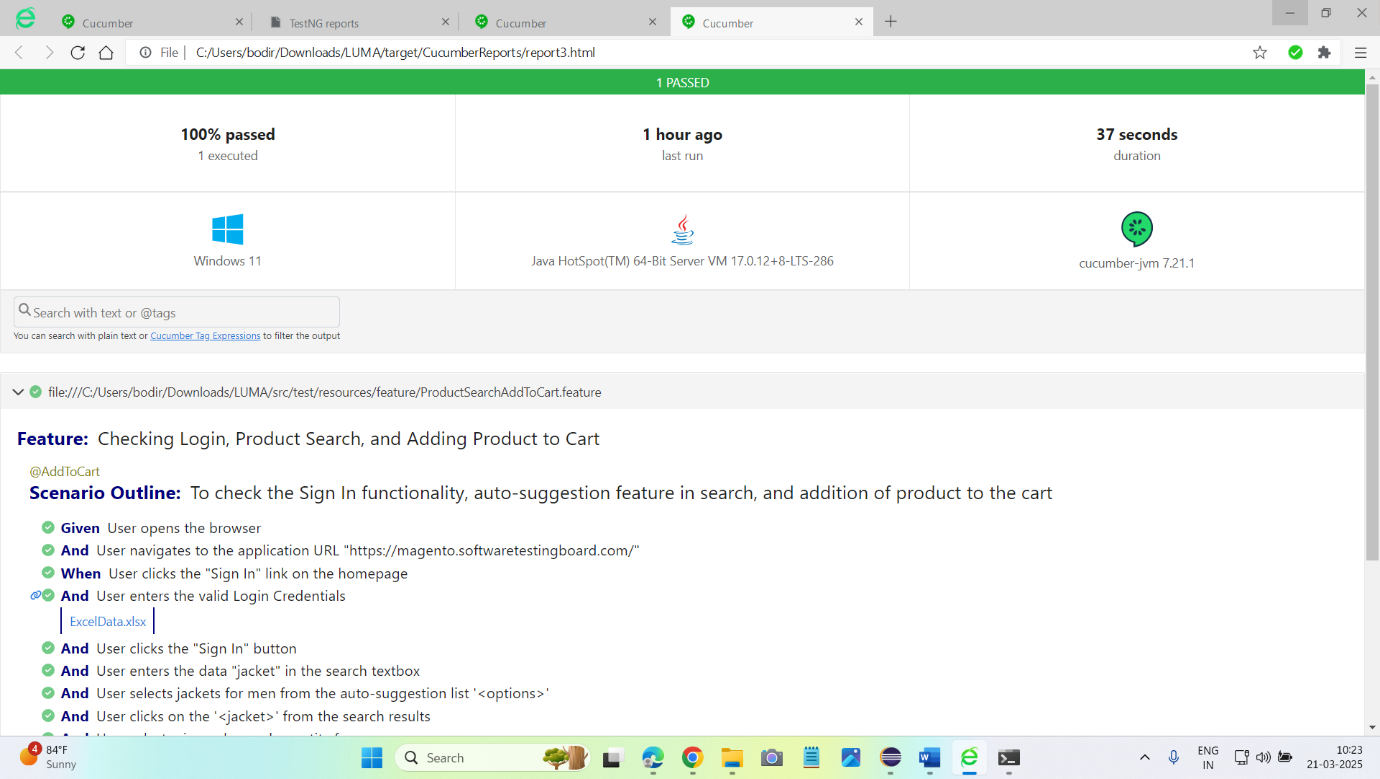
**Scenario 2:** Designed Cucumber framework for Login successful using POM, Data Properties, tags, hooks and generated extent reports.

**OUTPUT:** Cucumber Reports:



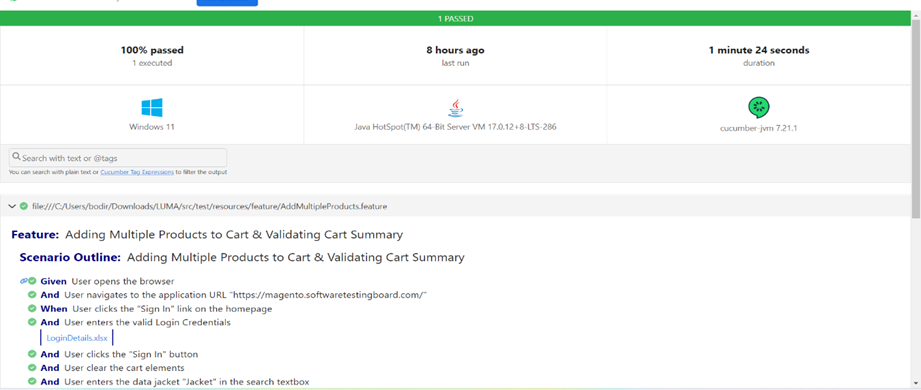
**Scenario 3:** Add to Cart using Cucumber Framework, POM, Data properties, generate allure reports.

**OUTPUT:** Cucumber Reports:

****

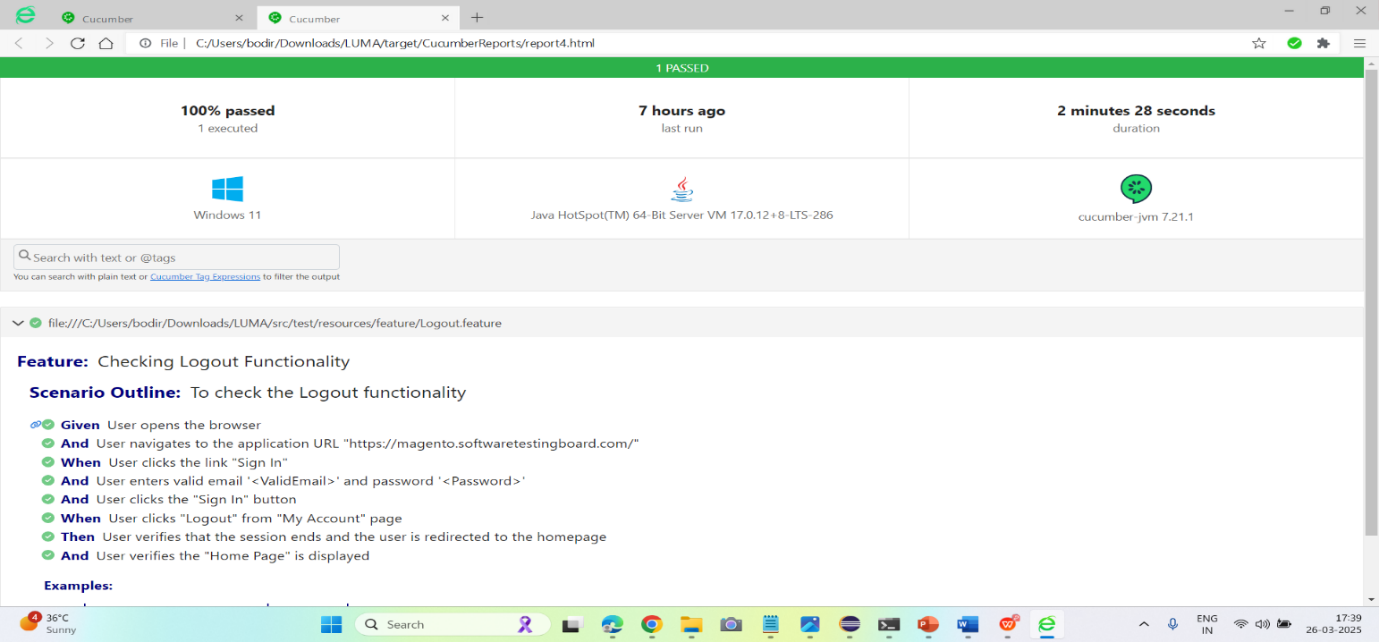
**Scenario 4:** Adding Multiple Products to Cart using Cucumber Framework, POM, Data properties, generate allure reports.

**OUTPUT:** Cucumber Reports:

****

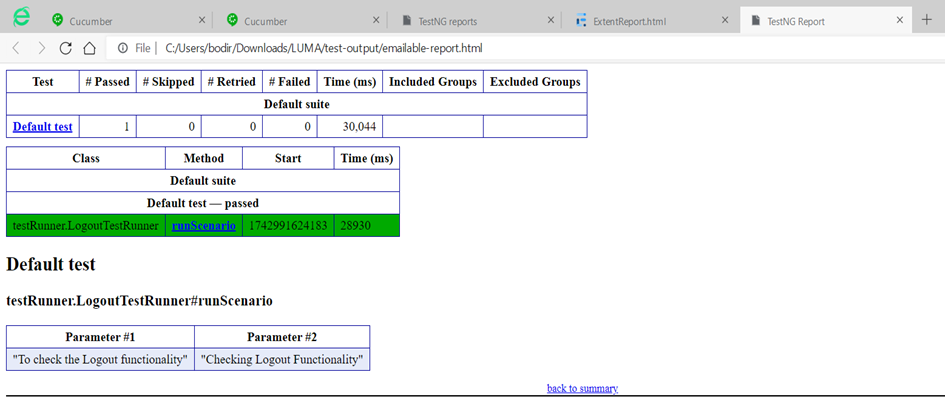
**Scenario 5**: Proceed to Checkout Using TestNG and generating allure reports.

**OUTPUT:** Cucumber Reports:



**Scenario 6:** Logout Using TestNG and generating allure report and Extent report.

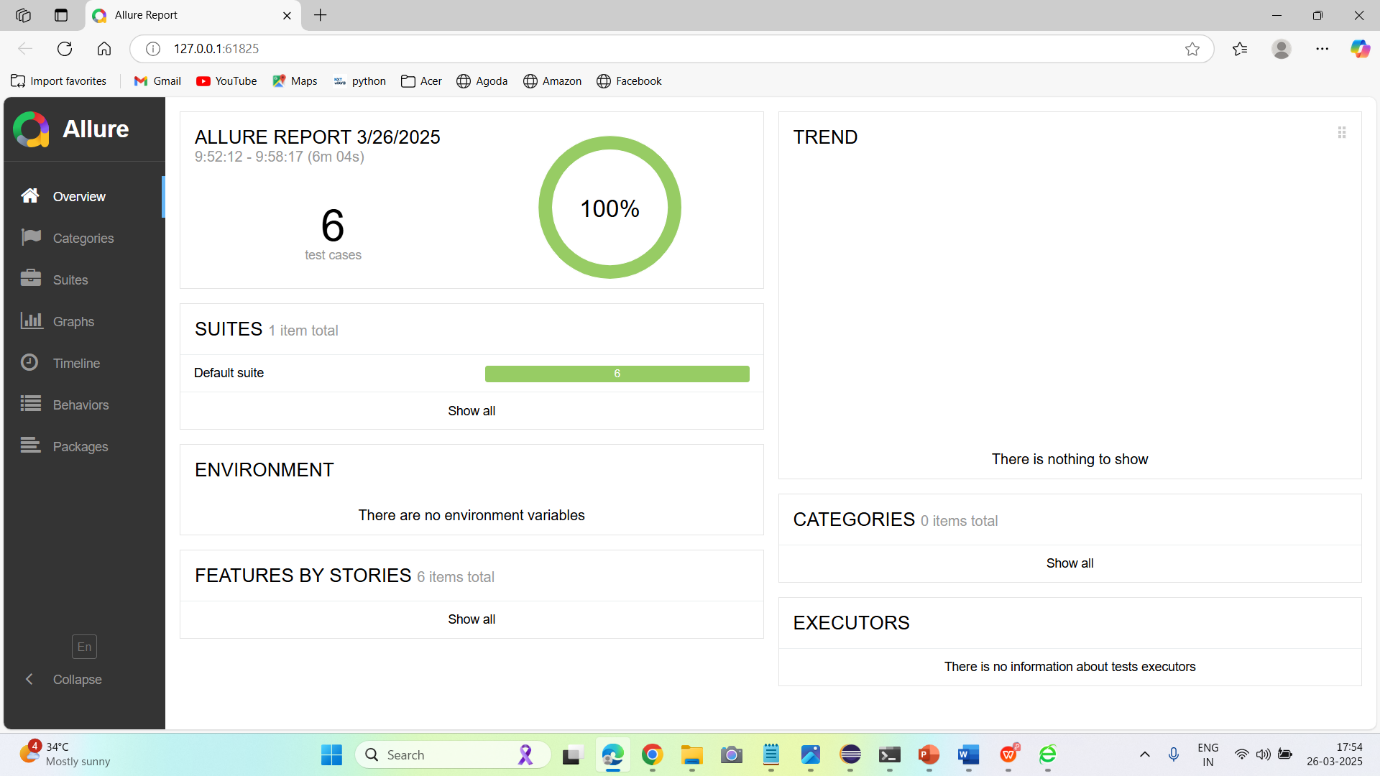
**OUTPUT:** TestNG Reports:



**Allure Report:**

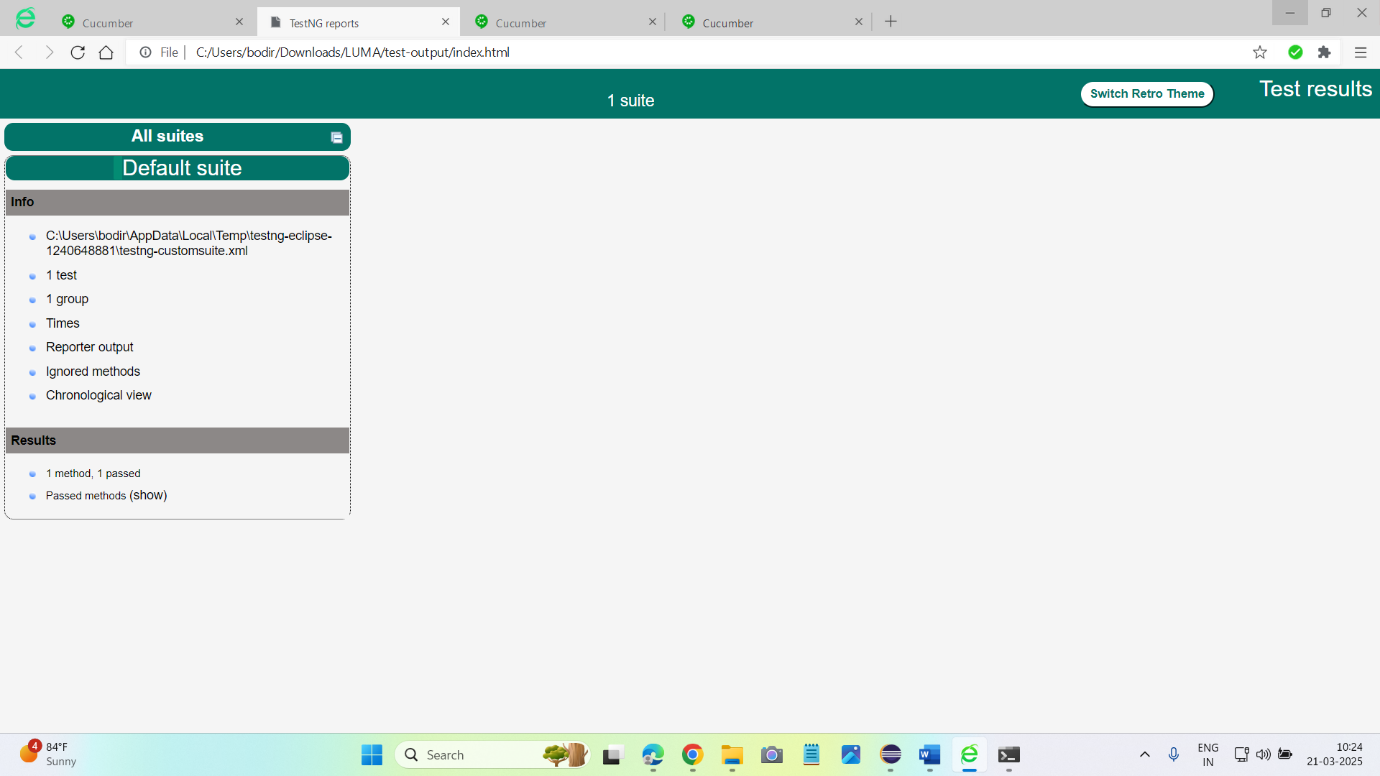
The allure-results folder contains **raw test execution data files** used to generate Allure Reports. These files are created after running your automated test cases with **Allure reporting integration** in your Maven-based Selenium TestNG framework.  
🡪 These files describe the **structure of test suites** and **test cases.** They represent how tests are organized and linked together in the report.  
🡪 These files hold **actual test results,** including:

* + Status (Passed/Failed/Skipped)
  + Execution time
  + Error messages (if any)
  + Links to attachments like screenshots or logs

****

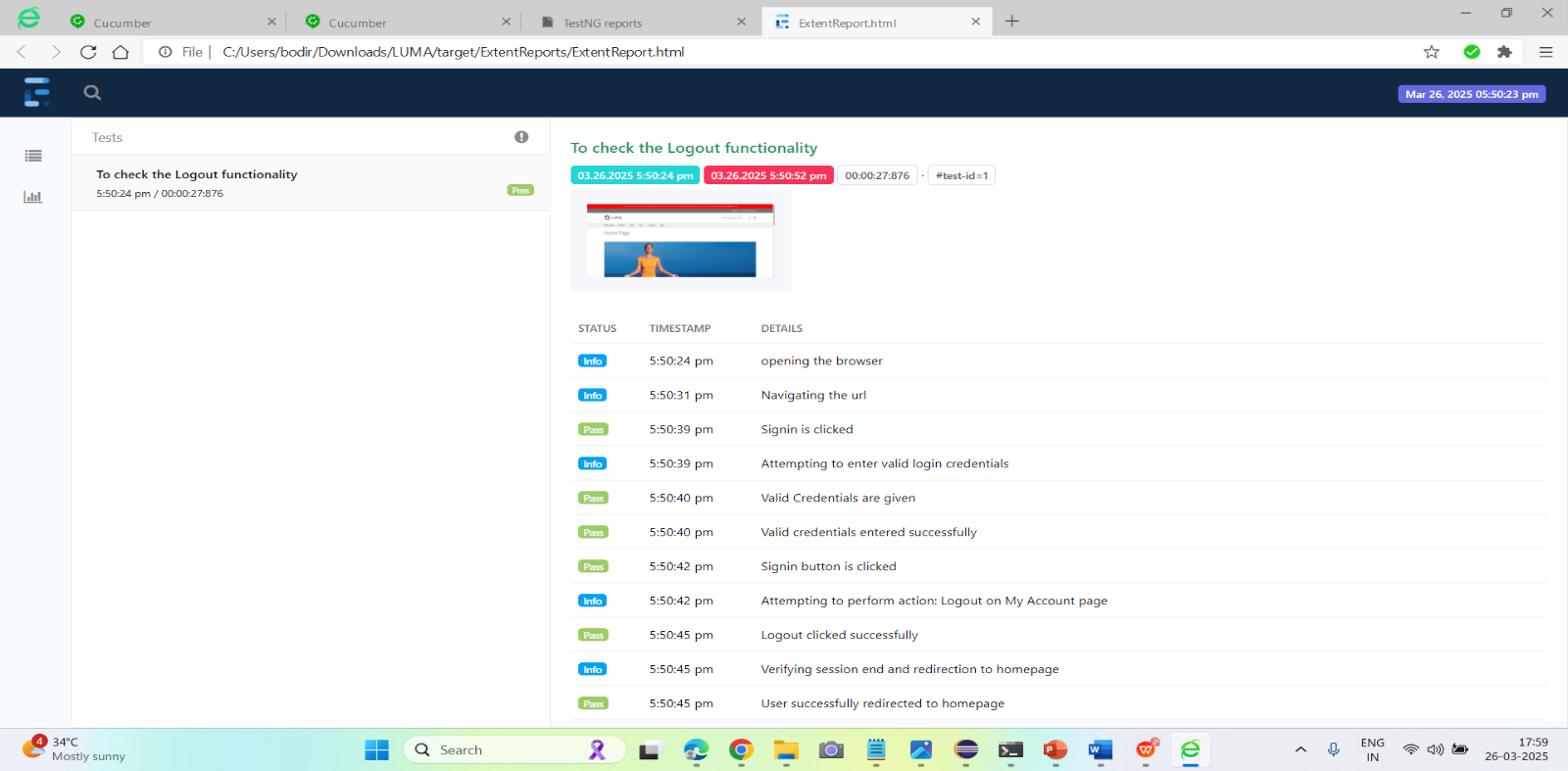
**TestNG-Report:**

Extent Report are integrated to provide detailed interactive and visually appealed reports.

****

**Extent Report:**

Extent Report are integrated to provide detailed interactive and visually appealed reports.

****

**Pom.xml:**

The pom.xml file is the **backbone of my Maven project**.

* Manages all the **dependencies** required for the project like Selenium, TestNG, Cucumber, Extent Reports, etc.
* Handles **build lifecycle, compilation**, and **execution** tasks.
* Defines **plugins** (if any), for example, Surefire for running TestNG suite.
* It helps me **automatically download and manage** library versions without adding jar files manually.
* I can easily **share** my project, and others just need to run **mvn** install to get everything working.
* It ensures **consistent builds,** making the project scalable and easy to maintain.

## **GitHub:**

## **Steps to Push Eclipse Project into GitHub**

### **Step 1: Initialize Git Repository in Eclipse**

1. Right-click on your project (e.g., Capstone Project) in the **Project Explorer**.
2. Navigate to:  
   Team 🡪 Share Project...
3. In the dialog box, select **Git** and click **Next.**
4. If you have an existing Git repository:
   * Select it from the list.  
     **OR**
   * Click **Create** to make a new repository inside the project directory.
5. Click **Finish.**

### **Step 2: Create a GitHub Repository**

1. Go to [GitHub](https://github.com).
2. Click on **New Repository.**
3. Enter the repository name (for example, CapstoneProject).
4. Click **Create Repository.**  
   we can now see the **remote repository URL** (e.g., https://github.com/yourusername/CapstoneProject.git

### **Step 3: Add GitHub Remote Repository in Eclipse**

1. In Eclipse, go to **Git Repositories** view:  
   Window 🡪 Show View 🡪 Other... 🡪 Git 🡪 Git Repositories
2. Right-click on **Remotes** under your local repo 🡪 Create Remote.
3. Name it origin 🡪 Click **OK.**
4. In **Configure Push:**
   * Click **Change...** next to URI.
   * Enter the **GitHub remote URL** (https://github.com/yourusername/CapstoneProject.git).
   * Enter your **GitHub username** and **personal access token (PAT)** (instead of a password).
   * Click **Next**, then **Finish.**

### **Step 4: Add Files to Git & Commit Changes**

1. Right-click on your project 🡪 Team 🡪 Commit....
2. Select the files to be committed or click **Select All.**
3. Write a **commit message** (e.g., "Initial Commit").
4. Click **Commit and Push....**

### **Step 5: Push Project to GitHub**

1. After clicking **Commit and Push**, the **Push Wizard** opens.
2. Ensure the **branch** is master or main.
3. Click **Next,** then **Finish.**
4. Eclipse pushes the project to GitHub.

## **Jenkins:**

## **Integrating Jenkins with a Local Project (Direct File Selection, Without Git)**

### **Step 1: Create a Freestyle Project in Jenkins**

1. Go to **Jenkins Dashboard** → Click **New Item.**
2. Enter a project name (e.g., CapstoneProject).
3. Select **Freestyle Project** → Click **OK.**

### **Step 2: Add an Advanced**

1. Click on Advanced.
2. Click on the use custom workspace.
3. Go to the eclipse and copy the path of the project.
4. Paste in the place of Directory.
5. Under **Source Code Management**, select **None.**

**Step 3: Add a Build Step for Your Local Project**

1. Under **Build** → Click **Add Build Step** → Select **Invoke top-level Maven targets.**
2. Select MAVEN\_HOME and the select the enter “test” in the Goals field.

### **Step 4: Post-build Actions**

* we can publish reports like testing, html.

### **Step 5: Save and Build**

1. Click **Save.**
2. Go to the job → Click **Build Now.**
3. Check **Console Output** for logs.

## **Integrating Jenkins with Git (Local Git / GitHub)**

### **Step 1: Create a Freestyle Project for Git**

1. Jenkins Dashboard → **New Item.**
2. Enter a job name → Select **Freestyle Project → OK.**

### **Step 2: Configure Source Code Management**

1. Under **Source Code Management,** select **Git.**
2. **Repository URL** (for local repo):

* file:///C:/Users/HP/eclipse-workspace/CapstoneProject/.git

1. Add **Credentials** (GitHub username and PAT).

**Step 3: Add a Build Step for Your Local Project**

1. Under **Build** → Click **Add Build Step** → Select **Invoke top-level Maven targets.**
2. Select MAVEN\_HOME and the select the enter “test” in the Goals field**.**

**Step 7: Add Post-build Actions (Optional)**

* We can publish reports like testing, html.

### **Step 8: Save and Build**

1. Save the job.
2. Click **Build Now** to trigger manually.
3. Check **Console Output** to verify the process.

**Synchronization:**

**Implicit Wait**

* **Definition:** It tells WebDriver to wait for a certain amount of time before throwing a NoSuchElementException. It applies to all elements in the script.
* **Scope:** Applied globally to the entire WebDriver session.
* **Default Behaviour:** If the element is found within the given time, WebDriver proceeds immediately without waiting for the full duration.
* **Drawback:** Cannot define different wait conditions for different elements.

**Explicit Wait**

* **Definition:** It tells WebDriver to wait for a specific condition (like element visibility, Clickability etc.) before proceeding.
* **Scope:** Applied to a specific element, not globally.
* **Flexible:** Can define different wait conditions for different elements.
* **Best Practice:** Preferred over Implicit Wait for handling specific scenarios.

**TestNG Annotations Used:**

* + @BeforeSuite – Initializes Extent Report.
  + @BeforeMethod – Sets up WebDriver before each test.
  + @Test – Executes the test case for account creation.
  + @AfterMethod – Closes browser after test execution.
  + @AfterSuite – Flushes the Extent Report.

**Conclusion:**

In this project, I have successfully implemented automation testing for the **MAGENTO LUMA** web application using Selenium WebDriver, TestNG, Cucumber, and Maven. The framework follows the Page Object Model (POM) design pattern for better code reusability and maintainability. Different testing scenarios, such as user registration, login, add to cart, Adding multiple products to the cart, checkout, and Logout, were automated and validated. Synchronization techniques like Thread.sleep(), Implicit Wait, and Explicit Wait were applied to handle dynamic web elements efficiently. Extent Reports were used to generate detailed test execution reports, and screenshots were captured for better analysis. This project demonstrates the capability to build an efficient and scalable automation framework for e-commerce applications.