**CAPSTONE PROJECT- AUTOMATION TESTING ON**

**DEMO-WEBSHOP-TRICENTIS**

**Problem Statement:**  
Demo Web Shop Tricentis is an online shopping platform where users can explore and purchase a wide range of products, including electronics, apparel, and books. Users can navigate through various categories, view detailed product information, add products to their cart, and proceed with order placement. Automating these key workflows is crucial to validate the platform's efficiency, accuracy, and reliability during the testing process, ensuring a seamless and error-free user experience.

**Objective**:

The objective of this project is to automate the user workflows of the Demoblaze online shopping platform using Selenium,TestNG,Page Object Model and Page Factory and BDD Cucumber.This includes automating tasks such as viewing product details,adding products to the cart,placing orders.The aim is to enhance the robustness of the testing process,reduce manual effort,and ensure a seamless user experience.

**Introduction**:

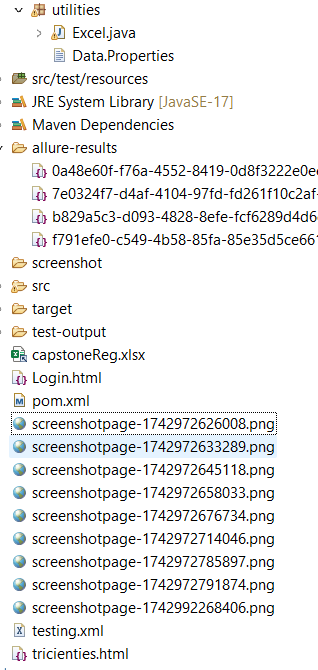
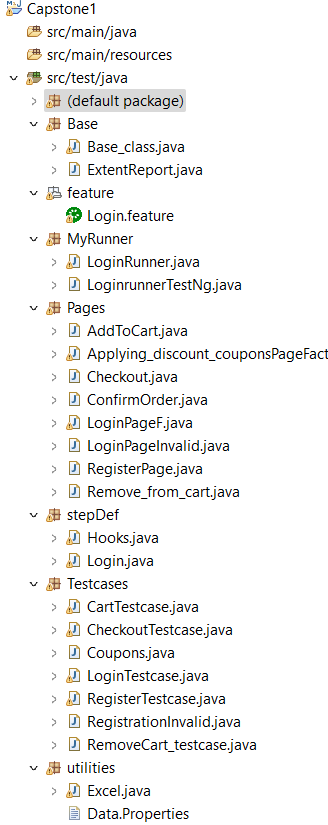
The Capstone Project is an automated testing framework built using Selenium,BDD Cucumber and TestNG. It follows the Page Object Model (POM) to structure test scripts efficiently. The project automates the testing of the “Demowebshop-tricentis” web application, handling functionalities like login, sign-up, cart operations, and reporting.

**Key Features:**

* Page Object Model (POM) for maintainability.
* Cucumber BDD Framework for test case execution.
* TestNG for test execution and reporting.
* Selenium WebDriver for browser automation.
* Extent Reports for test reporting.
* Maven for dependency management.
* Java
* Git/Github
* Jenkins

**Project Structure**

Project <url:https://www.demowebshop-tricentis.com/>



**2.** **Package summaries**

* **Base (Package):**

**1. BaseClass.java**

* Initializes **WebDriver** and browser configuration by using @parameter.
* Manages common setup and teardown functions.

**2. ExtentReport.java**

* Generates **automated test reports** using Extent Reports.
* Captures logs and screenshots for failed test cases.
* **Pages** (Page Object Model )- POM Classes:

**1. LoginPage.java**

* Handles user **Invalid Login functionality**.
* Contains methods to enter a username, password, and submit the form.
* Implements **element locators** using locators (By.id() ,By.className() andBy.xpath())

**2. RegisterPage.java**

* Implements **user authentication methods**.
* Interacts with **Register page elements** such as input fields and buttons.
* Validates **successful registrations**

**3. CartPage.java**

* Automates **shopping cart operations**.
* Handles **adding and removing products** from the cart.
* Verifies **cart contents and checkout functionality**

**4. LoginPageFactory.java**

* Implements **Page Factory Model** for Valid Log in Functionalities functionality.
* Uses @FindBy annotations for better **element handling**.
* Improves **Test script efficiency and maintainability**.

**5. ApplyingCoupons.java**

* **Purpose:** Validates the **Coupon Application Functionality**.
* Applying valid, invalid, and expired coupons.
* Ensuring that successful coupon application adjusts the cart total accordingly.
* Verifying error messages for invalid/expired coupon entries.
* Implements data-driven testing and integrates exception handling for dynamic content and flaky elements.

**6. ConfirmPage.java**

**Purpose:** Validates the order confirmation page after successful checkout.

* Verifies order confirmation message and order summary details (item names, prices, discounts, and total).
* Confirms correct display of payment, shipping information, and order number.
* Implements exception handling for flaky elements and dynamic content.

**7. RemoveFromCart.java**

**Purpose:** Automates and validates the "Remove from Cart" functionality.

* Tests single item, partial, and full cart removals, verifying cart updates accordingly.
* Validates UI feedback (e.g., loading spinners or success messages).
* Implements dynamic locators and handles exceptions like **StaleElementException**.
* Verifies appropriate error messages if issues occur (e.g., network failure).

**8. CheckoutPage.java**

**Purpose:** Automates and validates the checkout process.

* + Tests the entire checkout flow (cart review, address selection, payment, and order review).
  + Verifies order total calculations, shipping methods, and address fields.
  + Implements data-driven testing and dynamic locators for flexible test scenarios.
  + Exception handling for **TimeoutException** and **NoSuchElementException**.
  + Logs all steps and integrates with **Allure** or **Cucumber Reports** for detailed execution results.
* **utils (Utilities):**

1.data.properties(file) - Stores application URLs, user credentials, and

configurations.

2 ExcelReader.java

Reads and writes test data from Excel files for data-driven testing.

**Key Features:** Uses Apache POI to extract specific rows, columns, and cells

from .xls/.xlsx files.

**Reusable Methods:** Supports fetching multiple sets of input data dynamically.

* **features (Cucumber Feature File):**

**1.Login.feature:**

* A **Cucumber feature file** that defines **Login scenarios** in Gherkin syntax.
* Includes test cases like **successful Login** and **valid input validation**.
* **MyRunner (Cucumber Test Runner):**

**1. LoginRunner.java**

* Executes Cucumber feature files using @CucumberOptions.
* Generates Cucumber HTML and JSON reports.

2. **LoginTestNg.java**

* Executes login-related test cases using **TestNG** framework.
* Utilizes **@Test**, **@BeforeMethod**, and **@AfterMethod** annotations for setup, test execution, and teardown.
* Generates detailed logs and integrates with **TestNG Reports** for analyzing test results.
* **stepDef (Step Definitions for Cucumber):**

**1.LoginStepDef.java**

* Contains step definitions for Cucumber test scenarios.
* Maps Gherkin steps to Java methods.
* Uses TestNG assertions to validate results.

2. **Hooks.java**

* **Purpose:** Manages setup and teardown for Cucumber test scenarios.
* **Annotations:** Uses **@Before** and **@After** hooks to execute pre- and post-conditions (e.g., browser setup, closing).
* **Reusable Methods:** Defines methods for common tasks like initializing WebDriver, clearing cookies, and logging.
* **Scenario Tagging:** Supports conditional execution based on scenario tags (e.g., **@Smoke** or **@Regression**).
* **TestCase (Test Scripts - TestNG Based):**

**1. RegisterTestcase.java**

* A **TestNG test class** for validating sfunctionality.
* Calls methods from **RegisterPage.java** for test execution.

**2. RegistrationInvalid.java**

* Validates error messages for failed registration attempts.
* Uses data-driven testing to handle multiple invalid input sets.
* Integrates TestNG assertions to verify registration failures.

**3.LoginTestcase.java:**

* Logs in with a valid username and password.
* Validates login success.

**5.CartPageTestcase.java:**

* Adds an item to the cart.
* Validates if the item appears in the cart.

**6. Coupons.java**

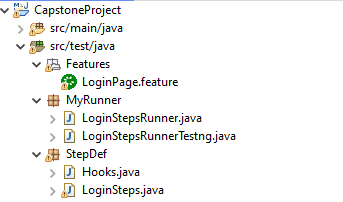
* + - Verifies cart total adjustments after applying coupons.
* Validates error messages for invalid or expired coupon entries.
* Implements data-driven testing for multiple coupon combinations.
* Logs test steps and generates execution reports.

**7. RemoveCart\_testcase.java**

* Tests removing single, multiple, and all items from the cart.
* Validates cart updates (item count, total price) after removal.

**8. CheckoutTestcase.java**

* Validates address selection, payment options, and order review details.
* Verifies order total calculations, shipping fees, and discounts.
* Checks successful order placement and order confirmation message.

 **3.Cucumber Framework for LoginInvalid Page**

**4. Other Important Files and Folders**

* src/test/resources - Stores test data, configurations.
* JRE System Library [JavaSE-17] - Java version used.
* Maven Dependencies - Required dependencies (Selenium, Cucumber, TestNG, Extent Reports).
* TestNG & JUnit 4 – Testing & Cucumber frameworks.
* Screenshots Folder - Stores screenshots of test cases.
* test-output Folder - Contains TestNG reports.
* xml Folder - May store TestNG XML configuration files.
* demoblaze.html - Local HTML file (results of extent reports).
* pom.xml - Maven configuration file managing dependencies.

**5. Execution Instructions**

**Run Cucumber Tests**

**Run from Eclipse:**

* Right-click SignUpRunner.java → Run As → JUnit Test.

2. **Run TestNG Tests**

**Run from Eclipse:**

* Right-click SignUpTest🡪Run AS 🡪TestNg Suite

Open test-output/index.html to check the **TestNG Reports**.

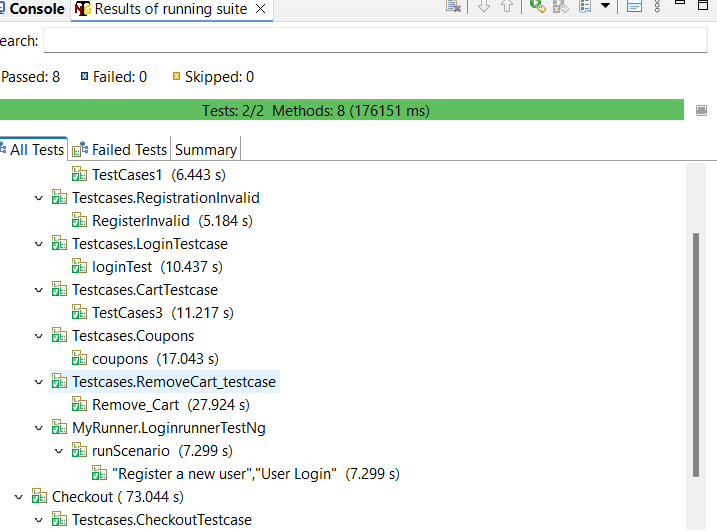
**6.Xml files**

* An xml folder has been created in the CapstoneProject directory to store all TestNG configuration files.
* It centralizes test suite files, improving project organization and structure.
* This setup simplifies test execution by maintaining a clean and streamlined directory hierarchy.
* It enhances maintainability by making configuration files easily accessible and manageable.
* The organized approach supports better scalability and efficient project workflow.

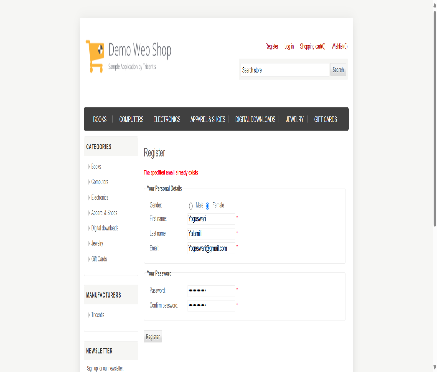
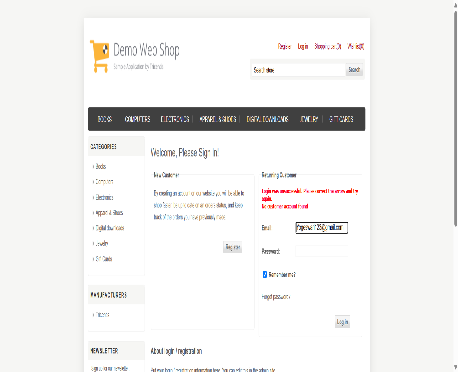
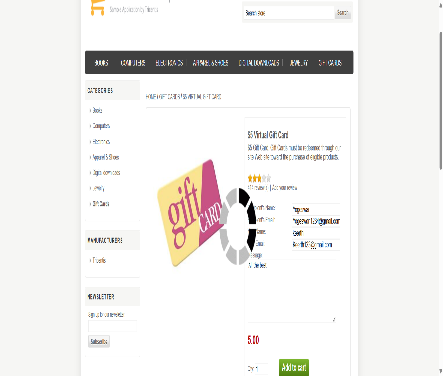
**Parallel Execution in xml**



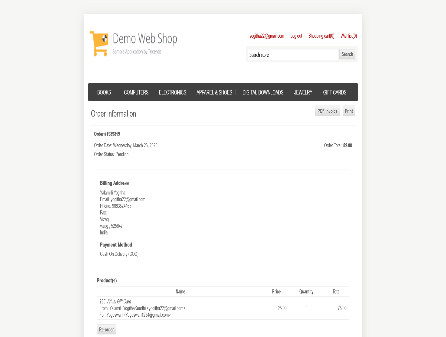
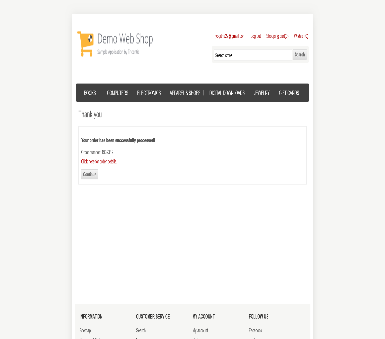
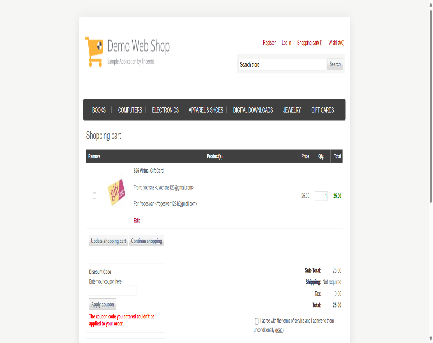
**Output of Xml**



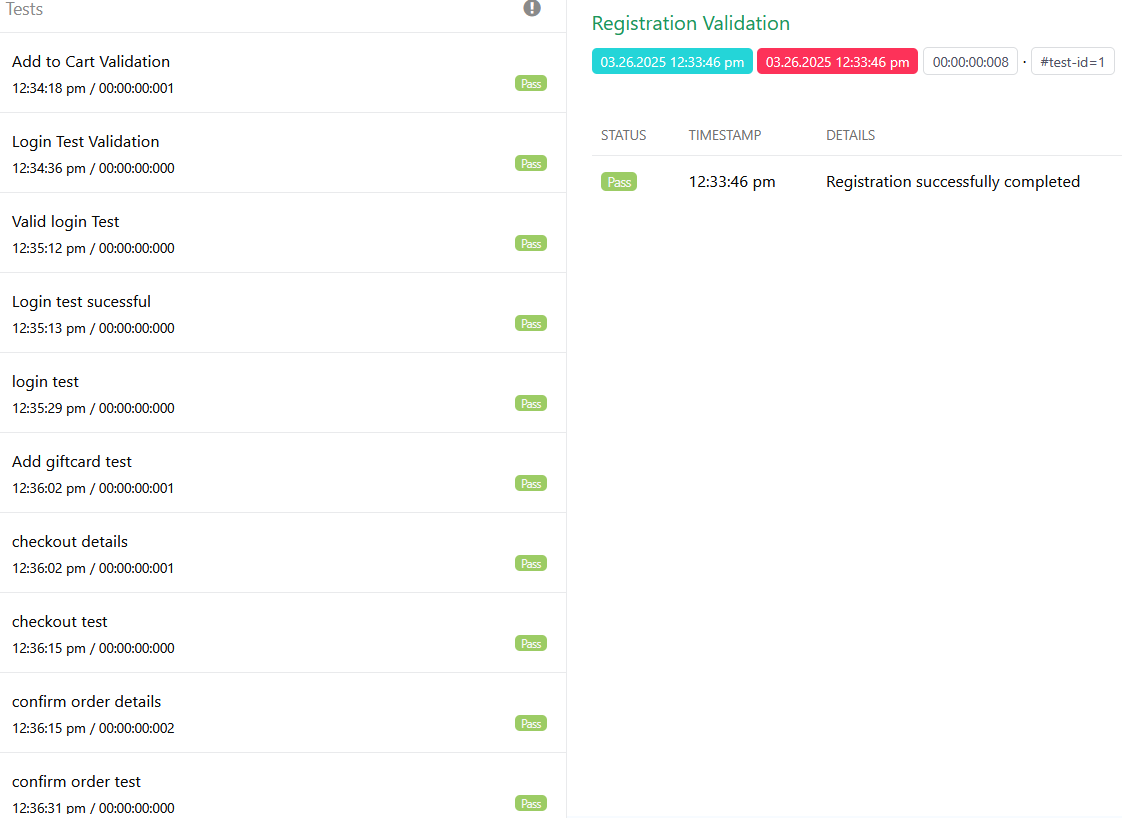
**7.Reports/OutPuts**

****

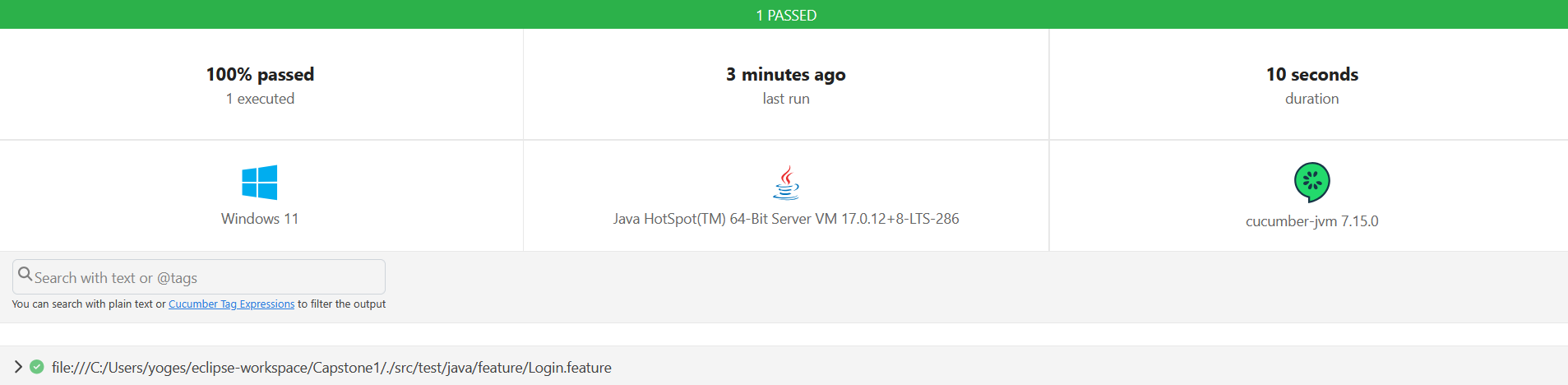
**Screenshots of Login page Register page AddToCart Page**

** InvalidCoupons confirmOrder checkout**

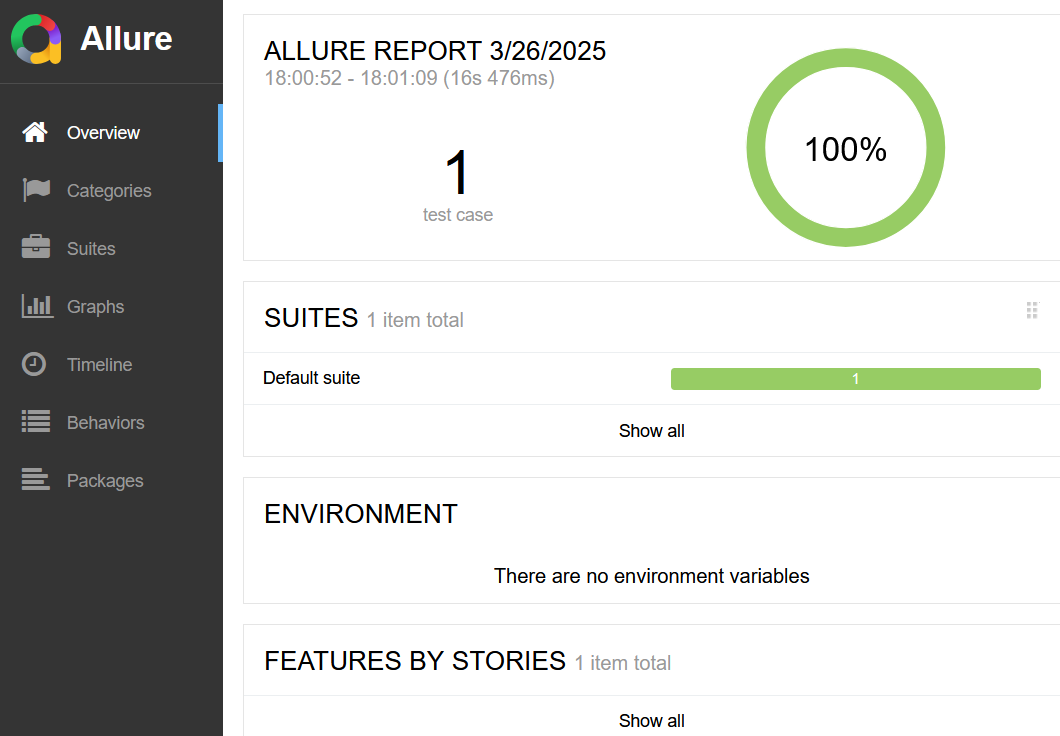
**ExtentReport: TestNg Testcases**

****

**HTML report: Cucumber framework for login page**

****

**AllureReport for CucumberFramework**

****

**8.Challenges Faced**

**1.Web Element Synchronization**

* + Handling **dynamic elements** and waiting strategies ( Thread.Sleep,Explicit & Implicit Waits,FluentWait).
  + Managing page load times and asynchronous operations.

**2.Cross-Browser Compatibility**

* + Ensuring tests run consistently across **Chrome, Firefox, and other browsers**.
  + Managing browser-specific behaviors and driver compatibility issues.

**3.Exception Handling**

* Managing **unexpected pop-ups, timeouts, and element not found exceptions**.
* Implementing robust try-catch blocks for smooth test execution.

**4.Data-Driven Testing**

* Implementing **parameterized tests using external data sources** (e.g., properties files).
* Managing test data efficiently for **multiple test scenarios**.

**5.Integration & Reporting**

* Generating **Cucumber and Extent Reports** with clear and detailed logs.
* Debugging test failures based on captured **screenshots and logs**.

**6.Scalability & Maintainability**

* Structuring the framework to support **future test cases** with minimal code changes.
* Managing large test suites efficiently with modularized code.

**9. Conclusion**

* **Capstone Project** integrates **Java, Selenium WebDriver, TestNG, and Cucumber** for structured and efficient test automation.
* It follows the **Page Object Model (POM)** to enhance **maintainability, reusability, and scalability**.
* Key components like **feature files, step definitions, and test runners** automate functional testing seamlessly.
* **Extent Reports and screenshots** improve test execution transparency and debugging.
* The framework enhances **test efficiency, reduces redundancy, and ensures high test coverage**