**Capstone-Project**

**Name: Atif Ali Khan**

**Email:** [**atifalikhankhan4173@gmail.com**](mailto:atifalikhankhan4173@gmail.com)

**Batch: 4**

**Website Name: Demo Web Shop**

**Website Url:** [**https://demowebshop.tricentis.com/**](https://demowebshop.tricentis.com/)

**GitHub Url:** [**https://github.com/atifalikhan1946/capstone-webShop.git**](https://github.com/atifalikhan1946/capstone-webShop.git)

## **1. Introduction**

This project automates various functionalities of the Demo Web Shop website using Selenium WebDriver, TestNG, and Cucumber. The goal is to ensure smooth user interactions, including user registration, login, book purchase, cart operations, and filtering options.

## **2. Problem Statement**

Manual testing is time-consuming and prone to human errors. Automating these tests:  
- Reduces repetitive efforts.  
- Ensures faster execution.  
- Increases test coverage.  
- Provides quick feedback on software stability.

## **3. Tools & Technologies Used**

- Programming Language: Java 17

- Automation Framework: Selenium WebDriver

- Testing Frameworks: TestNG, JUnit

- Behavior-Driven Testing (BDD): Cucumber

- Build Tool: Maven

- Reporting Tools: Extent Reports, Allure Reports

- IDE: Eclipse / IntelliJ

- Data Handling: Excel, Properties Files

- Version Control / CI-CD: GitHub / Jenkins

- Cross-Browser Testing: Chrome & Firefox

**4. Project Objective**

The objective of this project is to automate the testing of key functionalities of an e-commerce Demo Web Shop website. The project covers the following test cases:

* **User Registration**
* **User Login**
* **Adding Books to Cart**

## **5. Project Structure**

The project is organized into multiple packages and classes for better maintainability and readability.

src/test/java

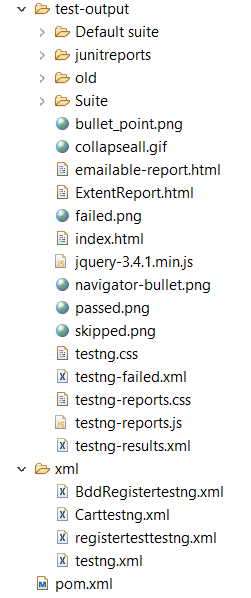
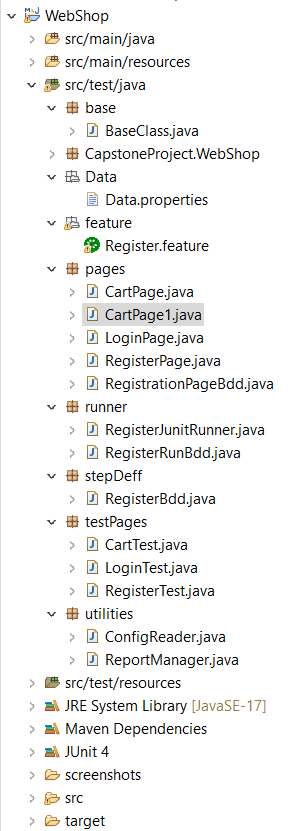
|── base # Base class for WebDriver initialization

|── pages # Page Object Model (POM) classes for UI interactions

|── testpages # Test classes for different scenarios

|── utilities # Utility classes (configuringreader, Extent Report generator)

|── data #here we have the properties file



**6. Key Components**

**6.1. Base Class**

* The BaseClass.java handles the WebDriver initialization and browser configurations.

1. Browser Initialization (invokeBrowser)

* Accepts a browser name as input (chrome, firefox, or edge).
* Launches the specified browser using WebDriver and maximizes the window.
* Displays an error message for invalid browser names.

2. Screenshot Capture (screenshot)

* Takes a screenshot of the current page using TakesScreenshot.
* Saves the screenshot in the ./screenshot/ folder with a timestamped filename.

**6.2. Utility Classes**

* **ConfigReader.java:**
  + Used for reading test data from an properties file.
* **ExtentReport.java:**

The ExtentReport.java file is responsible for generating detailed HTML reports for the test execution.

* **Report Initialiation:**
  + Uses ExtentReports and ExtentSparkReporter to generate reports.
* **Report Creation:**
  + Creates a new HTML report named ExtentReport.html.
  + Adds test cases dynamically to the report using createTest().
* **Report Attachment:**
  + Attaches the sparkReporter to the ExtentReports instance.
  + Allows tracking of test execution status with visual logs.

**7. Test Cases**

**7.1. User Registration Test**

**Class:** RegisterTest.java  
**Description:** Automates the registration process using Selenium WebDriver ,testing and Page Object Model.  
**Steps:**

1. **Reading Properties File:**

* The @BeforeTest method reads the **URL** from the data.properties file using FileInputStream

2.**Launching the Browser:**

* The @Test method uses the @Parameters annotation to run on different browsers.
* It invokes the browser, navigates to the URL, and applies **implicit wait** for stable execution.

3.**Registering a User:**

* Uses the RegisterPage class to interact with the **registration page** elements.
* Reads **user details** (first name, last name, email, password, and confirm password) from an properties file.
* Fills the form and clicks the **register button**.

4. **Logging Execution Status:**

* The test logs execution status using **ExtentReport**, marking the test as **PASS** if registration is successful.

5.**Closing the Browser:**

* The @AfterTest method closes the browser using driver.quit() and flushes the Extent Report data.

**7.2. User Login Test**

**Class:** LoginPageTest.java  
**Description:** Automates the login process.  
**Steps:**

1. **Reading Properties File:**

* The @BeforeTest method reads data from the data.properties file using FileInputStream.
* It extracts the **URL, email, and password** for the test and initializes

2. **Launching the Browser:**

* The @Test method uses the @Parameters annotation to run the test on different browsers.
* The invokeBrowser() method opens the specified browser and navigates to the URL.
* It applies **implicit wait** for smooth execution.

3.**Performing Login Actions:**

* Uses LoginPage to interact with the **login page** elements.
* Enters the **email and password**, clicks and logs in.

4. **Validating and Logging Test Status:**

* Uses ExtentReport.createTest() to log the execution status as **PASS**.
* Sleeps for 2 seconds to allow the actions to complete.

5. **Closing the Browser:**

* The @AfterTest method closes the browser using driver.quit() and flushes the Extent Report data.
* This ensures proper cleanup after the test execution.

**7.3. Add to Cart Test**

**Class:** CartTest.java  
**Description:** The CartTest.java file contains the test cases to automate the "Add to Cart" functionality for electronis products using Selenium and TestNG.  
**Steps:**

1. **Property File Reading:**
   * Reads the application URL from the data.properties file using FileInputStream and Properties class.
   * Initializes the Extent Report instance for test logging.
2. **Browser Launching:**
   * Uses the @Parameters annotation to run tests on multiple browsers.
   * Opens the specified browser and navigates to the given URL.
3. **Homepage Navigation:**
   * Instantiates the HomePage class.
   * Clicks on the **Electronics** section.
4. **Product Filtering & Sorting:**
   * Interacts with the AddToCartElectronicsPage to:
     + Sort products by **Name: A-Z**.
     + Display **4 products per page**.
     + Apply filters and switch the view.
5. **Add to Cart Functionality:**
   * Adds a cell phone product to the cart.
   * Verifies the **success message** and logs the result in the Extent Report.
6. **Browser Cleanup & Report Generation:**
   * Closes the browser after the test execution.
   * Flushes the Extent Report to save the results.

**7.4 Logging out after valid login**

Description: Automates the Logout Functionality.

**Steps:**

**1.Test Setup and Configuration:**

Loaded the URL, email, and password from the data.properties file and initialized the ReadMethods class for test execution.

**2. Browser Invocation and Navigation:**

Launched the browser using TestNG parameters, navigated to the provided URL, and applied implicit wait for stability.

**3.Login with Valid Credentials:**

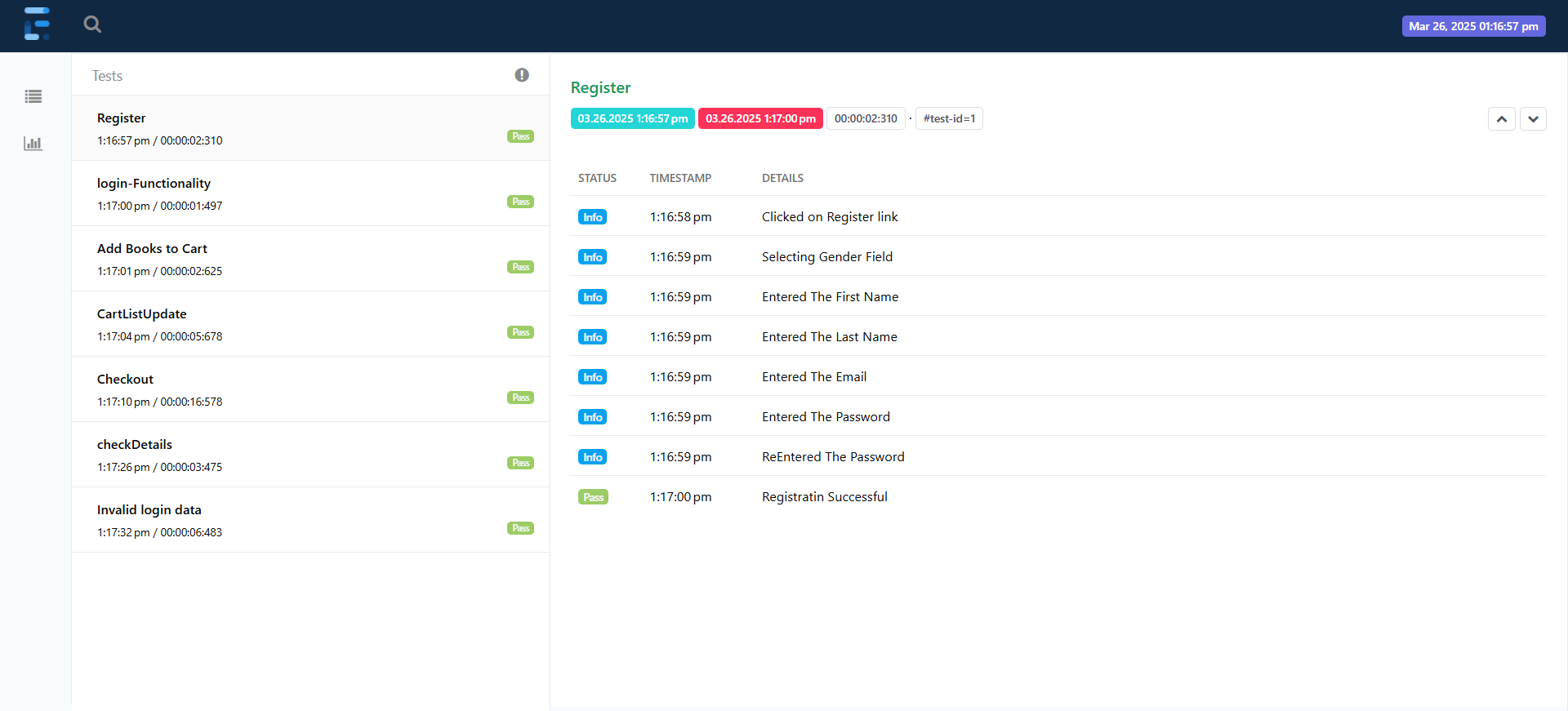
Used the LoginMethod from the ReadMethods class to log in with valid credentials and verified the login process.

**4.Logout Functionality:**

Automated the logout process by interacting with the HomePage class, capturing a screenshot upon successful logout.

**8. Reporting**

* **Extent Reports**:
  + The project uses Extent Reports to generate detailed and customizable test reports.
  + Each test execution generates a new report with the pass/fail status, execution time.



**9. Execution Flow**

1. **Execution :**
   * The project uses TestNG for test execution, with parameters for browser configuration.
   * Execute with .xml file.
2. **Reports:**
   * After execution, Extent Reports are generated in the test-output folder.
   * Reports include details of each test case, execution time.

**10. Challenges and Solutions**

* **Synchronization Issues:**
  + Handled using Thread.sleep() and driver.manage().timeouts().implicitlyWait() for stability.

**11. Conclusion**

This Selenium-based Java automation project demonstrates the ability to perform end-to-end testing of a Demo Web Shop e-commerce website. It effectively uses TestNG, Selenium WebDriver, and Extent Reports to automate and validate key functionalities. The project is modular, easy to maintain, and generates detailed reports for efficient debugging and test case validation.