**Demoblaze-Capstone Project**

**By Chandrika David**

**1. Introduction**

This document provides a detailed overview of the **Demoblaze Automation Project**, which is built using **Selenium WebDriver, TestNG, Cucumber, and the Page Object Model (POM)**. The goal is to **automate and validate** core functionalities of the **Demoblaze e-commerce website (**[**https://www.demoblaze.com**](https://www.demoblaze.com)**).**

**Key Objectives:**

* Automate end-to-end e-commerce workflows (**Login, Signup, Cart, Checkout**)
* Enhance **test efficiency** and **maintainability** using **Page Object Model (POM)** and **Page Factory**.
* Implement **data-driven testing** with **Excel (Apache POI).**
* Support **parallel execution** using **ThreadLocal WebDriver** and **TestNG XML**.
* Generate detailed reports using **Extent Reports, Allure Reports and Cucumber Reports**.
* Integrate with **Jenkins** for CI/CD (**Continuous Integration and Continuous Deployment/Delivery)** execution.

**2. Tools & Technologies Used**

| **Category** | **Tools & Technologies** |
| --- | --- |
| **Programming Language** | Java |
| **Automation Tool** | Selenium WebDriver |
| **Testing Framework** | TestNG, Cucumber |
| **Design Pattern** | Page Object Model (POM), Page Factory |
| **Build Tool** | Maven |
| **Version Control** | Git |
| **Reporting Tools** | Extent Reports, Cucumber Reports, Allure Reports |
| **Data Handling** | data.properties, Excel (Apache POI) |
| **Browser Drivers** | ChromeDriver, EdgeDriver, FirefoxDriver |
| **CI/CD Tool** | Jenkins |
| **Execution Management** | TestNG XML for parallel execution |

**3. Project Structure**

Wipro\_Capstone\_Project/

│── src/main/java

│ ├── base/

│ │ ├── BaseClass.java

│ │ ├── data.properties

│ │

│ ├── features/

│ │ ├── signup.feature

│ │

│ ├── pages/

│ │ ├── AboutUsPage.java

│ │ ├── CartPage.java

│ │ ├── CategoryPage.java

│ │ ├── ContactPage.java

│ │ ├── LoginPage.java

│ │ ├── SignUpPage.java

│ │

│ ├── utilities/

│ │ ├── ExtentReportManager.java

│ │ ├── ExcelUtils.java

│

│── src/test/java

│ ├── hooks/

│ │ ├── Hooks.java

│ │

│ ├── MyRunner/

│ │ ├── signUpRunner.java

│ │

│ ├── stepDefinitions/

│ │ ├── SignupSteps.java

│ │

│ ├── TestCases/

│ │ ├── AboutUsTest.java

│ │ ├── CartTest.java

│ │ ├── CategoryTest.java

│ │ ├── ContactTest.java

│ │ ├── LoginTest.java

│

│── reports/

│── screenshots/

│── target/

│── test-output/

│── pom.xml

│── testing.xml

**Explanation of Key Directories:**

* **base/** - Contains BaseClass.java for browser setup, teardown, and data.properties and excel sheet for test data.
* **features/** - Holds Gherkin feature files (signup.feature implemented using BDD).
* **pages/** - Implements Page Object Model:
  + **Signup Page:** Uses **BDD with Cucumber**.
  + **About Us, Contact, Category, and Login Pages:** Use **Page Factory (PF)**.
  + **Cart Page:** Uses **POM without Page Factory**.
* **utilities/** - Contains ExtentReportManager.java for test reporting.
* **hooks/** - Includes Hooks.java for test setup and teardown.
* **MyRunner/** - Contains Cucumber test runners (signUpRunner.java).
* **stepDefinitions/** - Step definitions mapping Gherkin steps to Java methods.
* **TestCases/** - Contains TestNG test classes for validation (LoginTest.java, etc.).

**4. Key Automation Concepts & Techniques**

**4.1 Wait Mechanisms**

* Thread.sleep() → Pauses execution for a fixed time.
* Implicit Wait → Waits for elements globally before throwing an error.
* Explicit Wait → Waits for a specific condition before proceeding.
* Fluent Wait → Polls for an element at regular intervals until a timeout.

**4.2 Exception Handling**

* StaleElementReferenceException → Handled using explicit waits or retry mechanisms.
* NoSuchElementException → Handled using presenceOfElementLocated().
* TimeoutException → Handled by increasing explicit wait time dynamically.

**4.3 Web Element Interactions**

* Element is Clickable → Ensures the element is in an actionable state before clicking.
* Element is Located → Ensures an element is present in the DOM before interacting.
* visibilityOfElementLocated() → Waits until an element is visible.
* stalenessOf() → Waits until an element is removed from the DOM.
* presenceOfElementLocated() → Waits until an element is present.

**4.4 Page Object Model (POM) & Page Factory**

* Page Factory (@FindBy annotations) → Efficiently initializes web elements.
* POM Design Pattern → Enhances maintainability by separating test logic from UI interactions.

**4.5 Alerts Handling**

* JavaScript alerts are managed using Alert alert = driver.switchTo().alert();

**4.6 Assertions**

* Soft Assertions (does not stop execution upon failure).
* Hard Assertions (stops execution on failure).

**4.7 Cucumber Features**

* Hooks (@Before & @After) → Handles test setup and teardown.
* Tags (@Regression, @SmokeTest) → Categorizes test execution.

**5. Parallel Execution Setup**

Parallel execution is achieved using ThreadLocal WebDriver to manage separate browser instances.

**TestNG XML Configuration (testng.xml):**

<suite name="ParallelSuite" parallel="tests" thread-count="2">

<test name="ChromeTest">

<parameter name="browser" value="chrome"/>

<classes>

<class name="TestCases.LoginTest"/>

</classes>

</test>

<test name="FirefoxTest">

<parameter name="browser" value="firefox"/>

<classes>

<class name="TestCases.CartTest"/>

</classes>

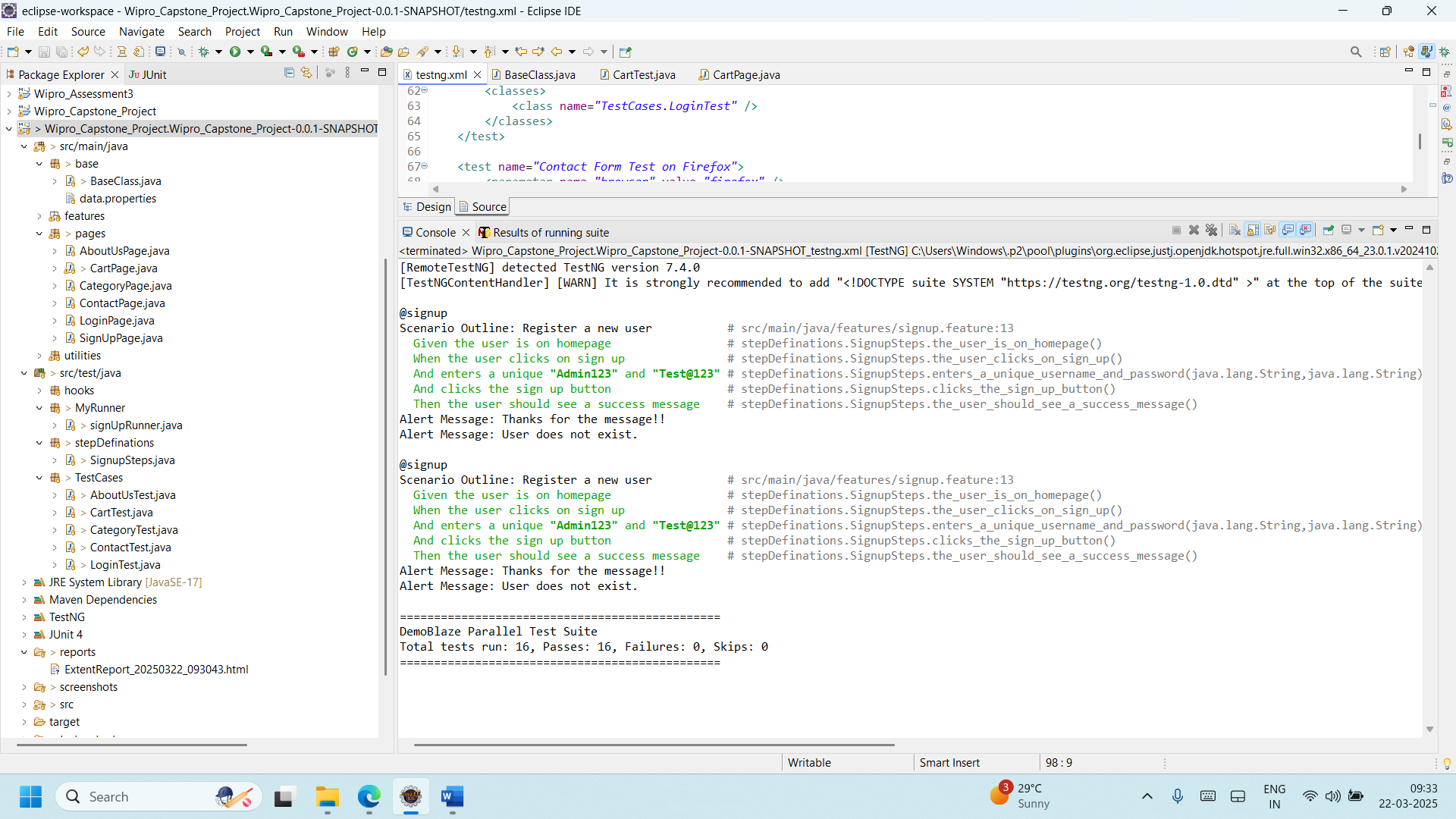
</test>

</suite>

**6. Test Execution Flow & Management**

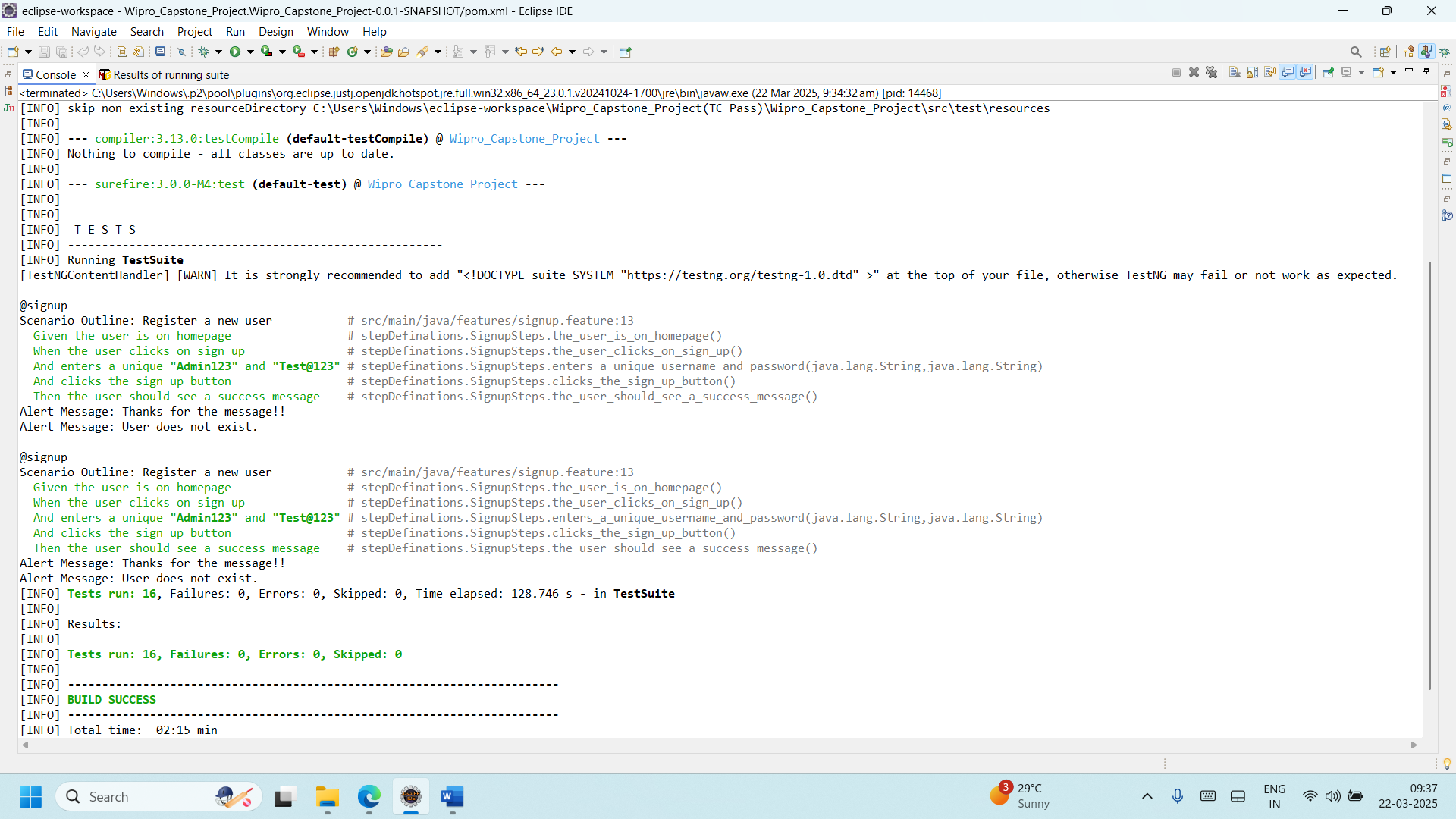
**6.1. Executing Tests Using TestNG XML:**

The TestNG XML file defines the execution flow of test cases by organizing them into test suites and specifying execution order. It allows running specific test classes, setting test parameters, and managing dependencies between tests. By executing the XML file, the framework ensures a structured and automated test execution process. It also enables grouping test cases for efficient management and reporting.



**6.2. Executing Tests Using Maven (pom.xml):**

The project supports test execution using Maven by configuring dependencies and plugins in pom.xml. The Maven Surefire Plugin is used to execute TestNG tests, while Cucumber dependencies enable BDD-based execution. Running mvn test automatically triggers the test execution based on the testng.xml file. Additionally, specific test cases can be executed using Maven command-line options. This ensures seamless automation execution, dependency management, and efficient reporting integration.



**7.Reporting Mechanism**

* **Extent Reports**: Generates HTML reports with test execution details.
* **Cucumber Reports**: Provides feature-wise execution results.
* **Console Logs:** Provides real-time debugging insights.
* **Allure Reports:**

**Step 1: Add Allure Dependencies to pom.xml**

* + Add the following dependencies inside the <dependencies> section:

<dependency>

<groupId> io.qameta.allure </groupId>

<artifactId>allure-testng</artifactId>

<version>2.21.0</version>

</dependency>

* + Add the Allure Maven plugin inside <build>:

<plugin>

<groupId>io.qameta.allure</groupId>

<artifactId>allure-maven</artifactId>

<version>2.10.0</version>

<executions>

<execution>

<goals>

<goal>report</goal>

</goals>

</execution>

</executions>

</plugin>

**Step 2: Run Test Cases & Generate Allure Results**

* + Execute the test cases using Maven:

🡪mvn clean test

* + After execution, the test results will be stored in the allure-results folder inside your project.

**Step 3: Generate Allure Report**

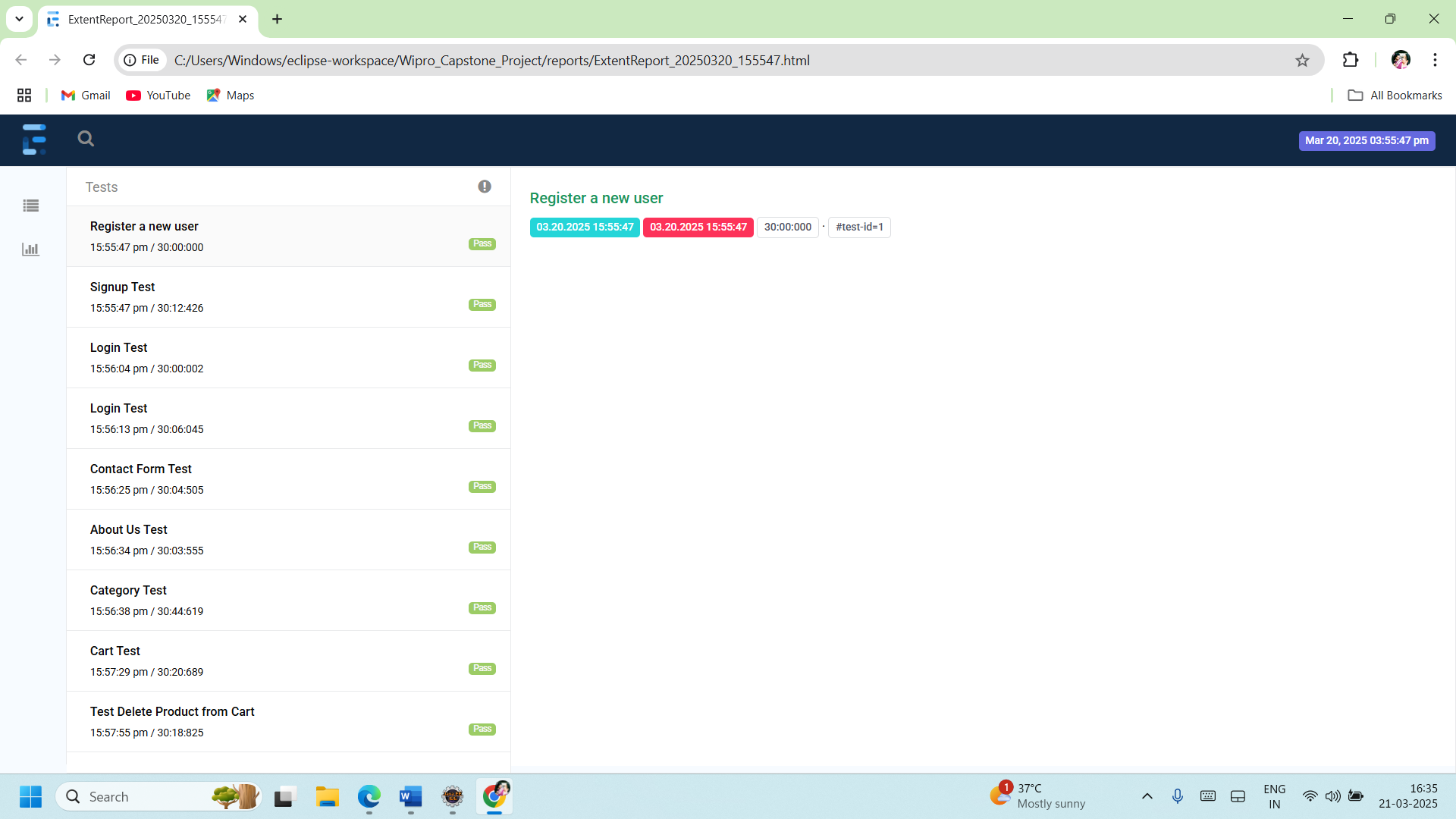
* + To generate the HTML report, run:

🡪allure serve allure-results

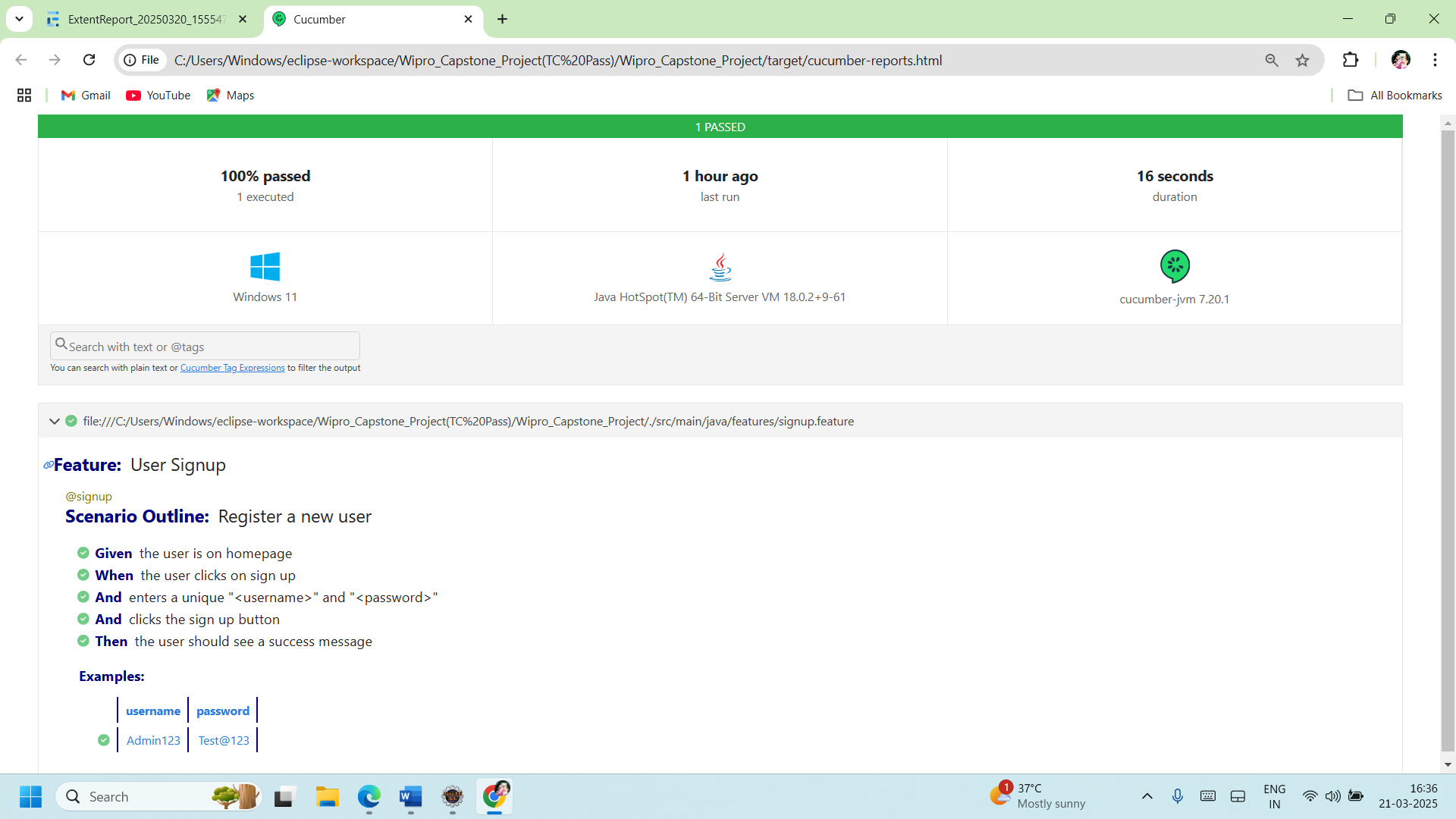
* + This command automatically opens the Allure Report in a web browser.

These are the Reports generated by the project:

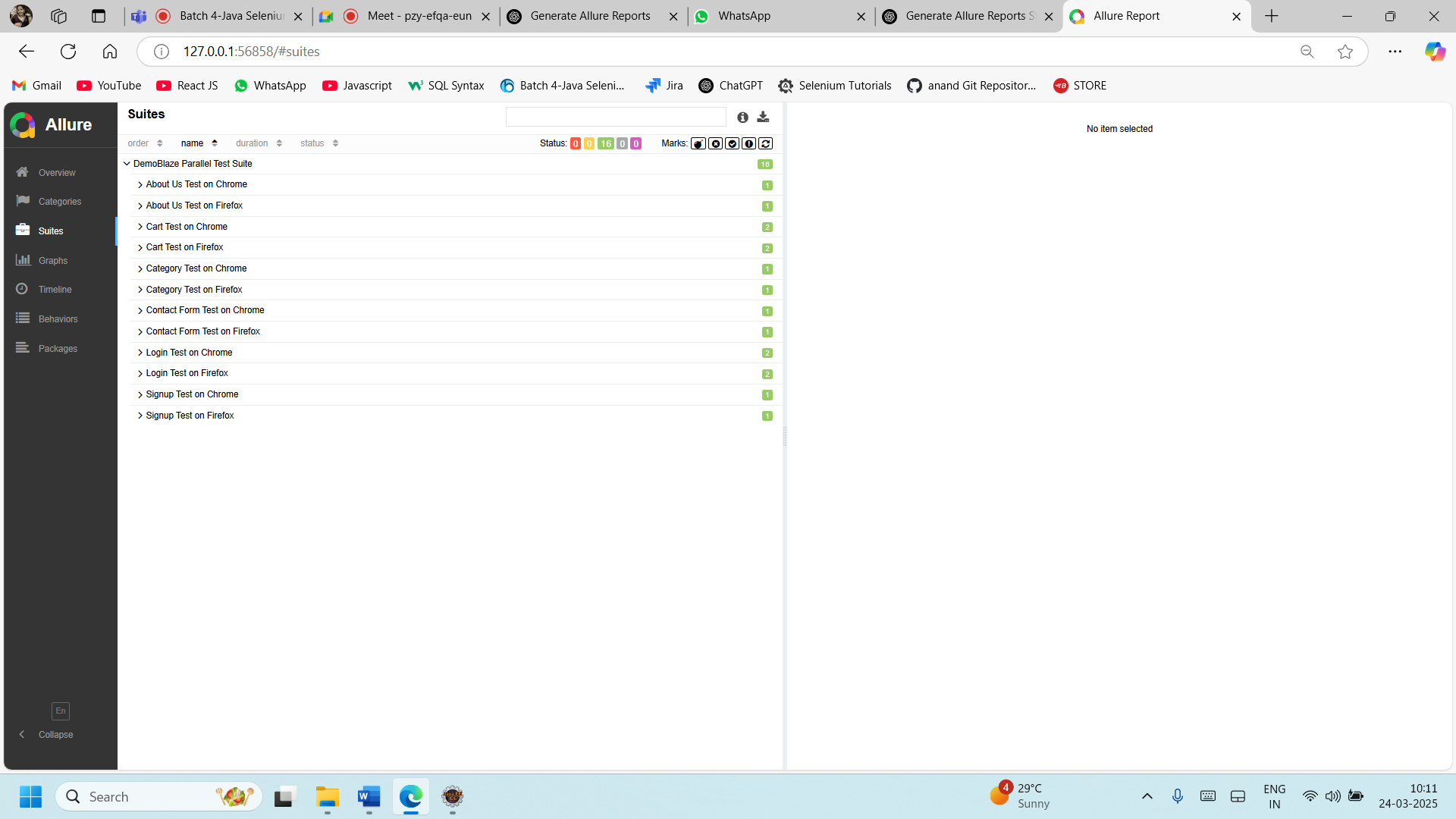
**Extent Report:**



**Cucumber Report:**



**Allure Reports:**



**8. Jenkins Integration for CI/CD**

**8.1 Setting Up Jenkins Job**

1. Open Jenkins Dashboard → Click New Item.
2. Select Freestyle Project and name it Demoblaze\_Automation.
3. Configure Source Code Management (SCM) → Select Git and enter repo URL.
4. Under Build, add Execute Shell and enter: mvn clean test
5. Click Save and trigger the build.

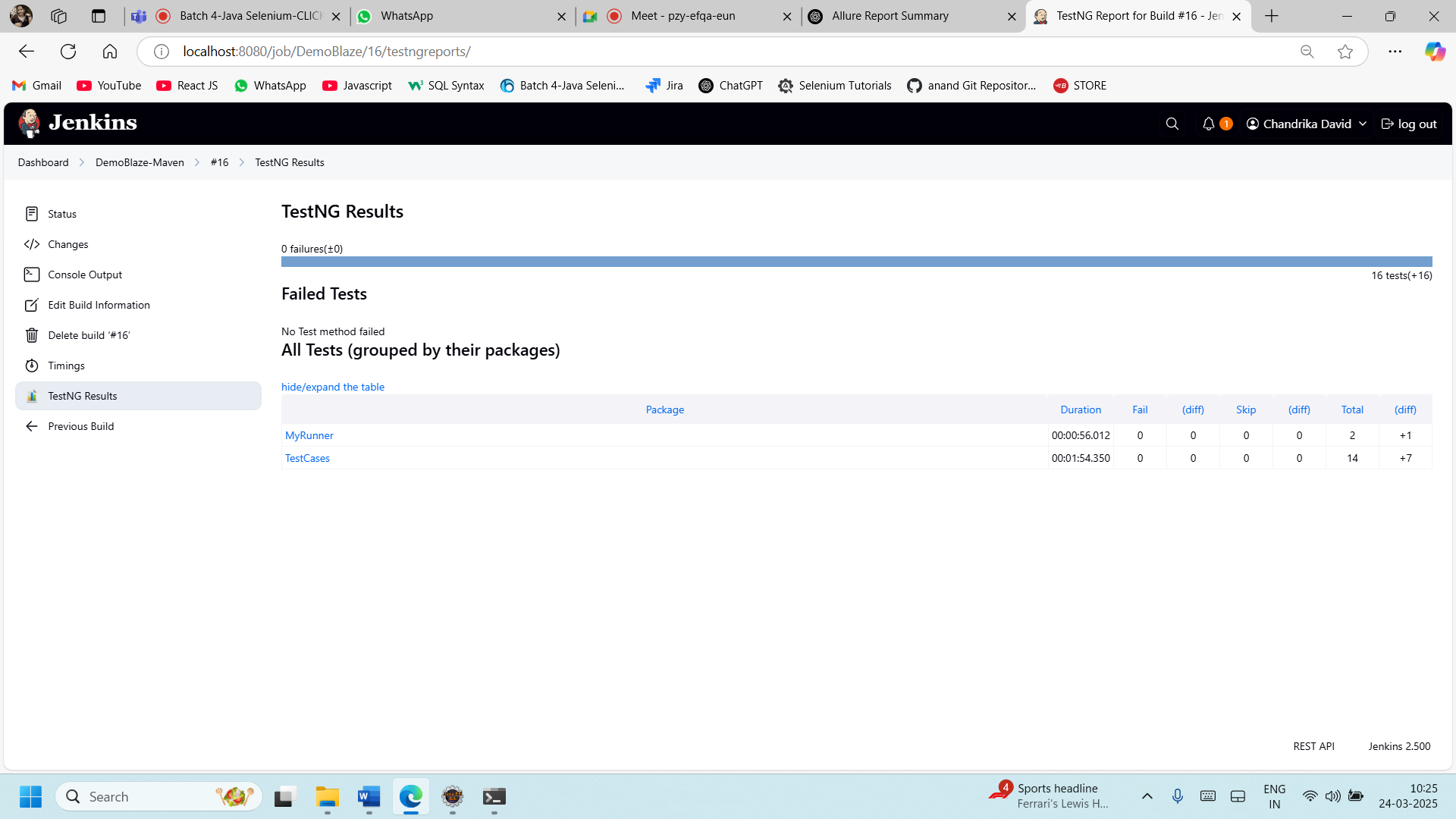
**8.2 Running Tests via Jenkins**

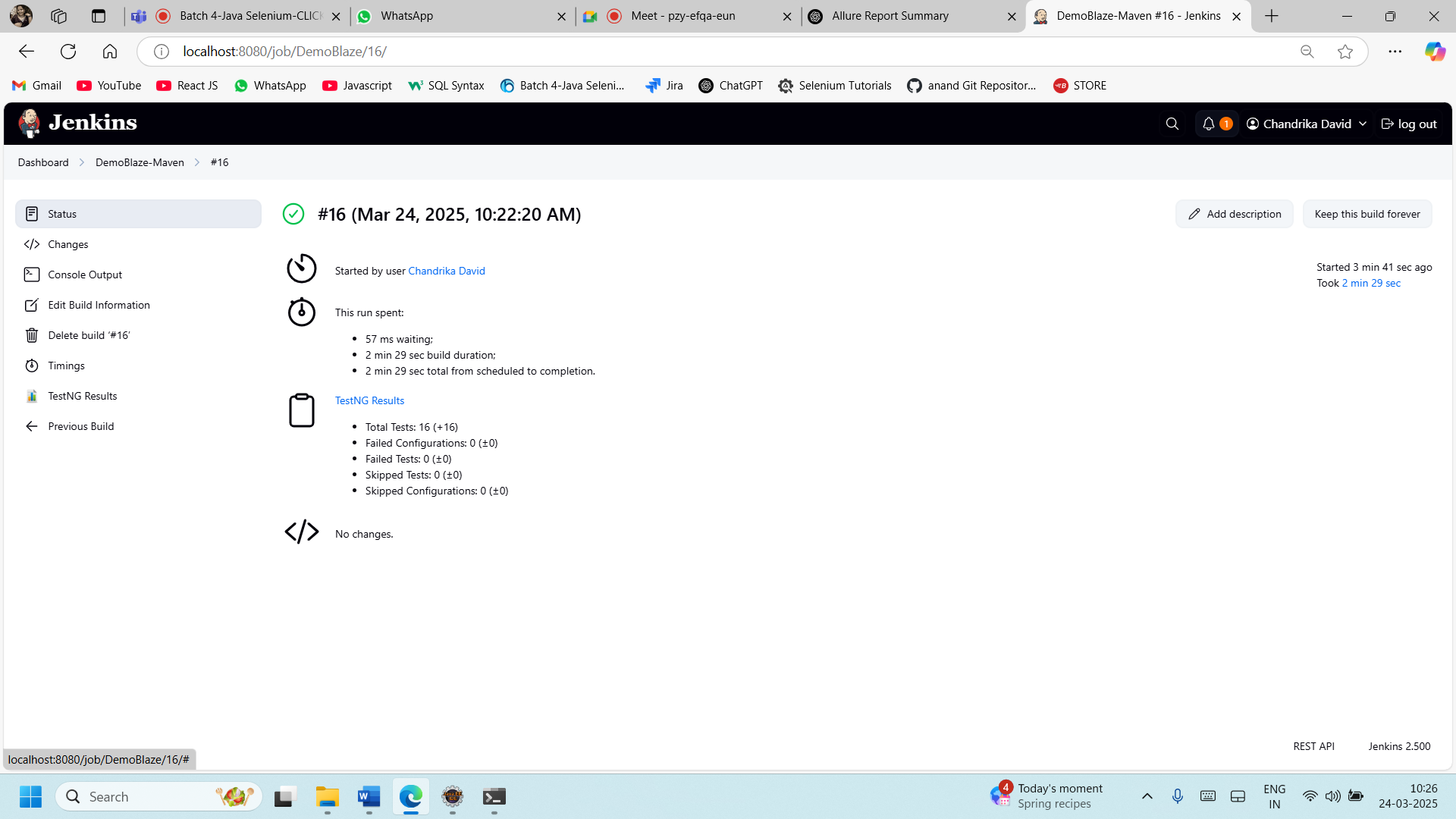
* Open Jenkins Dashboard → Select Demoblaze\_Automation
* Click Build Now
* View console output and Cucumber reports

**9. Execution Flow**

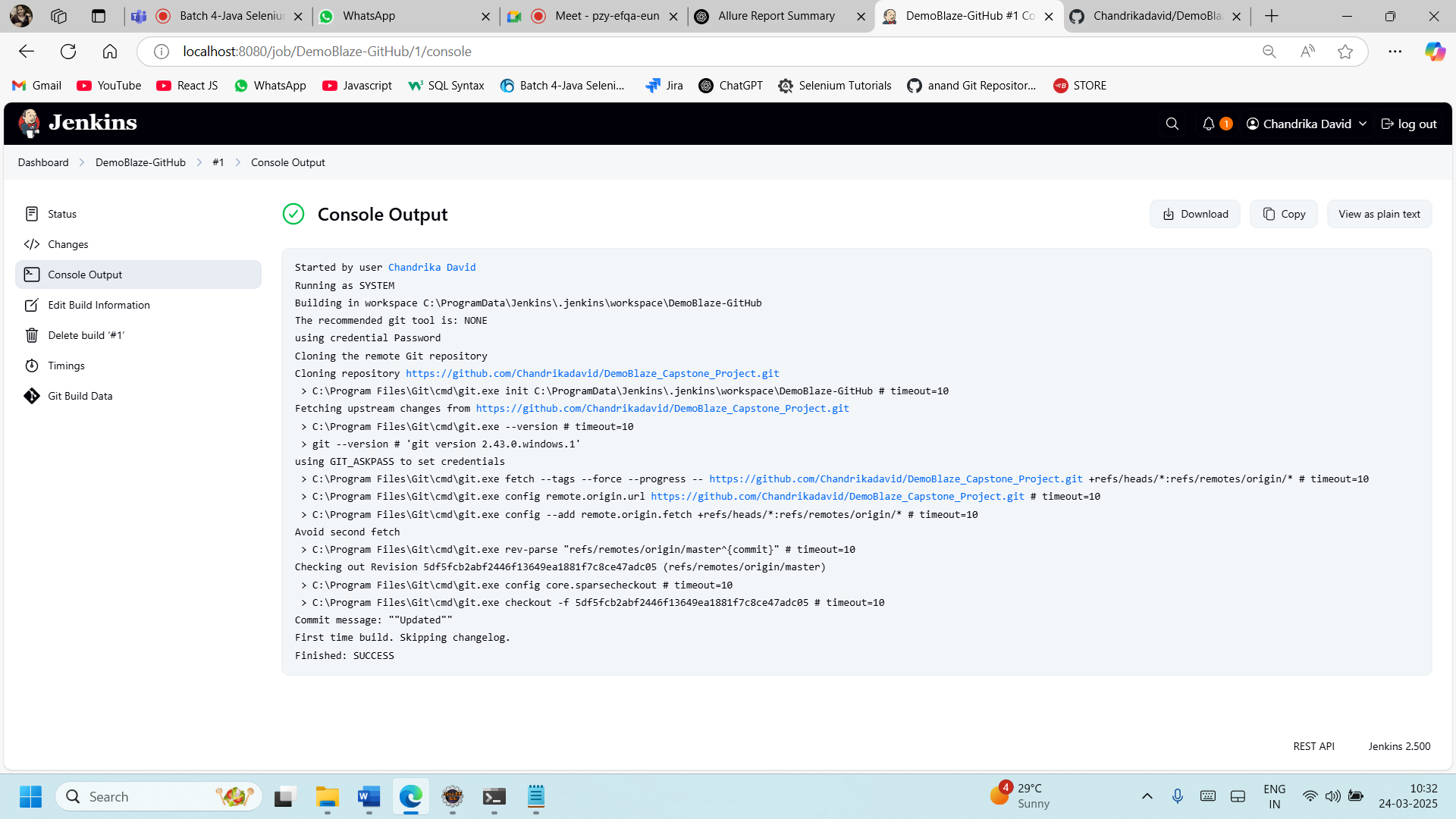
1. Jenkins fetches the latest code from GitHub.
2. Maven builds the project and executes TestNG tests in parallel.
3. Cucumber generates reports while screenshots are captured for failed tests.
4. Reports are sent via email or stored in Jenkins workspace.

**Integrate with Maven:**





**Integrate with GitHub:**



**10. Challenges Faced & Solutions**

| Challenge | Solution |
| --- | --- |
| Element Not Found Issues | Used Explicit & Fluent Waits. |
| Handling Alerts & Popups | Used Selenium Alert handling. |
| Parallel Execution Issues | Resolved using ThreadLocal WebDriver. |
| Test Data Management | Integrated Excel (Apache POI) for data-driven testing. |

**11. Conclusion**

This framework provides a scalable, efficient, and maintainable automation solution for Demoblaze.

Future Enhancements:

* Database Validation
* API Testing (REST Assured)
* Cloud Execution (Selenium Grid, AWS Lambda)