21CS3257P – SOFTWARE PROJECT MANAGEMENT

A Project Report

On

# Selenium Testing using Java To Run Automated Tests

Under the Guidance of

## Dr. T.Vignesh Sir

by


| I.D NUMBER | NAME |
| --- | --- |
| 2100030127 | D.Govardhan |

KONERU LAKSHMAIAH EDUCATION FOUNDATION

DEPARTMENT OF COMPUTER SCIENCE AND ENIGNEERING

(DST-FIST Sponsored Department)

Green Fields, Vaddeswaram, Guntur District-522 502

# Declaration

We here by declare that this Project report entitled Selenium testing using java to run automated tests has been prepared by us in the course 21CS3257P – Software Project Management in COMPUTER SCIENCE AND ENGINEERING during the Even Semester of the academic year 2023-2024. We also declare that this project-basedlab report is of our own effort.

| Student Name | Id Number |
|---|---|
| D. Govardhan | 2100030127 |

# CERTIFICATE

This is to certify that the project based Lab report entitled Selenium testing using java to run automated tests is a bonafide work done Mr. Govardhan bearing Regd. No. 2100030127 to the course 21CS3257P Software Project Management in COMPUTER SCIENCE AND ENGINEERING during the Even Semester of Academic year 2023-2024.

FACULTY IN CHARGE

# ACKNOWLEDGEMENTS

Our sincere thanks to Vignesh Sir in the lab sessions for her outstanding support throughout the project for the successful completion of the work.

We express our gratitude to Dr.Vignesh Sir, Course Co-Ordinator for the course 21CS3257P - Software Project Management in the Department of Computer Science and Engineering for providing us with adequate planning and support and means by which we can complete this project.

We express our gratitude to Prof. Senthil Sir, Head of the Department for Computer Science and Engineering for providing us with adequate facilities, ways and means by which we can complete this project.

We would like to place on record the deep sense of gratitude to the Vice Chancellor, K L University for providing the necessary facilities to carry out the project.

Last but not the least, we thank all Teaching and Non-Teaching Staff of our department and especially our classmates and our friends for their support.

# INDEX

# ABSTRACT

Automated testing has become an integral part of software development, aiding in detecting bugs, ensuring functionality, and enhancing overall product quality. Selenium WebDriver, a popular open-source testing tool, provides a platform-independent framework for automating web applications across different browsers and operating systems. Leveraging Java as a programming language, Selenium offers a robust and versatile solution for implementing automated tests.

This abstract outlines the process and benefits of utilizing Selenium WebDriver with Java for automated testing of web applications. The proposed approach involves setting up the Java development environment, configuring Selenium WebDriver, and writing test scripts to simulate user interactions and validate expected behaviors.

Key components of the Selenium testing framework include WebDriver, which serves as the core automation engine, and various supporting libraries for element identification, navigation, and assertion. By employing object-oriented programming principles in Java, testers can design modular, maintainable, and scalable test suites, facilitating efficient test creation and execution.

In conclusion, the integration of Selenium WebDriver with Java offers a powerful solution for implementing automated tests, enabling organizations to achieve greater efficiency, reliability, and agility in their software testing endeavors.

# INTRODUCTION

In today's rapidly evolving technological landscape, ensuring the quality and reliability of software applications is paramount. Automated testing has emerged as a cornerstone in the pursuit of delivering high-quality software efficiently and effectively. Among the plethora of tools available for automated testing, Selenium WebDriver stands out as a versatile and powerful solution for web application testing. When combined with the robustness and flexibility of Java, Selenium offers a compelling framework for implementing automated tests that can simulate user interactions, validate application functionality, and detect defects with precision.

This introduction serves as a primer to explore the realm of Selenium testing using Java. It provides an overview of the significance of automated testing in modern software development practices, emphasizing the need for reliable and efficient testing methodologies to meet the demands of today's dynamic digital landscape.

In this document, we delve into the process of Selenium testing using Java, elucidating the steps involved in setting up the testing environment, configuring Selenium WebDriver, and crafting automated test scripts. Through comprehensive examples and practical guidance, this document aims to equip software testers, developers, and quality assurance professionals with the knowledge and tools necessary to leverage Selenium WebDriver with Java effectively for automated testing of web applications.

By harnessing the combined power of Selenium WebDriver and Java, organizations can streamline their testing processes, accelerate release cycles, and enhance the overall quality and reliability of their software products.

# SYSTEM REQUIREMENT SPECIFICATION

- SOFTWARE REQUIREMENTS:

  The major software requirements of the project are as follows:

  Language: Any OOPS Language (preferably Java)
  IDE: Eclipse or Intellij
  Driver: ChromeDriver, EdgeDriver or other Driver for testing
  Dependencies: Dependencies like JUnit, Cucumber, Selenium etc.

- HARDWARE REQUIREMENTS:

The hardware requirements that map towards the software are as follows:

- Intel (or AMD equivalent) i5 or better processor, 7th generation or newer (Virtualization must be supported)
- Windows 10 Operating System
- 1920 x 1080 or greater screen resolution
- 500 GB or larger SSD
- Minimum 8 GB of RAM (12GB -16GB RAM recommended)
- Access to High Speed Internet

# METHODOLOGY

Implementing Selenium testing for a website involves several steps, from setting up the project to writing feature files and step definitions. Here's a step-by-step methodology to guide you through the process:

1.Requirement Analysis:

- Understand the functional and non-functional requirements of the web application to be tested.
- Identify the features and functionalities that need to be tested using Selenium WebDriver.

2.Environment Setup:

- Install Java Development Kit (JDK) on your machine.
- Configure your Java Integrated Development Environment (IDE) such as IntelliJ IDEA or Eclipse.
- Download the Selenium WebDriver JAR files and add them to your project's build path.
- Download the browser drivers (e.g., ChromeDriver, GeckoDriver) and set their paths in your system environment variables.

3.Test Planning:

- Define test scenarios and test cases based on the identified requirements.
- Prioritize test cases and determine the scope of automated testing.
- Design a test strategy considering factors like browser compatibility, test data management, and reporting.

4.Test Script Development:

- Create a new Java class or package for your test suite.
- Implement test scripts using Selenium WebDriver APIs to perform actions like navigating to web pages, interacting with elements, and validating page content.
- Utilize Java's object-oriented programming features for modularizing test scripts, enhancing maintainability and reusability.
- Incorporate assertions to verify expected outcomes and ensure the correctness of test results.

5.Test Execution:

- Execute the test scripts locally on your development environment.
- Run tests on different browsers and platforms to ensure cross-browser compatibility.
- Utilize test execution frameworks (e.g., TestNG, JUnit) to manage test suites, execute tests in parallel, and generate test reports.
- Implement logging and error handling mechanisms to capture and report test failures effectively.

6.Test Result Analysis:

- Analyze test results to identify and prioritize defects.
- Investigate failed tests to understand the root causes of failures.
- Update test scripts or application code as necessary to address identified issues.

7.Continuous Integration and Deployment:

- Integrate automated tests into the continuous integration/continuous deployment (CI/CD) pipeline.
- Configure CI tools (e.g., Jenkins, Bamboo) to trigger automated tests on code commits or scheduled intervals.
- Monitor test execution status and integrate test reports with CI/CD dashboards for visibility and transparency.

8.Maintenance and Iteration:

- Maintain test scripts by updating them to accommodate changes in the application under test or the testing environment.
- Review and refactor test code regularly to ensure readability, efficiency, and adherence to best practices.
- Iterate on the testing process based on feedback, evolving requirements, and emerging best practices to continually improve test coverage and effectiveness.

9.Test Data Management:

- Develop strategies for managing test data effectively, including techniques for generating, retrieving, and maintaining test data.
- Implement data-driven testing approaches using external data sources (e.g., CSV files, databases) to execute test scenarios with different datasets.

10.Cross-browser and Cross-platform Testing:

- Expand test coverage by conducting cross-browser testing on popular web browsers (e.g., Chrome, Firefox, Safari, Edge) to ensure compatibility.
- Perform cross-platform testing on various operating systems (e.g., Windows, macOS, Linux) to validate application behavior across different environments.

11.Parallel Test Execution:

- Utilize parallel execution capabilities offered by testing frameworks (e.g., TestNG, JUnit) to run multiple tests simultaneously, reducing overall test execution time.
- Leverage cloud-based testing services (e.g., Sauce Labs, BrowserStack) to scale test execution across multiple browsers and platforms in parallel.

12.Integration with Test Management Tools:

- Integrate automated tests with test management tools (e.g., Jira, TestRail) to maintain test artifacts, track test execution results, and

collaborate with team members effectively.

- Automatically update test statuses and generate test reports within the test management system to provide visibility into testing progress and outcomes.

13.Continuous Monitoring and Feedback:

- Implement monitoring solutions to continuously monitor application performance and user experience in production environments.
- Collect feedback from end-users, stakeholders, and quality assurance teams to identify areas for improvement and inform future testing efforts.

14.Security Testing Integration:

- Incorporate security testing practices into automated testing workflows to identify vulnerabilities such as injection flaws, cross-site scripting (XSS), and authentication bypass.
- Integrate security testing tools (e.g., OWASP ZAP, Burp Suite) into the automated testing pipeline to assess application security posture and mitigate risks

15.Training and Knowledge Sharing:

- Provide training sessions and workshops to empower team members with the skills and knowledge required to create and maintain Selenium tests effectively.
- Foster a culture of knowledge sharing and collaboration within the team to exchange best practices, tips, and lessons learned from Selenium testing experiences.

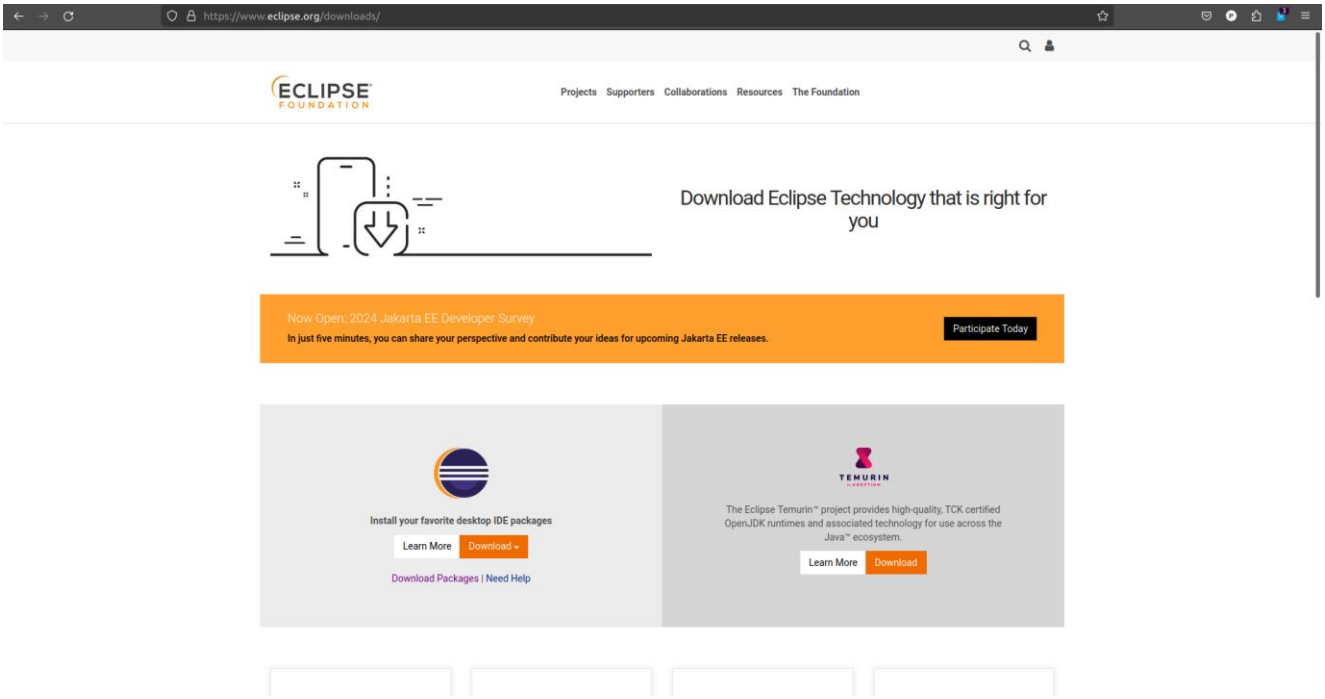
By following this methodology, organizations can establish a structured and systematic approach to Selenium testing using Java, enabling them to achieve reliable, scalable, and maintainable automated testing practices for web applications.

# CODING AND IMPLEMENTATION

Step1:

**Install Eclipse**

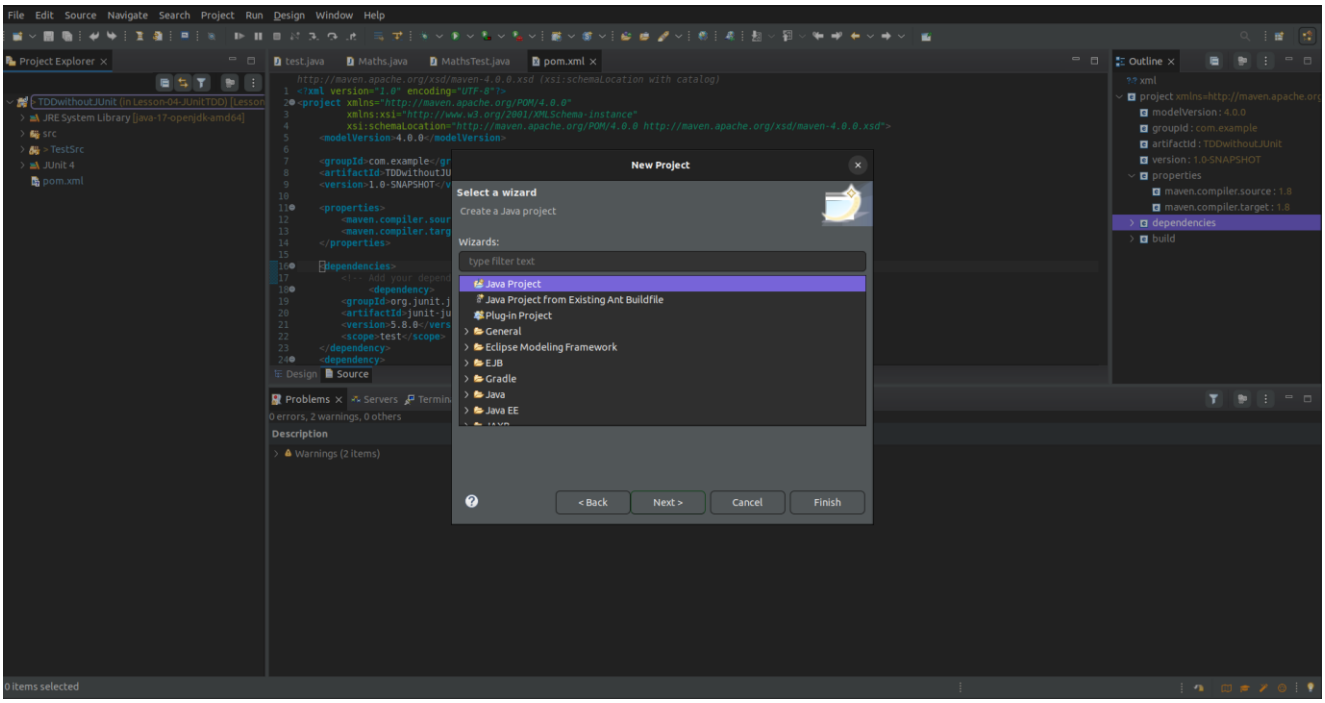
Download and install Eclipse IDE from the official website: Eclipse Downloads.



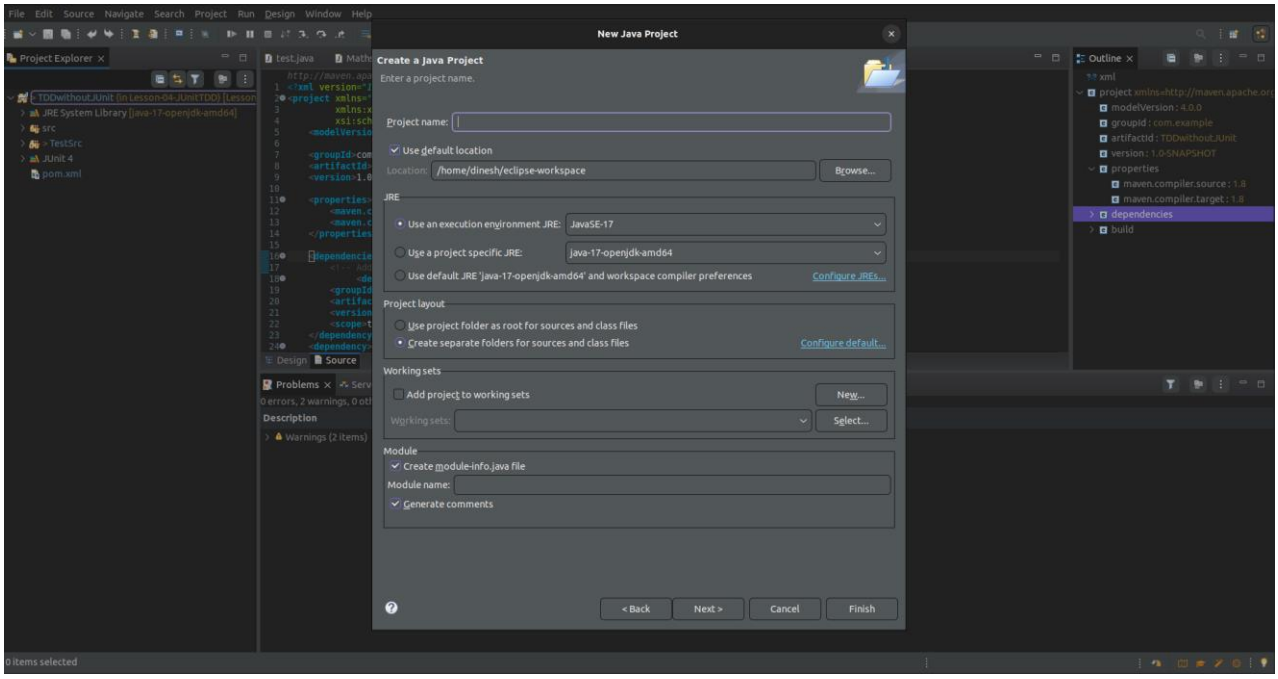

**Step 2: Create a New Java Project**

- Open Eclipse IDE.
- Go to `File > New > Java Project`.

- Enter a name for your project (e.g., "SeleniumProject") and click "Next".

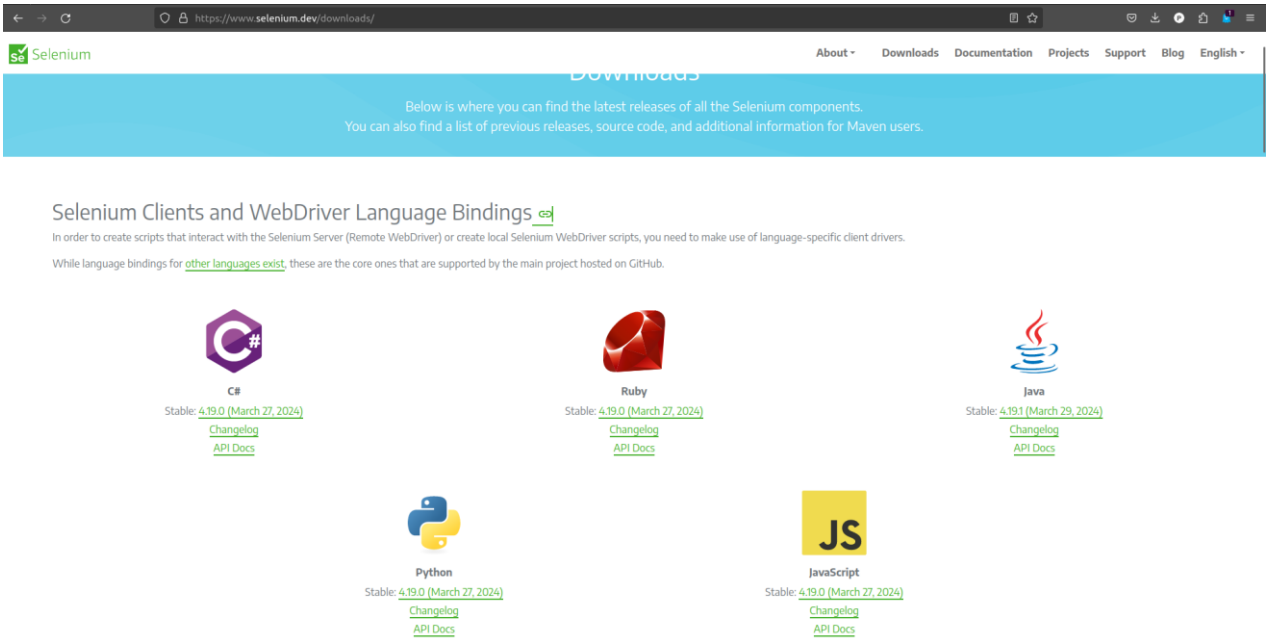



- Choose the default JRE or select a specific JRE version and click "Finish".



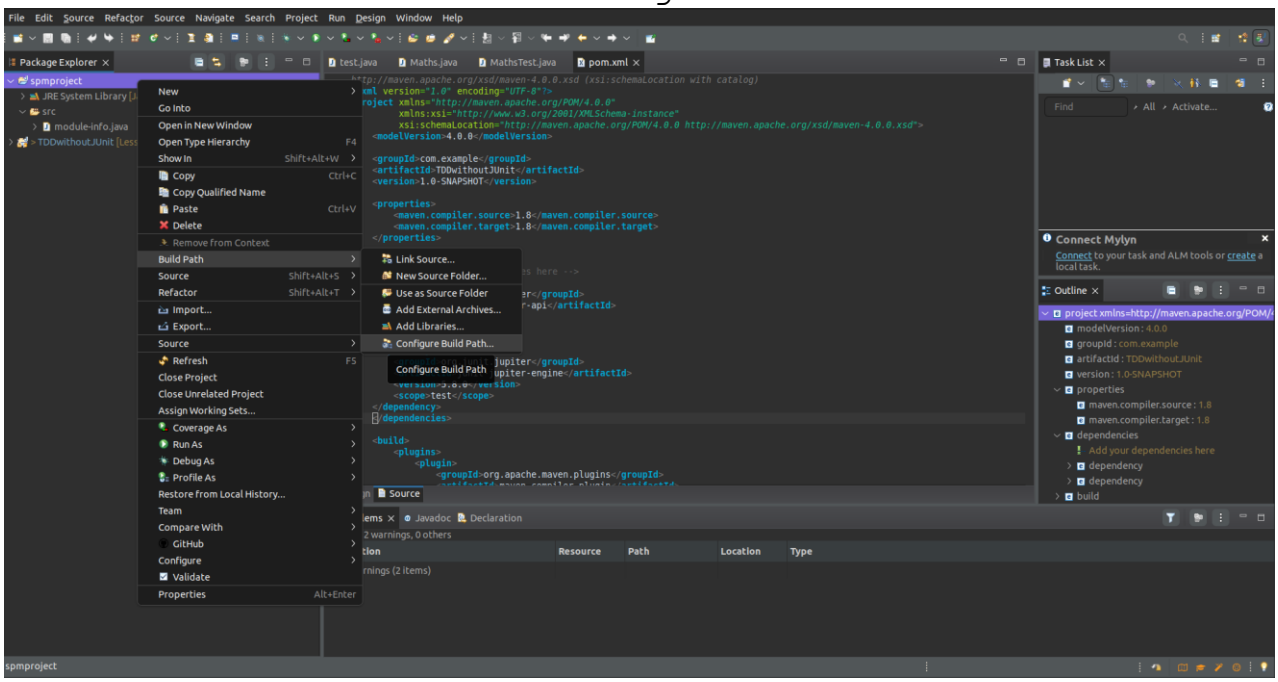
### Step 3: Configure Selenium WebDriver

Download the Selenium WebDriver JAR files from the Selenium website:

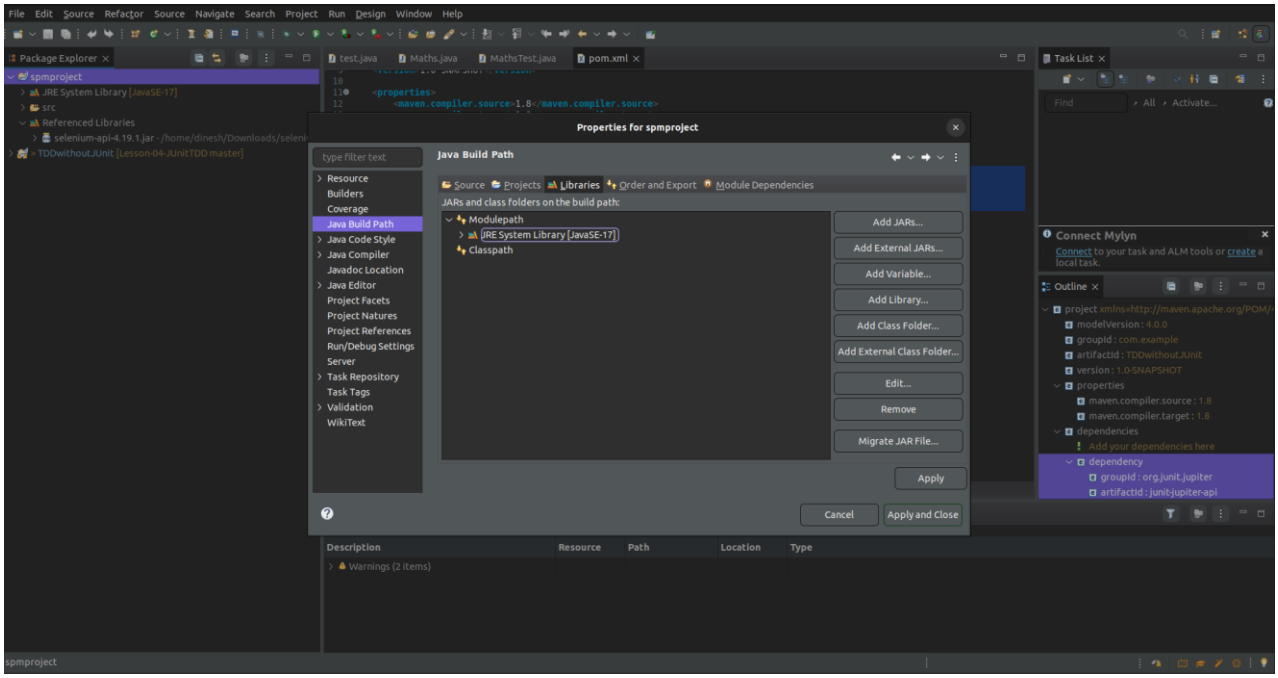



Selenium Downloads.

- In Eclipse, right-click on your project folder in the "Package Explorer" view.
- Select `Build Path > Configure Build Path`.



- In the "Libraries" tab, click on "Add External JARs" and navigate to the location where you saved the Selenium WebDriver JAR files.
- Select all the JAR files and click "Open" to add them to your project's




build path.

- Click "Apply and Close" to confirm the changes.

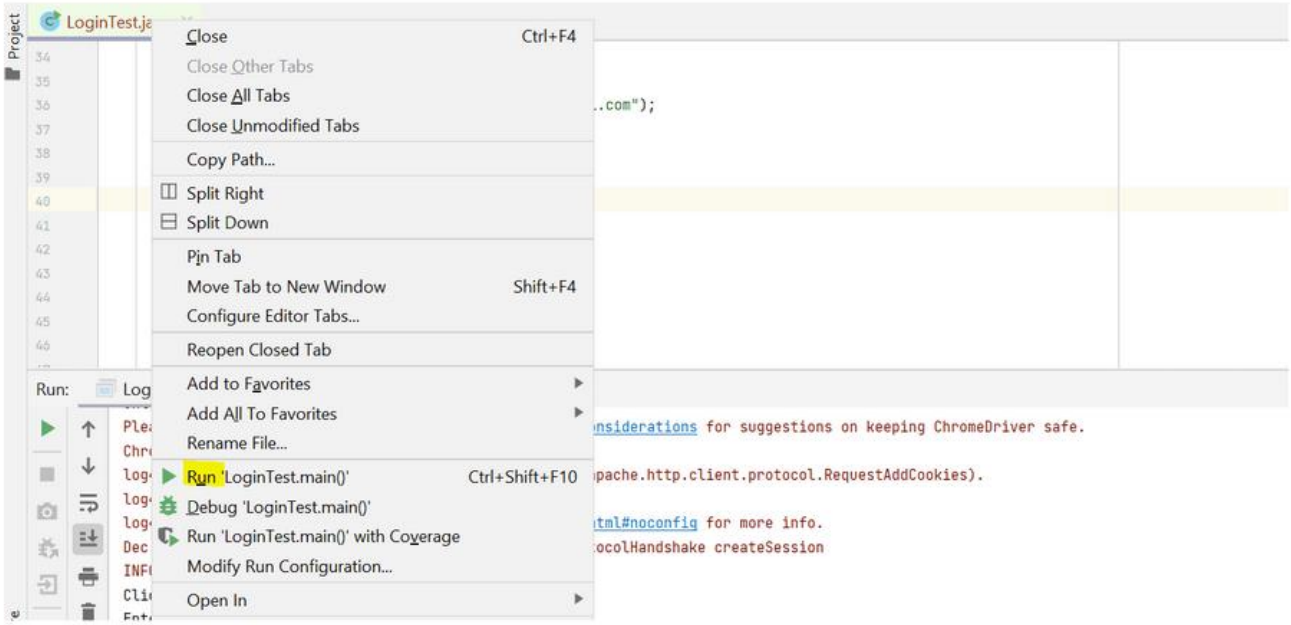
**Step 5: Run Test Scripts**

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;

import java.util.concurrent.TimeUnit;



public class LoginTest {
    public static void main(String[] args) {

        WebDriver driver = new ChromeDriver();
```

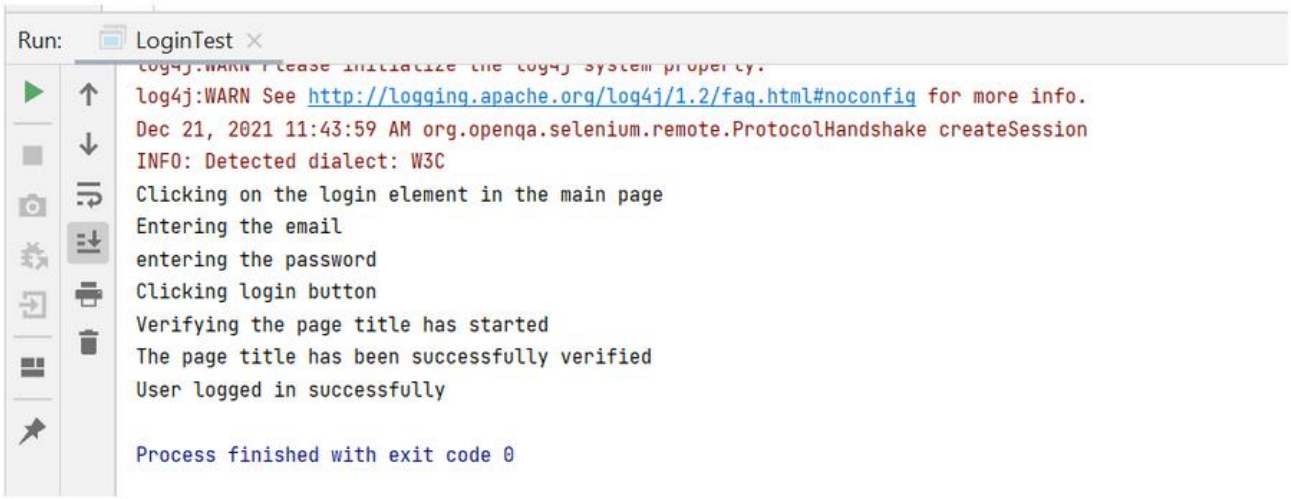
```
String url = "https://www.lambdatest.com/";

driver.get(url);
driver.manage().window().maximize();
driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);


WebElement login = driver.findElement(By.linkText("Login"));
System.out.println("Clicking on the login element in the main page");
login.click();

driver.manage().timeouts().pageLoadTimeout(10,TimeUnit.SECONDS);

WebElement email = driver.findElement(By.id("email"));
WebElement password = driver.findElement(By.id("password"));
WebElement loginButton = driver.findElement(By.id("login-button"));

email.clear();
System.out.println("Entering the email");
email.sendKeys("your_email");

password.clear();
System.out.println("entering the password");
password.sendKeys("your_password");

System.out.println("Clicking login button");
loginButton.click();

String title = "Welcome - LambdaTest";

String actualTitle = driver.getTitle();

System.out.println("Verifying the page title has started");
Assert.assertEquals(actualTitle,title,"Page title doesn't match");

System.out.println("The page title has been successfully verified");
```

```
        System.out.println("User logged in successfully");

        driver.quit();
    }
}
```

- To execute the tests locally, right-click on your class name and select the Run option. This initiates the test execution process, allowing you to observe the outcomes of the login page automation.



Output:


Run:
LoginTest
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Dec 21, 2021 11:43:59 AM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Clicking on the login element in the main page
Entering the email
entering the password
Clicking login button
Verifying the page title has started
The page title has been successfully verified
User logged in successfully
Process finished with exit code 0

## Conclusion:

In conclusion, Selenium testing using Java offers a robust and efficient approach to automate the testing of web applications. Throughout this process, we have explored the various steps involved in setting up a Selenium project, configuring Selenium WebDriver, and writing test scripts to simulate user interactions and validate application behavior.

By leveraging the power of Java programming language and Selenium WebDriver, organizations can realize numerous benefits in their testing endeavors:

1. **Improved Efficiency**: Automated testing with Selenium reduces the time and effort required for repetitive manual testing tasks, allowing teams to focus on more complex and critical aspects of software development.

2. **Enhanced Reliability**: Automated tests executed through Selenium WebDriver offer consistent and reliable results, minimizing the risk of human error associated with manual testing.

3. **Increased Test Coverage**: Selenium testing enables comprehensive test coverage across various browsers, platforms, and scenarios, ensuring the application's compatibility and functionality under diverse conditions.

4. **Accelerated Feedback Loops**: Automated tests provide rapid feedback on application changes, allowing for quicker identification and resolution of defects, thereby expediting the software delivery process.

5. **Cost Savings**: By automating testing processes, organizations can achieve cost savings associated with reduced manual effort, improved productivity, and decreased time-to-market.

6. **Facilitated Continuous Integration**: Selenium tests seamlessly integrate into continuous integration and continuous deployment (CI/CD) pipelines, enabling automated regression testing and ensuring the

stability of software releases.

7. Scalability and Maintainability: Selenium test suites written in Java can be easily scaled and maintained, thanks to the language's object-oriented nature and robust testing frameworks such as TestNG and JUnit.

In summary, Selenium testing using Java empowers organizations to achieve higher software quality, faster release cycles, and improved overall efficiency in their software development lifecycle. By embracing automated testing practices, teams can drive innovation, mitigate risks, and deliver exceptional user experiences in today's competitive digital landscape.