**Run Selenium Test end-to-end**

**1. Set Up Maven Project**

**Step:**

1. Open your IDE (e.g., IntelliJ IDEA or Eclipse).
2. Create a new Maven project.

**Example:**

* Use the Maven **archetype-quickstart** template for simplicity.
* Name your project, e.g., SeleniumTestNGProject.

**2. Update pom.xml with Dependencies**

**Step:** Add Selenium, TestNG, and WebDriverManager dependencies to pom.xml.

**Example:**

xml

Copy code

<dependencies>

<!-- Selenium Java -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.10.0</version>

</dependency>

<!-- TestNG -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.8.0</version>

<scope>test</scope>

</dependency>

<!-- WebDriverManager -->

<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>

<version>5.6.2</version>

</dependency>

</dependencies>

**3. Create a Test Class**

**Step:** Set up a basic test using TestNG annotations.

**Example:**

java

Copy code

package tests;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

import io.github.bonigarcia.wdm.WebDriverManager;

public class GoogleSearchTest {

WebDriver driver;

@BeforeMethod

public void setUp() {

WebDriverManager.chromedriver().setup();

driver = new ChromeDriver();

driver.manage().window().maximize();

}

@Test

public void searchGoogle() {

driver.get("https://www.google.com");

System.out.println("Title of the page is: " + driver.getTitle());

}

@AfterMethod

public void tearDown() {

if (driver != null) {

driver.quit();

}

}

}

**4. Configure TestNG XML**

**Step:** Create a testng.xml file to define test execution.

**Example:**

xml

Copy code

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="TestSuite">

<test name="GoogleTests">

<classes>

<class name="tests.GoogleSearchTest"/>

</classes>

</test>

</suite>

**5. Run Tests**

**Step:** Use TestNG to execute the tests.

**Example:**

1. Right-click on testng.xml → **Run as TestNG Suite**.
2. Alternatively, use Maven:

bash

Copy code

mvn test

**6. Add a Page Object Model (Optional)**

**Step:** Structure your project using the Page Object Model for better maintainability.

**Example:**

* **Page Class:** GoogleHomePage.java

java

Copy code

package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class GoogleHomePage {

WebDriver driver;

By searchBox = By.name("q");

public GoogleHomePage(WebDriver driver) {

this.driver = driver;

}

public void search(String query) {

driver.findElement(searchBox).sendKeys(query);

driver.findElement(searchBox).submit();

}

}

* **Test Class:** GoogleSearchTest.java

java

Copy code

@Test

public void searchGoogle() {

GoogleHomePage googlePage = new GoogleHomePage(driver);

googlePage.search("Selenium TestNG Maven tutorial");

System.out.println("Title of the page is: " + driver.getTitle());

}

**7. Generate Reports**

**Step:** Use TestNG’s default reporting or add additional reporting frameworks like **ExtentReports**.

**Example for ExtentReports:** Add the dependency to pom.xml:

xml

Copy code

<dependency>

<groupId>com.aventstack</groupId>

<artifactId>extentreports</artifactId>

<version>5.0.9</version>

</dependency>

Update the test code to include ExtentReports:

java

Copy code

import com.aventstack.extentreports.ExtentReports;

import com.aventstack.extentreports.ExtentTest;

import com.aventstack.extentreports.reporter.ExtentSparkReporter;

public class GoogleSearchTest {

ExtentReports extent;

@BeforeSuite

public void setupReport() {

ExtentSparkReporter spark = new ExtentSparkReporter("ExtentReport.html");

extent = new ExtentReports();

extent.attachReporter(spark);

}

@Test

public void searchGoogle() {

ExtentTest test = extent.createTest("Search Google Test");

test.info("Navigating to Google");

driver.get("https://www.google.com");

test.pass("Test Passed: Title is " + driver.getTitle());

}

@AfterSuite

public void tearDownReport() {

extent.flush();

}

}

**8. Wrap Up**

1. Show the folder structure of your project.
2. Summarize the benefits of using Maven, TestNG, and Selenium.
3. Provide a link to your GitHub repository for viewers to clone.

This structure ensures your video is engaging and provides viewers with actionable steps. Let me know if you'd like more details or assistance!

4o

**You said:**

class lombok.javac.apt.LombokProcessor (in unnamed module @0x4ed9f7b1) cannot access class com.sun.tools.javac.processing.JavacProcessingEnvironment (in module jdk.compiler) because module jdk.compiler does not export com.sun.tools.javac.processing to unnamed module @0x4ed9f7b1

**ChatGPT said:**

ChatGPT

The error occurs because newer versions of the JDK (starting from Java 9) use the module system, and some libraries like **Lombok** access internal APIs that are no longer publicly accessible without explicit permission.

Here’s how you can resolve this issue:

**1. Update Lombok Version**

Ensure you're using the latest version of Lombok, as recent versions are better equipped to handle module-related changes.

* Update Lombok in your pom.xml (for Maven projects):

xml

Copy code

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.30</version> <!-- Replace with the latest version -->

<scope>provided</scope>

</dependency>

* For Gradle, update the Lombok dependency:

groovy

Copy code

implementation 'org.projectlombok:lombok:1.18.30'

annotationProcessor 'org.projectlombok:lombok:1.18.30'

**2. Add JVM Arguments**

Pass additional JVM arguments to allow Lombok to access the internal APIs.

**Maven Projects:**

Add the following to your pom.xml under the <build> section:

xml

Copy code

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<compilerArgs>

<arg>--add-exports=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED</arg>

<arg>--add-opens=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED</arg>

</compilerArgs>

</configuration>

</plugin>

</plugins>

</build>

**Gradle Projects:**

Add the following to your build.gradle:

groovy

Copy code

tasks.withType(JavaCompile) {

options.compilerArgs += [

'--add-exports=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED',

'--add-opens=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED'

]

}

**3. Configure Your IDE**

Some IDEs (e.g., IntelliJ IDEA) may also need these arguments explicitly.

* **For IntelliJ IDEA:**
  1. Go to File > Settings > Build, Execution, Deployment > Compiler > Java Compiler.
  2. Add the following options in "Additional Command Line Parameters":

css

Copy code

--add-exports=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED

--add-opens=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED

* **For Eclipse:**
  1. Go to Window > Preferences > Java > Installed JREs.
  2. Edit your JRE and add the following VM arguments:

css

Copy code

--add-exports=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED

--add-opens=jdk.compiler/com.sun.tools.javac.processing=ALL-UNNAMED

**4. Verify Java Version Compatibility**

Ensure that Lombok is compatible with your JDK version. If you are using an older Lombok version or a very new JDK (e.g., JDK 20+), compatibility issues might persist.

**5. Alternative: Use Annotation Processors**

If Lombok still fails, consider using alternatives like annotation processors explicitly defined in your build tools.

Let me know if you need further clarification!

4o

**You said:**

I am not getting any thing in extent reports

**ChatGPT said:**

ChatGPT

If you are not seeing any results in your ExtentReports output, the issue might stem from incorrect initialization, improper reporting setup, or failure to flush the report. Here’s how you can troubleshoot and fix it step by step:

**1. Ensure Proper ExtentReports Initialization**

Make sure you’ve initialized **ExtentReports** and attached a reporter properly.

**Correct Example:**

java

Copy code

import com.aventstack.extentreports.ExtentReports;

import com.aventstack.extentreports.ExtentTest;

import com.aventstack.extentreports.reporter.ExtentSparkReporter;

public class TestReport {

ExtentReports extent;

ExtentTest test;

@BeforeSuite

public void setupReport() {

// Initialize the reporter and attach it to ExtentReports

ExtentSparkReporter spark = new ExtentSparkReporter("ExtentReport.html");

extent = new ExtentReports();

extent.attachReporter(spark);

}

@Test

public void sampleTest() {

// Create a test in the report

test = extent.createTest("Sample Test");

test.info("Starting the test");

test.pass("Test passed successfully");

}

@AfterSuite

public void tearDownReport() {

// Flush the report to write data to the file

extent.flush();

}

}

**2. Verify Report File Location**

Ensure the output file (e.g., ExtentReport.html) is being created in the specified location.

* Check if the file path is correct:

java

Copy code

ExtentSparkReporter spark = new ExtentSparkReporter("reports/ExtentReport.html");

If the path doesn’t exist, create the folder manually (reports in this case).

* Verify if the report file contains any content.

**3. Add Logs to the Tests**

For the report to contain meaningful information, you need to log actions in your test cases.

**Example:**

java

Copy code

@Test

public void searchTest() {

test = extent.createTest("Search Test");

test.info("Navigating to Google");

driver.get("https://www.google.com");

String title = driver.getTitle();

test.info("Page title is: " + title);

if (title.contains("Google")) {

test.pass("Title verification passed");

} else {

test.fail("Title verification failed");

}

}

**4. Ensure extent.flush() is Called**

Calling extent.flush() is crucial; without it, the report won’t save data to the output file.

**Example:**

java

Copy code

@AfterSuite

public void tearDownReport() {

extent.flush(); // Finalize the report

}

**5. Debugging Steps**

* **Check Console Logs:** Ensure there are no errors or exceptions during the test execution that could prevent the report from being written.
* **Test with a Minimal Example:** Run this simple example to confirm ExtentReports is working:

java

Copy code

import com.aventstack.extentreports.ExtentReports;

import com.aventstack.extentreports.ExtentTest;

import com.aventstack.extentreports.reporter.ExtentSparkReporter;

public class MinimalTest {

public static void main(String[] args) {

ExtentSparkReporter spark = new ExtentSparkReporter("SimpleReport.html");

ExtentReports extent = new ExtentReports();

extent.attachReporter(spark);

ExtentTest test = extent.createTest("Basic Test");

test.info("This is a basic test");

test.pass("The test passed");

extent.flush(); // Finalize the report

System.out.println("Report generated.");

}

}

**6. Verify Dependencies**

Ensure you’re using the correct version of ExtentReports.

Add this to your pom.xml if not already present:

xml

Copy code

<dependency>

<groupId>com.aventstack</groupId>

<artifactId>extentreports</artifactId>

<version>5.0.9</version>

</dependency>

**7. Check for Browser or WebDriver Issues**

If you're performing Selenium tests, ensure that the WebDriver actions are being logged properly. Use appropriate try-catch blocks to log failures.

**8. Use a Valid HTML File**

Ensure the ExtentSparkReporter points to an accessible and valid HTML file path. If using a relative path, ensure the directories exist.

By following these steps, you should be able to see content in your ExtentReports output. Let me know if you face any specific issues or errors!

4o

**You said:**

any code to automatically display extent report after test finishes

**ChatGPT said:**

ChatGPT

To automatically display the **Extent Report** in your default web browser after the test execution finishes, you can use the Desktop class in Java. This will open the report file (e.g., ExtentReport.html) once the extent.flush() method has been called.

Here’s the updated code:

**Code Example to Auto-Display Extent Report**

java

Copy code

import com.aventstack.extentreports.ExtentReports;

import com.aventstack.extentreports.ExtentTest;

import com.aventstack.extentreports.reporter.ExtentSparkReporter;

import java.awt.\*;

import java.io.File;

import java.io.IOException;

public class AutoDisplayExtentReport {

ExtentReports extent;

ExtentTest test;

@BeforeSuite

public void setupReport() {

// Define the path for the Extent Report

String reportPath = "ExtentReport.html";

ExtentSparkReporter spark = new ExtentSparkReporter(reportPath);

extent = new ExtentReports();

extent.attachReporter(spark);

}

@Test

public void sampleTest() {

test = extent.createTest("Sample Test");

test.info("Executing a sample test");

test.pass("Test executed successfully");

}

@AfterSuite

public void tearDownReport() {

// Flush the report to finalize it

extent.flush();

// Automatically open the report in the default browser

File reportFile = new File("ExtentReport.html");

if (reportFile.exists()) {

try {

Desktop.getDesktop().browse(reportFile.toURI());

} catch (IOException e) {

System.err.println("Failed to open report: " + e.getMessage());

}

} else {

System.err.println("Report file does not exist!");

}

}

}

**Explanation**

1. **Generate Report:**
   * extent.flush() ensures the report is written to the specified file.
2. **Open Report in Browser:**
   * Use Desktop.getDesktop().browse() to open the report file in the system's default browser.
   * Ensure the path to the file (ExtentReport.html) is correct and accessible.
3. **Handle Exceptions:**
   * If the file cannot be found or the browser cannot open it, log an appropriate error message.

**Pre-Requisites**

* **Java AWT Desktop Class:** The Desktop class works on most systems, but you need a GUI-enabled environment. It won’t work in headless servers.
* **Report File Path:** Ensure the ExtentReport.html file path is correct. If needed, use absolute paths.

**Sample Output**

After the test finishes:

1. The Extent Report (ExtentReport.html) is generated.
2. The file automatically opens in the default web browser.

This approach provides a seamless experience for running tests and immediately reviewing results. Let me know if you need further assistance!

4o

**You said:**

need dependnecy for spotless apply

**ChatGPT said:**

ChatGPT

To use **Spotless** in your project, you need to add the appropriate dependencies and configuration to your build tool. Spotless is a code formatter that can enforce consistent formatting for various languages, including Java. It can integrate with Maven or Gradle.

**For Maven: Spotless Dependency**

Add the following plugin to your pom.xml:

xml

Copy code

<build>

<plugins>

<plugin>

<groupId>com.diffplug.spotless</groupId>

<artifactId>spotless-maven-plugin</artifactId>

<version>2.43.0</version> <!-- Use the latest version -->

<configuration>

<java>

<googleJavaFormat>

<version>1.15.0</version> <!-- Use the latest version -->

</googleJavaFormat>

</java>

</configuration>

</plugin>

</plugins>

</build>

**Usage in Maven**

* Run Spotless to check the code:

sh

Copy code

mvn spotless:check

* Automatically format the code:

sh

Copy code

mvn spotless:apply

**For Gradle: Spotless Dependency**

Add the following plugin to your build.gradle file:

groovy

Copy code

plugins {

id 'com.diffplug.spotless' version '6.22.0' // Use the latest version

}

spotless {

java {

googleJavaFormat('1.15.0') // Use the latest version

}

}

**Usage in Gradle**

* Run Spotless to check the code:

sh

Copy code

./gradlew spotlessCheck

* Automatically format the code:

sh

Copy code

./gradlew spotlessApply

**Custom Formatting**

If you want to customize formatting rules, Spotless supports a wide range of formatters and configurations. For example:

**Custom Configuration (Optional):**

groovy

Copy code

spotless {

java {

target 'src/\*\*/\*.java' // Specify target files

importOrder('java', 'javax', 'org', 'com') // Define import order

removeUnusedImports() // Automatically remove unused imports

googleJavaFormat('1.15.0')

}

}

Let me know if you need help configuring Spotless further or resolving formatting issues!

4o

**You said:**

if i want to run this i using gighub actions what shall i do

**ChatGPT said:**

ChatGPT

To run **Spotless** in your project using **GitHub Actions**, you need to set up a workflow that triggers the appropriate commands (e.g., spotless:check or spotlessApply) during your CI pipeline. Here's how you can set it up:

**Step 1: Create a GitHub Actions Workflow**

1. Navigate to your repository.
2. Go to .github/workflows/ (create this directory if it doesn't exist).
3. Create a file called spotless.yml (or any name you prefer).

**Step 2: Define the Workflow**

Here’s a sample workflow for both **Maven** and **Gradle** projects:

**For Maven Projects**

yaml

Copy code

name: Spotless Check

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

spotless-check:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v3

- name: Set up Java

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '17' # Use the appropriate JDK version for your project

- name: Run Spotless Check

run: mvn spotless:check

**For Gradle Projects**

yaml

Copy code

name: Spotless Check

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

spotless-check:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v3

- name: Set up Java

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '17' # Use the appropriate JDK version for your project

- name: Run Spotless Check

run: ./gradlew spotlessCheck

**Step 3: Automatically Fix Issues with Spotless**

If you want Spotless to automatically apply fixes (e.g., formatting corrections), update the command:

**Maven:**

yaml

Copy code

- name: Run Spotless Apply

run: mvn spotless:apply

**Gradle:**

yaml

Copy code

- name: Run Spotless Apply

run: ./gradlew spotlessApply

**Step 4: Add Workflow Results to Pull Requests**

To make Spotless comments visible in the pull request:

1. Use the github-script action to annotate the code or log the results.
2. Example for Maven:

yaml

Copy code

- name: Run Spotless Check

run: |

mvn spotless:check || echo "::error file=Spotless failed to validate code. Run mvn spotless:apply to fix formatting."

**Step 5: Trigger and Review**

Once set up:

* On every push or pull request to the specified branch (e.g., main), Spotless will run.
* The workflow will fail if the code is not properly formatted.

**Optional: Customize Triggers**

You can modify the on: section of the workflow to customize triggers. For example:

* To run only on pull requests:

yaml

Copy code

on:

pull\_request:

* To run on specific branches:

yaml

Copy code

on:

push:

branches:

- main

- dev

This setup ensures that Spotless checks (and optionally applies fixes) are automatically enforced during your CI pipeline on GitHub Actions. Let me know if you'd like further assistance!

4o

**You said:**

I want to run this complete project i made inusing github cations , can u help

**ChatGPT said:**

ChatGPT

To run your **complete Selenium TestNG project with Maven** on **GitHub Actions**, you need to configure a workflow that automates the following steps:

1. Checkout your project from the repository.
2. Set up the required Java environment.
3. Run the Maven build to execute your TestNG tests and generate reports (like Extent Reports).
4. Optionally, upload reports (e.g., Extent Reports) as artifacts for easier access.

Here’s how to set up your **GitHub Actions workflow**:

**Step 1: Create the Workflow File**

1. In your project repository, navigate to .github/workflows/.
2. Create a file called maven-tests.yml.

**Step 2: Define the Workflow**

Here’s a workflow example for running your Selenium TestNG project:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: ubuntu-latest

steps:

# Step 1: Checkout repository

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up Java environment

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '17' # Use the JDK version required for your project

# Step 3: Cache Maven dependencies to speed up builds

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: ~/.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Extent Reports (or other generated files) as artifacts

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: |

target/surefire-reports/ # Default TestNG report path

ExtentReport.html # Path to your Extent Report

**Step 3: Key Features of the Workflow**

* **Java Setup:** The workflow ensures that the required JDK version is installed using actions/setup-java.
* **Maven Dependency Caching:** Caching the .m2 directory speeds up subsequent builds by avoiding repeated downloads of dependencies.
* **Test Execution:** The mvn clean test command runs your TestNG tests.
* **Artifact Upload:** Generated reports, such as ExtentReport.html or surefire-reports, are uploaded as artifacts. You can download them from the **Actions** tab after the workflow finishes.

**Step 4: Trigger the Workflow**

The workflow runs automatically:

* **On Push**: Whenever you push changes to the main branch.
* **On Pull Request**: When you create or update a pull request targeting the main branch.

**Step 5: Review Test Results**

1. Navigate to the **Actions** tab in your GitHub repository.
2. Select the workflow run.
3. Check the logs for test execution results.
4. Download the uploaded artifacts (e.g., Extent Report or TestNG report) to review test output.

**Optional Enhancements**

1. **Parallel Testing with GitHub Runners:** If your Selenium project supports parallel execution, you can run tests faster by splitting them across multiple runners.
2. **Integrate with BrowserStack/Sauce Labs:** If using remote WebDriver services like BrowserStack, add their credentials to your repository's secrets and modify the test configurations.
3. **Send Notifications:** Use tools like Slack or email to notify the team of test results.

**Example Directory Structure**

Ensure your project has the expected structure:

bash

Copy code

.

├── src/

│ ├── test/

│ │ ├── java/ # Test classes

│ │ └── resources/ # TestNG XML config

├── target/

│ ├── surefire-reports/ # TestNG reports (generated automatically)

│ └── ExtentReport.html # Extent report file (custom path if set)

├── pom.xml # Maven build file

├── .github/

│ └── workflows/

│ └── maven-tests.yml # GitHub Actions workflow file

**Let Me Know**

If you need help with specific configurations (e.g., environment variables, integrating BrowserStack, or customizing reports), feel free to ask!

4o

**You said:**

I dont have .github/workflows/ in my project

**ChatGPT said:**

ChatGPT

If your project does not have the .github/workflows/ directory, you can create it manually. Here's how:

**Step-by-Step Instructions**

1. **Navigate to Your Project Directory:** Open the root directory of your project.
2. **Create the Directory Structure:**
   * Inside the root directory of your project, create the .github folder.
   * Inside .github, create a workflows folder.

The final structure should look like this:

css

Copy code

your-project/

├── src/

├── pom.xml

├── .github/

└── workflows/

1. **Add Your Workflow File:**
   * Inside the workflows folder, create a new YAML file, such as maven-tests.yml or any other name ending in .yml.
   * Paste the workflow content from my earlier response into this file.
2. **Commit and Push Changes:**
   * Add the .github/workflows/maven-tests.yml file to your repository:

bash

Copy code

git add .github/workflows/maven-tests.yml

git commit -m "Add GitHub Actions workflow for running tests"

git push origin main

**Verify in GitHub**

1. Go to your GitHub repository in a web browser.
2. Navigate to the **Actions** tab.
3. You should see the workflow listed there. If it's not triggered automatically, you can manually trigger it by clicking "Run workflow."

Let me know if you encounter any issues while setting it up!

4o

**You said:**

got this error in github action ubunto machine afer pusing my chnages Error: COMPILATION ERROR : [INFO] ------------------------------------------------------------- Error: Source option 5 is no longer supported. Use 7 or later. Error: Target option 5 is no longer supported. Use 7 or later. [INFO] 2 errors [INFO] ------------------------------------------------------------- [INFO] ------------------------------------------------------------------------ [INFO] BUILD FAILURE [INFO] ------------------------------------------------------------------------ [INFO] Total time: 10.320 s [INFO] Finished at: 2024-12-13T05:35:27Z [INFO] ------------------------------------------------------------------------ Error: Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.1:compile (default-compile) on project dec13th: Compilation failure: Compilation failure: Error: Source option 5 is no longer supported. Use 7 or later. Error: Target option 5 is no longer supported. Use 7 or later. Error: -> [Help 1] Error: Error: To see the full stack trace of the errors, re-run Maven with the -e switch. Error: Re-run Maven using the -X switch to enable full debug logging. Error: Error: For more information about the errors and possible solutions, please read the following articles: Error: [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException Error: Process completed with exit code 1.

**ChatGPT said:**

ChatGPT

The error Source option 5 is no longer supported. Use 7 or later occurs because your project is configured to use an outdated Java version (source and target set to 5). GitHub Actions machines use a more modern Java version (e.g., Java 11 or Java 17), which no longer supports these older options.

Here's how you can fix this:

**Step 1: Update maven-compiler-plugin Configuration in pom.xml**

Ensure your pom.xml specifies a supported Java version (e.g., 11 or 17). Update the maven-compiler-plugin section in your pom.xml:

xml

Copy code

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>11</source> <!-- Set this to 11, 17, or your desired version -->

<target>11</target> <!-- Ensure this matches the source version -->

</configuration>

</plugin>

</plugins>

</build>

If the maven-compiler-plugin section does not exist, add it under <build>.

**Step 2: Match the Java Version in GitHub Actions**

Ensure your GitHub Actions workflow specifies a Java version that matches the source and target versions in pom.xml. Update your workflow:

yaml

Copy code

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11' # Match this to your `pom.xml` version

**Step 3: Commit and Push Changes**

After updating pom.xml and the workflow file, commit and push the changes:

bash

Copy code

git add pom.xml .github/workflows/maven-tests.yml

git commit -m "Fix Java version compatibility in Maven and GitHub Actions"

git push origin main

**Step 4: Verify the Fix**

1. Go to your GitHub repository.
2. Navigate to the **Actions** tab.
3. Verify that the workflow runs successfully.

**Optional: Ensure Compatibility with Older Code**

If your codebase requires compatibility with older Java versions, you may need to:

* Refactor legacy code to align with newer Java versions.
* Ensure dependencies are updated to versions compatible with the selected Java version.

Let me know if you encounter further issues!

4o

**You said:**

Error: Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test (default-test) on project dec13th: There are test failures.

**ChatGPT said:**

ChatGPT

The error There are test failures indicates that some of your TestNG or JUnit tests failed during the mvn test phase. This issue can arise due to various reasons, such as test logic, environment configuration, or missing dependencies.

Here’s how you can troubleshoot and address the issue:

**Step 1: Review the Detailed Error Log**

* Look for the specific test failure messages in the GitHub Actions logs. They will provide details such as:
  + The test class and method that failed.
  + The error message or exception (e.g., NullPointerException, AssertionError).
  + Stack traces or logs related to the failure.

**Step 2: Reproduce Locally**

1. Run the tests locally on your machine to reproduce the issue:

bash

Copy code

mvn clean test

1. Check if the same tests fail. If they pass locally, the issue might be environment-specific (e.g., a missing file, incorrect configuration, or platform differences).

**Step 3: Common Issues and Fixes**

**1. Environment-Specific Issues**

If your tests rely on environment-specific settings, make sure the required dependencies or configurations are present in GitHub Actions:

* **Browser WebDriver:** If your tests depend on a WebDriver (e.g., ChromeDriver for Selenium), ensure the correct version is installed and set up in your workflow:

yaml

Copy code

- name: Set up ChromeDriver

uses: actions/runner-tool-cache@v2

with:

tool: chromedriver

version: 114 # Replace with your desired version

* **Environment Variables:** Ensure required environment variables are set in the workflow using env:

yaml

Copy code

env:

BASE\_URL: "https://your-testing-url"

API\_KEY: "your-api-key"

**2. Missing Files or Resources**

Ensure all required files (e.g., config.properties, test data) are included in the repository and correctly located in the test environment.

**3. Dependency Version Mismatches**

If your tests fail due to dependency issues, try upgrading the dependencies in pom.xml to compatible versions:

xml

Copy code

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.8.0</version>

</dependency>

**4. Assertions and Test Logic Issues**

If the failure is due to an assertion error (e.g., expected vs. actual values mismatch), review the failing test's logic and update as needed.

**Step 4: Optional - Allow Build to Continue on Test Failures**

If you want the workflow to complete even when tests fail (useful for debugging), you can temporarily set the -Dmaven.test.failure.ignore=true flag in your workflow:

yaml

Copy code

- name: Run Maven Tests

run: mvn clean test -Dmaven.test.failure.ignore=true

This approach allows you to:

* Review logs for the failing tests without stopping the build.
* Access reports (e.g., Extent Reports or TestNG XML) to diagnose issues.

**Step 5: Upload Reports for Debugging**

If you're generating reports (e.g., TestNG or Extent Reports), ensure they're uploaded as artifacts for debugging:

yaml

Copy code

- name: Upload Test Reports

if: always()

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: target/surefire-reports/

**Step 6: Fix the Tests**

Based on your findings:

1. Update test logic, configurations, or dependencies.
2. Re-run the workflow to verify fixes.

**Final Workflow File Example**

Here's a combined example:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

jobs:

run-tests:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

- name: Run Maven Tests

run: mvn clean test

- name: Upload Test Reports

if: always()

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: target/surefire-reports/

Let me know the specific test failures you encounter, and I can assist further!

4o

**You said:**

I want to run on windows jobs: run-tests: runs-on: ubuntu-latest

**ChatGPT said:**

ChatGPT

If you want to run your tests on a **Windows environment** in GitHub Actions instead of Ubuntu, you can simply change the runs-on field in your workflow configuration to use windows-latest. Here's how to update your GitHub Actions workflow:

**Updated Workflow for Windows**

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # Change from ubuntu-latest to windows-latest

steps:

# Step 1: Checkout repository

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11' # Match this to the version you're using

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: |

target\surefire-reports\

ExtentReport.html # Or other report paths

**Key Changes:**

1. **Set runs-on to windows-latest:** This tells GitHub Actions to run the job on a Windows environment. The default image used for Windows is windows-latest, which is currently based on Windows Server 2022.

yaml

Copy code

runs-on: windows-latest

1. **Adjust Paths for Windows:**
   * Windows uses backslashes (\) instead of forward slashes (/). For instance, change the cache path to use the appropriate Windows path:

yaml

Copy code

path: C:\Users\runner\.m2

* + Also, update any paths in your reports or file locations accordingly (for example, target\surefire-reports\).

1. **Maven on Windows:** Maven will work on Windows in the same way as it does on Ubuntu, as long as the JDK is set up correctly.

**Additional Considerations:**

1. **WebDriver and Browser Setup:** If you're using WebDriver (e.g., ChromeDriver) on Windows, ensure that you either:
   * Include the correct version of the WebDriver in your repository.
   * Download the WebDriver during the setup phase in the workflow.

For example, to set up ChromeDriver:

yaml

Copy code

- name: Set up ChromeDriver

uses: actions/runner-tool-cache@v2

with:

tool: chromedriver

version: 114 # Specify the version you need

1. **Java Setup:** The actions/setup-java step should work similarly across different operating systems, so make sure you're using the Java version compatible with your project.

**Commit and Push Changes**

After updating your workflow file, commit and push the changes to your GitHub repository:

bash

Copy code

git add .github/workflows/maven-tests.yml

git commit -m "Update GitHub Actions to run tests on Windows"

git push origin main

**Verify**

1. Go to the **Actions** tab in your GitHub repository.
2. Check if the workflow triggers and runs on Windows.
3. Review the logs for any issues and verify the test results.

Let me know if you encounter any problems while setting up the Windows job!

4o mini

**You said:**

add reports folder to gitignore

**ChatGPT said:**

ChatGPT

To add a **reports folder** to your .gitignore file, follow these steps:

**Step 1: Open or Create .gitignore**

If you already have a .gitignore file in the root of your project, open it. If it doesn’t exist, you can create it.

**Step 2: Add the reports Folder to .gitignore**

Add the following line to the .gitignore file to exclude the reports folder (and all its contents) from version control:

bash

Copy code

/reports/

This will ignore the reports folder at the root of the project. If your reports folder is located elsewhere, adjust the path accordingly (e.g., target/reports/ or \*/reports/ for nested folders).

**Step 3: Commit the Change**

After updating the .gitignore file, commit and push the changes:

bash

Copy code

git add .gitignore

git commit -m "Add reports folder to .gitignore"

git push origin main

**Step 4: Remove reports Folder from Git (if it’s already tracked)**

If the reports folder has already been tracked by Git, you'll need to remove it from version control. Run the following commands:

bash

Copy code

git rm -r --cached reports

git commit -m "Remove reports folder from Git tracking"

git push origin main

This will remove the folder from Git tracking but leave it on your local machine.

**Verify the Changes**

1. Ensure that the reports folder is no longer being tracked by Git:

bash

Copy code

git status

1. Make sure that no new reports folder changes appear in your GitHub repository after you push the updates.

Let me know if you need any further assistance!

4o mini

**You said:**

No files were found with the provided path: target/surefire-reports/ # Default TestNG report path ExtentReport.html # Path to your Extent Report. No artifacts will be uploaded.

**ChatGPT said:**

ChatGPT

The error **"No files were found with the provided path: target/surefire-reports/"** means that GitHub Actions couldn't find the specified report files to upload as artifacts, which could happen if:

1. The tests failed before the reports were generated.
2. The report files are in a different directory or have a different name than expected.

Here’s how to resolve this issue:

**Step 1: Ensure Reports Are Generated**

By default, the **Surefire plugin** generates test reports in the target/surefire-reports/ directory, but if you're using **Extent Reports** for custom reporting, ensure you're configuring it correctly in your tests.

**For Surefire Reports (TestNG)**

* Ensure the tests are running correctly and the Surefire plugin is generating reports in the target/surefire-reports/ folder.

The directory path should be:

bash

Copy code

target/surefire-reports/testng-\*.xml

**For Extent Reports**

* If you're generating **Extent Reports**, ensure you're creating and saving the report in the correct location (e.g., target/ExtentReport.html).
* Verify your test code or Extent Reports configuration to ensure the report is being generated.

Example of setting up Extent Reports in Java:

java

Copy code

ExtentReports extent = new ExtentReports();

ExtentHtmlReporter htmlReporter = new ExtentHtmlReporter("target/ExtentReport.html");

extent.attachReporter(htmlReporter);

**Step 2: Adjust Report Path in GitHub Actions Workflow**

Once you're sure the reports are generated in the right locations, update the file paths in the **GitHub Actions workflow** to match where the reports are stored.

**Example of Adjusted Workflow:**

yaml

Copy code

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

**Step 3: Debugging Report Generation**

* Add a debug step to your workflow to inspect the contents of the target directory after running the tests. This can help you verify where the reports are being stored:

yaml

Copy code

- name: List target directory

run: dir target

This will list the contents of the target directory, allowing you to confirm the paths of the generated reports.

**Step 4: Commit and Push Changes**

Once you’ve confirmed the correct paths and updated the workflow, commit and push the changes:

bash

Copy code

git add .github/workflows/maven-tests.yml

git commit -m "Fix report paths for artifact upload"

git push origin main

**Step 5: Verify**

1. Go to the **Actions** tab in your GitHub repository.
2. Check the workflow run to see if the reports are now correctly uploaded as artifacts.
3. Ensure that the reports appear under the **Artifacts** section at the end of the workflow.

Let me know if you need further assistance or if the issue persists!

4o mini

**You said:**

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

**ChatGPT said:**

ChatGPT

The error **"SLF4J: Failed to load class 'org.slf4j.impl.StaticLoggerBinder'"** typically occurs when SLF4J (Simple Logging Facade for Java) cannot find an implementation of the logging backend, such as **Logback** or **Log4j2**, at runtime.

SLF4J is a logging abstraction that requires a concrete logging implementation to handle the actual logging. If you don't provide one, you'll see this warning.

To fix this, you need to add an appropriate SLF4J binding to your pom.xml. Here are the steps to resolve the issue:

**Step 1: Add a SLF4J Binding Dependency**

Choose one of the commonly used SLF4J bindings (Logback or Log4j) and add it to your pom.xml.

**Option 1: Add Logback (Recommended)**

Logback is one of the most widely used and recommended logging frameworks for SLF4J.

Add this dependency to your pom.xml for Logback:

xml

Copy code

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version> <!-- Use the latest version -->

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.6</version> <!-- Use the latest version -->

</dependency>

**Option 2: Add Log4j2**

If you prefer to use Log4j2, you can add the following dependency:

xml

Copy code

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version> <!-- Use the latest version -->

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.17.1</version> <!-- Use the latest version -->

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.17.1</version> <!-- Use the latest version -->

</dependency>

**Step 2: Clean and Rebuild the Project**

Once you've added the appropriate dependencies, clean and rebuild your project:

bash

Copy code

mvn clean install

This will download the required libraries and resolve the SLF4J issue.

**Step 3: Verify**

* After rebuilding the project, the SLF4J warning should be resolved.
* If you still see the warning, make sure there are no conflicting versions of SLF4J or logging libraries in your project’s dependencies.

**Step 4: Optional - Configure Logging**

If you are using Logback or Log4j2, ensure you have the corresponding configuration file in the src/main/resources directory:

* **For Logback:** logback.xml or logback-spring.xml
* **For Log4j2:** log4j2.xml or log4j2-spring.xml

These configuration files define how the logging is handled, including log levels, loggers, and appenders (e.g., console, file, etc.).

Let me know if you need further assistance or if this resolves your issue!

4o mini

**You said:**

what to add in yml file to send report to my gmail id

**ChatGPT said:**

ChatGPT

To send the test report to your Gmail address via GitHub Actions, you need to add a step in your GitHub Actions workflow that will send an email after the tests are completed. You can use an SMTP server (like Gmail’s SMTP server) to send an email with an attachment (e.g., your test report).

You can use an email-sending action like [**mailer**](https://github.com/marketplace/actions/send-email) or [**msm-email-action**](https://github.com/msm-technology/msm-email-action) to send the report through Gmail. Here’s a step-by-step guide to send an email with the report attached:

**Step 1: Add Gmail SMTP Settings**

To send emails via Gmail's SMTP server, you'll need the following details:

* **SMTP server address:** smtp.gmail.com
* **Port:** 587 (for TLS)
* **Username:** Your Gmail address (e.g., youremail@gmail.com)
* **Password:** Your Gmail App Password (which can be generated via Google's 2-Step Verification).

**Step 2: Set Up Secrets in GitHub**

Before you use your Gmail credentials in the GitHub Actions workflow, store sensitive data (like your Gmail username and App Password) in GitHub secrets.

1. Go to your repository on GitHub.
2. Click **Settings** > **Secrets and variables** > **Actions**.
3. Add the following secrets:
   * GMAIL\_USERNAME: Your Gmail address (e.g., youremail@gmail.com)
   * GMAIL\_APP\_PASSWORD: Your Gmail App Password (don't use your regular Gmail password)
   * Optionally, you can add other secrets like TO\_EMAIL (recipient email) for flexibility.

**Step 3: Create the Workflow to Send the Email**

Add the following steps to your GitHub Actions .yml file:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # or ubuntu-latest depending on your environment

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

# Step 6: Send the Report via Email (using an email sending action)

- name: Send Test Report Email

uses: dawidd6/action-send-mail@v3

with:

server\_address: smtp.gmail.com

server\_port: 587

username: ${{ secrets.GMAIL\_USERNAME }}

password: ${{ secrets.GMAIL\_APP\_PASSWORD }}

to: ${{ secrets.TO\_EMAIL }} # Optionally set the recipient email

subject: "Test Report"

body: "Please find the attached test report."

attachments: target/ExtentReport.html # Add the report file here

**Explanation:**

1. **dawidd6/action-send-mail@v3:** This is a GitHub Action to send emails using an SMTP server.
   * **server\_address:** The SMTP server (Gmail's SMTP server).
   * **server\_port:** Port 587 (for TLS).
   * **username/password:** These are your Gmail credentials stored in GitHub secrets.
   * **to:** The recipient email, which can be a secret (${{ secrets.TO\_EMAIL }}).
   * **subject:** The subject of the email.
   * **body:** The body of the email.
   * **attachments:** Attach the report generated during the tests (e.g., target/ExtentReport.html).
2. **Upload Report:** The test report is uploaded and passed to the email action as an attachment.

**Step 4: Verify**

1. After pushing your changes, check the **Actions** tab on GitHub.
2. Once the workflow finishes, check your Gmail inbox to see the email with the attached report.

Let me know if you encounter any issues or need further clarification!

4o mini

**You said:**

what about my outlook email id if i want to share in that

**ChatGPT said:**

ChatGPT

If you want to send the report to your **Outlook** email address instead of Gmail, you can modify the GitHub Actions workflow to use Outlook's SMTP settings.

Here are the steps to update the workflow for Outlook email:

**Step 1: Set Up Secrets in GitHub (Outlook)**

You’ll need the following details for Outlook's SMTP server:

* **SMTP server address:** smtp-mail.outlook.com
* **Port:** 587 (for TLS)
* **Username:** Your Outlook email address (e.g., youremail@outlook.com)
* **Password:** Your Outlook app password (or regular password if you don’t use two-factor authentication)

**Set the Secrets:**

1. Go to your repository’s **Settings**.
2. Click **Secrets and variables** > **Actions**.
3. Add these secrets:
   * OUTLOOK\_USERNAME: Your Outlook email (e.g., youremail@outlook.com).
   * OUTLOOK\_APP\_PASSWORD: Your Outlook app password (for enhanced security, if 2FA is enabled).
   * Optionally, you can set a secret for the recipient's email, like TO\_EMAIL, if it's dynamic.

**Step 2: Update GitHub Actions Workflow**

Update your .yml workflow file to use **Outlook’s SMTP server** to send the report.

Here’s an updated version of the workflow:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # or ubuntu-latest depending on your environment

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

# Step 6: Send the Report via Email (using Outlook SMTP server)

- name: Send Test Report Email

uses: dawidd6/action-send-mail@v3

with:

server\_address: smtp-mail.outlook.com # Outlook SMTP server

server\_port: 587 # TLS port

username: ${{ secrets.OUTLOOK\_USERNAME }} # Outlook email address

password: ${{ secrets.OUTLOOK\_APP\_PASSWORD }} # Outlook app password or regular password

to: ${{ secrets.TO\_EMAIL }} # Recipient email (use secret for flexibility)

subject: "Test Report"

body: "Please find the attached test report."

attachments: target/ExtentReport.html # Attach the report file (adjust if necessary)

**Explanation:**

1. **SMTP Server for Outlook:** The SMTP server address is smtp-mail.outlook.com, and port 587 is used for sending the email securely over TLS.
2. **Secrets:** We are using GitHub secrets (${{ secrets.OUTLOOK\_USERNAME }}, ${{ secrets.OUTLOOK\_APP\_PASSWORD }}, ${{ secrets.TO\_EMAIL }}) for the email username, password, and recipient's email address.
3. **Attachment:** The target/ExtentReport.html is attached to the email as the test report.

**Step 3: Verify**

1. After pushing the changes to GitHub, go to the **Actions** tab of your repository.
2. Once the workflow finishes running, check your **Outlook inbox** to see the email with the attached test report.

**Outlook Authentication:**

If you are using Outlook with two-factor authentication (2FA), you need to create an **App Password** instead of using your regular password:

1. Go to your Microsoft account security settings.
2. Under **App passwords**, generate a new app password.
3. Use the app password in place of your regular password in the GitHub Actions secrets.

Let me know if you encounter any issues or need further assistance!

4o mini

**You said:**

Run dawidd6/action-send-mail@v3 with: server\_address: smtp-mail.outlook.com server\_port: 587 username: \*\*\* password: \*\*\* to: \*\*\* subject: Test Report body: Please find the attached test report. attachments: target/ExtentReport.html env: JAVA\_HOME: C:\hostedtoolcache\windows\Java\_Temurin-Hotspot\_jdk\11.0.25-9\x64 JAVA\_HOME\_11\_X64: C:\hostedtoolcache\windows\Java\_Temurin-Hotspot\_jdk\11.0.25-9\x64 Error: Input required and not supplied: from

**ChatGPT said:**

ChatGPT

The error message **"Input required and not supplied: from"** occurs because the dawidd6/action-send-mail action requires a **"from"** field (the sender’s email address) in addition to the to, subject, body, and attachments fields.

To resolve this, you need to provide the **sender's email** address in the workflow file. If you're sending the email from your Outlook account, you can use the same email address as the **sender**.

Here’s how to modify the workflow:

**Step 1: Update the Workflow with "from" Field**

You need to add a from field to specify the sender’s email address (which is typically the same as the username for Outlook). Update the GitHub Actions YAML file as follows:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # or ubuntu-latest depending on your environment

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v3

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

# Step 6: Send the Report via Email (using Outlook SMTP server)

- name: Send Test Report Email

uses: dawidd6/action-send-mail@v3

with:

server\_address: smtp-mail.outlook.com # Outlook SMTP server

server\_port: 587 # TLS port

username: ${{ secrets.OUTLOOK\_USERNAME }} # Outlook email address

password: ${{ secrets.OUTLOOK\_APP\_PASSWORD }} # Outlook app password or regular password

from: ${{ secrets.OUTLOOK\_USERNAME }} # Sender's email address (same as username)

to: ${{ secrets.TO\_EMAIL }} # Recipient email (use secret for flexibility)

subject: "Test Report"

body: "Please find the attached test report."

attachments: target/ExtentReport.html # Attach the report file (adjust if necessary)

**Explanation of Changes:**

1. **from**: The from field has been added to specify the sender's email address, which is typically the same as the username.
2. The value of from is set to ${{ secrets.OUTLOOK\_USERNAME }}, which refers to your Outlook email address stored in GitHub secrets.

**Step 2: Verify**

1. Push the changes to your repository on GitHub.
2. After the workflow runs, check the **Actions** tab in GitHub to see if the email has been sent successfully.
3. Verify that the email has arrived in the recipient's inbox with the attached report.

Let me know if you face any further issues!

4o mini

**You said:**

Deprecation notice: v1, v2, and v3 of the artifact actions The following artifacts were uploaded using a version of actions/upload-artifact that is scheduled for deprecation: "Test-Reports". Please update your workflow to use v4 of the artifact actions

**ChatGPT said:**

ChatGPT

To address the deprecation notice for actions/upload-artifact, you need to update your GitHub Actions workflow to use **version 4** of the actions/upload-artifact action, as versions v1, v2, and v3 are being deprecated.

**Update the Action Version**

Replace the existing actions/upload-artifact@v3 with actions/upload-artifact@v4 in your workflow YAML file.

Here's the updated step:

yaml

Copy code

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v4

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

**Explanation:**

* The version v4 of actions/upload-artifact is the latest and should be used instead of v1, v2, or v3 to avoid the deprecation warning.

**Final Workflow Example:**

Here is the updated workflow file with the new version of actions/upload-artifact:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # or ubuntu-latest depending on your environment

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v4

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

# Step 6: Send the Report via Email (using Outlook SMTP server)

- name: Send Test Report Email

uses: dawidd6/action-send-mail@v3

with:

server\_address: smtp-mail.outlook.com # Outlook SMTP server

server\_port: 587 # TLS port

username: ${{ secrets.OUTLOOK\_USERNAME }} # Outlook email address

password: ${{ secrets.OUTLOOK\_APP\_PASSWORD }} # Outlook app password or regular password

from: ${{ secrets.OUTLOOK\_USERNAME }} # Sender's email address (same as username)

to: ${{ secrets.TO\_EMAIL }} # Recipient email (use secret for flexibility)

subject: "Test Report"

body: "Please find the attached test report."

attachments: target/ExtentReport.html # Attach the report file (adjust if necessary)

**Next Steps:**

1. Commit and push the changes to your repository.
2. Check the **Actions** tab in GitHub to ensure the workflow runs successfully.
3. Ensure the artifact upload and email sending steps complete without any warnings.

Let me know if you need further assistance!

4o mini

**You said:**

Invalid login: 535 5.7.139 Authentication unsuccessful, basic authentication is disabled.

**ChatGPT said:**

ChatGPT

The error message **"Invalid login: 535 5.7.139 Authentication unsuccessful, basic authentication is disabled"** indicates that **basic authentication** (using your email and password) is no longer allowed by Outlook's SMTP server. Microsoft has disabled basic authentication for security reasons, and now Outlook requires **OAuth2 authentication** or an **App Password** when sending emails programmatically.

**Solution: Use an Outlook App Password**

If you're not using **OAuth2** but want to use **SMTP** for sending emails, you'll need to generate an **App Password** for your Outlook account.

**Steps to Use an App Password:**

1. **Enable Two-Factor Authentication (2FA) on Your Microsoft Account:**
   * Go to your Microsoft Account Security page.
   * Enable two-factor authentication (2FA) if it's not already enabled.
2. **Generate an App Password:**
   * After enabling 2FA, go to the same **Manage Security Settings** page.
   * Under **App passwords**, click **Create a new app password**.
   * An app password will be generated for you. Make sure to copy this password because it will not be shown again.
3. **Update Your GitHub Secrets:**
   * Go to the **Secrets and variables** > **Actions** section of your repository's settings.
   * Add a new secret called **OUTLOOK\_APP\_PASSWORD** and set its value to the app password you just generated.
4. **Update the GitHub Actions Workflow:**

Now, update the dawidd6/action-send-mail action to use the app password for authentication instead of your regular Outlook password.

Here’s how you update the workflow:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # or ubuntu-latest depending on your environment

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests

- name: Run Maven Tests

run: mvn clean test

# Step 5: Upload Test Reports (if necessary)

- name: Upload Test Reports

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v4

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/ExtentReport.html # Adjust this path if necessary

# Step 6: Send the Report via Email (using Outlook SMTP server)

- name: Send Test Report Email

uses: dawidd6/action-send-mail@v3

with:

server\_address: smtp-mail.outlook.com # Outlook SMTP server

server\_port: 587 # TLS port

username: ${{ secrets.OUTLOOK\_USERNAME }} # Outlook email address

password: ${{ secrets.OUTLOOK\_APP\_PASSWORD }} # Use the app password here

from: ${{ secrets.OUTLOOK\_USERNAME }} # Sender's email address (same as username)

to: ${{ secrets.TO\_EMAIL }} # Recipient email (use secret for flexibility)

subject: "Test Report"

body: "Please find the attached test report."

attachments: target/ExtentReport.html # Attach the report file (adjust if necessary)

**Summary of Changes:**

1. Generate an **App Password** from your Microsoft account (required due to 2FA).
2. Update the GitHub secrets with the generated **App Password**.
3. Use the **App Password** in the password field of the workflow instead of your regular Outlook password.

**Step 7: Push the changes**

After making these changes, push your updated workflow to GitHub, and it should authenticate successfully using the **App Password** and send the email.

Let me know if this resolves your issue or if you need further assistance!

4o mini

**You said:**

th artifact genrated in github action are in xml format. I need html repot , what needs to be done

**ChatGPT said:**

ChatGPT

To generate an HTML report instead of XML artifacts from your TestNG tests in GitHub Actions, you need to adjust your TestNG configuration and ensure that **TestNG** is set to generate an HTML report after the tests are run.

**Steps to Generate HTML Report with TestNG:**

1. **Configure TestNG to Generate HTML Reports**

TestNG can generate both XML and HTML reports by default. However, you need to ensure that **TestNG** is configured to output the HTML report in the correct location.

* + **Add a testng.xml file (if you don't have one):** This file defines your test configuration, and you can specify the report formats (HTML and/or XML) inside it.

**Example of testng.xml (make sure it’s in your project’s root or the location you prefer):**

xml

Copy code

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Test Suite" verbose="1" parallel="tests" thread-count="2">

<test name="TestNG Tests">

<classes>

<class name="com.example.tests.TestClass1" />

<class name="com.example.tests.TestClass2" />

<!-- Add all your test classes here -->

</classes>

</test>

</suite>

1. **Configure TestNG Listener for HTML Reports**

You can use the org.uncommons.reportng.HTMLReporter to generate a nice HTML report after the tests have finished. You can do this by adding it to your TestNG configuration in your pom.xml.

**Add the Maven dependency for ReportNG in pom.xml:**

xml

Copy code

<dependencies>

<dependency>

<groupId>org.uncommons</groupId>

<artifactId>reportng</artifactId>

<version>1.1.4</version>

</dependency>

</dependencies>

**Configure the testng.xml file** to use the HTMLReporter listener:

xml

Copy code

<suite name="Test Suite" verbose="1" parallel="tests" thread-count="2">

<listeners>

<listener class-name="org.uncommons.reportng.HTMLReporter" />

<listener class-name="org.uncommons.reportng.JUnitXMLReporter" />

</listeners>

<test name="TestNG Tests">

<classes>

<class name="com.example.tests.TestClass1" />

<class name="com.example.tests.TestClass2" />

</classes>

</test>

</suite>

1. **Configure the GitHub Actions Workflow to Capture HTML Reports**

In your GitHub Actions workflow, make sure to capture and upload the generated HTML report.

Example GitHub Actions YAML file to capture and upload the HTML report:

yaml

Copy code

name: Run Selenium Tests

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

run-tests:

runs-on: windows-latest # or ubuntu-latest depending on your environment

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v3

# Step 2: Set up JDK

- name: Set up JDK

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '11'

# Step 3: Cache Maven dependencies

- name: Cache Maven dependencies

uses: actions/cache@v3

with:

path: C:\Users\runner\.m2

key: ${{ runner.os }}-m2-${{ hashFiles('\*\*/pom.xml') }}

restore-keys: |

${{ runner.os }}-m2-

# Step 4: Run Maven Tests (ensure HTML report is generated)

- name: Run Maven Tests

run: mvn clean test -DsuiteXmlFile=testng.xml

# Step 5: Upload Test Reports (Upload HTML Report)

- name: Upload Test Report (HTML)

if: always() # Always run, even if tests fail

uses: actions/upload-artifact@v4

with:

name: Test-Reports

path: |

target/surefire-reports/\*.xml

target/test-classes/TEST-\*.html # Adjust path if necessary

**Key Adjustments:**

1. **Generate HTML reports**: Ensure TestNG is configured to generate an HTML report using the HTMLReporter listener.
2. **Upload HTML report**: Modify the upload step to capture the HTML report from the appropriate folder (target/test-classes/TEST-\*.html or wherever your HTML report is located).

**Note:**

1. **Directory for HTML Reports**: TestNG's HTML report is usually placed in the target/test-classes/ directory. If you have a custom path, adjust the workflow to point to that directory.
2. **Maven Surefire Plugin**: If you're not using ReportNG, the maven-surefire-plugin also supports generating HTML reports through the testng-report configuration.

By following these steps, you should be able to generate and upload the HTML reports from your TestNG tests in your GitHub Actions pipeline. Let me know if you face any issues!