Step 1: Environment Setup

- Install Java: Make sure Java is installed on your machine. You can download it from the official Oracle website and follow the installation instructions.
- Install Maven: Download Maven from its official website and follow the setup instructions. Maven will manage your project's build and dependencies.
- IDE Setup: Install an IDE that supports Java and Maven (e.g., IntelliJ IDEA, Eclipse).
- Selenium WebDriver: Selenium WebDriver will be added as a dependency in your Maven project, so it's automatically downloaded by Maven.
- TestNG and Cucumber: Similarly, TestNG and Cucumber will be added as dependencies in your project.

Step 2: Create Maven Project

a. Create a New Maven Project: In your IDE, create a new Maven project.
b. Add Dependencies: Edit your pom.xml file to include Selenium, TestNG, and Cucumber dependencies.
c. Example **pom.xml** dependencies section:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>CucumberDemo</groupId>
 <artifactId>CucumberDemo</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <dependencies>
        <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.13.2</version>
          <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
        <dependency>
          <groupId>io.cucumber</groupId>
          <artifactId>cucumber-junit</artifactId>
          <version>7.16.1</version>
          <scope>test</scope>
        </dependency>

  <!-- Selenium WebDriver -->
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.19.1</version>
  </dependency>
  <!-- TestNG -->
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.9.0</version>
  </dependency>
  <!-- Cucumber -->
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.16.1</version>
  </dependency>
  <dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-testng</artifactId>
    <version>7.16.1</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-compiler-plugin -->
        <dependency>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.13.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-plugin -->
        <dependency>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>3.2.5</version>
        </dependency>


</dependencies>

<properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
</properties>

</project>
```

Step 3: Configure Cucumber and TestNG

a. Feature File: Create a new directory named **features** inside **src/test/resources**. Here, you will store your .feature files, which define your test scenarios in Gherkin language.

b. Example feature file (psaWebsite.feature):

```
Feature: Testing PSA website functionality

 Scenario: Home page title test
  Given The user is on the PSA home page
  When The user retrieves the title of the page
  Then The title of the page should be " Home | Philippine Statistics Authority | Republic of the Philippines"

 # Add more scenarios for each test case
```

c. Step Definitions: Create a new package under src/test/java for your step definitions. Implement the steps defined in your .feature files.

d. Inside **src/test/java** create a new package (“stepDefinitions”) and a new class inside (“HomePageSteps.java”):

```
package stepDefinitions;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.cucumber.java.en.*;
import static org.testng.Assert.assertEquals;

public class HomePageSteps {
  WebDriver driver;

  @Given("^The user is on the PSA home page$")
  public void the_user_is_on_the_psa_home_page() {
    System.setProperty("webdriver.chrome.driver", "path_to_chromedriver");
    driver = new ChromeDriver();
    driver.get("https://psa.gov.ph");
  }

  @When("^The user retrieves the title of the page$")
  public void the_user_retrieves_the_title_of_the_page() {
    String title = driver.getTitle();
    System.out.println("Page Title: " + title);
  }

  @Then("^The title of the page should be \"([^\"]*)\"$")
  public void the_title_of_the_page_should_be_something(String expectedTitle) {
    assertEquals(driver.getTitle(), expectedTitle);
    driver.quit();
  }
}
```

Step 4: Test Runner Class

a. Inside **src/test/java** create a new package (“runner”) and a new class inside (“RunnerTest.java”) that allows you to run the tests with TestNG.

b. IMPORTANT: Append your test class name with “test”, example “Runner**Test**.java”

c. Example (RunnerTest.java):

```
package runner;

import org.testng.annotations.DataProvider;
import io.cucumber.testng.AbstractTestNGCucumberTests;
import io.cucumber.testng.CucumberOptions;

@CucumberOptions(
    features = "src/test/resources/features",
    glue = "stepDefinitions"
)
public class TestRunner extends AbstractTestNGCucumberTests {
  @Override
  @DataProvider(parallel = false)
  public Object[][] scenarios() {
    return super.scenarios();
  }
}
```

Step 5: Writing Test Cases

You now have the basic structure set up. You can proceed to write additional test cases in the .feature file and implement their step definitions. Here are some test ideas:

- Navigation Test: Test navigation to different parts of the site, like "About Us" or "Statistics" sections.
- Search Functionality Test: Test the search functionality, if available, by searching for specific terms and verifying the results.
- Contact Form Test: If there's a contact form, test submitting it with valid and invalid data.
- Responsive Design Test: Test the website's responsiveness by adjusting the browser's size and ensuring the site adjusts correctly.

Step 6: Running Tests

In Eclipese, run your tests by executing:

- ✓ The Test Runner class.
- ✓ The pom.xml (Run→Maven test)

Alternatively, you can run your tests from the command line by navigating to your project directory and running mvn test.

```
mvn clean test
```

Step 7: Running the test from Jenkins

1. Install Jenkins
   - If you haven't already, download and install Jenkins from the official Jenkins website.
   - Follow the installation instructions for your specific operating system.

2. Install Required Jenkins Plugins
   - Maven Integration plugin: This is usually installed by default.
   - JUnit plugin: For TestNG and Cucumber test report integration.

   To install plugins in Jenkins:
   a. Navigate to Jenkins Dashboard > Manage Jenkins > Manage Plugins.
   b. Go to the Available tab, search for the plugins, and install them.

3. Configure Maven in Jenkins
   a. Go to Manage Jenkins > Global Tool Configuration.
   b. Scroll down to Maven installations and click Add Maven.
   c. Provide a name for the Maven installation and ensure Install automatically is selected.

   d. Jenkins will download and install Maven based on the settings provided. You can also configure a specific version of Maven if necessary.

4. Create a New Jenkins Job
   a. Go back to the Jenkins Dashboard and click New Item.
   b. Enter a name for your project, select Maven project, and click OK.
   c. In the Source Code Management section, select the type of repository you use (e.g., Git) and provide the repository URL and credentials if required.

5. Configure the Build
   a. In the Build section, specify the Root POM as pom.xml and Goals and options as clean test. This tells Jenkins to clean the previous build artifacts and run the tests.
   b. In the Post-build Actions section, add a post-build action to Publish JUnit test result report. Configure it to find the Cucumber test reports. You might need to specify the TestNG XML report pattern, which is typically **/testng-results.xml or for Cucumber JSON reports, **/cucumber.json.

6. Run the Build
   a. Once the job is configured, click Save.
   b. Back at the job summary page, click Build Now to start the build process.
   c. Jenkins will check out your code, run the Maven goals specified, and execute your Cucumber tests.

7. View the Results
   a. After the build completes, you can view the results directly in Jenkins. The build history will show the status of each build.
   b. Click on a specific build and navigate to the Test Result link to view detailed test reports.