Aim: To Setup and Run Selenium Tests in Jenkins Using Maven

Objective: Objective is to setup enables seamless integration of automated testing into the CI/CD pipeline, facilitating faster feedback loops and promoting a culture of continuous improvement in software development.

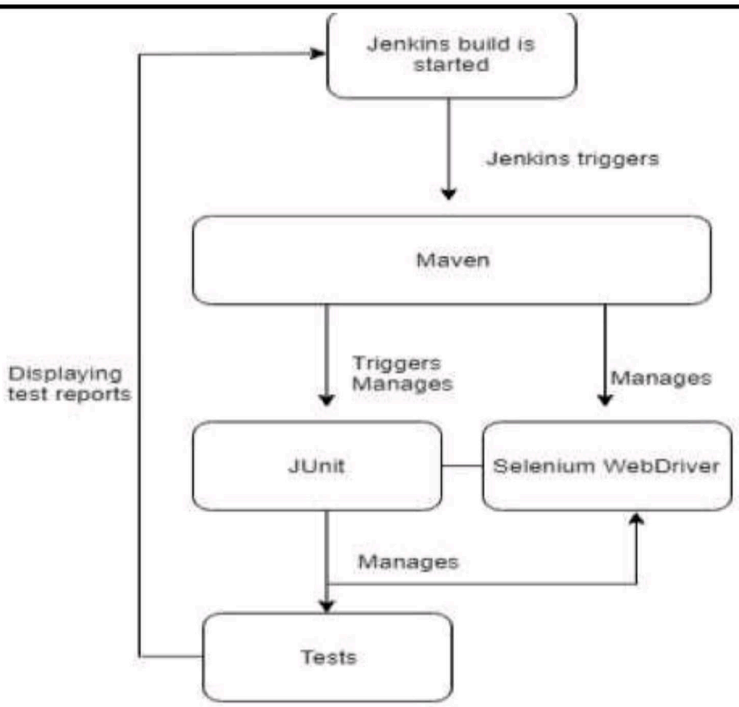
Theory:

Jenkins is the leading open-source continuous integration tool developed by Hudson lab. It is cross-platform and can be used on Windows, Linux, Mac OS and Solaris environments. Jenkins is written in Java. It has taken the place as one of the best open-source tools that allow continuous integration and build management.

Running Selenium tests in Jenkins allows you to run your tests every time your software changes and deploy the software to a new environment when the tests pass. Jenkins can schedule your tests to run at specific time. You can save the execution history and Test Reports. Jenkins supports Maven for building and Testing a project in continuous integration

Maven is a powerful project / build management tool, based on the concept of a POM (Project Object Model) that includes project information and configuration information for Maven such as construction directory, source directory, dependency, test source directory, Goals, plugins, etc.

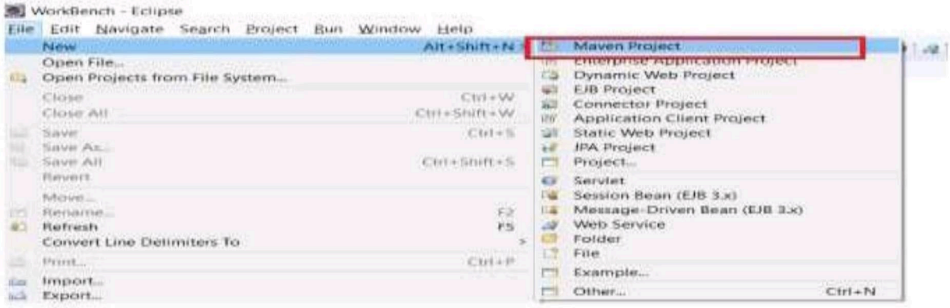
Integrating Maven with Selenium provides following benefits Apache Maven provides support for managing the full lifecycle of a test project. Maven is used to define project structure, dependencies, build, and test management. Using pom.xml(Maven) you can configure dependencies needed for building testing and running code. Maven automatically downloads the necessary files from the repository while building the project.



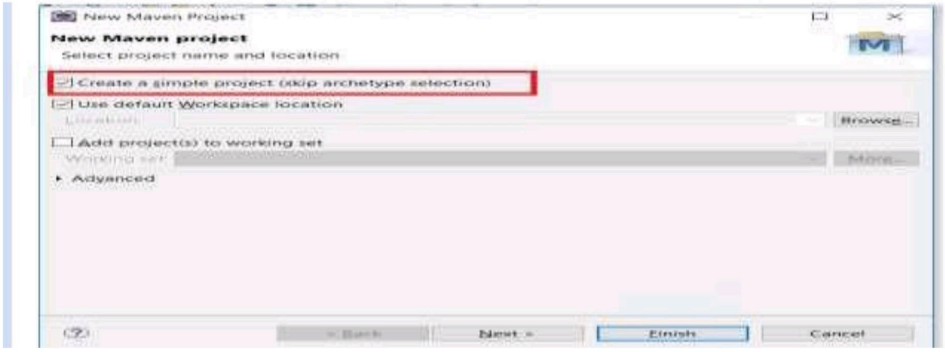
Steps:

---Create a Maven Selenium script---

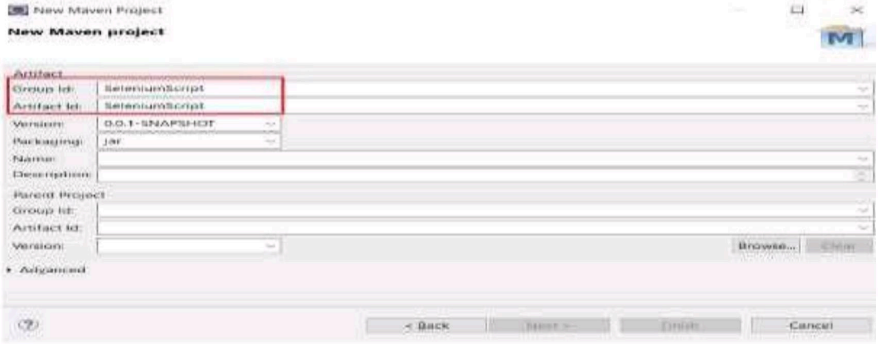
1. In Eclipse IDE, create a new project by selecting **File | New | Maven Project** from Eclipse menu.



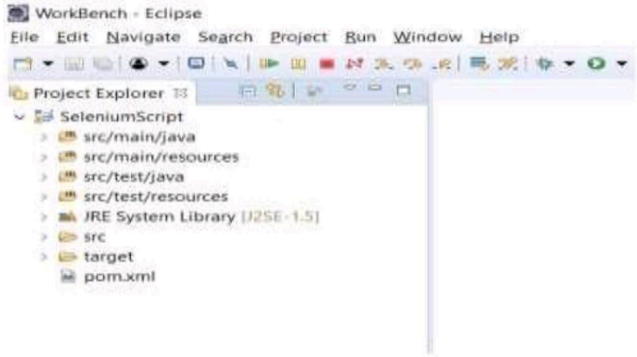
2. On the **New Maven Projec**t dialog select the **Create a simple project** and click **Next**



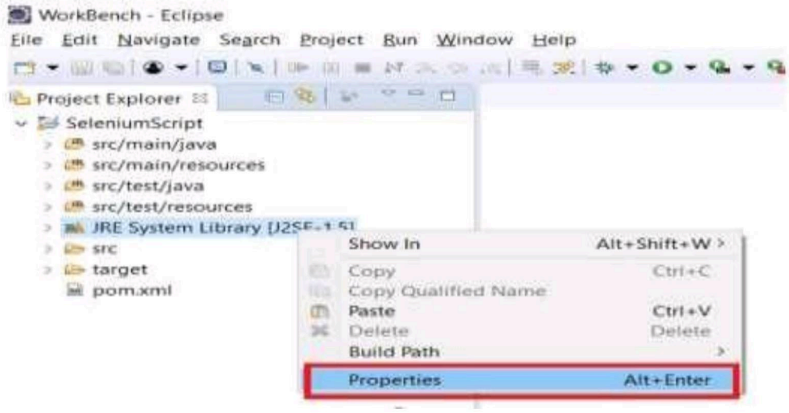
3. Enter **SeleniumScript** in **Group Id**: and **Artifact Id**: and click finish



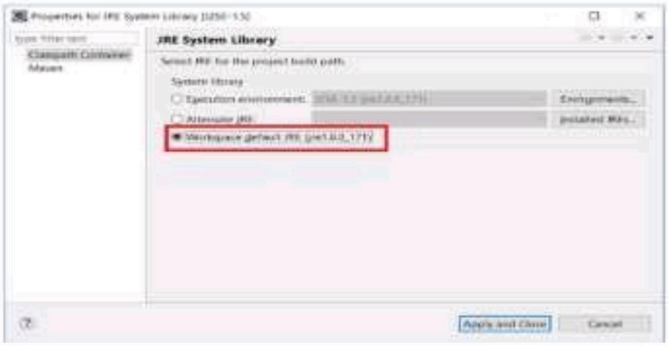
4. Eclipse will create webdriverTest.



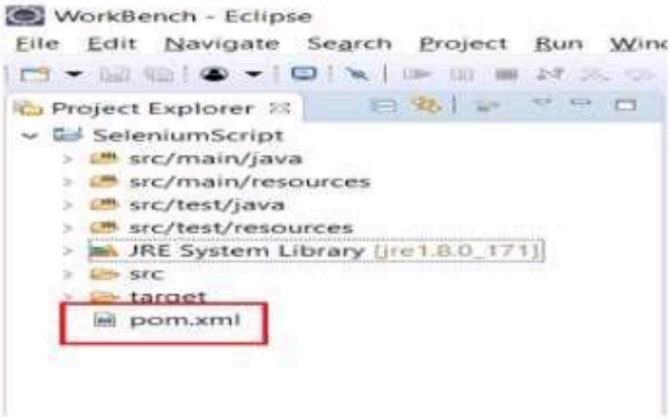
5. Right click on JRE System Library and select the Properties option from the menu.



6. On the Properties for JRE System Library dialog box , make sure Workspace default JRE is selected and click ok.



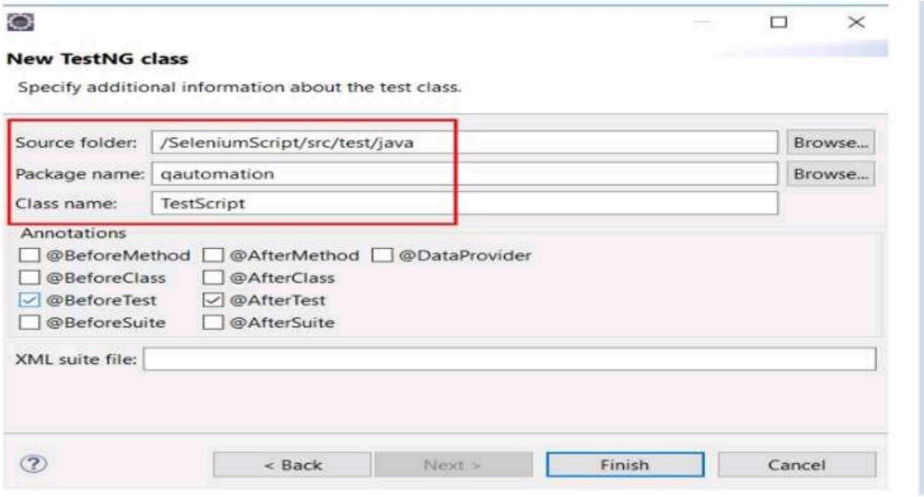
7. Select pom.xml from project explorer.



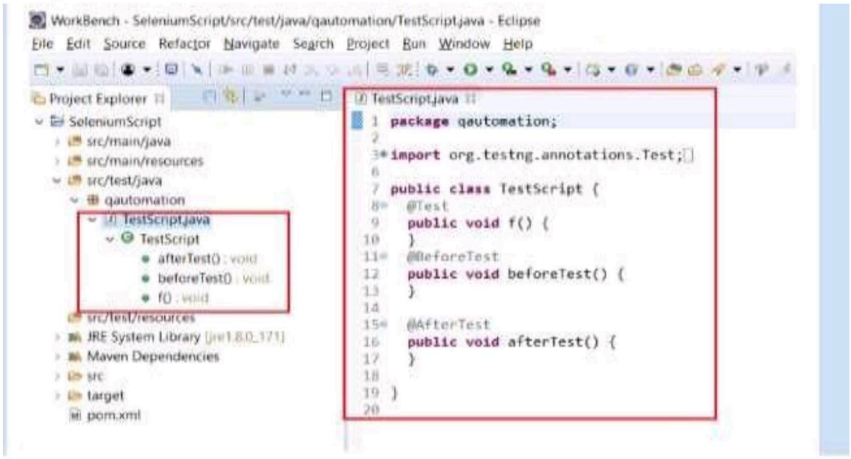
8. Add selenium, Maven, TestNG, Junit dependencies to pom.xml in the code.

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.45.0</version>
                </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.8</version>
            <scope>test</scope>
        </dependency>
</dependencies>
```

9. Create a new file TestNG class File|New|Others|TestNG|TestNG Class. Enter Package name as "Qautomation" and "TestScript" in the Name:textbox and click on the Finish button.



10. Eclipse will create the TestScript class



11. Add following code to the TestScript class and respective browser drivers for chrome,firefox and IE.

```
package qautomation;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.firefox.FirefoxProfile;
import org.openqa.selenium.ie.InternetExplorerDriver;
```

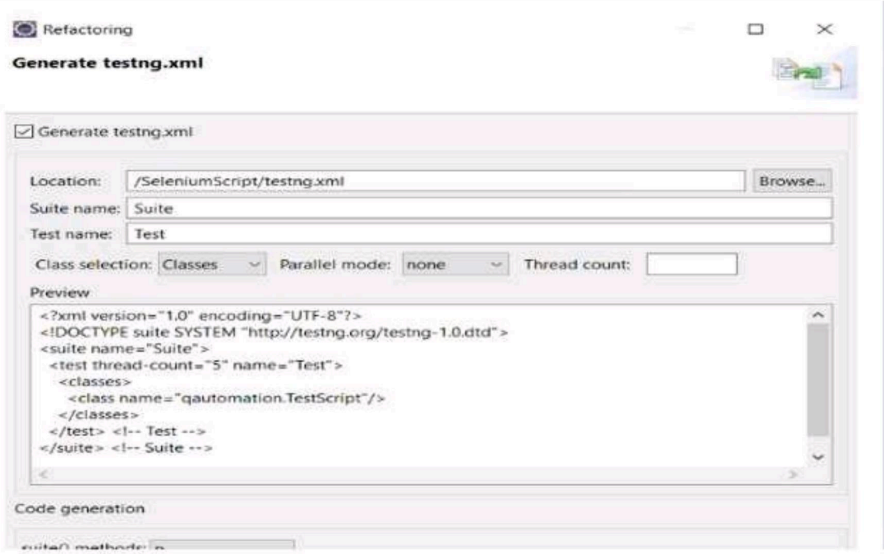
```
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
public class TestScript {
        public static WebDriver driver=null;
        public String browser = System.getProperty("browser");
        public String url = System.getProperty("URL");

 @BeforeTest
  public void beforeTest() {

        if(browser.equalsIgnoreCase("Chrome"))
        {
        System.setProperty("webdriver.chrome.driver",
        System.getProperty("user.dir")+"\\chromedriver.exe");
        Map<String, Object> prefs = new HashMap<String, Object>();
        ChromeOptions options = new ChromeOptions();
        options.setExperimentalOption("prefs", prefs);
        options.addArguments("--disable-arguments");
        options.addArguments("--test-type");
        options.addArguments("test");
        options.addArguments("disable-infobars");
        driver = new ChromeDriver(options);
        }
        else if(browser.equalsIgnoreCase("FireFox"))
        {
        System.setProperty(FirefoxDriver.SystemProperty.DRIVER_USE_MARIONETTE
        ,"true");
        System.setProperty(FirefoxDriver.SystemProperty.BROWSER_LOGFILE,Syste
        m.getProperty("user.dir")+"\\FireFoxLogs.txt");
        System.setProperty("webdriver.gecko.driver",
        System.getProperty("user.dir")+"\\geckodriver_v23.exe");
        FirefoxProfile profile = new FirefoxProfile();
        profile.setAcceptUntrustedCertificates(false);
        FirefoxOptions options = new FirefoxOptions().setProfile(profile);
        driver = new FirefoxDriver(options);
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        }
        else if (browser.equalsIgnoreCase("IE"))
        {
        System.setProperty("webdriver.ie.driver",
        System.getProperty("user.dir")+"\\IEDriverServer351.exe");
        DesiredCapabilities caps = DesiredCapabilities.internetExplorer();
```

```
                caps.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNO
                RING_SECURITY_DOMAINS,true);
                caps.setCapability(InternetExplorerDriver.IGNORE_ZOOM_SETTING,true);
                caps.setCapability(InternetExplorerDriver.UNEXPECTED_ALERT_BEHAVIOR,"
                accept");
                caps.setCapability(InternetExplorerDriver.REQUIRE_WINDOW_FOCUS,true);
                caps.setCapability(InternetExplorerDriver.INITIAL_BROWSER_URL,"http://www.
                google.com/");
                driver = new InternetExplorerDriver(caps);
                driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
                driver.manage().window().maximize();
                }
                driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
                driver.manage().window().maximize();
        }
        @Test
        public void TestApplication() {
        driver.get(url);
        String title = driver.getTitle();
        System.out.println("Title="+title);
        Assert.assertTrue(title.contains("QAutomation"));
        }
        @AfterTest
        public void afterTest() {
                driver.quit();
        }
}
```

12. Right click on the WebdriverTest and select TestNG| Convert to TestNG. Eclipse will create testing.xml which says that you need to run only one test with the name TestApplication.

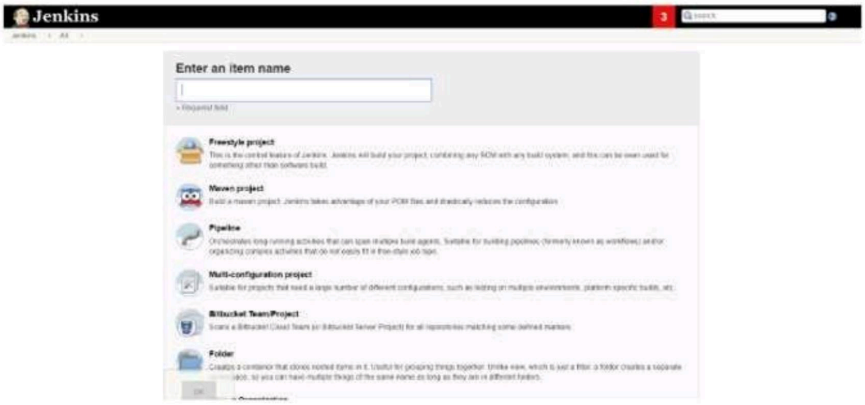


13. Adding dependencies and plugins
    Additionally we need to add
    1. Maven-compiler-plugin
    2. Maven-surefire-plugin
    3. Testng.xml

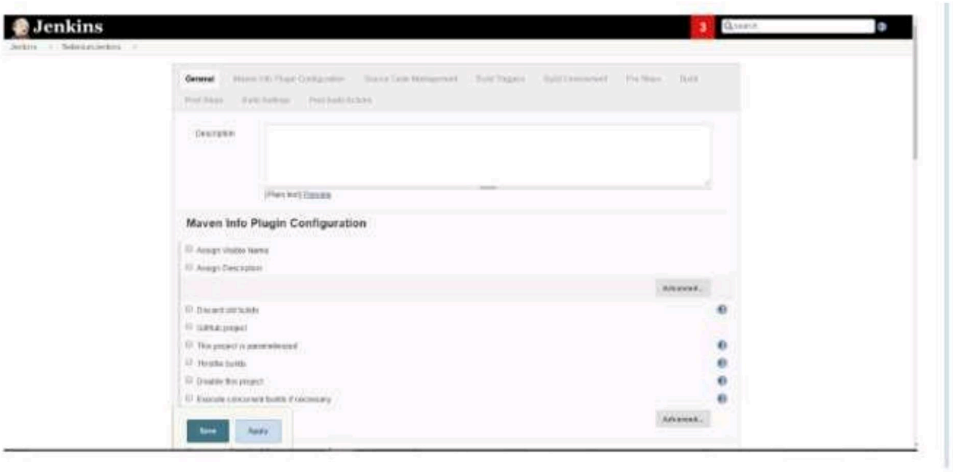
---------Integrating your test to Jenkins----------

1. Launch and login into jenkins URL – http://localhost:8080/

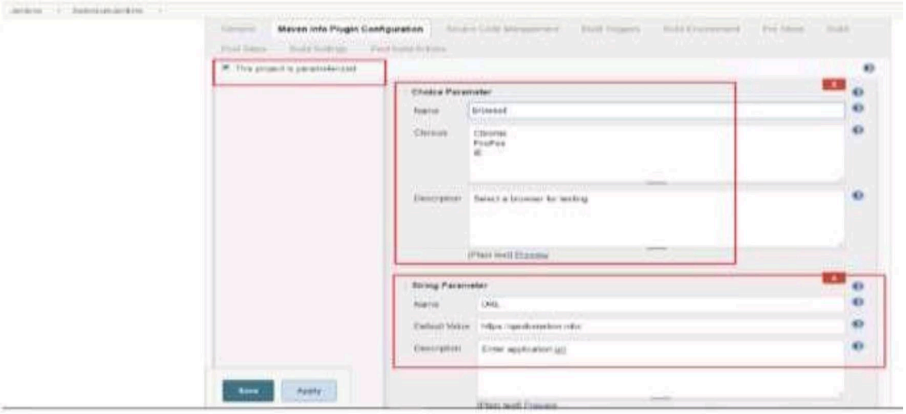
2. Click on new item and enter an appropriate name for the new job , select Maven Project and click on save.



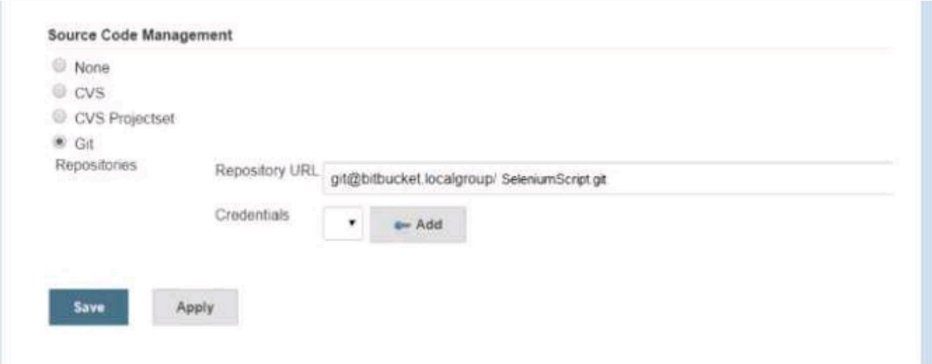
3. A new empty job has been created at this point.



4. Jenkins Parameterized Build in Jenkins just check the checkbox **This project is paramerized** and add the parameter by **Add Parameter** as per your project requirement.



5. If code is located on Git Under **Source Management** , select the appropriate repository for the location of project and pass the URL and credentials.



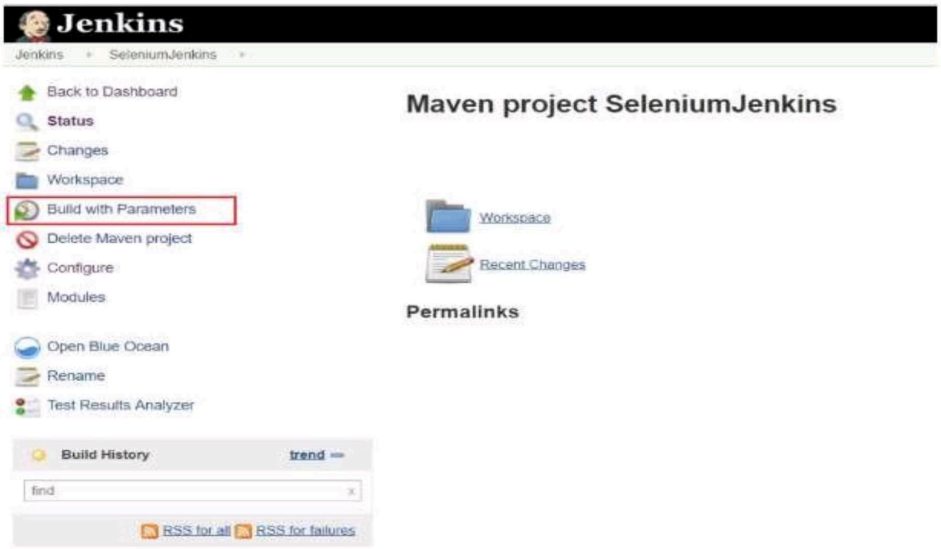
6. In the "pre-steps" build section another set of parameters can be passed to the Jenkins build. Specify the Maven targets that need to be executed in order to run test.

if your source code is located on Git the do below setting under **Build** section:

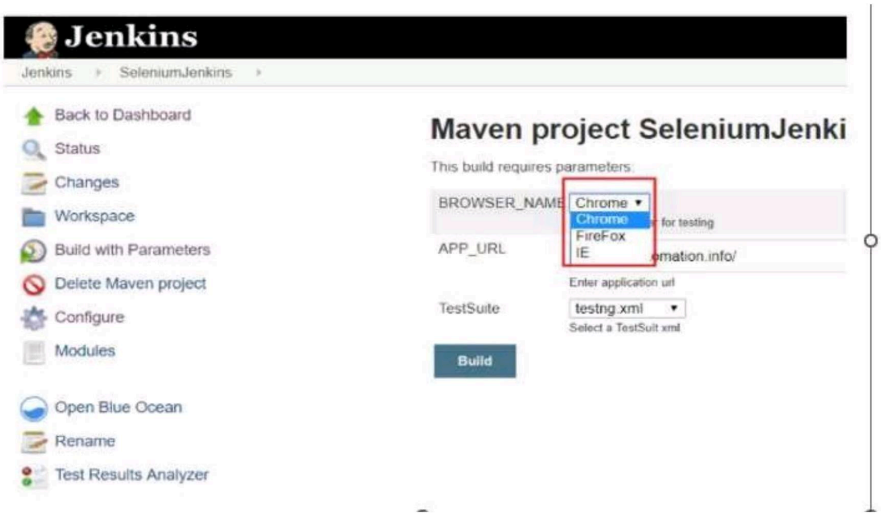
If you have selenium code on your local just pass the pom.xml path in ***Root POM***.

7. Run the test in Jenkins by clicking on Building with Parameters.

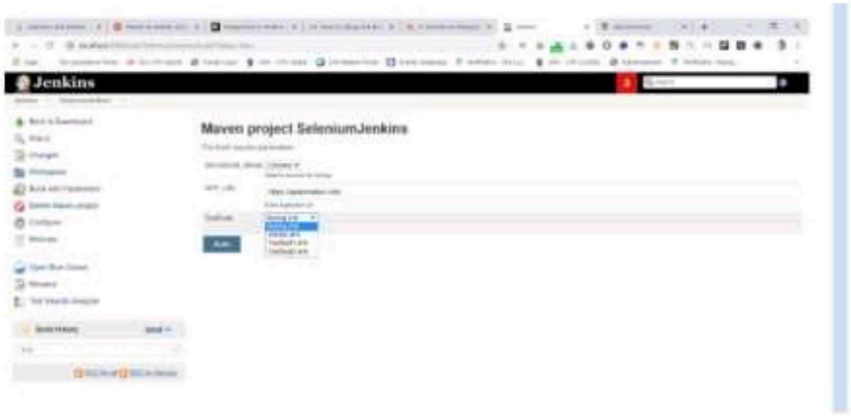
8. Run the test in Jenkins by clicking on ***Build with Parameters*** .



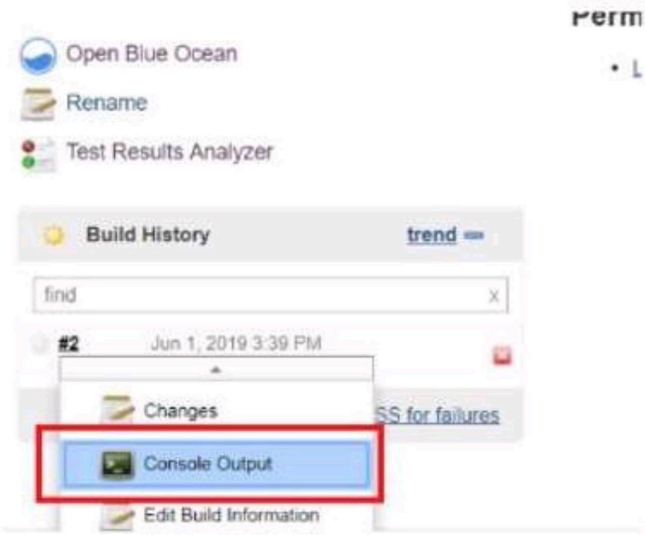
8. Select the browser you want to run from dropdown.



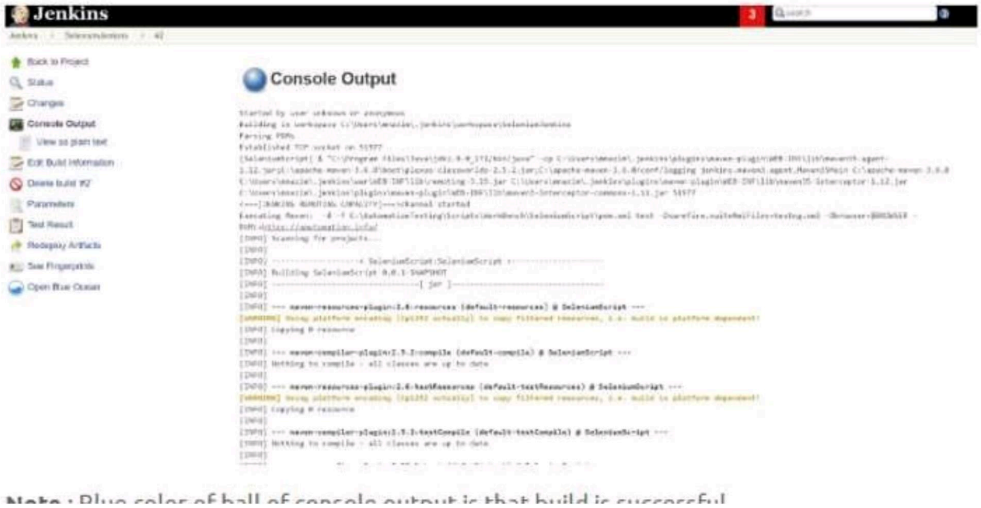
9. Select the TestSuit file.



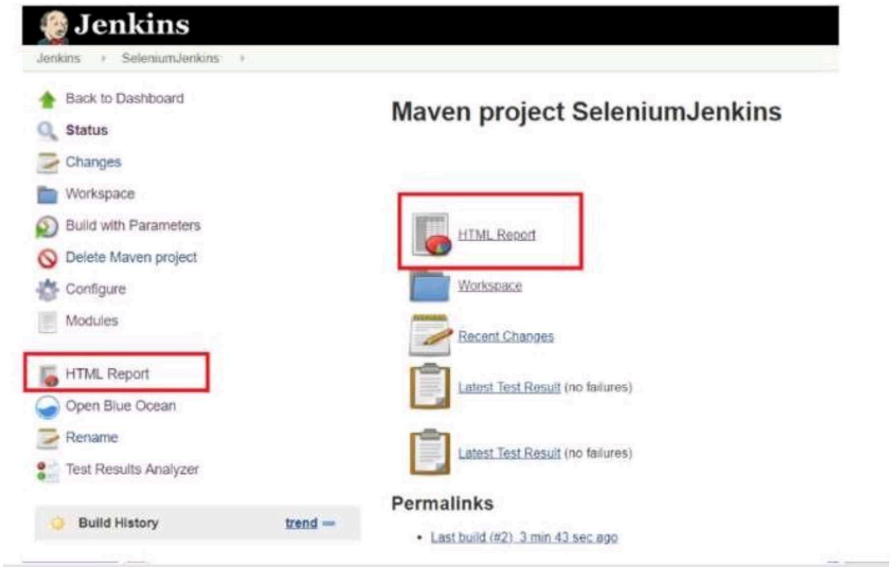
10. Click the build button and go to console output .

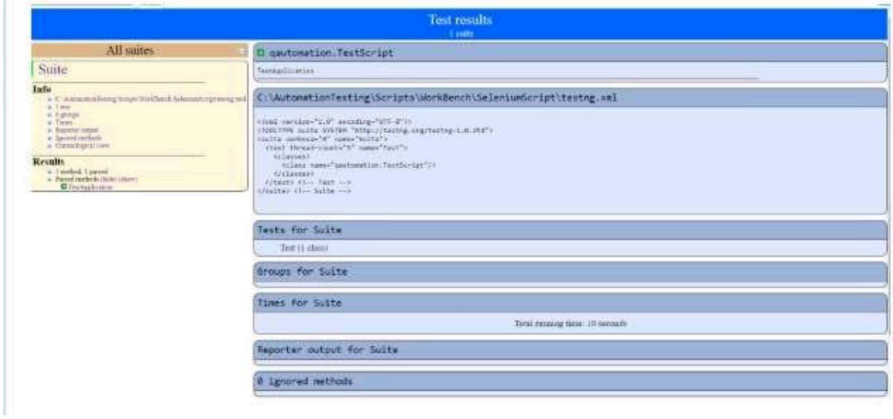


11. See the logs from **Console Output** window.

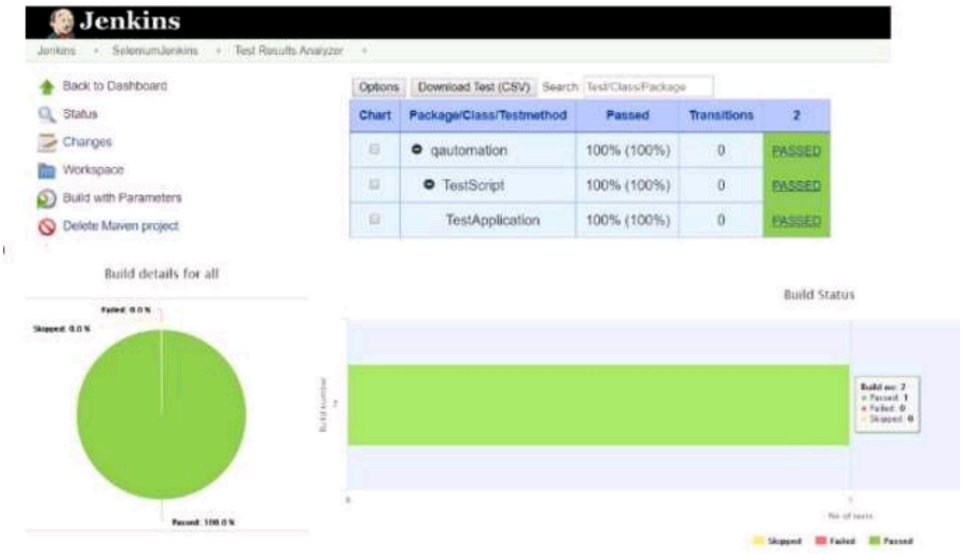


12. View the html report just click on the link.





13. Click Test Analyzer to analyse the result.



Conclusion:

Q1. Which browsers are supported by selenium webdriver?

Selenium WebDriver supports a wide range of browsers, making it a versatile tool for web application testing. The major browsers supported include Google Chrome, Mozilla Firefox, Internet Explorer, Safari, and Microsoft Edge. Selenium also provides support for Opera and various headless browsers like Headless Chrome and Headless Firefox, which are useful for test environments without a graphical user interface. To interact with these browsers, Selenium uses browser-specific drivers, such as ChromeDriver for Chrome, GeckoDriver for Firefox, and others. Each driver acts as a bridge between the Selenium WebDriver and the browser, enabling automation and control of browser activities within the scope of testing scenarios.

Q2. What are some features of selenium 4?

Selenium 4, the latest major release of the popular web automation framework, introduces several significant enhancements and new features. It includes an updated WebDriver interface which now fully conforms to the W3C WebDriver standard, improving compatibility and consistency across different browser drivers. Selenium 4 also introduces improved support for modern web technologies through enhanced interactions with the browser's DevTools, enabling features like network interception and more detailed emulation capabilities. The revamped Selenium Grid offers easier setup and more robust scalability for distributed testing, including better support for Docker containers. Additionally, Selenium 4 improves debugging and observability with more comprehensive logging and screenshot capabilities. The inclusion of relative locators (also known as friendly locators) allows for more intuitive element selection based on spatial relationships, enhancing the ease of test script development.