The automated testing of a website can be built by creating Java page objects representing the web application using the Selenium WebDriver and a headless environment such as PhantomJS Driver, and then porting the solution into a continuous integration environment (Jenkins).

Below are the Maven dependencies to execute an automated testing solution using Selenium WebDriver, PhatomJS Driver, and JUnit.

|  |  |
| --- | --- |
| Description | Maven Dependency |
| Selenium WebDriver | <dependency> <groupId>org.seleniumhq.selenium</groupId>  <artifactId>selenium-java</artifactId>  <version>2.53.0</version>  </dependency>  <dependency> <groupId>org.seleniumhq.selenium</groupId>  <artifactId>htmlunit-driver</artifactId>  <version>2.21</version>  </dependency> |
| PhatomJSDriver (Ghost Driver) | <dependency>  <groupId>com.github.detro</groupId> <artifactId>phantomjsdriver</artifactId>  <version>1.2.0</version>  </dependency> |
| JUnit | <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency> |

The steps to install PhantomJS below are to be executed on the server where Jenkins is hosted.

|  |  |
| --- | --- |
| Install PhantomJS | As Referenced from <https://gist.github.com/julionc/7476620>.  Version: 2.1.1  Platform: x86\_64  First, install or update to the latest system software.  sudo apt-get update  sudo apt-get install build-essential chrpath libssl-dev libxft-dev  Install these packages needed by PhantomJS to work correctly.  sudo apt-get install libfreetype6 libfreetype6-dev  sudo apt-get install libfontconfig1 libfontconfig1-dev  Get it from the [PhantomJS website](http://phantomjs.org/).  cd ~  export PHANTOM\_JS="phantomjs-2.1.1-linux-x86\_64"  wget https://bitbucket.org/ariya/phantomjs/downloads/$PHANTOM\_JS.tar.bz2 sudo tar xvjf $PHANTOM\_JS.tar.bz2  Once downloaded, move Phantomjs folder to /usr/local/share/ and create a symlink:  sudo mv $PHANTOM\_JS /usr/local/share  sudo ln -sf /usr/local/share/$PHANTOM\_JS/bin/phantomjs /usr/local/bin  Now, It should have PhantomJS properly on your system.  phantomjs –version |

Selenium WebDriver (Selenium 2.0) makes calls to a web browser’s native support for automation using a browser-specific browser driver. Selenium documentation lists the available [Selenium Drivers](http://www.seleniumhq.org/docs/03_webdriver.jsp#selenium-webdriver-s-drivers). Firefox Driver, ChromeDriver, and HtmlUnit Driver are an example of a few Selenium Drivers. The example project uses the PhantomJS Driver. The PhantomJS Driver utilizes Ghost Driver, a Remote WebDriver that uses PhantomJS as back-end (github.com/detro/ghostdriver). The HtmlUnit Driver has been installed, but is not currently be used. HtmlUnit is a java-based implementation of a WebBrowser without a GUI. I found the HtmlUnit Driver did not consistently represent the target WebBrowser, especially in regards to emulating JavaScript events, and I switched to the PhantomJS Driver for a similar headless environment.

Below are highlights of some functionality as demonstrated in the [MavenJunitPrototypePractice](https://github.com/gdombchik/MavenJunitPrototypePractice) GitHub repository. The JUnit files can be located in the com.selenium.mavenJunitPrototypePractice package.

|  |  |  |
| --- | --- | --- |
| Functionality | Description | JUnit File |
| Writing a Basic JUnit Script Using Selenium WebDriver | Navigate to the landing page of the web application and verify the title of the page. | Four\_WritingABasicJUnitScriptUsingSelenium.java |
| Selenium Locators | Selenium Locators is a mechanism used to locate elements within a document. The locators include by id, by class name, by tag name, by name, by link text, by partial link text, by css, and by XPath. | Five\_WebDriverLocators.java |
| JUnit Assertions | JUnit Assert class provides a set of assertion methods which are useful for writing tests. Example Assert methods are Assert.assertTrue(), Assert.assertFalse(), Assert.assertEquals(), or Assert.assertNotEquals(). | Six\_WebDriverAssertions.java |
| Page Object Design Pattern | As specified in the Selenium documentation:  “Page Object is a Design Pattern which has become popular in test automation for enhancing test maintenance and reducing code duplication.  The Page Object Design Pattern provides the following advantages  1. There is a clean separation between test code and page specific code such as locators (or their use if you’re using a UI Map) and layout.  2. There is a single repository for the services or operations offered by the page rather than having these services scattered throughout the tests.” | Test File: Seven\_PageObject.java  Page Objects located in the com.selenium.pageObjectZoo package:  - AboutPage.java  - AbstractPage.java  - AdoptionPage.java  - ContactConfirmationPage.java  - ContactPage.java  - LandingPage.java |
| JUnit Test Suite | A collection of Selenium JUnit tests. | TestNine\_AllSuite.java |
| XPath Selenium Locator | XPath Selenium Locator is the means identifying and interacting with elements on a page. | Ten\_WebDriverXPath.java |
| Css Selenium Locator | Css Selector Selenium Locator is a locator strategy by css. | Eleven\_CssSelector.java |
| Alerts and Windows | How Selenium handles alerts and external windows. | Twelve\_PopUpsAndWindows.java |
| Collections | Write Java Collections to manage Selenium WebElements. | Thirteen\_Collections.java |
| Parameter Management | Manage parameter management through a properties file. | Fourteen\_ParameterManagement.java |
| Page Factory Pattern | Page Factory Pattern is a factory class to make using Page Objects simpler and easier. | Fifteen\_PageFactory.java |
| Select Class | The Select Class provides helper methods to select and deselect options. | Sixteen\_SelectClass.java |
|  |  | Seventeen\_TakingScreenshots.java  WebDriverUtils.java |
|  |  | Eighteen\_HeadlessBrowser.java |
|  |  | TwentyTwo\_Waiting.java |
| Selenium WebDriver | Specify the Selenium WebDriver implementation. The example uses PhantomJSDriver and returns the WebDriver interface. Specify the landing page of the web application. Quit the WebDriver after JUnit test. | AbstractPageStepDefinition.java  AbstractWebDriver.java  Located in the  com.selenium.utils package:  WebDriverList.java |
| Properties File | The main functionality is to specify the PhantomJS binary package located on my machine versus the location on the AWS Ubuntu server. | PropertyManager.java  parameters.properties |

In reviewing how to run the Java tests using a continuous integration (ci) tool such as Jenkins, an option is to use Maven to run the Selenium tests using a headless environment using tools such as PhantomJS.

The steps to install Jenkins, Java, Maven, and Git on AWS E2 Ubuntu server: <https://gist.github.com/jsuwo/9038610>.

The Jenkins project setup for a Java project in GitHub. This example Jenkins has been installed on an AWS E2 Ubuntu server.

|  |  |
| --- | --- |
| Description | Command |
| *AWS E2 Ubuntu server* | |
| Switch to the Jenkins user. | $ sudo su – jenkins |
| Create public private key pair. | $ cd .ssh  $ ssh-keygen -t dsa |
| Cat the public key and copy the results. | $ cat MavenJunitPrototypePractice.pub |
| *GitHub* | |
| Add and paste deploy key in GitHub project. | <https://github.com/gdombchik/MavenJunitPrototypePractice/settings/keys> |
| *Manage Jenkins – Configure System* | |
| Jenkins Location | Scroll to the “Jenkins Location” section.  Jenkins URL:  <http://ec2-52-25-64-69.us-west-2.compute.amazonaws.com/>  System Admin e-mail address:  [greg@gregorydombchik.com](mailto:greg@gregorydombchik.com) |
| E-mail Notification | Scroll to the “E-mail Notification” section.  SMTP server:  mail.gregorydombchik.com |
| *Jenkins Server – Global Tool Configuration* | |
| Maven Installation | Scroll to the “Maven” section.  Select “Maven installation…” button.  Name:  Apache Maven 3.0.5  MAVEN\_HOME:  /usr/share/maven/ |
| JDK Installation | Scroll to the “JDK” section.  Select “JDK installation…” button.  Name:  Oracle JDK 1.7  JAVA\_HOME:  /usr/lib/jvm/java-7-oracle/ |
| *Jenkins Server – New Project* | |
| Add A Jenkins Project for a Java project. | Select New Item.  Enter an item name.  Select Freestyle project.  Select Ok. |
| Specify the URL of the remote GitHub repository. | Scroll to the “Source Code Management” section.  Select the “Git” option.  Repository URL:  <https://github.com/gdombchik/MavenJunitPrototypePractice.git> |
| Update the Poll SCM. | Scroll to the “Build Triggers” section.  Select the “Poll SCM” option.  Enter the following in the “Schedule” text box:  H \*/3 \* \* \* |
| Update the Build. | Scroll to the “Build” section.  Select from the “Add build step” and select the “Invoke top-level Maven targets” option.  Maven Version:  (Default)  Goals:  clean install  POM:  pom.xml |
| Update E-mail Notification. | Scroll to the Post-build Actions.  Select from the “Add post-build action” and select the “E-mail Notification” option.  Enter email address in the “Recipients” text box:  [greg@gregorydombchik.com](mailto:greg@gregorydombchik.com) |
| *GitHub* | |
| Add Jenkins GitHub Plugin you can automatically trigger build jobs when  pushes are made to GitHub. | <https://github.com/gdombchik/MavenJunitPrototypePractice/settings/hooks>  Select from the “Add service” and select the “Jenkin’s (Git plugin)” option. (NOT GITHUB PLUGIN. This service does not appear to work).  Enter the following in the “Jenkins url” text box:  <http://ec2-52-25-64-69.us-west-2.compute.amazonaws.com/> |

