**LAS3007**

Software test automation and continuous integration

*Course Assignment*

Anna Maria Portelli Cassola

Contents

[Overview 3](#_Toc440299125)

[Implementation of automated tests 3](#_Toc440299126)

[Project structure 3](#_Toc440299127)

[Overall structure 3](#_Toc440299128)

[Website testing 5](#_Toc440299129)

[Mobile application testing 7](#_Toc440299130)

[Assumptions, difficulties encountered and possible improvements 8](#_Toc440299131)

[Website testing 8](#_Toc440299132)

[Mobile application testing 10](#_Toc440299133)

[Installation of Testing and Continuous Integration tools 11](#_Toc440299134)

[Installation of Selenium Grid 11](#_Toc440299135)

[Installation of mobile application testing tools 12](#_Toc440299136)

[Difficulties during installation 12](#_Toc440299137)

[Installation and configuration of Jenkins CI 13](#_Toc440299138)

[Jenkins system configuration 13](#_Toc440299139)

[Jenkins job configuration 13](#_Toc440299140)

[Installation or configuration issues 15](#_Toc440299141)

# Overview

This document provides a description of the work done to fulfil the requirements of the course assignment. These included the installation of Jenkins CI and Selenium Grid as well as the implementation of automated testing of a website and mobile application. Website testing had to be done using both Mozilla Firefox and Google Chrome, while mobile application testing had to be done using Appium and an Android emulator.

Note that installations and development were done on a ***Windows 8.1*** machine.

# Implementation of automated tests

## Project structure

### Overall structure

Step definitions

Feature files

Page object models

Selenium WebDriver

Appium Android Driver

Mozilla Firefox

Output (success/fail)

Google Chrome

Chrome Driver

Android Device VM (Oracle VM VirtualBox)

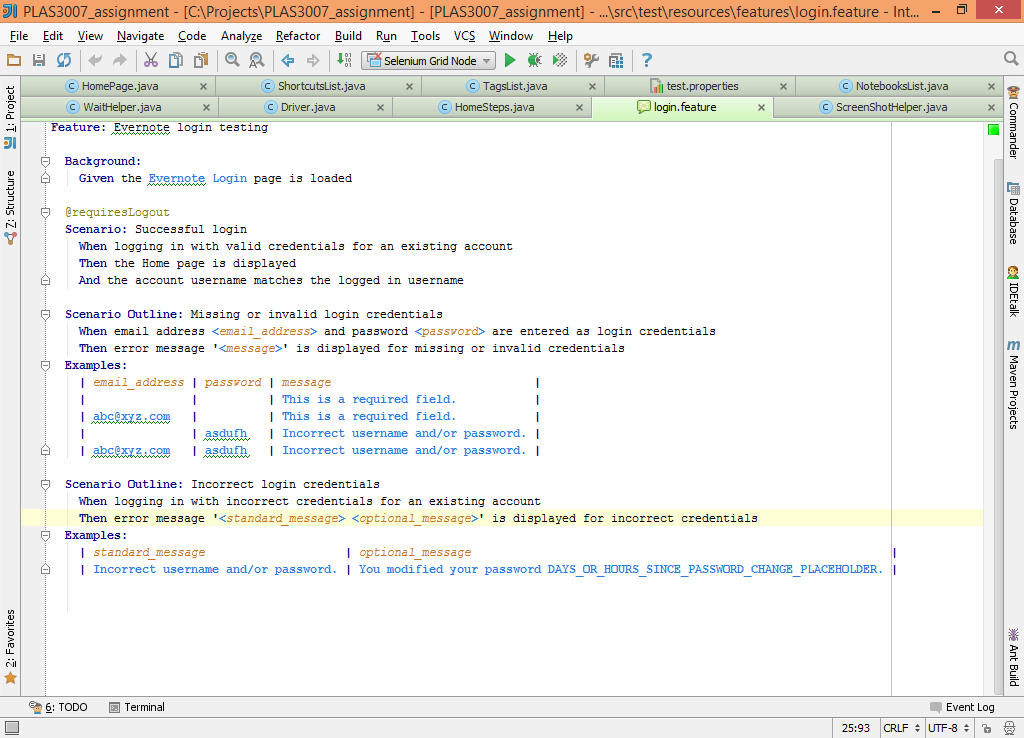
Android Emulator (Genymotion)

Appium Server

#### Feature files

Feature files, written in Gherkin, define a set of scenarios and cases which must be tested. Upon running the tests, the files are parsed using Cucumber and the respective action related to the test steps defined will be executed.

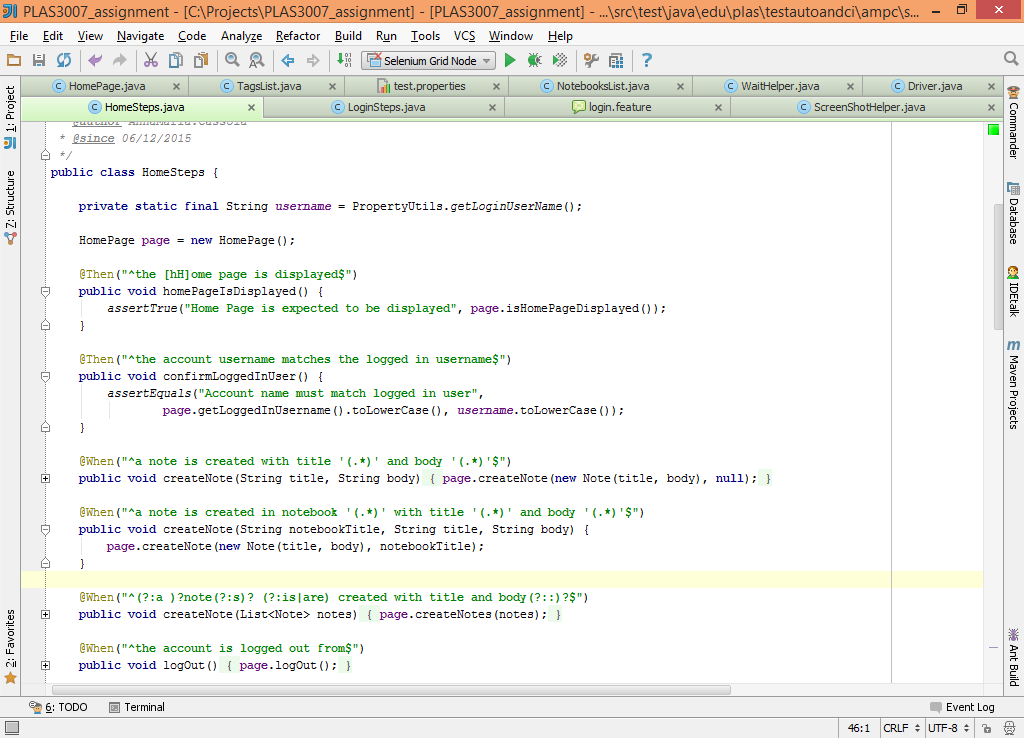
Example:



#### Step Definitions

Step definition classes include annotations which will attempt to match test steps in the feature files to a single Java method, using regular expressions, in order to automatically perform an action or assert the success or fail of a scenario or test case. The step definitions should typically only contain code directly related to the action or assertion of the test step and should not contain any particular logic to achieve a result. The step definitions should also not be concerned with interfacing with any underlying drivers or tools used to execute the tests.

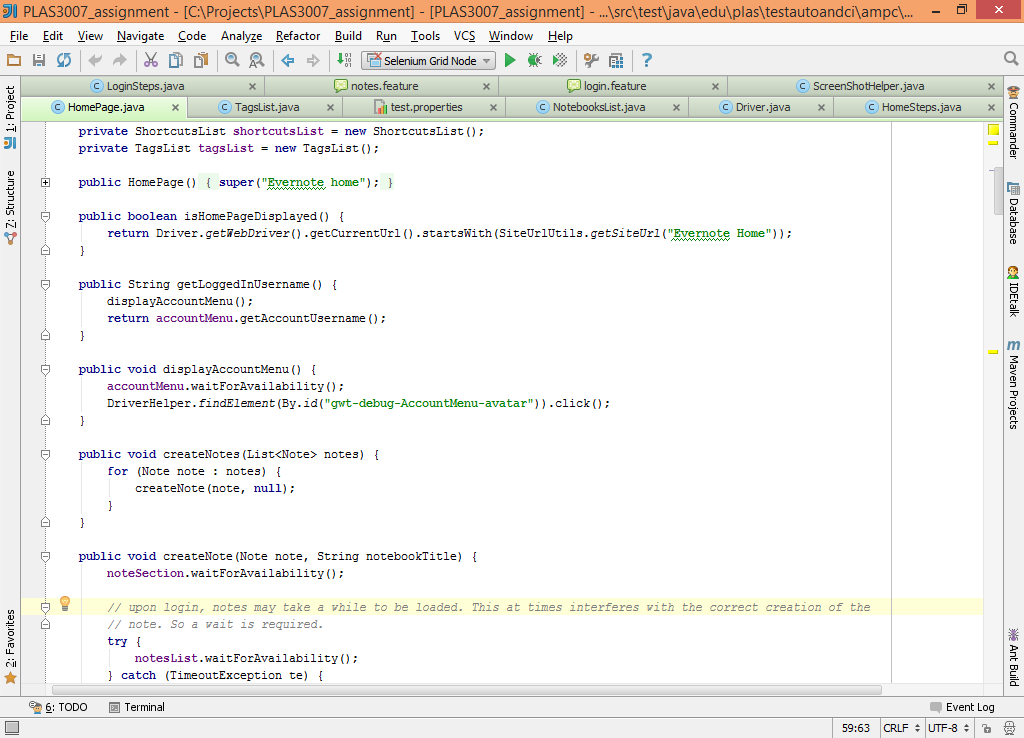
Examples:



#### ***Page Object Models***

Page object models abstract the intricacies involved when performing an action or determining the result of a test. These may range from driver connection setup and configuration to more granular steps that must be executed to perform a more generic action.

Examples:



#### ***Selenium***

Selenium provides the necessary interfacing between the Java application and the web browsers. It provides functionality that enables the application to traverse the HTML DOM in order to retrieve the current state of elements and to perform actions on those elements. In essence, it provides a way to mimic user actions on a website programmatically

#### Appium Android Driver

The Android Driver is similar in function to the Selenium WebDriver. It provides the necessary interfacing for interaction between the Java application and the Android mobile application, also using the Selenium API to achieve this. The Appium Android Driver must connect to the Appium Server to finally interact with the Android Emulator.

#### Browsers and Emulators

Mozilla Firefox and Google Chrome are the two browsers used for website testing within the scope of this assignment. Selenium WebDriver integrates with Firefox without any particular configuration. However an additional Chrome driver must be provided for Selenium to integrate with Google Chrome. The path to the Chrome driver executable must be provided as a parameter to the WebDriver upon initialisation of the driver if tests must be run using Google Chrome.

An Android Emulator and an Android virtual machine are needed to run the mobile application tests. Genymotion is used as the emulator, while Oracle VM VirtualBox is used to host an Android device virtual machine. Genymotion makes use of the Android SDK to communicate with the Android device. The device selected for the purpose of these tests was *Google Nexus 4 (Android version 4.2.2 – API 17)*.

#### Other

A set of utility and helper classes have also been made available to simplify code elsewhere in the project and especially to avoid repetition of similar code. Such classes include the SiteUtils and DriverHelper class.

A HouseKeeper class was introduced to take care of any pre-test preparation or post-test cleanup. @Before and @After hooks will be executed before or after any matching test case, depending on the tagging of scenarios in the feature files.

### Website testing

A set of test cases for the Evernote website were provided in text format. These were converted into feature files and the code required to mimic user actions was developed to finally determine the success or failure of the actions.

#### Feature files

Two feature files were created:

1. *login.feature*

Defines scenarios related to the successful and unsuccessful login for the Evernote website.

1. *notes.feature*

Defines scenarios related to the creation, modification, organisation, searching and deletion of notes.

#### Step definition classes

Step definition classes created for this project are:

1. *NavigationSteps*

Defines steps related to generic website navigation.

1. *LoginSteps*

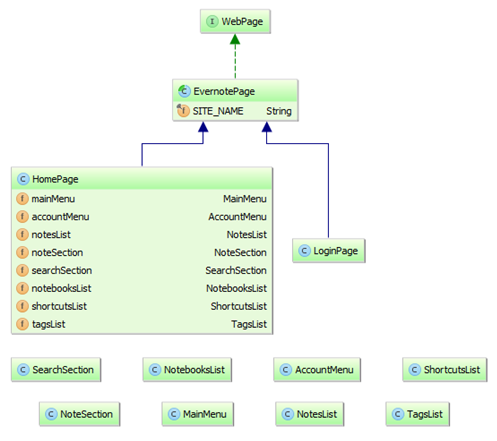
Contains step implementation for actions or assertions directly related to the Evernote Login page.

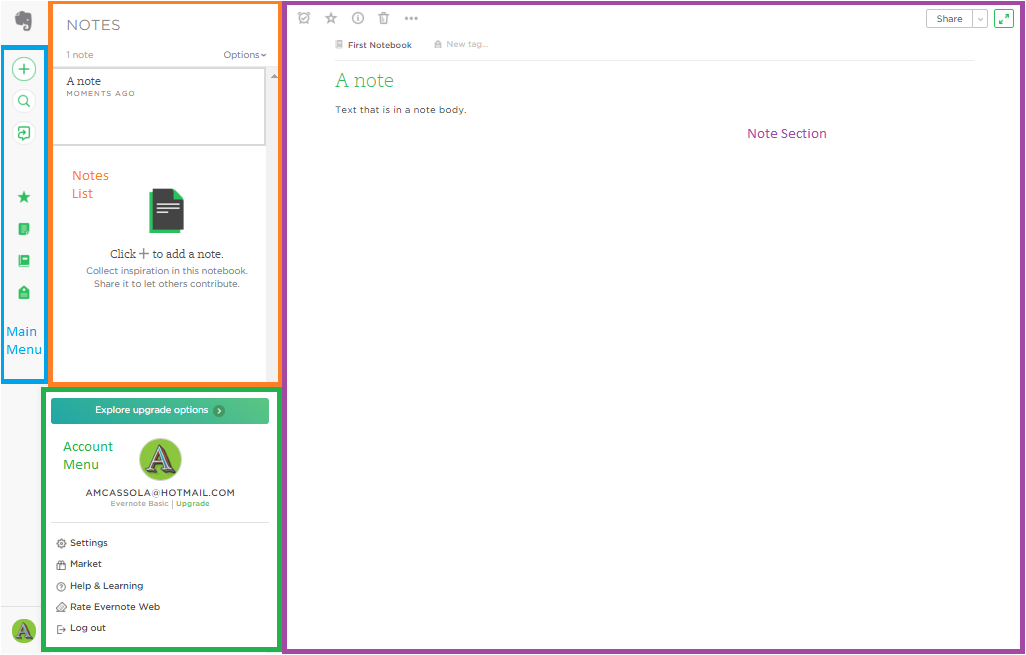
1. *HomeSteps*

Contains step implementation for actions or assertions directly related to the Evernote Home page, that is, to the creation, modification, organisation, searching and deletion of notes.

#### Page object model classes

The LoginPage and HomePage classes are the page object models that the step definitions classes interact with. The Evernote Home page consists of a number of sections which are each represented by a Section class. The HomePage class contains each of the section classes. In this way detail of each section is abstracted within its representation, leaving the HomePage class cleaner and easier to manage. The below class diagram represents the relationship between the page object model classes. The following image shows some of the sections of the Evernote Home page.





#### Other classes

Helper classes and utility classes have also been implemented to simplify code in the page object model and step definition classes and to reduce repetition. Some classes are described below:

1. *DriverHelper*

Wraps a few functions offered by the Selenium WebDriver such as navigating to sites and searching for elements in the HTML DOM.

1. *WaitHelper*

Helps with the management of implicit and explicit waits.

1. *SiteUtils*

Provides a function for the translation of a site name (eg. Evernote home) to the actual URL of the web page.

### Mobile application testing

Mobile application testing involved the download and installation of a number of tools, which are necessary to have a virtual setup of a mobile device on which to perform the tests. Once installed, the set of test cases for the Android Contacts App that were provided in text format were converted into feature files, and the code required to mimic user actions was developed to finally determine the success or failure of the actions.

#### Feature files

A single feature files were created (*contacts.feature*). The file defines scenarios related to the management of contacts using the Android Contacts app.

#### Step definition classes

The *ContactSteps* class defines steps related to contact management.

#### Page object model classes

The *ContactsApp* class abstracts the logic and granular interaction with the mobile device necessary to perform the steps defined in the step definition class.

## Assumptions, difficulties encountered and possible improvements

### Website testing

#### Assumptions

1. ***The Evernote user account exists and is active***

The user account configured as the existing account is expected to be a valid Evernote account.

1. ***Titles of notes created are unique***

It is assumed that titles of notes created are unique throughout the test run. If, for some reason, more than one note with the same title exists in the list of notes, existence checks will not fail if a previously existing note element is returned by the findElement method. Similarly, when searching the notes list for a note with a particular title, the first note found will be returned if more than one note exists with the same title.

1. ***Notes are deleted successfully after each test case***

Deletion of notes is done as part of one of the Cucumber @After hooks. If the deletion of the notes generates an error the test cases will continue to be executed. This should not pose an issue for most cases, since note titles used are unique and notes created for different tests will not interfere. However, this may cause the sorting tests to fail, since only the 3 notes created for that particular test case will be expected to be found.

#### Difficulties encountered

Difficulties encountered, amongst other are:

1. ***Implicit vs explicit waits***

The Evernote website is a highly dynamic website, making heavy use of JavaScript to change the view of the Home page depending on the user’s action. This often results in elements not being available in the HTML DOM immediately, or, if available, the elements cannot be interacted with immediately. Due to difficulty towards the beginning of the project to, at times, determine what should be explicitly waited for, while still requiring waits for specific events at other times, it was decided that the implicit wait will be set upon initialisation of the WebDriver, but will be disabled (set to 0) and re-enabled (reset to original value) whenever an explicit wait is needed. Although this may not follow the Selenium best practices, wait issues were resolved in the majority of cases.

1. ***Determining what to wait for before performing the next action***

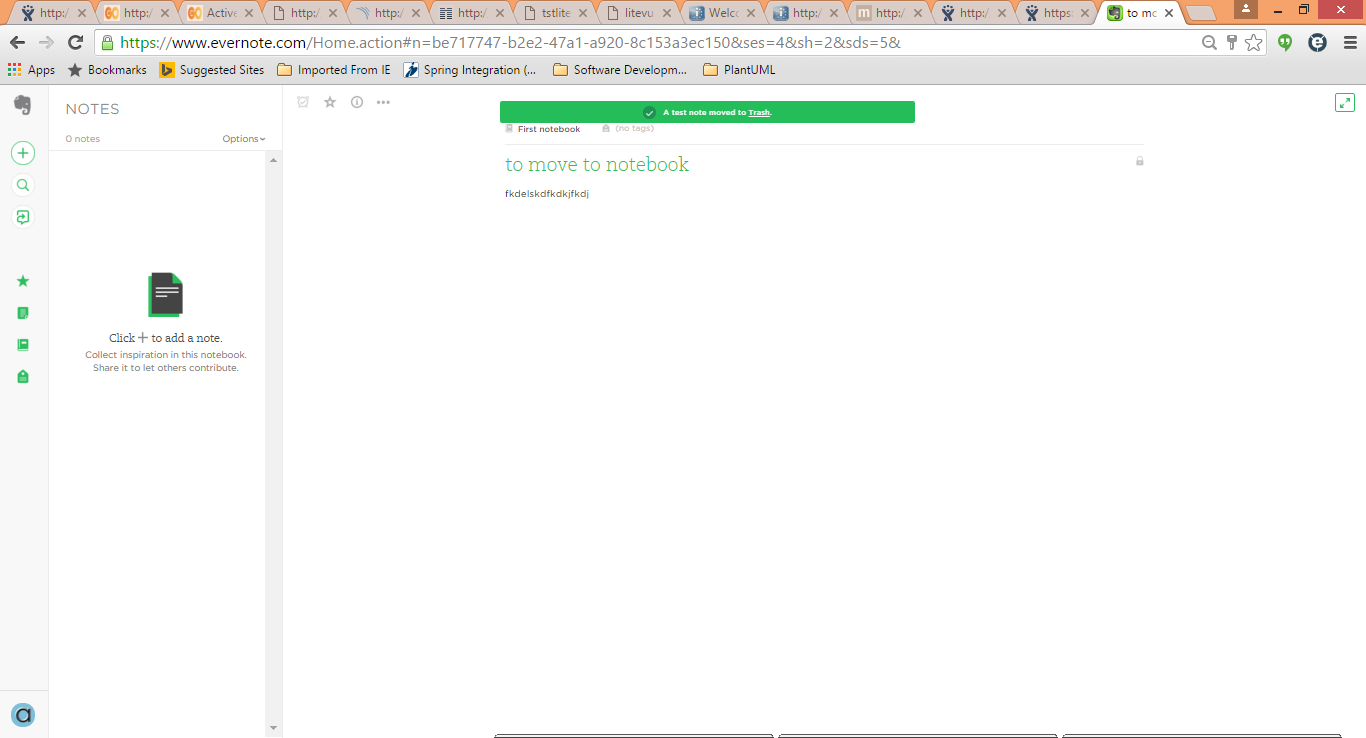
In relation to point 1, there were several instances during development when an action is performed but does not yield the expected result. This often happened because the action would be sent to the browser before the browser would have completed loading the page fully. One such example was when trying to tag a note – upon selection of the note, the note details are populated in the note section. The action to click the tag field (in order to add text to it) would be sent before the note has been loaded completely. Once the body of the note is populated, focus moves from the tag field to body. However, this happens before text is written to the tag field, leaving the tag field blank. This particular case was resolved by waiting for a non-empty body before moving focus to the tag field and populating it.

1. ***Switching notes views***

When switching from one list of notes to another (eg. from list of notes in trashcan to the full list of notes), the notes displayed may take a moment to be refreshed. Since a list of notes may be empty or may have 1 or more note, it was not possible to either wait for a list to be empty or a list not to be empty. However, a momentary wait was required to avoid failure of tests. A simple wait was implemented using Thread.sleep() in such cases. The wait time is typically set to 1 second but this is configurable through properties.

1. ***Deletion of notes***

The deletion of notes (which is done at after each test case) would often fail due to a variety of errors including StaleElementReferenceException and MoveTargetOutOfBoundsException. This would happen due to the dynamic nature of the list – elements previously retrieved would have been removed from the DOM in the meantime. A wait for the visibility followed by the invisibility of the ‘<noteTitle> moved to Trash’ message was introduced to allow some time for the list to be refreshed before proceeding with deleting the next note.



#### Possible improvements

1. ***Error handling***

The Selenium WebDriver throws a WebDriverException for several types of error. The same type of exception is thrown when an element is not present in the HTML DOM, when an element is not visible, when an element cannot be interacted with at the time, when an element is stale, etc. More specific custom exceptions could be implemented and thrown depending on the WebDriverException message. This would facilitate having different logic within the page object models depending on the outcome of the element search or interaction.

1. ***Use of explicit waits throughout***

Selenium best practices suggest that the implicit wait timeout is set upon initialisation of the WebDriver and is left as is throughout the test run. It is also suggested that implicit and explicit wait timeouts are best not used in conjunction. However, due to difficulties encountered towards the beginning of the implementation whenever an explicit wait will be used, the implicit wait timeout is disabled (set to 0) and re-enabled immediately after. Since the Evernote website is a very dynamic website, the implicit wait may be eliminated and replaced by appropriate explicit waits. This will not only abide by best practices but will also provide a more consistent way of waiting for changes in the HTML DOM to happen.

1. ***Parallelisation***

The current implementation allows very little parallelisation of test runs. It is possible to run the tests tagged @login and @notes in parallel, but the notes tests will still take around 6-7 times longer than the login tests. It is also not possible to run the same notes tests on different browsers at the same time. Solutions to this problem include:

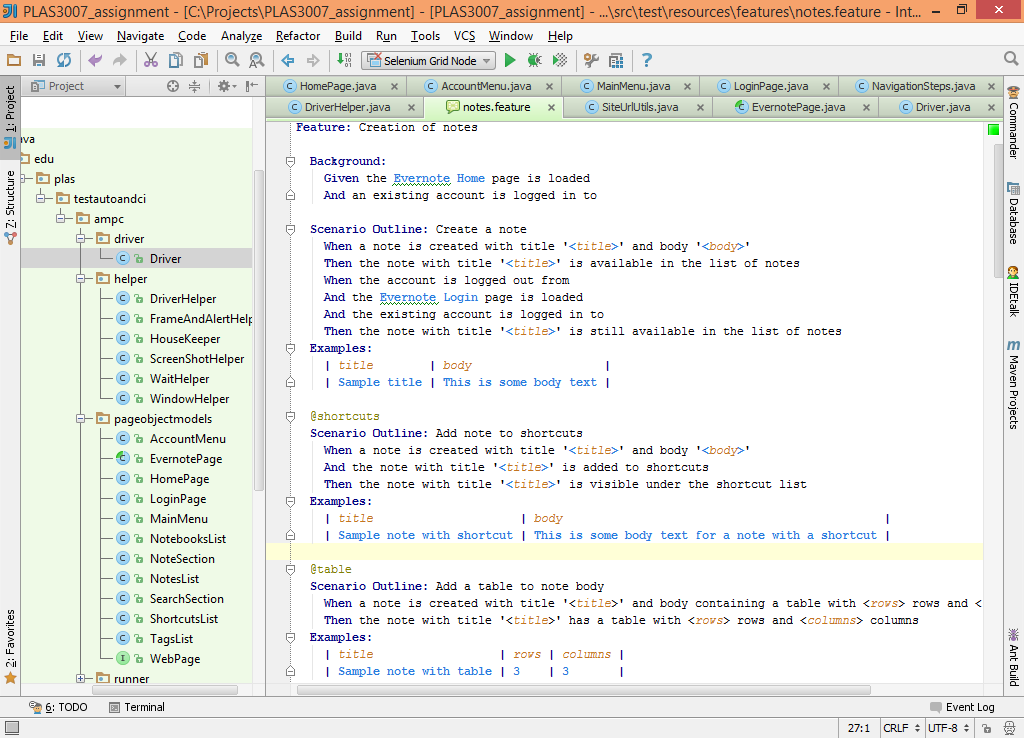
1. Setting up different user accounts and configuring builds to specify which account must be used. This will avoid clashes in creation and deletion of notes within the same account if tests are run in parallel. The downside to this is that there may be several accounts created, possibly 1 for each different build that will be run in parallel.
2. Keeping a list of the notes created during a test and modifying or deleting only the notes pertaining to the test. The list of notes would have to be maintained within the test run, and some test cases may have to be reworked (eg. sorting tests) to ignore notes created by other builds, resulting in more complex code.
3. ***Performance***

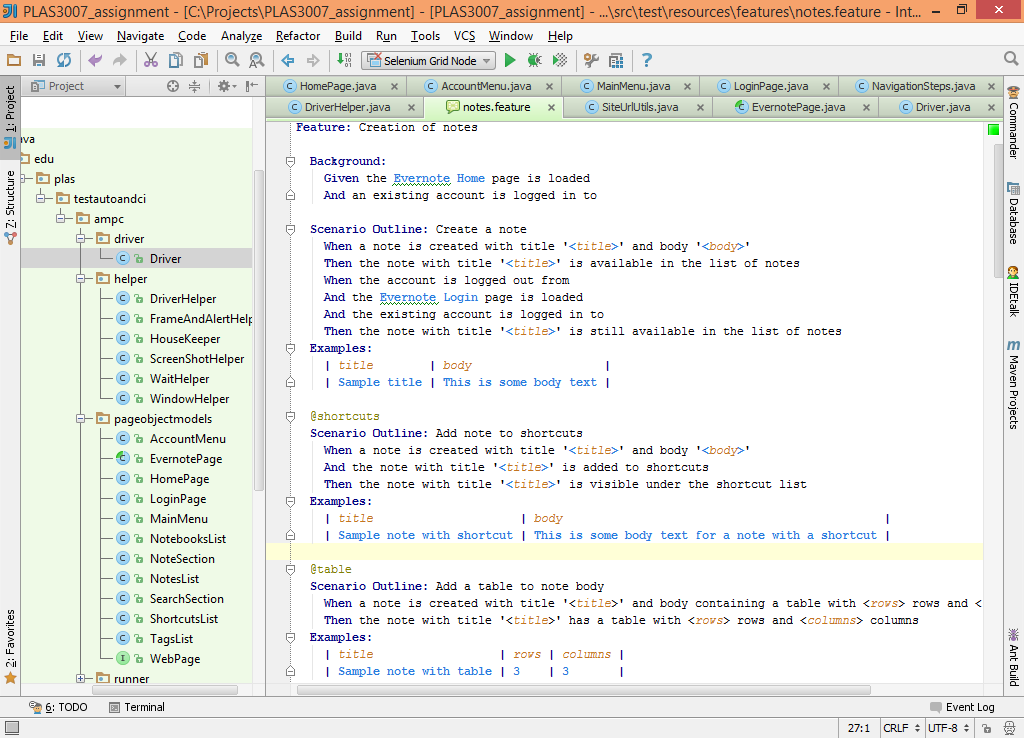
Test scenarios implemented very much reflect the scenarios described in the requirements document. That is, a scenario was added to the feature file for each test description provided. For example:

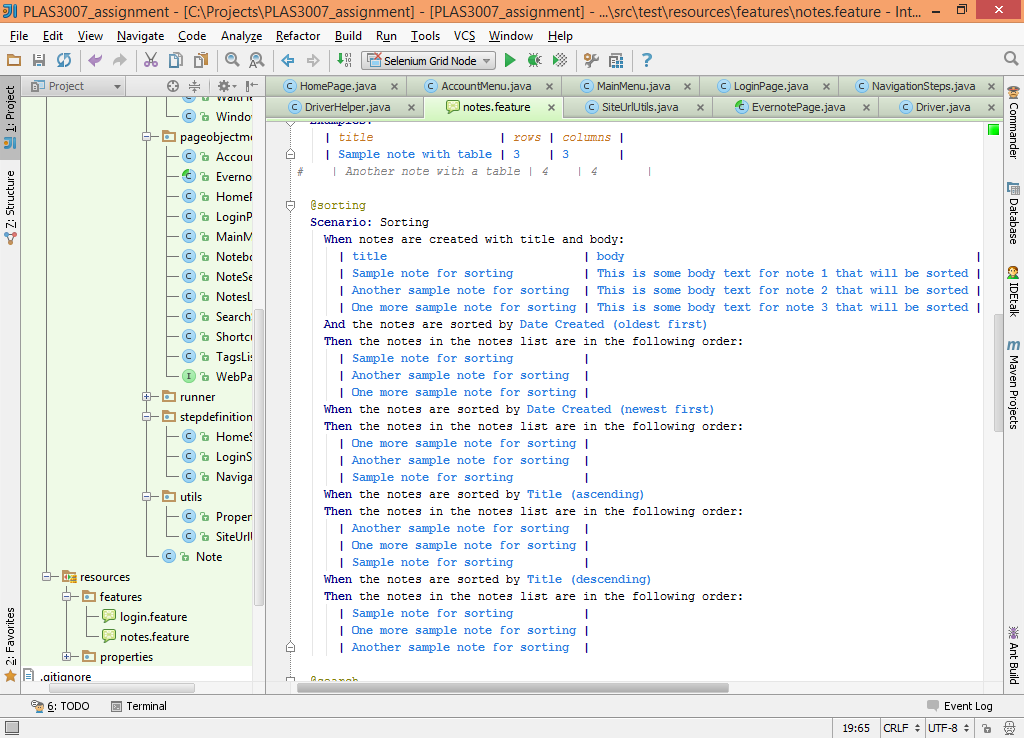
Required tests:

1. ‘Create a note ­ set note title and body. Make sure that the note just created is displayed in the Notes list.’
2. ‘Create a note, log out, log in again and make sure that the note is still saved in the Notes list.’
3. ‘Create a note, mark it as favourite (shortcut), then make sure that it is listed under Shortcuts.’
4. ‘Create 3 notes and ensure that sorting works well in the Notes list…’

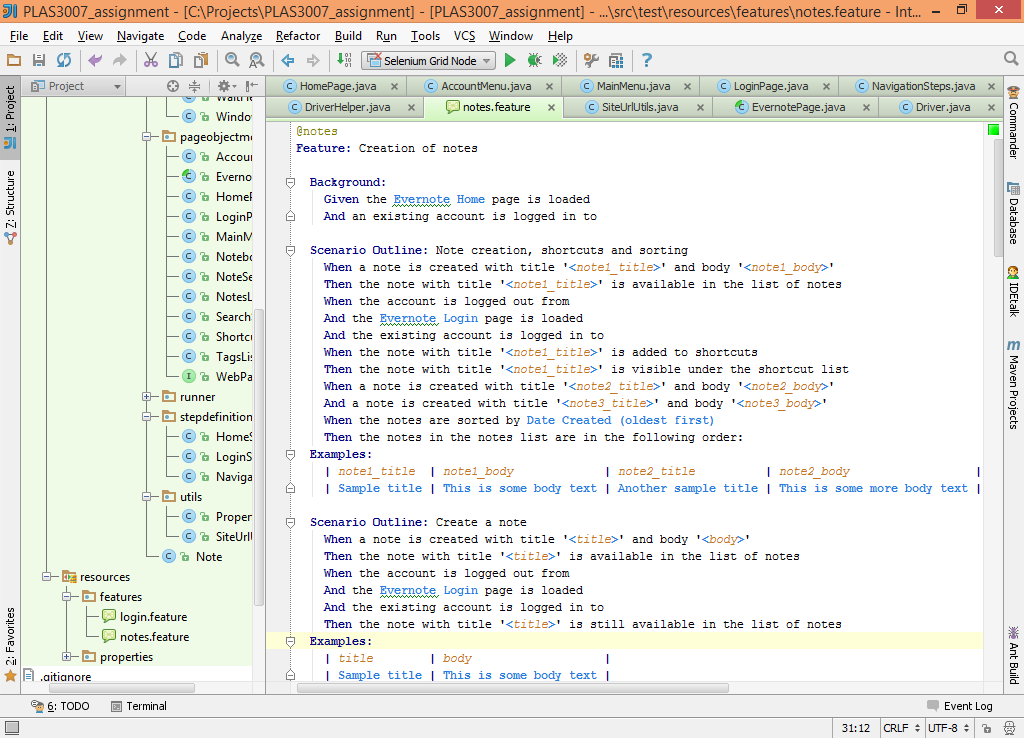
Feature scenarios:







With the current implementation, notes are created at the beginning of each scenario, as is required by the test case, and all created notes are deleted upon completion of each scenario. Using the above scenarios as examples, 5 notes are created in total, and the note deletion function is invoked 3 times. The 3 scenarios may be merged into a single scenario similar to the following:



With a scenario similar to this, only 3 notes will be created and notes will be deleted only once, reducing the overheads involved in doing so.

Although this would optimise performance, a balance between performance optimisation and the scope of a scenario must be found, so that tests that do not depend on previous ones may still be tested if a previous test fails.

1. ***Test Maintainability***

Element locator strings are currently hard coded within the page object model classes. Changes to the locators would require a code change. This may be improved by introducing a configurable way of determining the element locators, using a properties or XML file for example.

### Mobile application testing

#### Assumptions

1. ***Contacts created are unique***

It is assumed that contacts created are unique throughout the test run. If, for some reason, more than one contact with the same name exists under ‘All contacts’, existence checks will not fail if a previously existing contact element is returned. Similarly, when for a contact with a particular name, the first contact found will be returned if more than one contact exists with the same name.

1. ***Contacts are deleted successfully after each test case***

Deletion of contacts is done as part of one of the Cucumber @After hooks. If the deletion of the contacts generates an error the test cases will not fail.

#### Difficulties encountered

The main difficulty encountered during mobile test development was understanding how to locate elements present on the screen. The *uiautomatorviewer* tool that comes with the Android SDK facilitated this process and once interaction with the first few elements was done from the Java application, finding other elements to interact with became a trivial task.

#### Possible improvements

1. ***Error handling***

Errors are not always handled gracefully when an exception is thrown. For example, if an element is not found, a NoSuchElementException is thrown by the Appium AndroidDriver. Such error are not always handled, resulting in the exception bubbling up and being printed to the console, rather than having a more descriptive message being returned.

1. ***Moving to the expected screen view***

There are currently no checks in place to determine that the screen currently displayed is in fact the screen that must be interacted with next. That is, for example, if a new contact must be added, one would expect to find the ‘Add contact’ button on screen. However this is not available if viewing an existing contact. This could be improved by adding navigation to the expected screen, so that, if a new contact must be added, the contact list is first navigated to and then the ‘Add contact’ button may be clicked.

1. ***Test Maintainability***

Similar to the website testing implementation, element locator strings are currently hard coded within the page object model class. Changes to the locators would require a code change. This may be improved by introducing a configurable way of determining the element locators, using a properties or XML file for example.

# Installation of Testing and Continuous Integration tools

## Installation of mobile application testing tools

Additional tools needed to be installed in order to integrate the Java test application with the underlying Android device. These are:

1. Appium Server
2. Android SDK (Android version 4.2.2 – API17)
3. Genymotion and Oracle VM VirtualBox
4. An Android virtual device (Google Nexus 4, Android version 4.2.2 – API17)

The installation procedure was provided as part of the assignment document.

### ****Difficulties during installation****

1. *Hard disk space*

At the time when I needed to install the above tools, I was facing disk space issues. This made it impossible for me to have the setup done at one go, and I had to uninstall and reinstall some tools since the installation would halt once the disk got full. After having cleared sufficient space for the installations, the applications were installed with no particular issues.

1. *Environment variables*

Following the download of the Android SDK, the location of the tools and platform-tools directory had to be added to the PATH system variable. These were added and checked and rechecked, but the *adb* application found in the platform-tools directory continued not to be found when run from command line. Once the Android SDK directory paths were moved to the beginning of the PATH variable (rather than added at the end) the *adb* application was found, as expected.

1. *Virtual Device*

The first device I tried to use was the one provided in the assignment document. However, startup of the virtual device yielded no results – the device did not start up. I tried downloading a few other devices before settling for the Google Nexus 4. Issues encountered with other devices include devices not starting up as expected and the inability to take screenshots of the screen with the *uiautomatorviewer* tool (available with the Android SDK), thus not being able to find elements to interact with within the mobile screen.

## Installation of Selenium Grid

A Selenium Hub and Selenium Node was set up by following the instructions found on this site:

<https://code.google.com/p/selenium/wiki/Grid2>

After downloading the selenium server jar file, the hub was started by executing the following command:

java -jar selenium-server-standalone-2.48.2.jar -role hub

A node was started by running this command:

java -jar selenium-server-standalone-2.48.2.jar -role node -hub http://localhost:4444/grid/register -browser "browserName=firefox,maxInstances=5,platform=WINDOWS" -browser "browserName=chrome,maxInstances=5,platform=WINDOWS" -Dwebdriver.chrome.driver=C:\Projects\PLAS3007\_assignment\browser-driver\chrome\chromedriver.exe

Note that this setup is not used in conjunction with Jenkins.

Selenium WebDriver initialisation was modified to use the RemoveWebDriver class if tests must be run using the Selenium Grid. The Selenium Hub URL and the browser capabilities are passed as parameters to the RemoteWebDriver upon initialisation.

## Installation and configuration of Jenkins CI

Installation of the Jenkins automation server was easy and straightforward. The Jenkins ZIP file was downloaded from <https://jenkins-ci.org/> and the setup file was run. Upon completion of the installation wizard the Jenkins server started up automatically.

### Jenkins system configuration

Following the default installation, the Jenkins system configuration needed to be updated so that Jenkins will be capable of finding the tools required for its builds. This included the configuration of the JDK and Maven settings, namely the JAVA\_HOME and MAVEN\_HOME parameters.

#### Additional plugins

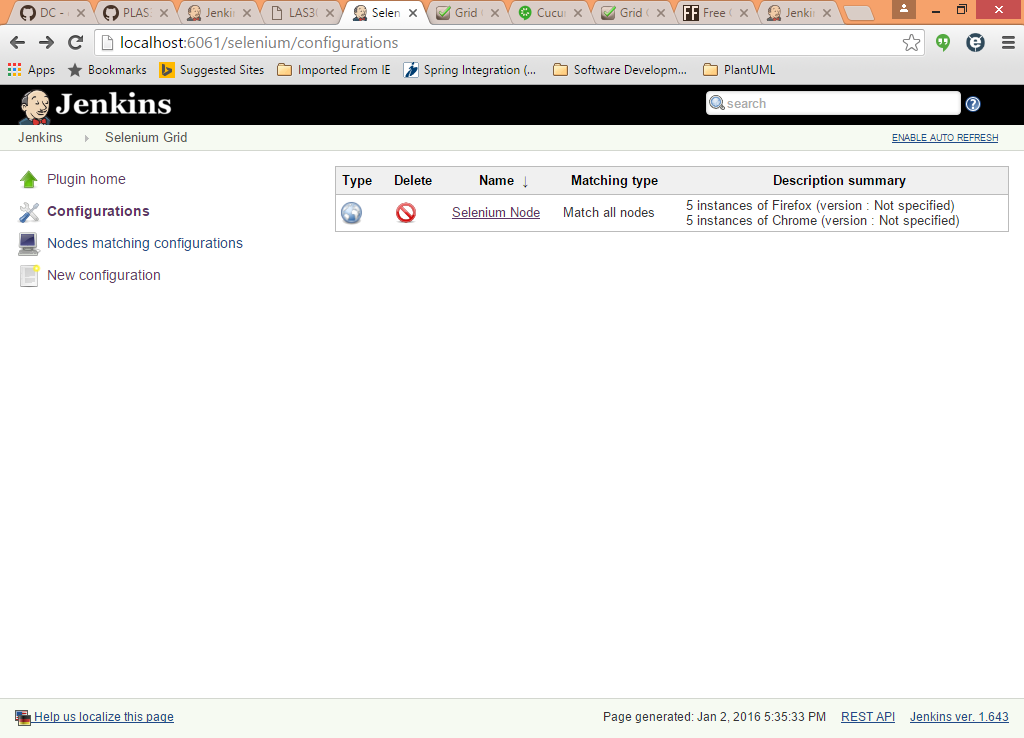
Additional Jenkins plugins installed are:

1. *Git Plugin*

Enables Jenkins to download the build code from a Git SCM rather than expecting the code to be hosted locally. The location of git.exe was added to the system configuration so that Jenkins could connect to the Git repository.

1. *Selenium Plugin*

Integrates the Selenium Grid setup within Jenkins. The Selenium Grid Hub and node/s are started up with Jenkins. Builds requiring the use of Selenium Grid will not need any additional servers to be running separately. Configuration of 1 Selenium node was added for which the browser types and number of browsers available on the node were set from the Selenium Grid plugin management screens.

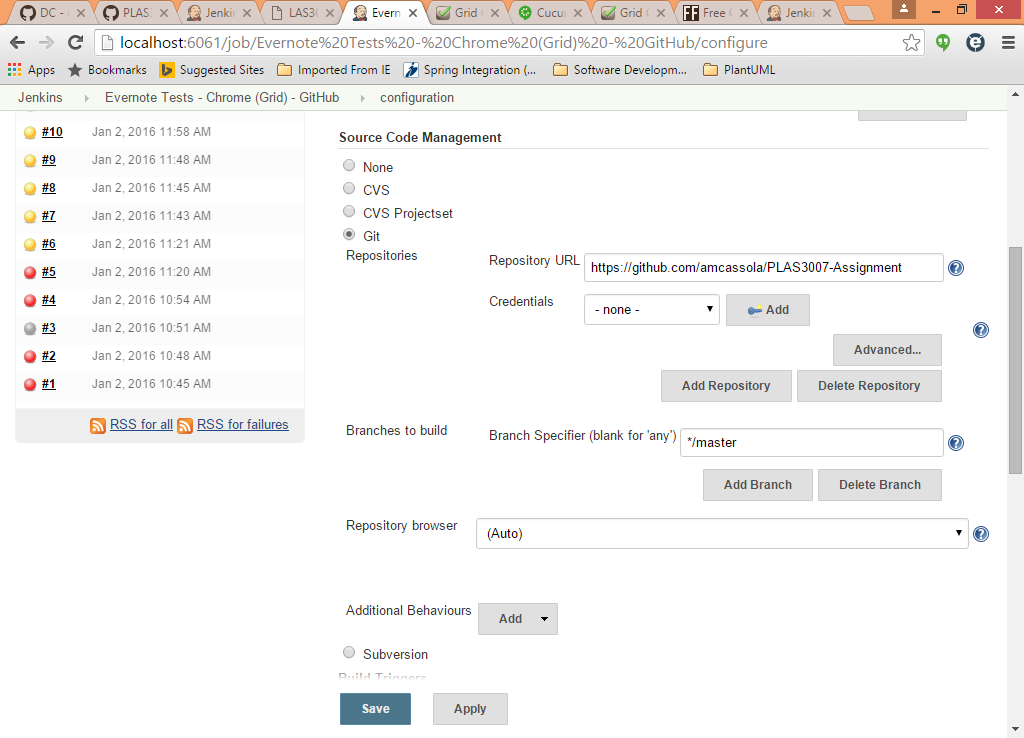


### Jenkins job configuration

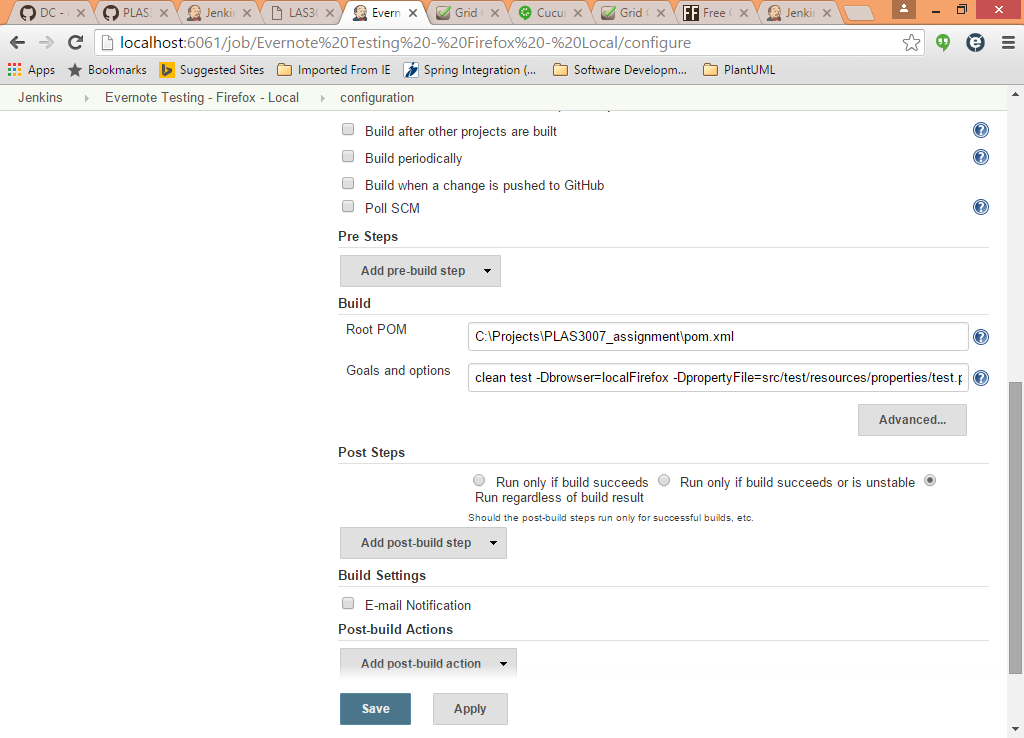
Jenkins job configuration specifies what jobs must be run and their schedules, if any. The following jobs were set up:

1. **Evernote Tests – Firefox – Local**: Configured as a Maven project that will build and execute the Maven test command using code in a local directory. Web application tests are run using a Firefox browser. Selenium Grid is not used.
2. **Evernote Tests – Firefox – GitHub**: Configured as a Maven project that will build and execute the Maven test command using code from a Git repository (GitHub). Web application tests are run using a Firefox browser. Selenium Grid is not used.
3. **Evernote Tests – Firefox (Grid) – GitHub**: Configured as a Maven project that will build and execute the Maven test command using code from a Git repository (GitHub). Web application tests are run using a Firefox browser available through Selenium Grid.
4. **Evernote Tests – Chrome – Local**: Configured as a Maven project that will build and execute the Maven test command using code in a local directory. Web application tests are run using a Chrome browser. Selenium Grid is not used.
5. **Evernote Tests – Chrome – GitHub**: Configured as a Maven project that will build and execute the Maven test command using code from a Git repository (GitHub). Web application tests are run using a Chrome browser. Selenium Grid is not used.
6. **Evernote Tests – Chrome (Grid) – GitHub**: Configured as a Maven project that will build and execute the Maven test command using code from a Git repository (GitHub). Web application tests are run using a Chrome browser available through Selenium Grid.
7. **Android Contacts App Testing – Local:** Configured as a Maven project that will build and execute the Maven test command using code in a local directory. The mobile application tests will be run.
8. **Android Contacts App Testing – GitHub:** Configured as a Maven project that will build and execute the Maven test command using code in a local directory. Tests are run using a Chrome browser. Selenium Grid is not used. The mobile application tests will be run.

For GitHub projects, the Source Code Management options were set as follows:



For local projects, ‘None’ was selected under Source Code Management, and the location of the Root POM was set to the absolute path of the pom.xml of the local code base.



The Maven mvn arguments for each build were placed under the Build Goals and options. Arguments for the website tests and the mobile application tests differ slightly.

#### Website test arguments

Arguments are:

clean test -DpropertyFile=src/test/resources/properties/test.properties -Dbrowser=localFirefox "-Dcucumber.options=--tags @login,@notes"

The browser argument varies per job as follows:

|  |  |
| --- | --- |
| *Job name* | *Browser value* |
| Evernote Tests – Firefox – Local | localFirefox |
| Evernote Tests – Firefox – GitHub | localFirefox |
| Evernote Tests – Firefox (Grid) – GitHub | gridFirefox |
| Evernote Tests – Chrome – Local | localChrome |
| Evernote Tests – Chrome – GitHub | localChrome |
| Evernote Tests – Chrome (Grid) – GitHub | gridChrome |

#### Mobile application test arguments

Arguments are:

clean test -DpropertyFile=src/test/resources/properties/test.properties "-Dcucumber.options=--tags @contacts"

No particular schedules were set up for either of these jobs. Jobs are run manually when required.

### Installation or configuration issues

Issues encountered when setting up or executing builds are:

1. *Missing settings*

Upon installing Jenkins and initially configuring jobs, builds would fail due to lack of JDK and Maven settings in the Jenkins system configuration. Download of the code from GitHub also failed initially due to missing configuration of the git.exe file. These were minor issues that were quickly resolved.

1. *Browser window not displayed while running tests*

When running builds from command line or from the IDE, the browser window would be displayed and it would be possible to see how the tests are progressing without relying only on the output logs. However, when running the tests through Jenkins the browser window was no longer loaded. It turned out that when Jenkins is started up using jenkins.exe, the browser window will not be displayed. Starting up Jenkins by running the WAR file would resolve the problem, which it did. Having said this, it was good to see the browser window while still setting up jobs, to ensure that they were running as expected. Once all was set up starting Jenkins using the EXE file rather than the WAR file and relying only on the test output logs was sufficient.

1. *Firefox Grid tests would fail when using the integrated Selenium Grid*

After installing the Jenkins Selenium Plugin and ensuring that the hub and node are up and running, jobs using the Selenium Grid Chrome browsers were executed without issues, but the Firefox tests were failing with the following error:

org.openqa.selenium.WebDriverException: Unable to connect to host 127.0.0.1 on port 7055 after 45000 ms.

The Jenkins Selenium Plugin does not use the latest version of the selenium-server-standalone JAR file. This was upgraded from version 2.41.0 to version 2.48.2 and the problem was resolved.