**LAS3007**

Software test automation and continuous integration

*Course Assignment*

Anna Maria Portelli Cassola

Contents

[Overview 3](#_Toc439586078)

[Implementation of automated tests 3](#_Toc439586079)

[Project structure 3](#_Toc439586080)

[Overall structure 3](#_Toc439586081)

[Website testing 5](#_Toc439586082)

[Mobile application testing 7](#_Toc439586083)

[Assumptions, difficulties encountered and possible improvements 7](#_Toc439586084)

[Website testing 7](#_Toc439586085)

[Mobile application testing 9](#_Toc439586086)

[Installation of Testing and Continuous Integration tools 9](#_Toc439586087)

[Installation of Selenium Grid 10](#_Toc439586088)

[Installation and configuration of Jenkins CI 10](#_Toc439586089)

[Jenkins system configuration 10](#_Toc439586090)

[Jenkins job configuration 11](#_Toc439586091)

[Installation or configuration issues 13](#_Toc439586092)

# 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

Android Driver

Mozilla Firefox

Android Emulator

Output (success/fail)

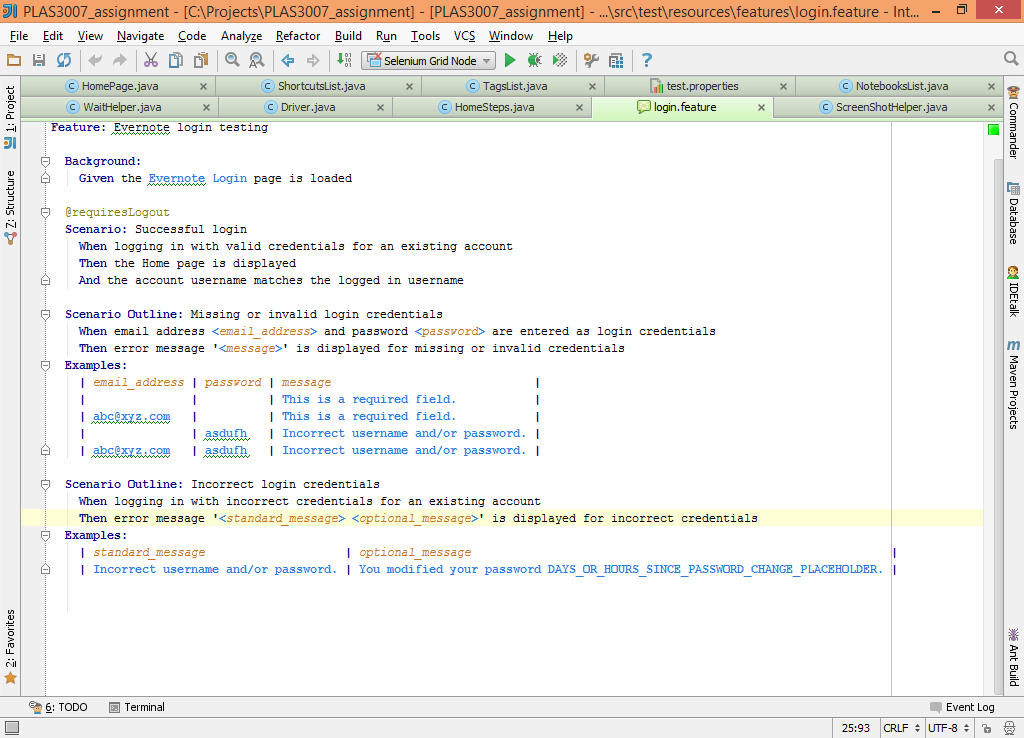
Google Chrome

Chrome Driver

#### 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.

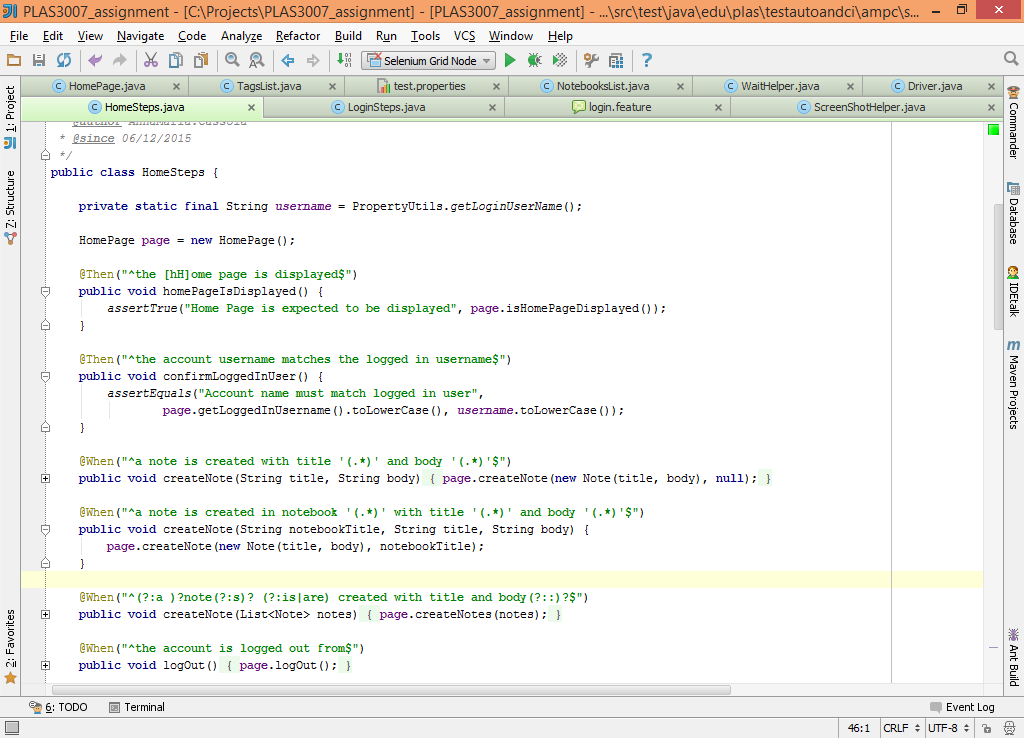
Example:



#### 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.

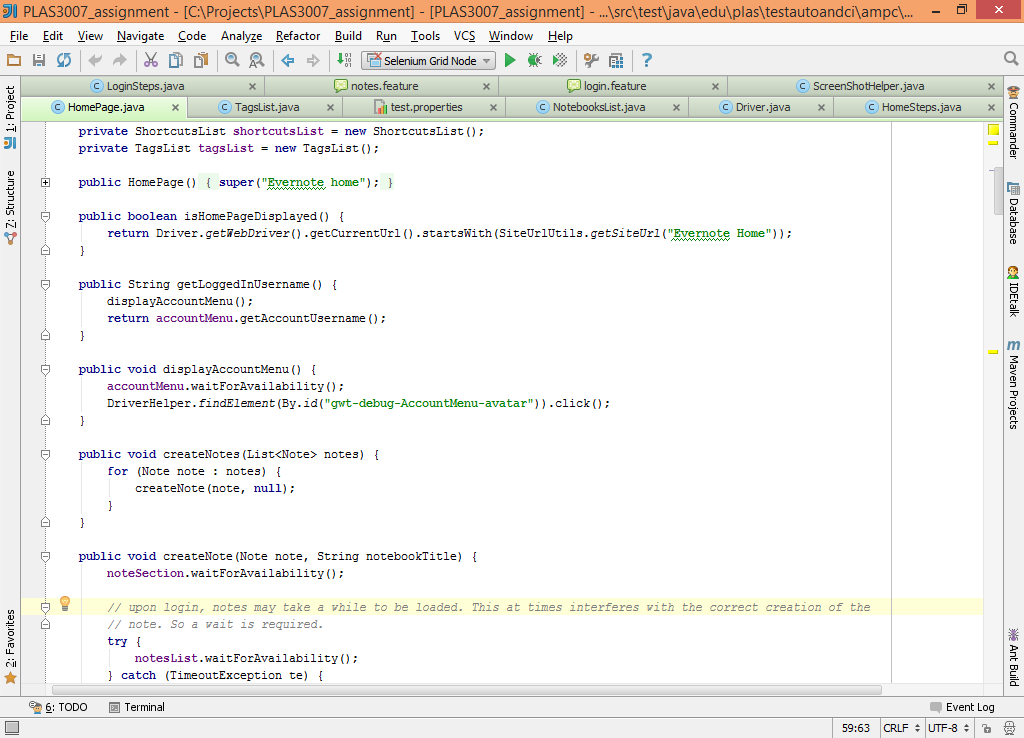
Examples:



#### ***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 the individual 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

#### 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*

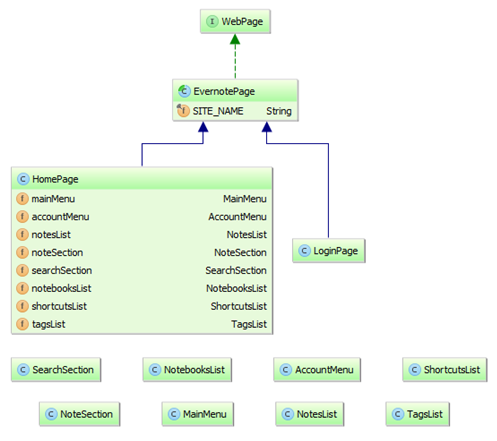
Contains step implementation for actions or assertions directly related to the Evernote Login page.

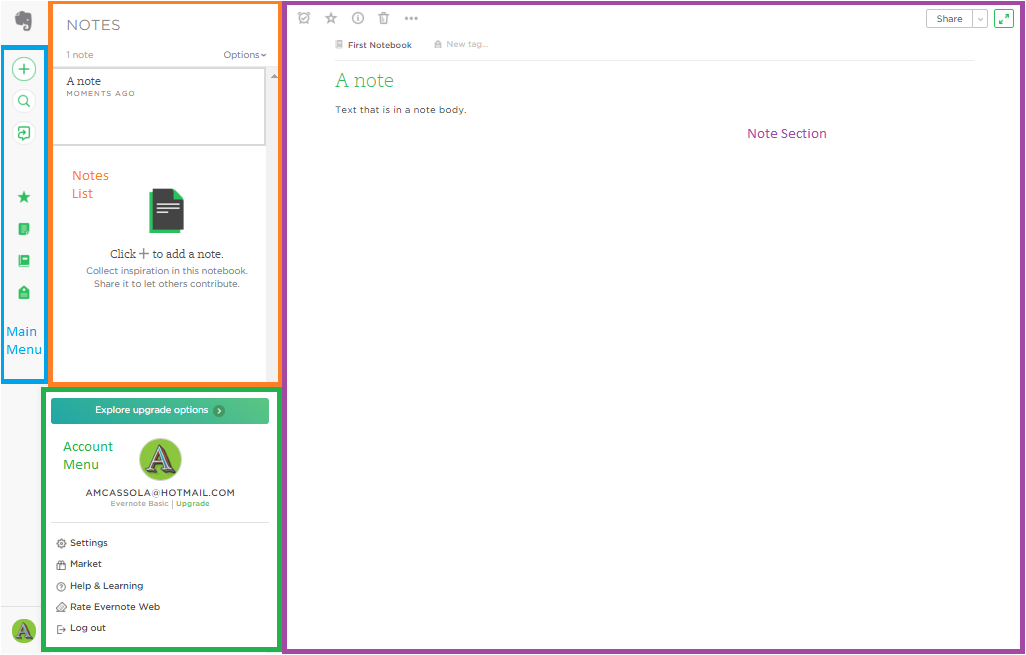
1. *HomeSteps*

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. *Driver and DriverHelper*

Wrap the Selenium web driver initialisation and shutdown and also 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. *PropertyUtils*

Offers a set of functions to retrieve values from property files

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

## 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.

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.

#### Issues and solutions

#### 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. ***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:

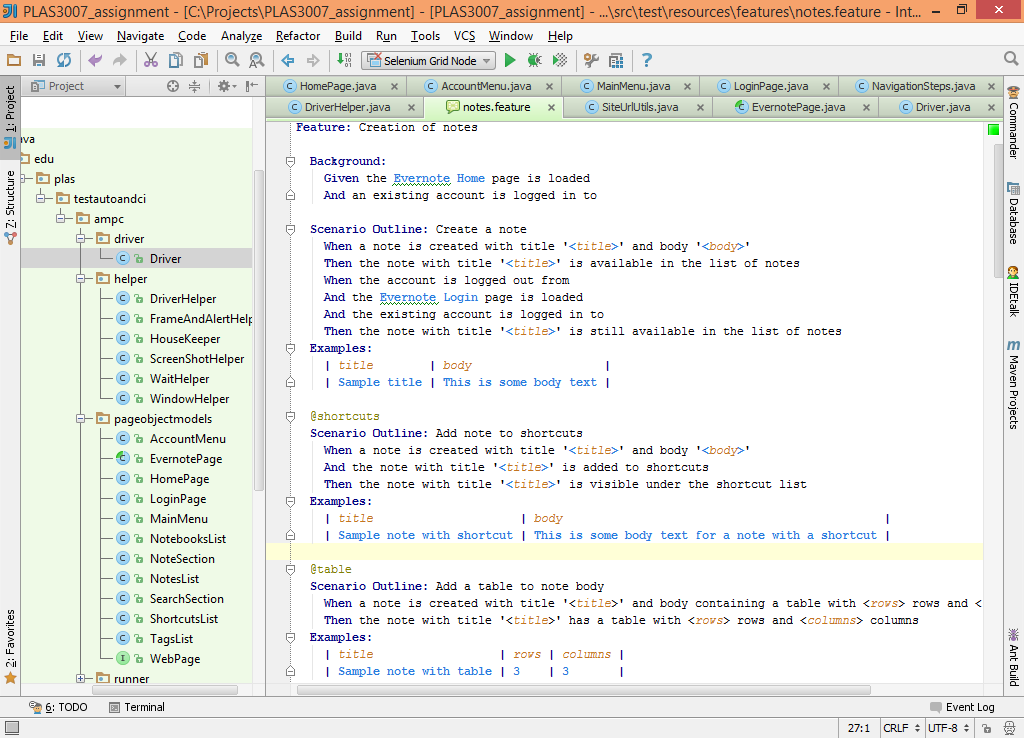
1. Setting up different user accounts and configuring builds to specify which account must be used. This will avoid clashes in creation and deletion of notes within the same account if tests are run in parallel. The downside to this is that there may be several accounts created, possibly 1 for each different 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***

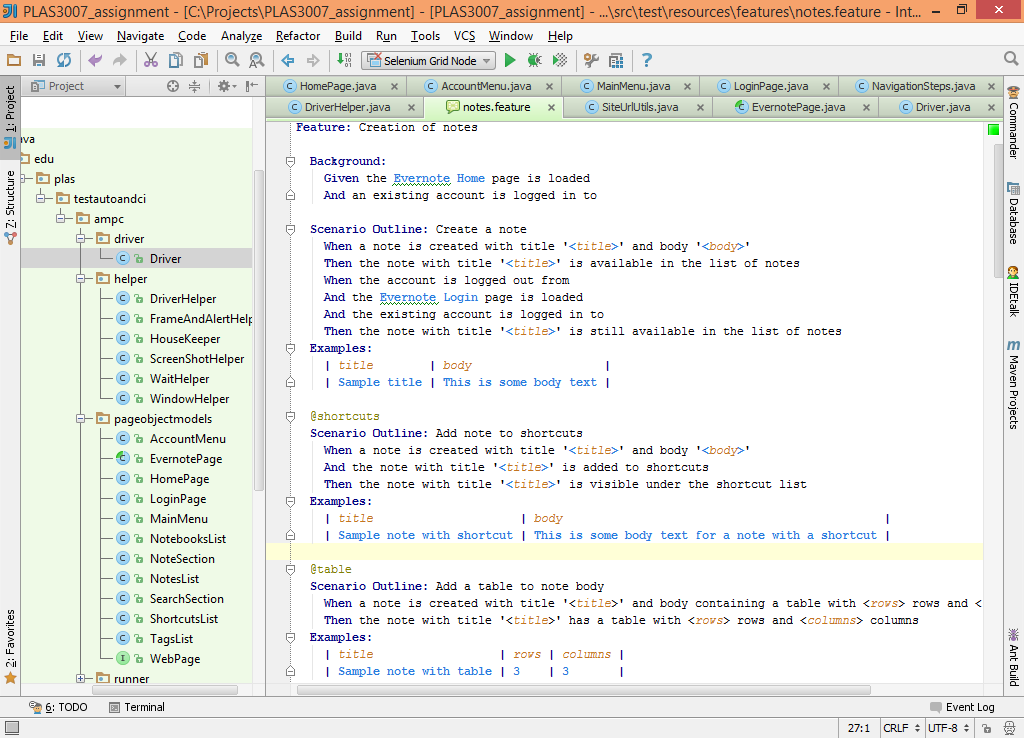
Test scenarios implemented very much reflect the scenarios described in the requirements document. That is, a scenario was added to the feature file for each test description provided. For example:

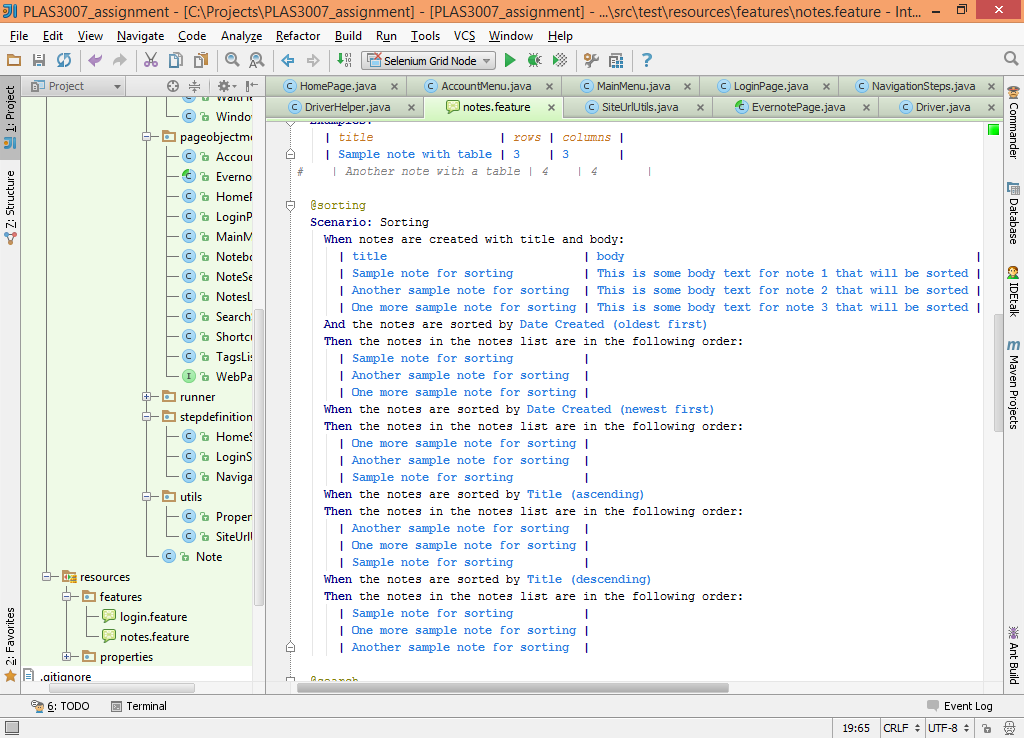
Required tests:

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…’

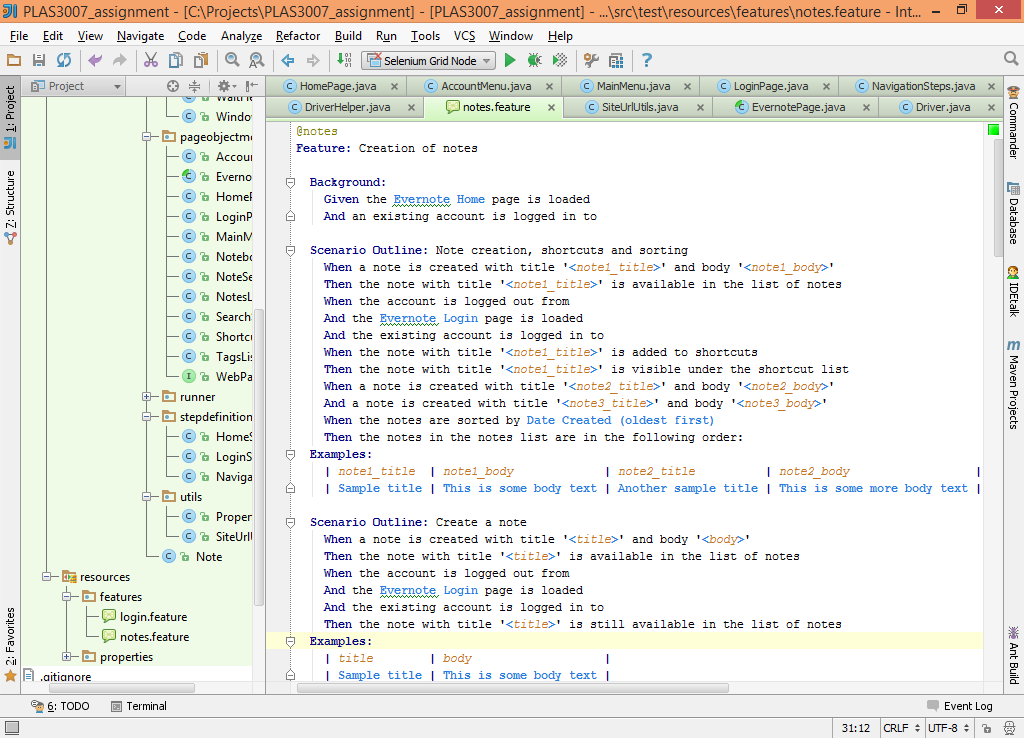
Feature scenarios:







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.

### Mobile application testing

# Installation of Testing and Continuous Integration tools

## 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

This setup is not used in conjunction with Jenkins.

## 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

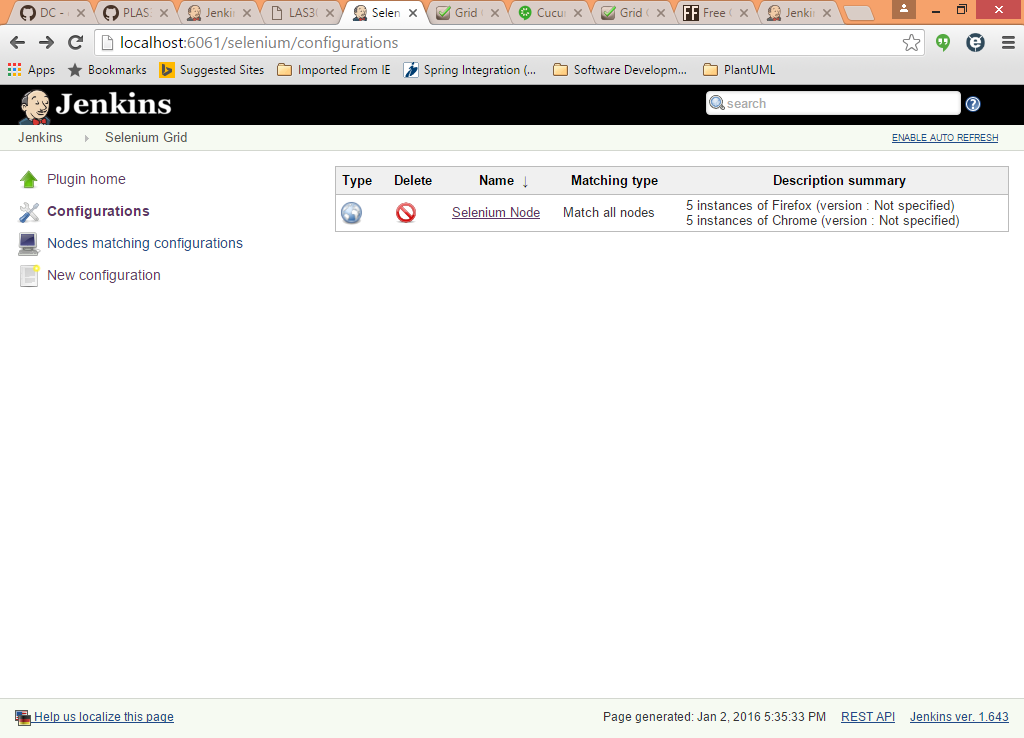
Additional Jenkins plugins installed are:

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.

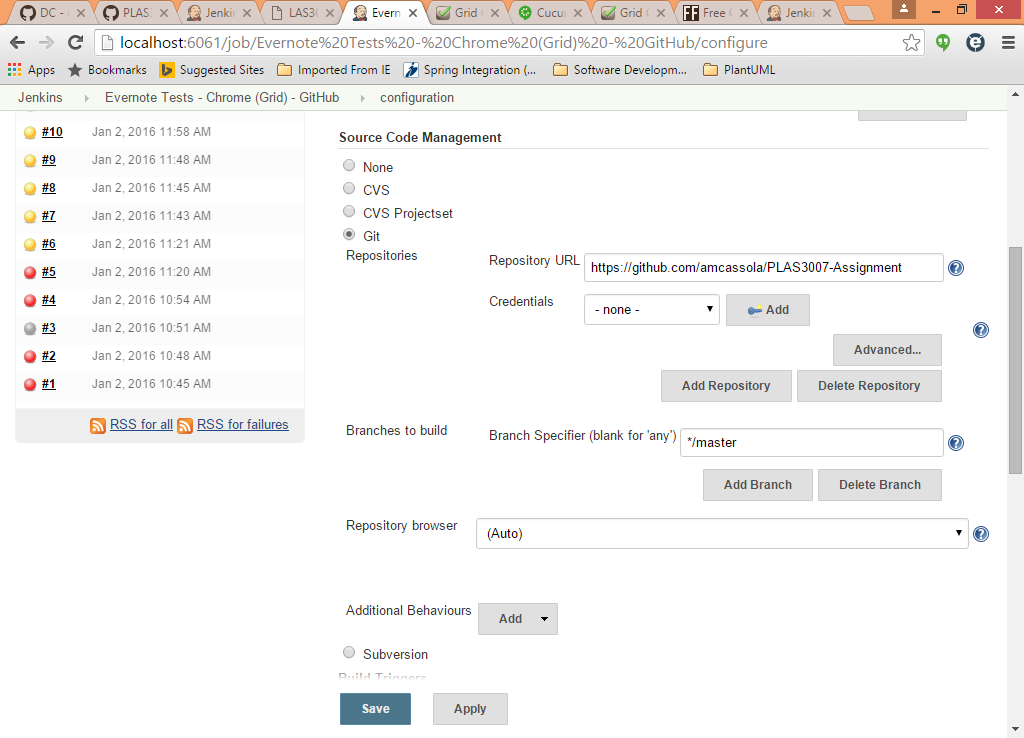


### Jenkins job configuration

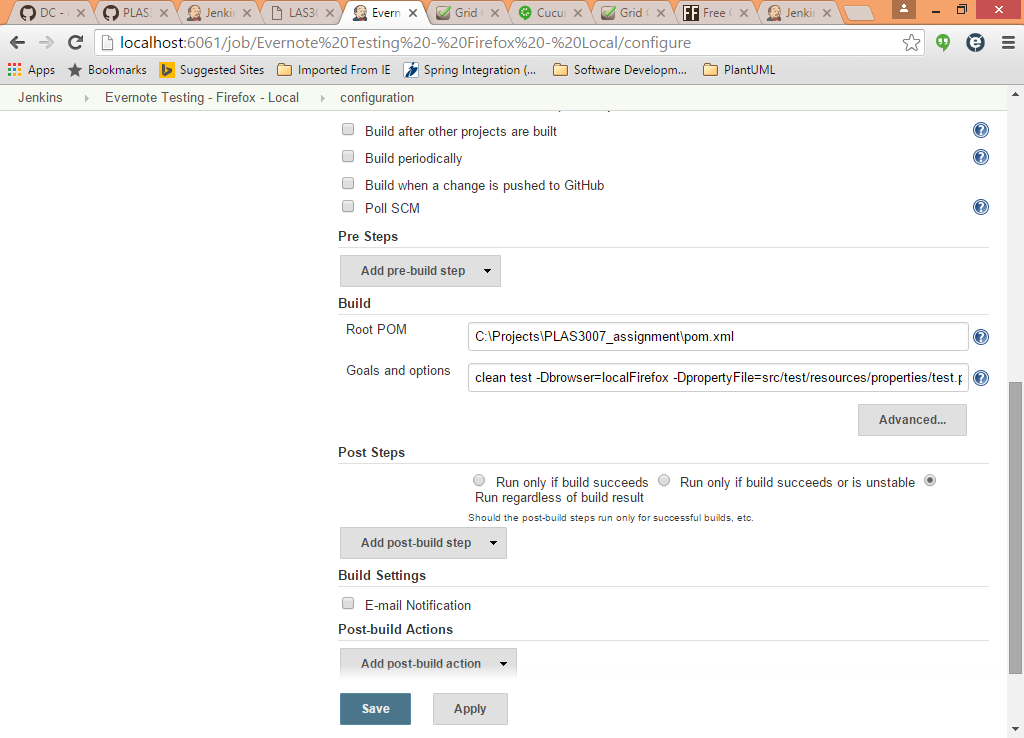
Jenkins job configuration specifies what jobs must be run and their schedules, if any. The following jobs were set up:

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. 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). 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). 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. 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). 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). Tests are run using a Chrome browser available through Selenium Grid.

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. The 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 |

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.