SOEN 345 - Software Testing, Verification, and Quality Assurance
Assignment 4

Ali Hanni - 40157164

Gina Cody School of Computer Science and Software Engineering
Concordia University, Montreal, QC, Canada

Winter 2022

Question 1 - Part A

In the `DemoTutorialTest` class, three methods are implemented, `startBrowser()` and `tearDown()` that respectively initiate and shutdown the driver in the browser, and `demoTest()` that perfoms a test on a web browser window using the driver namely *chromedriver*. In `demoTest()`, there is first a waiting time, then the browser window is maximized, then the driver goes to google home page (*https://www.google.com*), and finally, using the driver, we get the title of the page and assert that it must be equal to *"Google"*.
It is worth noting that all the interactions with the browser during the test are done via *chromedriver*, that has a multitude of tools to test user interface in a web browser environment.

```
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i
.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /Users/alihanni/GitHub/SOEN/SOEN345/A
4/UITest-tutorial-part-A/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ SOENT
utorials ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. bu
ild is platform dependent!
[INFO] Compiling 3 source files to /Users/alihanni/GitHub/SOEN/SOEN345/A4/UITest
-tutorial-part-A/target/test-classes
[INFO] /Users/alihanni/GitHub/SOEN/SOEN345/A4/UITest-tutorial-part-A/src/test/ja
va/tests/DemoTutorialTest.java: /Users/alihanni/GitHub/SOEN/SOEN345/A4/UITest-tu
torial-part-A/src/test/java/tests/DemoTutorialTest.java uses or overrides a depr
ecated API.
[INFO] /Users/alihanni/GitHub/SOEN/SOEN345/A4/UITest-tutorial-part-A/src/test/ja
va/tests/DemoTutorialTest.java: Recompile with -Xlint:deprecation for details.
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ SOENTutorials ---
[INFO] Surefire report directory: /Users/alihanni/GitHub/SOEN/SOEN345/A4/UITest-
tutorial-part-A/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running test.java.tests.DemoTutorialTest
Starting ChromeDriver 98.0.4758.102 (273bf7ac8c909cde36982d27f66f3c70846a3718-re
fs/branch-heads/4758@{#1151}) on port 30463
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggest
ions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
Feb. 23, 2022 6:08:15 P.M. org.openqa.selenium.remote.ProtocolHandshake createSe
ssion
INFO: Detected dialect: W3C
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.834 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.763 s
[INFO] Finished at: 2022-02-23T18:08:16-05:00
[INFO] ------------------------------------------------------------------------
```

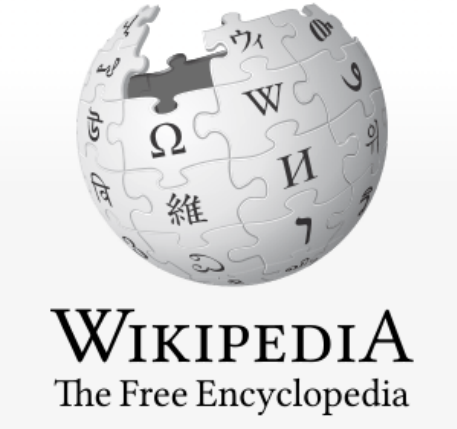
The test has passed.

Question 2 - Part B

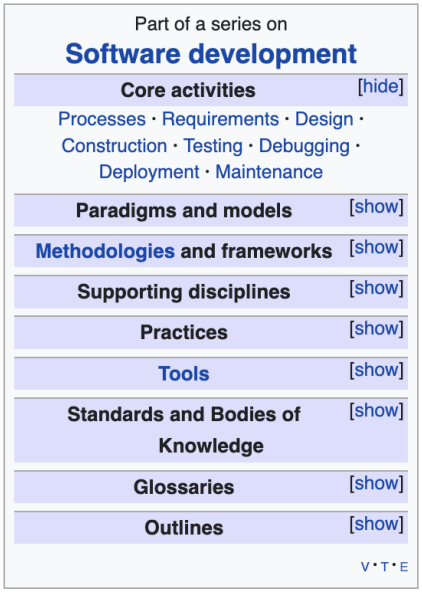
In part B, we use the selenium extension for Google Chrome to test the UI of a google search result page. Selenium let you record the UI and the different interaction the user makes with it, and then add *"commands"* that let you verify certain information about the web page that has been record, for example verifies if a certain element is present or not.
We had to verify that the elements *Images, Shopping, Videos, and News* are present on the search result page on Google. To do that, we start a new Selenium project, start create a new test, start the recording on Google's homepage, search something in Google (i.e. "arbitrary search"), once the search result page has loaded, we can stop the recording. From there, we must add commands to verify that all the elements we must check are present. To do so we use the *assert element present* command to verify that each element is present. When it's done, we can export the test (named testcase2) and save the Selenium project (which returns a .side file). Adding the test `testcase2()` in the `DemoTutorialTest` class will add the test to our existing maven project. Note that the project used in part A was copied and reused for part B.
The whole project, including the .side file and the `testCase2.java` file generated by selenium, are joined with this submission.

Question 3 - Part C

The two elements identified as different in the web application compared to the mobile applications are the Wikipedia logo on the top left of the Web app page, and the side bar on the right of the web app page. Both are not present in the mobile app version.



Wikipedia logo present on the web application



Side bar present on the web application.

The whole project used for part C is joined with this submission as `UITest-tutorial-part-C`. It includes the whole project, the /side file and both test cases created in this part.

Question 4

UI testing adds a layer to other testing methods by allowing the programmer to verify the correctness of certain element of the UI given different workflows or input from a user. It allows to test particularities concerning the UI that would have been harder or less intuitive to test otherwise, like the presence of certain element on the mobile version of the app. Also, with powerful tools such as Selenium, it is easy to perform a particular possible workflow and test throughout the whole process if the user interface reacts as expected.
UI testing definitely reduces the tester's workload. It allows the tester to quickly test everything there is to test on a certain workflow seamlessly. For example, if a certain UI element is to be retrieved in multiple pages, it is possible for the tester to verify the presence of such an element in all the pages quickly and in a single test.
Also, UI testing is a good practice since it participate in the separation of concerns, and thus reduces the chances of defects in the program not being found during testing phases.