Selenium Testing
Testing
Academy
Marco Blum.

14/04/2018
Copenhagen Business

Code can be found here:
https://github.com/Thug-Lyfe/test_SeleniumExerise/blob/master/README.md

Manual vs automatic testing.

Manual testing, excels at finding bugs that otherwise would not have been found, as it is a person doing it and people tend to do illogical things from a designer/developers point of view. It can also lead to better criticism as a machine cannot give proper feedback in the ways a person can. Whereas the automatic is all coded, the manual testing is much more adaptable to changes and can do things on the fly.

Automatic testing, excels at making sure the entire project, keeps running as it should and if anyone tampers with existing classes and files and something is not as it should be, then the automatic testing will find out soon enough. It is also a lot faster and uses much less resources than manual testing.

Test pyramid

As all the tests, are testing the ui… then no this exercise does not support the ideas of the test pyramid, we did not do any testing of the backend and client directly only through a proxy, chrome, and as such we have done no unit or service testing in this exercise, which we should have done more of than ui tests to fulfill the ideology of the test pyramid.

Automated GUI tests

There is one glaring problem with automated GUI tests. Most likely there will be a requirement to login, in at least one of the tests, if not more, and what we use to test our GUI is a third party program. This can be a serious security threat if not handled properly.

Demonstration of selenium tests

Some selenium tests can be found here:
https://github.com/Thug-Lyfe/test_SeleniumExerise/blob/master/SeleniumExerice/src/test/java/carTest.java
Usually a selenium tests works in the way that you find the out the route of your GUI test, and find a unique identifier for each the elements used in the route. Hereafter you can use the driver to get the element needed for whatever action and sendkeys, click, submit, clear etc.
fx. to test if the filter works we need two elements, the table and the filter input, from here we can find that both of them have a unique id which is the most optimal to use. So first we get the filter element from the driver and we sendkeys to it, after this we get the table from the driver and checks if it has the correct expected number of data rows. Next we clear the filter and send it a space (we do this as the clear method does not send a keystroke, so the app cannot respond to it) and check if it still has the correct amount of data rows.

It is also important to fix the waiting issue. fx. A website has to load and it can be both slow and fast based on numerous factors. Therefore it is very important to include a ***WebDriverWait*** method on the test classes, so that in the event that it cannot load the website it gives a corresponding error instead of just waiting for this one test. Now my tests are run sequentially, and it is inherently important that none of my tests freezes my testing environment.

```
62      @Test
63      public void test2(){
64          (new WebDriverWait(driver, 4)).until((ExpectedCondition<Boolean>) (WebDriver d) -> {
65              WebElement filt = driver.findElement(By.id("filter"));
66              filt.sendKeys("2002");
67              WebElement tableBody = driver.findElement(By.id("tbodycars"));
68              List<WebElement> rows = tableBody.findElements(By.tagName("tr"));
69              Assert.assertThat(rows.size(), is(2));
70              filt.clear();
71              filt.sendKeys(" ");
72              rows = tableBody.findElements(By.tagName("tr"));
73              Assert.assertThat(rows.size(), is(5));
74              return true;
75          });
76      }
```

Code for testing the filter in the GUI, also described further up in the report.

Above is an example of a ***WebDriverWait*** in action, where i have specified 4 seconds as the allotted maximum time to wait for it.
The ***WebDriverWait*** function works in the way that it ignores all instances of ***NotFoundExeptions*** until either in this case 4 seconds have passed or the ***ExpectedContition<Boolean>*** is returned which is the one returned on line 74.