Lab5: "Automated GUI testing"

Purpose

This lab aims to introduce automated webb GUI testing using Python and Selenium. Being able to automate the important GUI checks that needs to be execute often and quickly is a vital part of in most test strategies.

Task description

In this lab the front-end GUI will be tested using GUI automation. Tools to be used will be Python3, PyTest, Selenium

The front-end can be found in the virtual environment under: http://127.0.0.1/

Test automation and test data

One of the first problems one runs into when doing test automation is the test data. Test automation needs to be aware of the conditions beforehand in order to make the correct assertions or continue the test case, however a lot of times the test cases also change the underlying test data, affecting subsequent test runs of the same test case.

Imagine a test case that looks for the user "Bob Smith", changes the name to "Lisa Smith" and saves. Running the test case a second time would fail as the user "Bob Smith" is no longer present in the system. So how to work with this?

There are a few variants available to us:

1: The test case can at the end reset any changes made back into the original. Challenges with this might be that it is not something we desire in our test case, the GUI might not support it, or if the test automation end prematurely (ie crashes) then the reset might never happen.

2: The test case can query the database directly (or another intermediate layer like a REST API) about what current users are in the system. Ie it would be using secondary oracles for the information. This could be ok, but test cases can be harder to read (Instead of "Bob Smith" it could be "datarecord[0]". Also the second time the test case is executed "Lisa Smith" would be in the DB from the start and the test case would in fact never change anything ("Lisa Smith" -> "Lisa Smith").

3: The test case can reset the test environment into a known state. For example replace the database with an older known copy of it containing the user Bob Smith. If possible this solution has lots of upsides associated with it avoiding

some of the potential problems described in the points above. Technically though, reseting the test system into a know state can sometimes be a non-trivial matter.

Test cases for this lab

1: Write a test case that opens the already existing user for editing. Decide on a strategy for keeping the data consistent in between runs (see "Test automation and test data" above. For example, all customers could be fetched via the REST API or directly via the database, or the database could be overwritten with a clean backup copy of it (pos_bak.db)

- [GET] http://127.0.0.1:6399/customers
- [File] /home/pft/restapi/point-of-sale/pos.db (sqlite3)

2: Write a test case for creating a new subscriber via the Webb GUI.

Make sure to assert that data fields are shown properly in the Webb GUI for customer and equipment.

Follow up creating test cases that cover the critical paths in the system.

Python3, Selenium and Webdriver

On the virtual machine the Python bindings for Selenium is already pre-installed (pip3 install selenium).

Selenium requires a driver to interface with the chosen browser. Firefox, for example, requires geckodriver, which needs to be installed on any system where the tests are to be executed. The virtual machine already have "geckodriver" installed. For other drivers (Chrome, Edge, Firefox, etc) see: https://selenium-python.readthedocs.io/installation.html

Useful methods and modules when using Selenium

Some methods and modules that wil be of use are:

Starting the browser, creating a driver instance and closing

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.close()
```

Fetching a web page and asserting page title

Calling the get() method will wait until the page/url provided finishes loading.

```
driver.get("http://www.google.com/")
assert "Google" in driver.title
```

For more information in the Webdriver object and its available methods, see: https://selenium-python.readthedocs.io/api.html

Locating elements on a page

In order to interact with certain elements of a web page (buttons, input fields, etc), the element needs to be located first and assigned to an object.

There are various strategies to locate elements in a page. You can use the most appropriate one for your case. Selenium provides the following methods to locate elements in a page:

```
find_element_by_id
find_element_by_name
find_element_by_xpath
find_element_by_link_text
find_element_by_partial_link_text
find_element_by_tag_name
find_element_by_class_name
find_element_by_css_selector
```

For more information, see: https://selenium-python.readthedocs.io/locating-elements.html

For the system under test in the virtual machine elements of interest will have appropriate ID's associated with them. A typical strategy for test automators to identify which elements ID's or names to look for is to open the web page being automated in Google Chrome, opening the "developer tools" (CTRL+Shift+J) and then using the "Select element" tool to click on, and identify, various elements on the page.

Interacting with webelements

After finding an element on a page there typically exists a need to interact with it in various ways, some examples are:

```
elem = driver.find_element_by_id("new_button")
elem.click()

location_input = driver.find_element_by_id("Location")
assert location_input.get_attribute('value') == 'Ronneby'
```

```
location_input.clear()
location_input.send_keys('Karlskrona')
```

Other methods of interest are:

```
#Whether the element is visible to a user.
is_displayed()

#Returns whether the element is enabled.
is_enabled()

#Returns whether the element is selected.
#Can be used to check if a checkbox or radio button is selected.
is_selected()
```

There are several other methods available, for more information, see: https://seleniumhq.github.io/selenium/docs/api/py/webdriver_remote/selenium.webdriver.remote.webelement.html

Waiting for a page to reload

Sometimes (read: often) when interacting with a web page there is a need to wait for the page to be reloaded, or for certain elements to become available. Waiting for a specific element by ID can be achieved in the following way:

```
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.common.exceptions import TimeoutException

seconds_to_wait = 4
try:
    myElem = WebDriverWait(driver, seconds_to_wait).until(EC.presence_of_element_located((By
except TimeoutException:
    print("Loading took too much time!")
```