**READ ME**

==================================================================

**Title:** WW UI Automation Test Assignment

Table of contents:

1. [**Pre-requisites to be installed**](#_Pre-requisites_to_be)
2. [**HOW TO RUN?**](#_HOW_TO_RUN?)
3. [**Page Object Model**](#_Page_Object_Model:)
4. [**Project structure**](#_Project_structure:)
5. [**Test Objective**](#_Test_Objective:)
6. [**Test Case Steps:**](#_Test_Case_Steps:)
7. [**Test Case Main Script (WWnavigateTest1.py):**](#_Test_Case_Main)
8. [**Pages operations:**](#_Pages_operations:)
9. [**Scope for improvement:**](#_Scope_for_improvement:)

# Pre-requisites to be installed:

Python 3.8

**Browser:** Chrome

Selenium plugin (3.14) (Use pip install selenium)

Pycharm IDE (any Python IDE)

Note: Check Chrome browser version and install a compatible chrome driver.

In **WWnavigateTest1.py script, on line #12, change the chrome driver path**

**to the local path**

**‘**chrome\_path = r'/Users/chitrarapaka/PycharmProjects/pythonProject/venv/bin/chromedriver 2' to the local chrome driver path

For Chrome driver related issues, refer this link

<https://stackoverflow.com/questions/47148872/webdrivers-executable-may-have-wrong-permissions-please-see-https-sites-goo>

**Mac users:**

Download chrome driver for Mac from link:

<https://www.selenium.dev/documentation/getting_started/installing_browser_drivers/>

1: Set path by giving path of chrome driver.

export PATH=$PATH:/opt/WebDriver/bin >> ~/.profile

This statement modifies the PATH environmental variable. Find system’s path by executing the following command:

$ echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

2: Other Alternative is to place the chrome driver in the ‘python Project folder Pycharm -ex: python project/venv/bin/chromedriver (Applicable when python interpreter of PyCharm is used).

**Windows Users:**

Set the path in environment variables.

Ensure to have appropriate permissions for chrome driver

Chrome Driver Download for windows: <https://www.selenium.dev/documentation/getting_started/installing_browser_drivers/>

**Notes :**

Mac OS Users - while giving executable path for chrome driver, give chrome path = r'/Users/xxxx/PycharmProjects/pythonProject/venv/bin/chromedriver 2'

Windows Users - The .exe extension for chrome should be included in the path

chrome path = r'/Users/xxxx/PycharmProjects/pythonProject/venv/bin/chromedriver 2.exe’

Note:

Change permissions of chrome driver file on Mac or Windows.

Make sure to include “.exe” for chrome driver path in the script on windows but not for Mac.

Give command pip install selenium for installation of any packages

# HOW TO RUN?

Install pre-requisites, set the chrome driver path and change chrome driver path in WWnavigatetest1.py script (line 12)

Clone GIT repository “WWuiautomationproject” to the local machine.

Note: Make sure the python scripts folder has write permissions. If not, logs cannot be written into the logtextfile.

# Page Object Model:

The test case execution uses Page Object Model (POM). Page Object Model is a design pattern where the core focus is on reducing code duplication and minimization of the effort involved in code update/maintenance. Under the Page Object Model, page classes are created for all the webpages that are a part of the Automation Under Test (AUT).

* Page Object Element (Page Class/Page Object) – The Page Class is an object repository for the Web Elements/Web UI Elements of the web-pages under test. It also contains an implementation of the interfaces/methods to perform operations on these web elements. It is loosely based on the fundamentals of Object-Oriented Programming.
* Test Cases – As the name suggests, test cases contain the implementation of the actual test scenarios. It uses page methods/methods in the page class to interact with the page’s UI elements. If there is a change in the UI of the web page, only the Page Class needs to be updated, and the test code remains unchanged.

All the methods or operations are defined specific to each page. If any changes for these pages occur, only the function specific to operation need to be changed.

In Test cases file, these methods are called from each page to execute the test cases.

**Code Details:**

# Project structure:

* Pythonproject
  + WWTest (Test package that contains Pages directory, testcases directory, Logs directory)
    - Pages
      * Findstudiopage.py
      * Businesshourspage.py
    - Testcases
      * WWnavigateTest1.py

# Test Objective:

To validate WW web UI actions, record and log screenshots of the recorded actions.

# Test Case Steps:

* Step 1: Navigate to Main page -  <https://www.weightwatchers.com/us/find-a-workshop/>
* Step 2: Assert loaded page title contains “Find WW Studios & Meetings Near You | WW USA”
* Step 3: Under Find workshops - Click on Studio
* Step 4: In search field, search for meetings for zip code
* Step 5: Print title of first search result and distance located
* Step 6: Click on first search result
* Step 7: Verify if location title/name matches with the search result clicked on step 5
* Step 8: In the same page - click -Business hours-expand it
* Step 9: Define a method to print all Business hours of that studio - Output

# Test Case Main Script (WWnavigateTest1.py):

WWnavigateTest1.py is the main test case program which calls different page operations with page objects.

The script WWnavigateTest1.py has the initial setup for the web driver and calls different methods to execute the test cases. The script also initializes the logger to log the events into a log file “WWTestCaseLog.txt”. Logger level is set to info.

Screenshots: Screenshots would be taken at each step defined in the test cases.

*def set\_up():*

This method sets up the web driver.

*def test\_findstudio():*

This method calls the functions to be performed in the Findstudiopage.py. This also returns the title of the first search result.

*def test\_businesshours(search\_title):*

This method calls the functions to be performed in the Businesshourspage.py. It passes the title of the search result returned using the *test\_findstudio()* method.

# Pages operations:

1. Findstudiopage

class Findstudiopage:

This script has methods that use locators to perform operations on the page.

The methods are all defined in a class.

*def \_\_init\_\_(self, drv):*

This method is used to instantiate driver and implicitly wait, so that the time to load web elements will be taken care of. An alternate way to implement is to use webdriver wait and expected conditions.

Example for expected conditions -

*self.wait\_variable.until(E.presence\_of\_element\_located((By.CLASS\_NAME, "rightArrow-daPRP'"))).click()*

Step 2: Assert loaded page title contains “Find WW Studios & Meetings Near You | WW USA”

*def test\_title(self, logger):*

This method is used to assert the page title to check if the right page is loaded up.

Step 3: Under Find workshops-Click on Studio

*def click\_studio(self,logger):*

This method is used to click the “Studio”

Step 4: In search field, search for meetings for zip code -10011

*def search\_address(self,logger):*

This method is used to search for studios in the zip code ‘10011’. It also saves a screenshot.

Step 5: Print title of first search result and distance located

*def searchresult(self,logger):*

This method prints the title of the first search result and distance. The method also returns the title of the first search result to be used in Step 7.

Step 6: Click on first search result

*def clickfirstlink(self,logger):*

This method clicks the link to the first search result.

**Businesshourspage:**

This script has methods that use locators to perform operations on the page.

The methods are all defined in a class Businesshours

*def \_\_init\_\_(self, drv):*

This method is used to instantiate driver and implicitly wait, so that the time to load web elements will be taken care of. An alternate way to implement is to use webdriver wait and expected conditions.

Step 7: Verify if location title/name matches with the search result clicked on step 5

*def compare\_title(self,search\_title):*

This method takes the returned value in step 5 as an argument and compares it with the title.

Step 8: Click-Business hours-expand the down arrow

*def click\_businesshours(self, logger):*

This method locates then business hours down arrow and clicks to expand it.

Step 9: Prints business hours,as well as output is stored to the text file

*def print\_businesshours(self,logger):*

This method is used to get the business hours text and format it to print it and also store the output into a file “bh\_output.txt”

*def quit\_chrome(self):*

This method is used to close the quit the browser.

# Scope for improvement:

Try or Except can be used in other places to make the code more effective.

Implementation of Pytest or Unit test frameworks for validating above test case empowers the validation more.

**Enabling Automated UI testing using serverless technologies:**

The Selenium Python scripts can be integrated with AWS Lambda, post results, logs and screenshots to S3 DB and even CloudWatch Dashboard can be used for data visualization

<https://aws.amazon.com/blogs/devops/serverless-ui-testing-using-selenium-aws-lambda-aws-fargate-and-aws-developer-tools/>