Chapter 3

The Framework

Introduction

During this deliverable, we designed and built an automated testing framework to implement our test plan from the previous chapter. This is based on the **Team Term Project Specifications** document. To recap, we are testing the user interface of Wheelmap using Selenium and Python.

Framework Overview

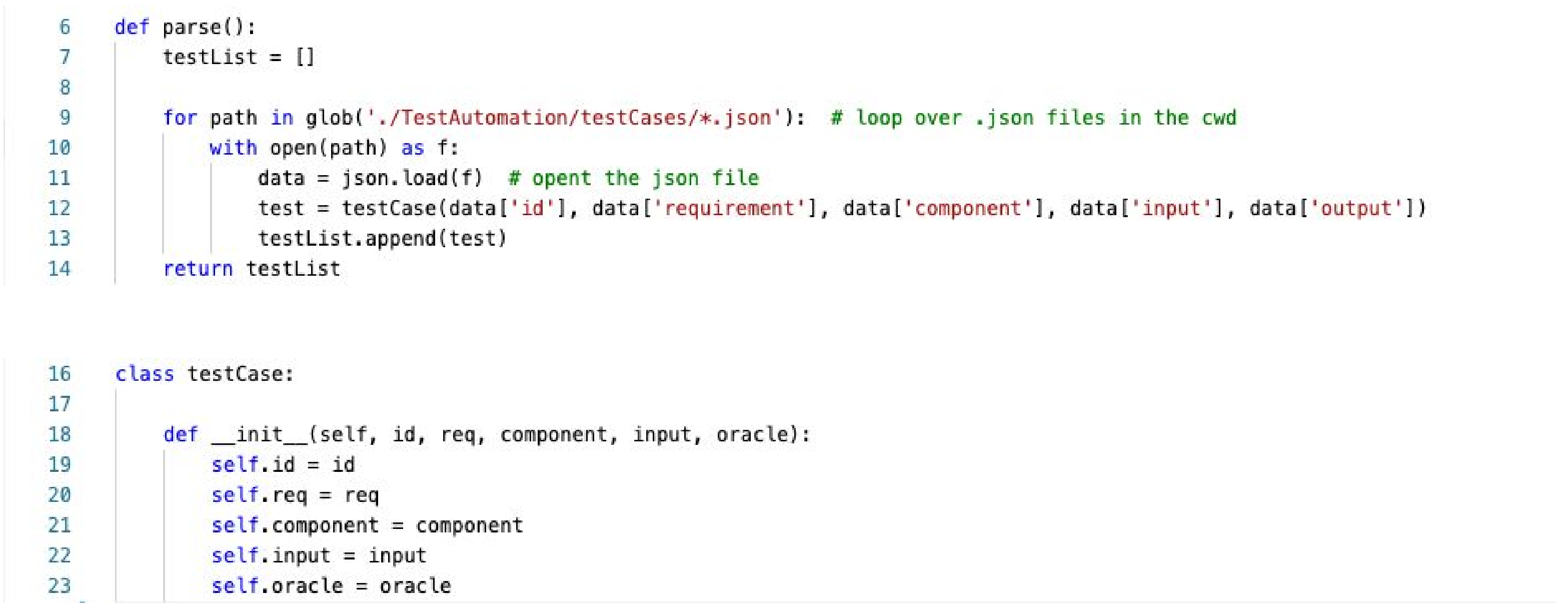
The Parser (Parser.py)

Key packages used:

- Glob

The parser is the first crucial piece of our framework. This program has a **parse** function which searches the testCases folder for all .json files. It then parses each file for the designated components of a test case (id, requirement, component, input, and oracle - in our case) and uses these attributes to create a **testCase** object.

The parse function ultimately appends each testCase object to a list which it returns.



```
def parse():
    testList = []

    for path in glob('./TestAutomation/testCases/*.json'):  # loop over .json files in the cwd
        with open(path) as f:
            data = json.load(f)  # opent the json file
            test = testCase(data['id'], data['requirement'], data['component'], data['input'], data['output'])
            testList.append(test)
    return testList
```

```
class testCase:

    def __init__(self, id, req, component, input, oracle):
        self.id = id
        self.req = req
        self.component = component
        self.input = input
        self.oracle = oracle
```

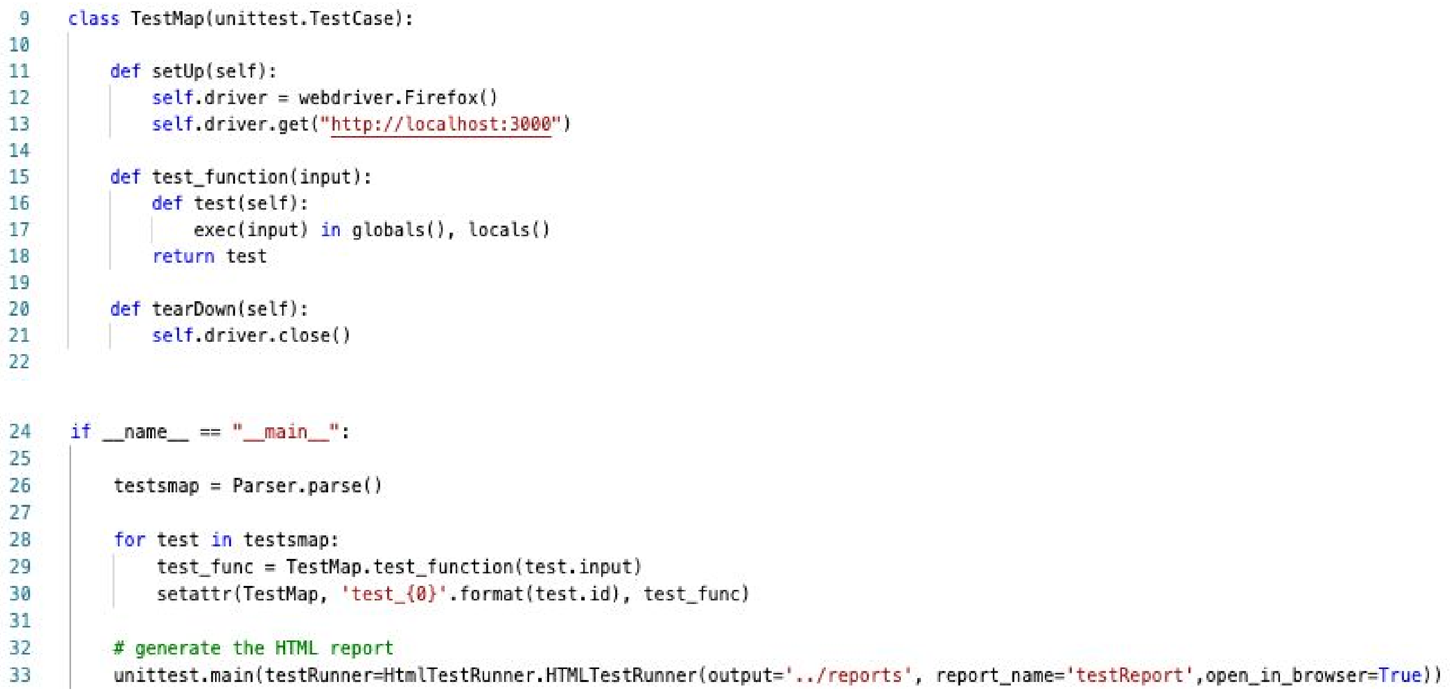
Figure 1: parse function and testCase class

The Driver

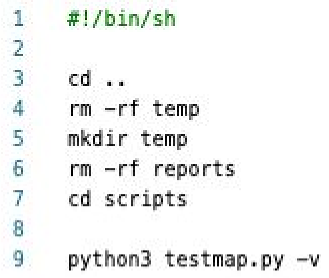
Key packages used:

- Selenium
- Unittest - a unit testing framework that is part of the standard Python library
- HTMLTestRunner

Our driver program first calls the parser to get the returned list of test case objects. It contains a class "TestMap" which then utilizes the Selenium and Unittest packages to perform the series of actions within the user interface given by the input of the test case. These are performed on the Firefox browser running Wheelmap locally.



```python
class TestMap(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.get("http://localhost:3000")

    def test_function(input):
        def test(self):
            exec(input) in globals(), locals()
        return test

    def tearDown(self):
        self.driver.close()


if __name__ == "__main__":

    testsmap = Parser.parse()

    for test in testsmap:
        test_func = TestMap.test_function(test.input)
        setattr(TestMap, 'test_{0}'.format(test.id), test_func)

    # generate the HTML report
    unittest.main(testRunner=HtmlTestRunner.HTMLTestRunner(output='../reports', report_name='testReport',open_in_browser=True))
```

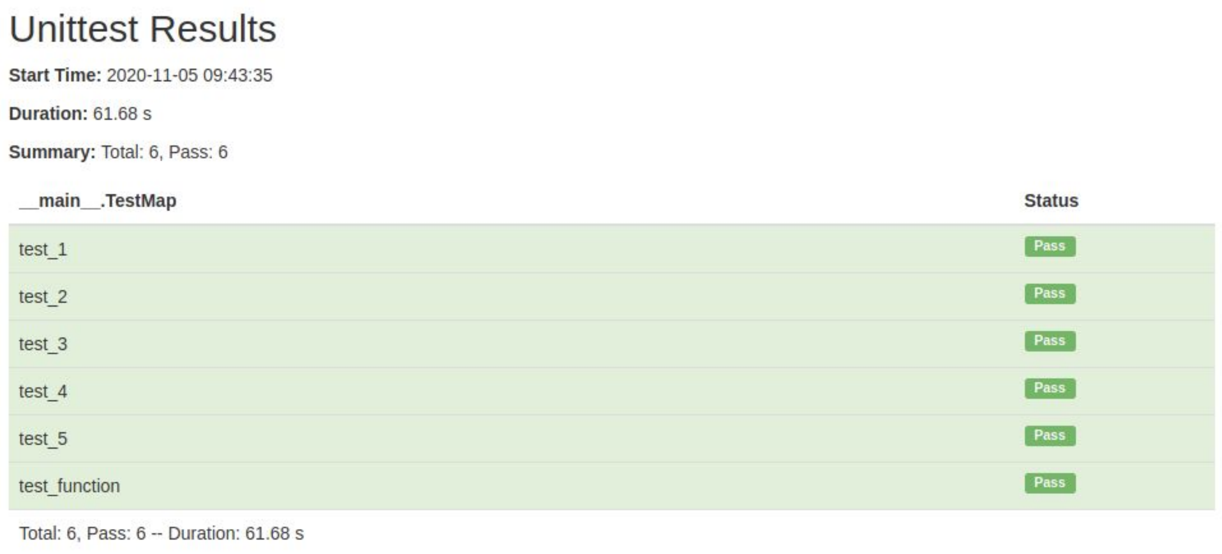


```sh
#!/bin/sh

cd ..
rm -rf temp
mkdir temp
rm -rf reports
cd scripts

python3 testmap.py -v
```

Figure 2: testmap.py and runAllTests.py

The Report

To create the report, we used a package called HTMLTestRunner. This package fits well with our framework as it integrates with Unittest, which we utilized as a framework for our test cases. It serves as a Unittest runner that saves test results in a user-friendly format within .html files.



Unittest Results

Start Time: 2020-11-05 09:43:35

Duration: 61.68 s

Summary: Total: 6, Pass: 6

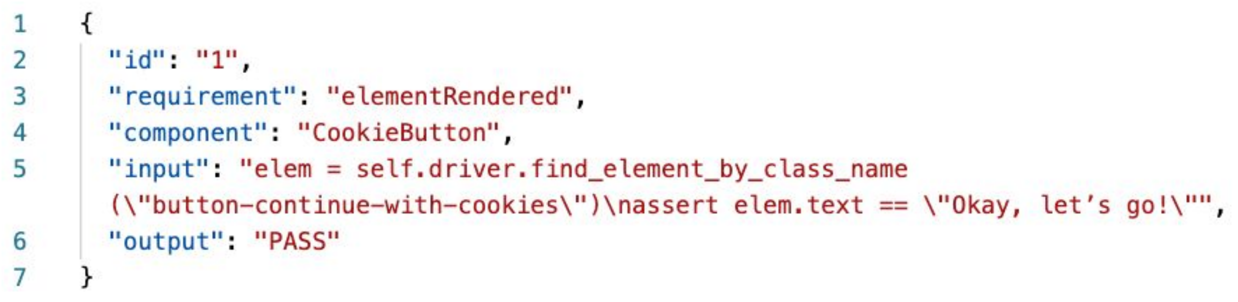
__main__.TestMap	Status
test_1	Pass
test_2	Pass
test_3	Pass
test_4	Pass
test_5	Pass
test_function	Pass

Total: 6, Pass: 6 -- Duration: 61.68 s

Figure 3: report output example

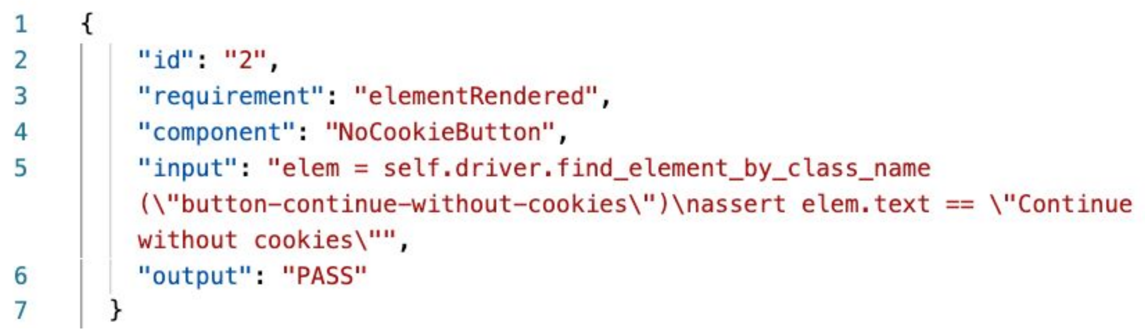
Test Cases

Test case #1: Finds the “accept cookies” button



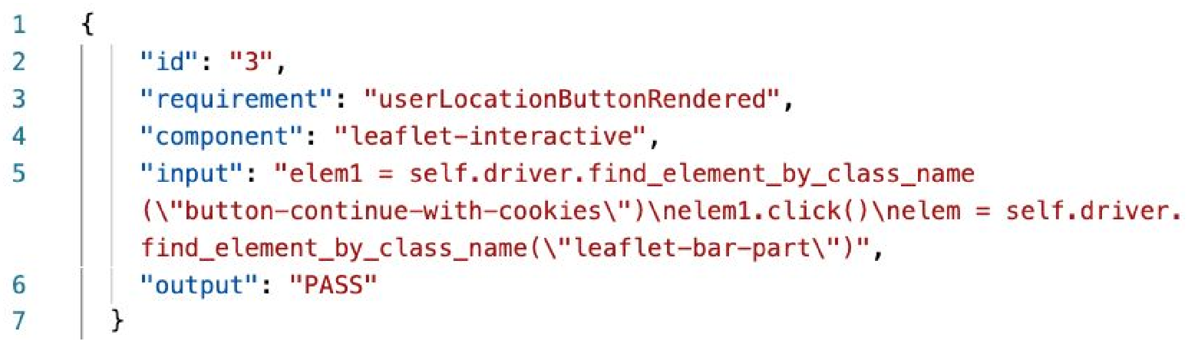
```
{
  "id": "1",
  "requirement": "elementRendered",
  "component": "CookieButton",
  "input": "elem = self.driver.find_element_by_class_name
  (\"button-continue-with-cookies\")\nassert elem.text == \"Okay, let’s go!\"",
  "output": "PASS"
}
```

Test case #2: Finds the “continue without cookies” button



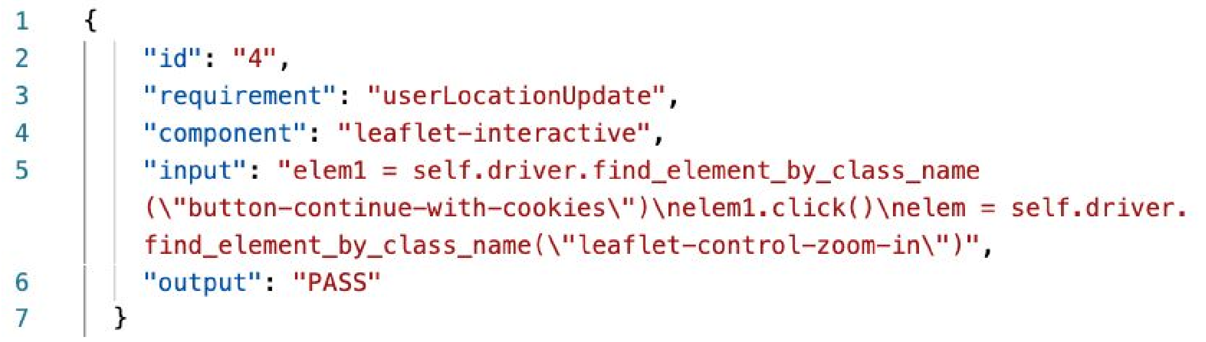
```
{
    "id": "2",
    "requirement": "elementRendered",
    "component": "NoCookieButton",
    "input": "elem = self.driver.find_element_by_class_name
    (\"button-continue-without-cookies\")\nassert elem.text == \"Continue
    without cookies\"",
    "output": "PASS"
  }
```

Test case #3: Turns the location button off



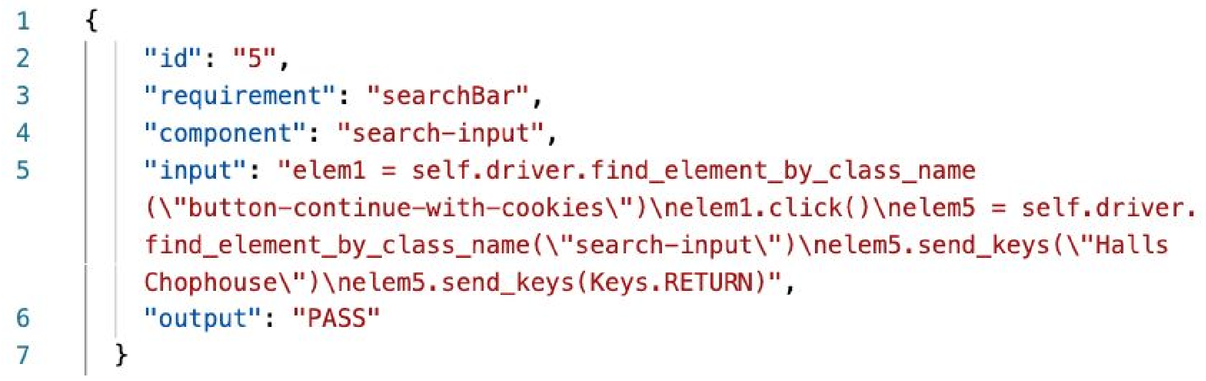
```
{
    "id": "3",
    "requirement": "userLocationButtonRendered",
    "component": "leaflet-interactive",
    "input": "elem1 = self.driver.find_element_by_class_name
    (\"button-continue-with-cookies\")\nelem1.click()\nelem = self.driver.
    find_element_by_class_name(\"leaflet-bar-part\")",
    "output": "PASS"
  }
```

Test case #4: Turns the location button on



```
{
    "id": "4",
    "requirement": "userLocationUpdate",
    "component": "leaflet-interactive",
    "input": "elem1 = self.driver.find_element_by_class_name
    (\"button-continue-with-cookies\")\nelem1.click()\nelem = self.driver.
    find_element_by_class_name(\"leaflet-control-zoom-in\")",
    "output": "PASS"
  }
```

Test case #5: Tests the search function



```
{
    "id": "5",
    "requirement": "searchBar",
    "component": "search-input",
    "input": "elem1 = self.driver.find_element_by_class_name
    (\"button-continue-with-cookies\")\nelem1.click()\nelem5 = self.driver.
    find_element_by_class_name(\"search-input\")\nelem5.send_keys(\"Halls
    Chophouse\")\nelem5.send_keys(Keys.RETURN)",
    "output": "PASS"
  }
```

How-to Guide

We have added some dependencies with this deliverable. First, please install the following:

- pip3 install html-testRunner
- pip3 install glob
- pip3 install selenium
- wget https://github.com/mozilla/geckodriver/releases/download/v0.27.0/geckodriver-v0.27.0-linux64.tar.gz

- tar -xvzf geckodriver*
- chmod +x geckodriver
- sudo mv geckodriver /usr/local/bin/

Next, get Wheelmap up and running by executing the following commands within the project folder:

- cp .env.example .env
- npm install
- npm run dev

Lastly, simply navigate to the top-level directory, TestAutomation, and execute "./scripts/runAllTests.py". This will ultimately open the .html report of the test case results.