Web UI Automation with Selenium for Beginners Documentation

Release 0.0.1

Og Maciel

Aug 18, 2018

Contents:

CHAPTER 1

About the Speaker

Fig. 1: Og Maciel

Og Maciel is a Senior Manager of Quality Engineering at **Red Hat**. He has spent the last 6+ years building a team of **Black Belt Quality Engineers** responsible for the automation of complex systems and delivering quality products through the use of Continuous Delivery of Processes. He is also an Avid Reader.

- Projects
  - Red Hat Satellite
  - Pulp Project
  - Ansible Tower
- Social
  - LinkedIn

- Gitlab
- Github
- Twitter
- GoodReads

CHAPTER 2

Objective

- Cover **Web UI automation** using **Selenium** with a focus on the **Python** programming language.
- Learn how to easily gather Web UI information, record their actions and play them back via **Selenium IDE** or **Scirocco**.
- Write **Python** code to perform the same actions **interactively**
- Write automated tests using Python's **Unittest** module 'style'
- Write automated tests using **pytest** 'style'
- Use **SauceLabs** to execute automated tests on multiple types of operating systems and web browser combinations.
- **BONUS Round**: Time permiting, use **Gitlab Pipelines** for a **Continuous Integration / Delivery** process

CHAPTER 3

Follow Along

You can get a copy of all files used in this tutorial by cloning this repository!

```
git clone https://gitlab.com/omaciel/DevConfUSA18.git
```

Then, make sure to install all the required Python modules, either using *pip*...

```
pip install -r requirements.txt
```

... or the *make* command:

```
make install
```

CHAPTER 4

(My) Definitions

4.1 When I say web UI automation...

... I mean:

- Control a web browser and access a web page's elements...
- **Programmatically** :)

4.2 DOM Object

DOM stands for Document Object Model and "is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document. The objects can be manipulated programmatically and any visible changes occurring as a result may then be reflected in the display of the document[1]."

When you're automating a web application, your job is to locate all elements that you need to interact with:

- Fields
- Buttons
- Menus
- Images
- Spinners
- Dropdowns
- Lists
- ...

[1] DOM https://en.wikipedia.org/wiki/Document_Object_Model

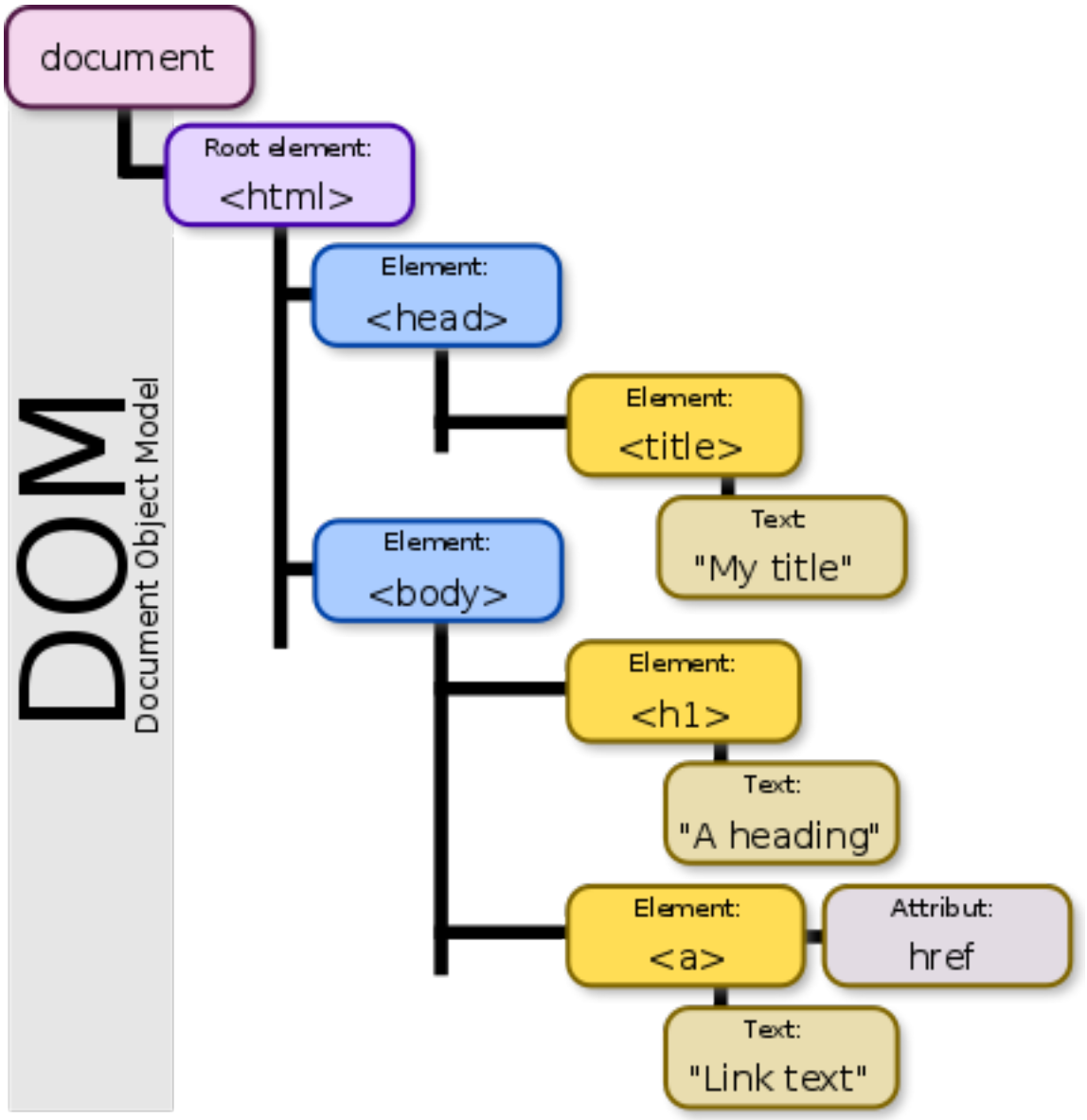


Fig. 1: Document Object Model By Birger Eriksson - Own work, CC BY-SA 3.0

4.3 Locating Web Elements

There are different schemes for locating a web element, each of them having different levels of complexity and reliability:

- Text
- Value
- CSS attributes
- Tag name
- XPath[2]

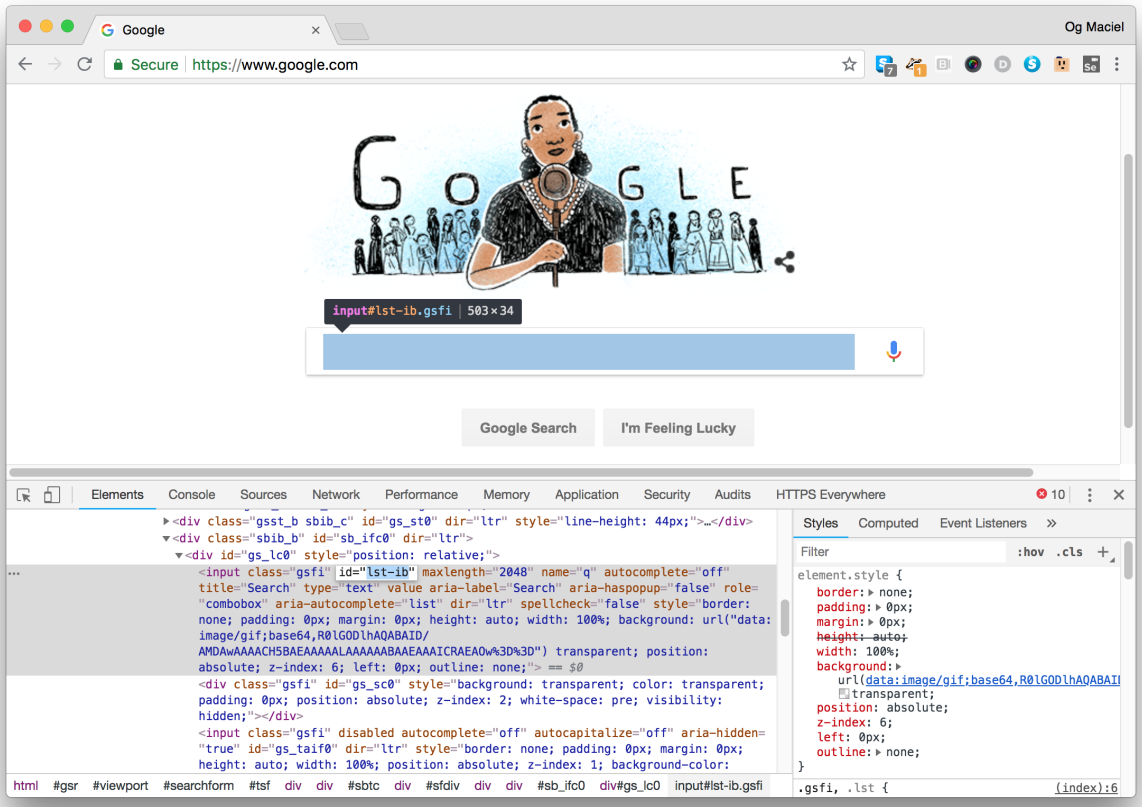


Fig. 2: Google's web page and source code for the search field.

There is a reason why I placed **XPath** at the bottom of this list:

- XPath is very fragile and any changes to a web element in the web page can break your locatiing strategy
- Xpath can be very **evil** too (i.e. hard to wrap your mind around it):

```
//a[contains(@href, 'compute_resources') and
normalize-space(.)='name']/../following-sibling::
td[@class='ellipsis']
```

[2] Xpath (XML Path Language) is a query language for selecting nodes from an XML document. https://en.wikipedia.org/wiki/XPath

CHAPTER 5

Selenium

- Selenium is a portable software-testing framework for web applications.
- It provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including:
  - C#
  - Groovy
  - Java
  - Perl
  - PHP
  - Python
  - Ruby
  - Scala
- Selenium deploys on Windows, Linux, and MacOS platforms.
- It is open-source software, released under the Apache 2.0 license[1]
- Seems to dominate the area for web UI automation
- Web 'drivers' let's you programmatically control different types of web browsers
  - Firefox
  - Chrome
  - Internet Explorer or Edge
  - Safari
  - HtmlUnit (headless browser)
- *Selenese* let's you interact with a web page's elements

[1] Selenium https://en.wikipedia.org/wiki/Selenium_(software)

- Click
- Select
- Type
- Mouse movement, hovering, dragging
- ...

CHAPTER 6

Using Selenium IDE

6.1 Overview

- Support for **Firefox** and **Chrome**
  - Requires Firefox == 61 or newer
- Records all interactions with browser
- Can replay all recordings
- Allows for multiple tests to be recorded
- **Used** to have tons of useful features but newer version has dropped most of them
- Older versions of **Selenium IDE** used to support exporting your tests to several formats, including **Python**
  - You could run the exported Python code and be done with it!

```
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import NoAlertPresentException
import unittest, time, re

class SeleniumIde(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "https://www.google.com/"
        self.verificationErrors = []
        self.accept_next_alert = True

    def test_selenium_ide(self):
        driver = self.driver
        driver.get(self.base_url + "/")
        driver.find_element_by_id("lst-ib").clear()
        driver.find_element_by_id("lst-ib").send_keys("Red Hat")
        driver.find_element_by_id("lst-ib").send_keys(Keys.ENTER)
        for i in range(60):
            try:
                if u"Red Hat® Success - Trusted by the Fortune 500 - redhat.com" == driver.find_el ➥
ement_by_id("vs0p1c0").text: break
            except: pass
            time.sleep(1)
        else: self.fail("time out")
        driver.find_element_by_id("vs0p1c0").click()

    def is_element_present(self, how, what):
        try: self.driver.find_element(by=how, value=what)
        except NoSuchElementException as e: return False
        return True

    def is_alert_present(self):
        try: self.driver.switch_to_alert()
        except NoAlertPresentException as e: return False
```

- Now only **JSON** format seems to be supported

```
{
  "id": "51a034da-3929-4bca-8581-3869764da024",
  "name": "Untitled Project",
  "url": "https://www.google.com",
  "tests": [{
    "id": "c5b80fac-3d33-4201-aa2e-28caa41f8f3b",
    "name": "Untitled",
    "commands": [{
      "id": "77f5e1a1-9363-4592-9cb2-a9c0dd67c520",
      "comment": "",
      "command": "open",
      "target": "/",
      "value": ""
    }, {
      "id": "22c0756b-ce6c-43af-bc12-1cb94bc85034",
      "comment": "",
      "command": "type",
      "target": "id=lst-ib",
      "value": "Red Hat"
    }, {
      "id": "a6368f72-3ccf-4592-96c8-68348739a453",
      "comment": "",
      "command": "sendKeys",
      "target": "id=lst-ib",
      "value": "${KEY_ENTER}"
    }, {
      "id": "c6b1c334-30b7-482f-a448-41eec304b504",
      "comment": "",
      "command": "assertText",
      "target": "css=h3.r > a",
      "value": "Red Hat - We make open source technologies for the enterprise"
    }, {
      "id": "0f0984fa-3a1c-47d5-80e1-275019e08ac1",
      "comment": "",
      "command": "clickAt",
      "target": "css=h3.r > a",
      "value": "Red Hat - We make open source technologies for the enterprise"
- 1.4k  Better Test.side  Fundamental : Helm WK Undo-Tree        1 :  0  Top
```


- Requires creativity to ensure that playing back actions will wait for web elements to be present

6.2 How to install it

6.2.1 Firefox

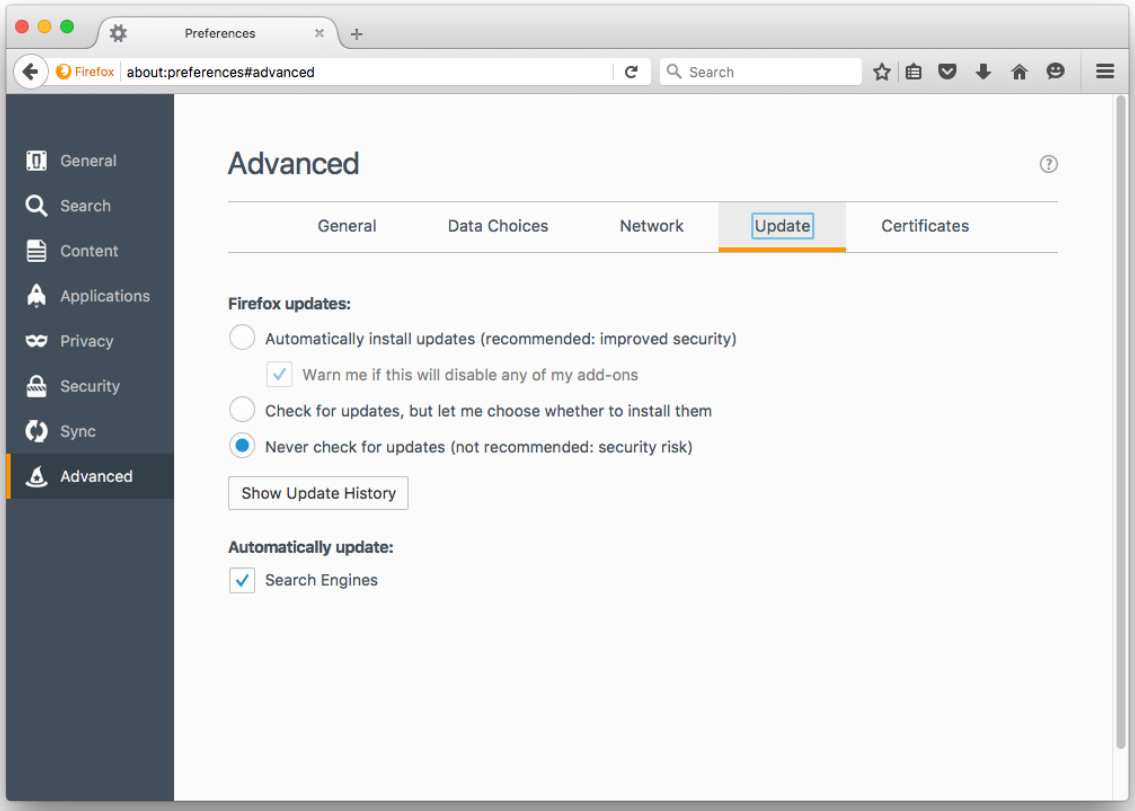
- Make sure that you're using an updated version of Firefox.
  - I recommend Firefox Quantum version 61 or greater

  - You **can** have multiple versions running side by side; just rename the executable to **Firefox45** for example

---

Important: If using an older version of **Firefox**, in order to preventing it from updating itself, you'll have to disable the automatic update feature. To do that:

- Immediately after starting Firefox, choose the **Preferences** menu
- Check the *Never check for updates (not recommended: security risk)* option



- Install Selenium IDE for Firefox

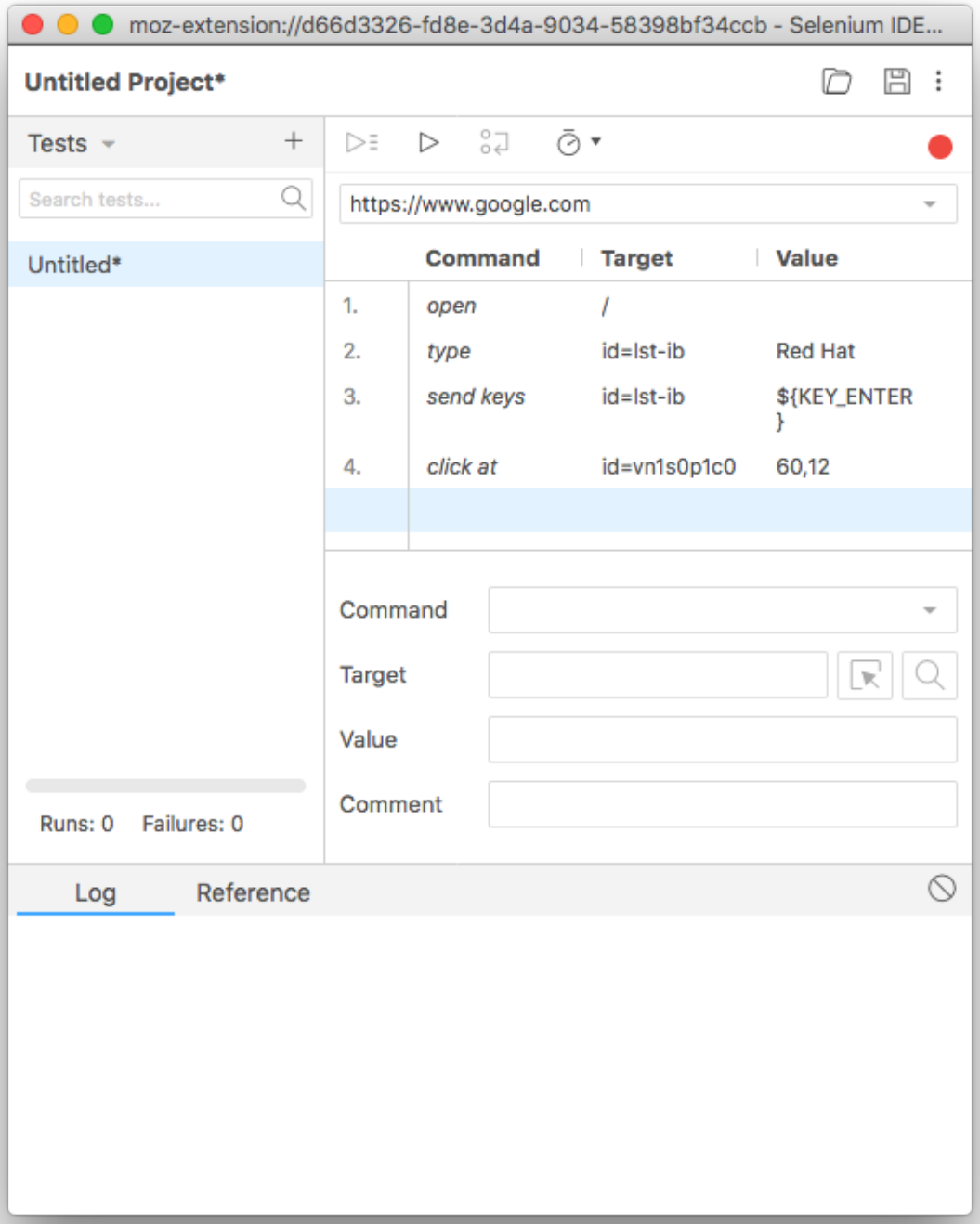
6.2.2 Google Chrome

- Make sure that you're using an updated version of Chrome
- Install Selenium IDE for Chrome

6.3 Demo

6.4 Selenium IDE: Simple Test

Here's an example of *Googling* for the term **Red Hat** and clicking the corresponding link.



This test can also be exported as **JSON** and be opened at a later time

```
{
"id": "b797dc14-e81e-4869-952f-678661776c02",
"name": "Untitled Project",
```

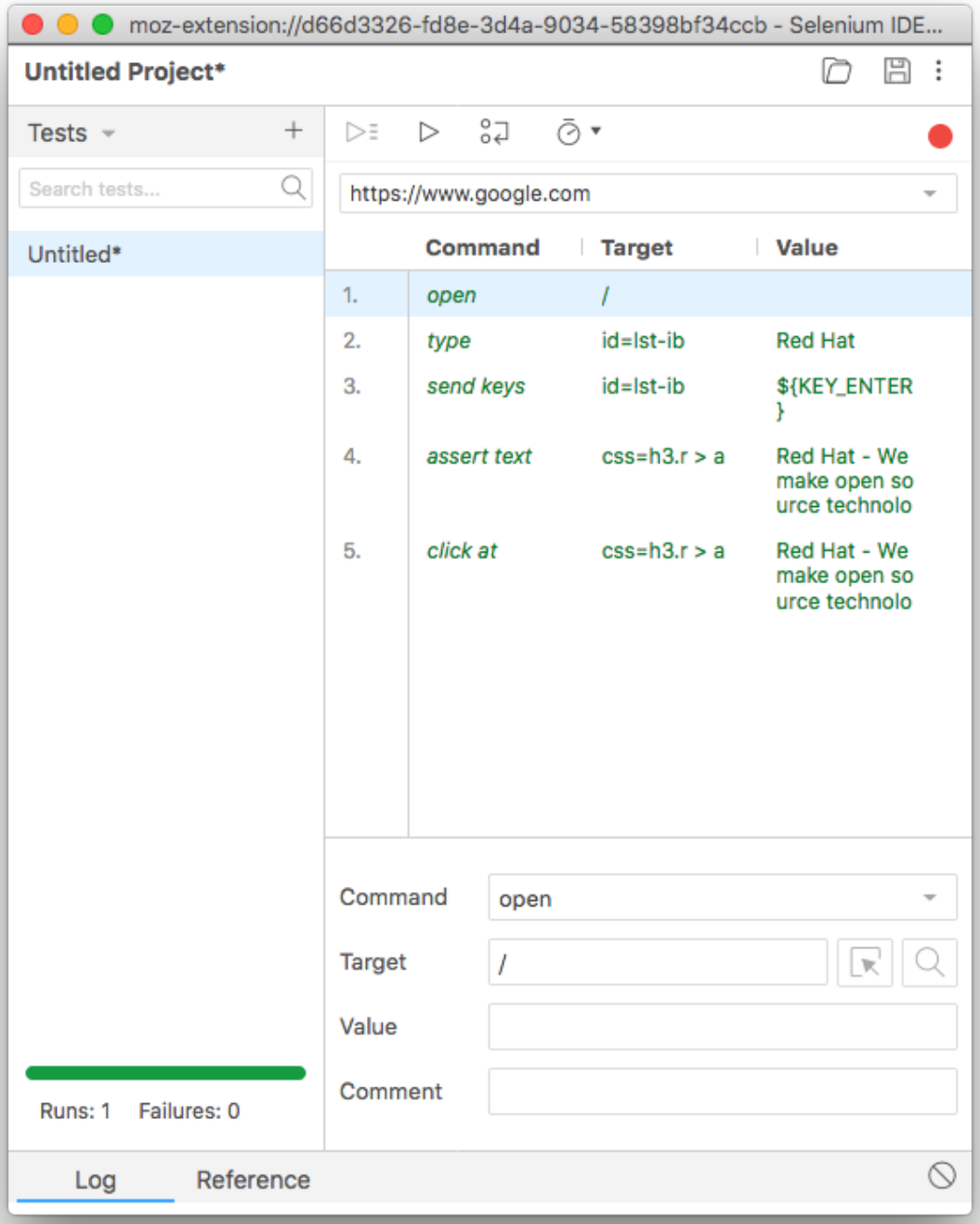
(continues on next page)

(continued from previous page)


```
"url": "https://www.google.com",
"tests": [{
    "id": "c73764c1-67ef-477f-991a-35ff97a0652b",
    "name": "Untitled",
    "commands": [{
    "id": "59e5e648-0b92-4480-99ac-eeb3ba456433",
    "comment": "",
    "command": "open",
    "target": "/",
    "value": ""
    }, {
    "id": "083c23db-032b-4495-9947-b0668924d4a1",
    "comment": "",
    "command": "clickAt",
    "target": "id=lst-ib",
    "value": "16,16"
    }, {
    "id": "0c8489a7-9b1c-40ba-9967-3b79fc3e7107",
    "comment": "",
    "command": "clickAt",
    "target": "css=h3.r > a",
    "value": "19,15"
    }]
}],
"suites": [{
    "id": "22654ef7-6e89-46e9-bd19-247030d0db7d",
    "name": "Default Suite",
    "parallel": false,
    "timeout": 300,
    "tests": ["c73764c1-67ef-477f-991a-35ff97a0652b"]
}],
"urls": ["https://www.google.com/"],
"plugins": [],
"version": "1.0"
}
```


6.5 Selenium IDE: Better Test

Here's the same example of *Googling* for the term **Red Hat** and clicking the corresponding link, but we're explicitly setting the text that must be clicked on



This test can also be exported as **JSON** and be opened at a later time

```
{
"id": "51a034da-3929-4bca-8581-3869764da024",
"name": "Untitled Project",
```

(continues on next page)

(continued from previous page)


```
"url": "https://www.google.com",
"tests": [{
    "id": "c5b80fac-3d33-4201-aa2e-28caa41f8f3b",
    "name": "Untitled",
    "commands": [{
    "id": "77f5e1a1-9363-4592-9cb2-a9c0dd67c520",
    "comment": "",
    "command": "open",
    "target": "/",
    "value": ""
    }, {
    "id": "22c0756b-ce6c-43af-bc12-1cb94bc85034",
    "comment": "",
    "command": "type",
    "target": "id=lst-ib",
    "value": "Red Hat"
    }, {
    "id": "a6368f72-3ccf-4592-96c8-68348739a453",
    "comment": "",
    "command": "sendKeys",
    "target": "id=lst-ib",
    "value": "${KEY_ENTER}"
    }, {
    "id": "c6b1c334-30b7-482f-a448-41eec304b504",
    "comment": "",
    "command": "assertText",
    "target": "css=h3.r > a",
    "value": "Red Hat - We make open source technologies for the enterprise"
    }, {
    "id": "0f0984fa-3a1c-47d5-80e1-275019e08ac1",
    "comment": "",
    "command": "clickAt",
    "target": "css=h3.r > a",
    "value": "Red Hat - We make open source technologies for the enterprise"
    }]
}],
"suites": [{
    "id": "23b8c9dd-03d5-4434-9646-21da2f664edd",
    "name": "Default Suite",
    "parallel": false,
    "timeout": 300,
    "tests": ["c5b80fac-3d33-4201-aa2e-28caa41f8f3b"]
}],
"urls": ["https://www.google.com/"],
"plugins": [],
"version": "1.0"
}
```

CHAPTER 7

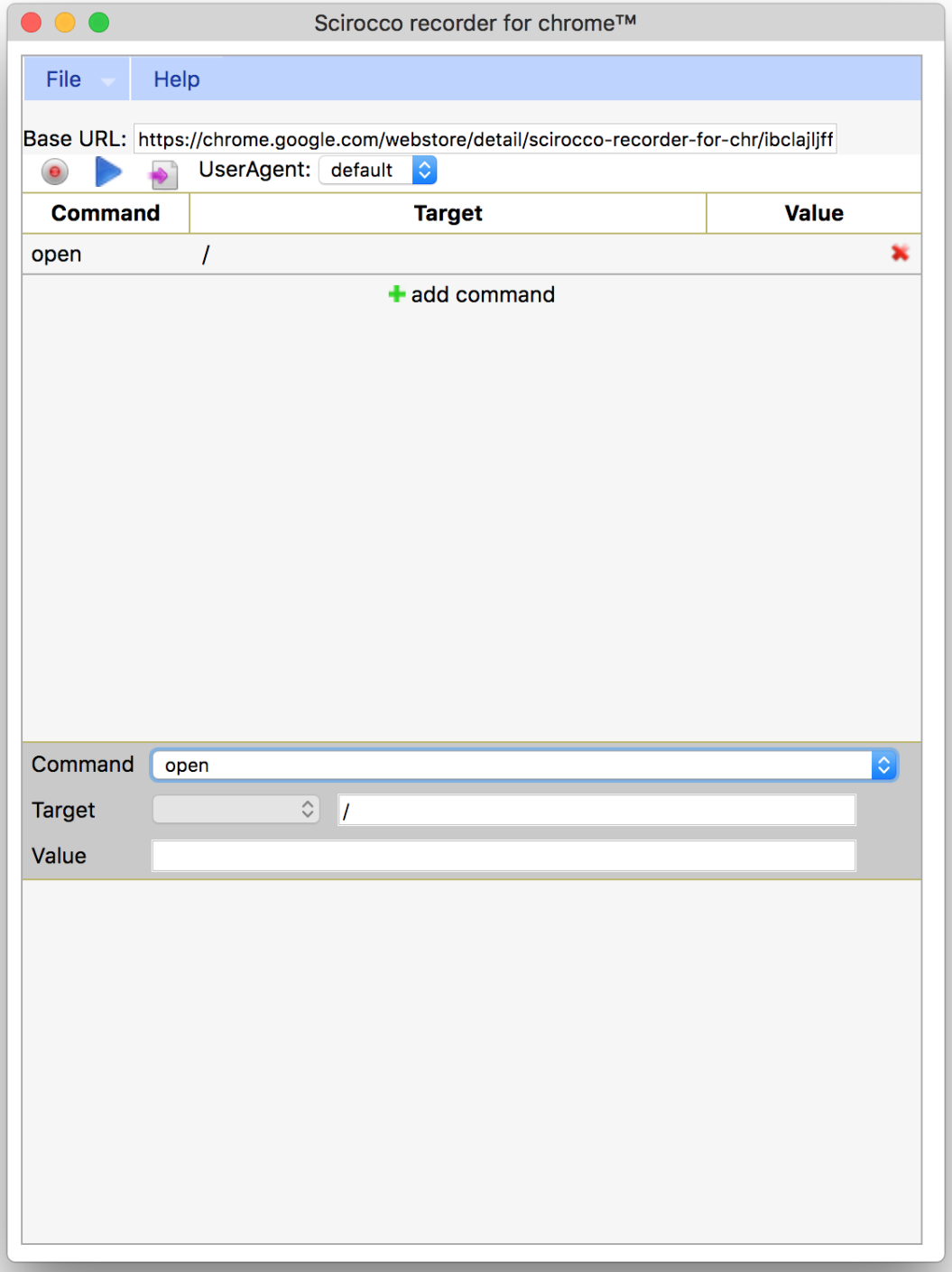
Scirocco Recorder IDE

7.1 Overview

- Records all interactions with browser
- Can replay all recordings
- **Not** as full fledged IDE as Selenium IDE
- Limited range of commands
- Limited options for exporting test cases (Python is NOT supported)

7.2 How to install it

- Make sure that you're using an updated version of Chrome
- Install Scirocco Recorder for Chrome


Scirocco recorder for chrome™
File
Help
Base URL: https://chrome.google.com/webstore/detail/scirocco-recorder-for-chr/ibclajljff
UserAgent: default
Command
Target
Value
open
/
add command
Command
open
Target
/
Value

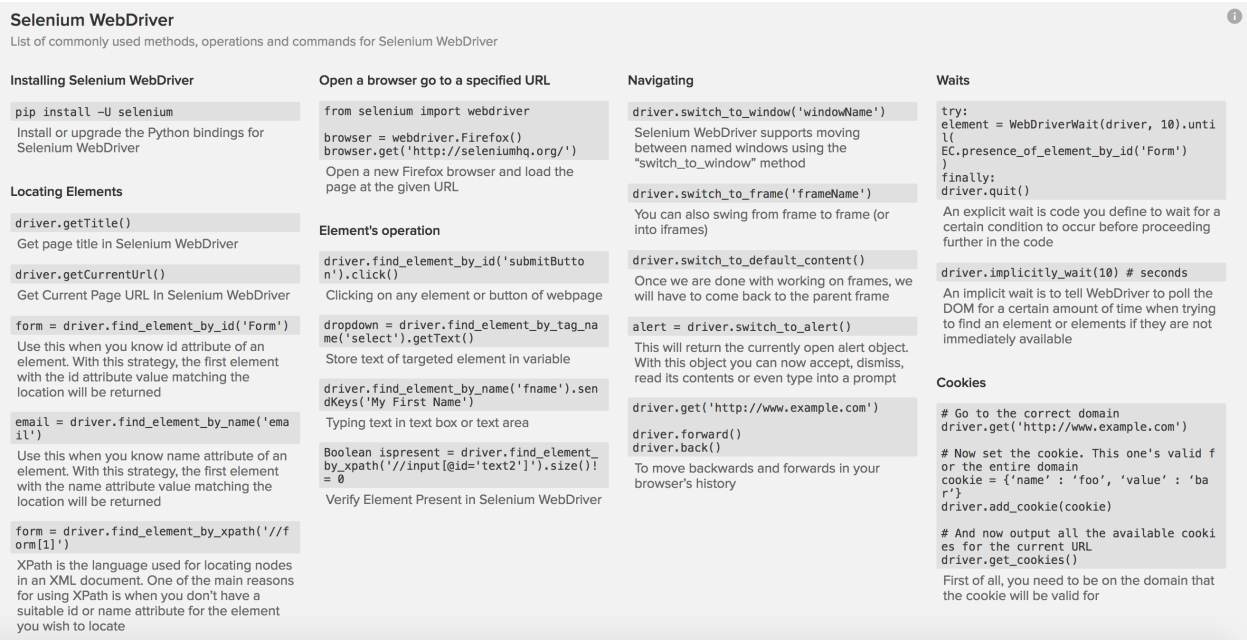
CHAPTER 8

Using Selenium with Python

Now we will write actual **Python** code to interact with a web browser.

8.1 Install Selenium Python module

The *selenium* module provides several useful methods that lets you browser web pages and interact with their web elements.



Some examples:

- find_element_by_id

- find_element_by_class_name
- find_element_by_tag_name
- find_element_by_css_selector
- find_element_by_partial_link_text
- find_element_by_xpath

For the following HTML fragment:

```
<div id="coolestWidgetEvah">...</div>
<div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></
↪div>
<iframe src="..."></iframe>
<div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></
↪div>
<a href="http://www.google.com/search?q=cheese">search for cheese</a>
<input type="text" name="example" />
```

We can use Selenium to locate several different elements:

```
element = driver.find_element_by_id("coolestWidgetEvah")
cheeses = driver.find_elements_by_class_name("cheese")
frame = driver.find_element_by_tag_name("iframe")
cheese = driver.find_element_by_css_selector("#food span.dairy.aged")
cheese = driver.find_element_by_partial_link_text("cheese")
inputs = driver.find_elements_by_xpath("//input")
```

For further information about the many more useful methods provided by Selenium, please refer to the documentation.

So let's go ahead and install the *selenium* module:

Note: This step can be skipped if you've cloned the repository and installed all Python dependencies.

```
pip install selenium
```

8.2 Install a Web Driver

Selenium requires a driver to interface with the chosen browser. Firefox, for example, requires geckodriver Some of the more popular ones are:

Web browser	Download URL
Google Chrome	https://sites.google.com/a/chromium.org/chromedriver/downloads
Microsoft Edge	https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox	https://github.com/mozilla/geckodriver/releases
Safari	https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Note: For this tutorial I have chosen to use the web driver for the Chrome web browser. On a Mac I can easily install it with:

```
brew cask install chromedriver
```

Note: Make sure to include the ChromeDriver location in your PATH environment variable

8.3 Interact with Chrome via Python Selenium

Open a Python shell and type the code below to search for Red Hat via Google interactively:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys


# Start Google Chrome
browser = webdriver.Chrome()

# Open Google.com
browser.get('https://www.google.com/')

# Locate the search box
element = browser.find_element_by_id('lst-ib')
assert element is not None

# Search for Red Hat
element.send_keys('Red Hat' + Keys.RETURN)
assert browser.title.startswith('Red Hat')

# Visit Google.com
browser.get('https://www.google.com')

# Locate the Google Logo
element = browser.find_element_by_id('hplogo')
assert element is not None

# Change the logo and adjust its height
browser.execute_script(
    "arguments[0].setAttribute('srcset', "
    "'https://omaciel.fedorapeople.org/ogmaciel.png')",
    element
    )
browser.execute_script(
    "arguments[0].setAttribute('height', '100%')",
    element
    )

# Close the browser
browser.quit()
```

CHAPTER 9

Using Python Selenium with Unittest

Let's create a couple of automated tests using Python's **unittest** module:

9.1 Use Cases

- As a user, when I open www.google.com on my web browser, the web browser tab will show the word **Google** as its title
- As a user, when I open www.google.com on my web browser and search for the word **Red Hat**, the web browser tab will show the word **Red Hat** as its title

9.2 Python Code

Here is the Python code that will automate these tests:

```
import unittest

from selenium import webdriver
from selenium.webdriver.common.keys import Keys


class GoogleTestCase(unittest.TestCase):

    def setUp(self):
        """Explicitly create a Chrome browser instance."""
        self.browser = webdriver.Chrome()
        self.addCleanup(self.browser.quit)

    def test_page_title(self):
        """Assert that title of page says 'Google'."""
        self.browser.get('http://www.google.com')
```

(continues on next page)

(continued from previous page)

```
        self.assertIn('Google', self.browser.title)

    def test_search_page_title(self):
        """Assert that Google search returns data for 'Red Hat'."""
        self.browser.get('http://www.google.com')
        self.assertIn('Google', self.browser.title)
        element = self.browser.find_element_by_id('lst-ib')
        assert element is not None
        element.send_keys('Red Hat' + Keys.RETURN)
        assert self.browser.title.startswith('Red Hat')


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Now we can execute these tests:

```
python tests/unittest/test_SeleniumUnittest.py
test_page_title (__main__.GoogleTestCase)
Assert that title of page says 'Google'. ... ok
test_search_page_title (__main__.GoogleTestCase)
Assert that Google search returns data for 'Red Hat'. ... ok

----------------------------------------------------------------------
Ran 2 tests in 7.765s

OK
```

CHAPTER 10

Using Python Selenium with pytest

Let's re-create those same automated tests but now using pytest:

10.1 Use Cases

- As a user, when I open www.google.com on my web browser, the web browser tab will show the word **Google** as its title
- As a user, when I open www.google.com on my web browser and search for the word **Red Hat**, the web browser tab will show the word **Red Hat** as its title

10.2 Install python-selenium

Note: This step can be skipped if you've cloned the repository and installed all Python dependencies.

pytest-selenium is a plugin for pytest that provides support for running Selenium based tests by providing a *selenium* fixture. You can find more information about this plugin by reading the official pytest-selenium documentation.

The most important reason for using it though is that through its *selenium* fixture we get automatically a *setup* and *teardown* without the need to write a single line of code (which can be overriden if needed of course!).

```
pip install python-selenium
```

10.3 Python Code

Here is the Python code that will automate these tests:

```python
from selenium.webdriver.common.keys import Keys


def test_page_title(selenium):
    selenium.get('https://www.google.com')
    assert 'Google' in selenium.title


def test_search_page_title(selenium):
    selenium.implicitly_wait(10)
    selenium.get('https://www.google.com')
    assert 'Google' in selenium.title
    element = selenium.find_element_by_id('lst-ib')
    assert element is not None
    element.send_keys('Red Hat' + Keys.RETURN)
    assert selenium.title.startswith('Red Hat')
```

Now we can execute these tests:

```
pytest -v --driver chrome tests/pytest/test_SeleniumPytest.py
============================================================================================␣
↪test session starts␣
↪============================================================================================
platform darwin -- Python 3.7.0, pytest-3.6.4, py-1.5.4, pluggy-0.6.0 -- /Users/
↪omaciel/.virtualenvs/devconfusa18/bin/python
cachedir: .pytest_cache
driver: chrome
sensitiveurl: .*
metadata: {'Python': '3.7.0', 'Platform': 'Darwin-17.7.0-x86_64-i386-64bit', 'Packages
↪': {'pytest': '3.6.4', 'py': '1.5.4', 'pluggy': '0.6.0'}, 'Plugins': {'xdist': '1.
↪22.5', 'variables': '1.7.1', 'selenium': '1.13.0', 'metadata': '1.7.0', 'html': '1.
↪19.0', 'forked': '0.2', 'base-url': '1.4.1'}, 'Base URL': '', 'Driver': 'chrome',
↪'Capabilities': {}}
rootdir: /Users/omaciel/hacking/devconfusa18, inifile:
plugins: xdist-1.22.5, variables-1.7.1, selenium-1.13.0, metadata-1.7.0, html-1.19.0,␣
↪forked-0.2, base-url-1.4.1
collected 2 items

tests/pytest/test_SeleniumPytest.py::test_page_title PASSED                              ␣
↪                                                                                        ␣
↪                                [ 50%]
tests/pytest/test_SeleniumPytest.py::test_search_page_title PASSED                       ␣
↪                                                                                        ␣
↪                                [100%]

============================================================================================␣
↪2 passed in 6.51 seconds␣
↪============================================================================================
```

10.4 Running All Tests Simultaneously

Now, let's execute all tests in multiple threads:

10.4.1 Install python-xdist

Note: This step can be skipped if you've cloned the repository and installed all Python dependencies.

```
pip install python-xdist
```

10.4.2 Execute All Tests

Run the following command:

```
pytest -v -n 2 --driver chrome tests/pytest/test_SeleniumPytest.py
============================================================================================␣
↪test session starts␣
↪============================================================================================
platform darwin -- Python 3.7.0, pytest-3.6.4, py-1.5.4, pluggy-0.6.0 -- /Users/
↪omaciel/.virtualenvs/devconfusa18/bin/python
cachedir: .pytest_cache
driver: chrome
sensitiveurl: .*
metadata: {'Python': '3.7.0', 'Platform': 'Darwin-17.7.0-x86_64-i386-64bit', 'Packages
↪': {'pytest': '3.6.4', 'py': '1.5.4', 'pluggy': '0.6.0'}, 'Plugins': {'xdist': '1.
↪22.5', 'variables': '1.7.1', 'selenium': '1.13.0', 'metadata': '1.7.0', 'html': '1.
↪19.0', 'forked': '0.2', 'base-url': '1.4.1'}, 'Base URL': '', 'Driver': 'chrome',
↪'Capabilities': {}}
rootdir: /Users/omaciel/hacking/devconfusa18, inifile:
plugins: xdist-1.22.5, variables-1.7.1, selenium-1.13.0, metadata-1.7.0, html-1.19.0,␣
↪forked-0.2, base-url-1.4.1
[gw0] darwin Python 3.7.0 cwd: /Users/omaciel/hacking/devconfusa18
[gw1] darwin Python 3.7.0 cwd: /Users/omaciel/hacking/devconfusa18
[gw0] Python 3.7.0 (default, Jun 29 2018, 20:13:13)  -- [Clang 9.1.0 (clang-902.0.39.
↪2)]
[gw1] Python 3.7.0 (default, Jun 29 2018, 20:13:13)  -- [Clang 9.1.0 (clang-902.0.39.
↪2)]
gw0 [2] / gw1 [2]
scheduling tests via LoadScheduling

tests/pytest/test_SeleniumPytest.py::test_search_page_title
tests/pytest/test_SeleniumPytest.py::test_page_title
[gw0] [ 50%] PASSED tests/pytest/test_SeleniumPytest.py::test_page_title
[gw1] [100%] PASSED tests/pytest/test_SeleniumPytest.py::test_search_page_title

============================================================================================␣
↪2 passed in 5.19 seconds␣
↪============================================================================================
```

CHAPTER 11

Using Python Selenium with Pytest and SauceLabs

Assuming that you have an account at SauceLabs and that you have exported your credentials via your system's environmental variables, let's re-create those same automated tests and execute them on **SauceLabs**:

11.1 Use Cases

- As a user, when I open www.google.com on my web browser, the web browser tab will show the word **Google** as its title
- As a user, when I open www.google.com on my web browser and search for the word **Red Hat**, the web browser tab will show the word **Red Hat** as its title

11.2 Python Code

Here is the Python code that will automate these tests:

11.2.1 Fixtures

```
import pytest


BROWSERS = [
        {
            'browserName': 'chrome',
            'platform': 'macOS 10.12',
        },
        {
            'browserName': 'MicrosoftEdge',
            'platform': 'Windows 10',
```

(continues on next page)

(continued from previous page)

```
        },
        {
            'browserName': 'firefox',
            'platform': 'Linux',
        },
        {
            'browserName': 'safari',
            'platform': 'macOS 10.12',
        },
]


def pytest_addoption(parser):
    """Add command line option to select webdriver capabilities.
    These options can be used to choose the webdriver capabilites
    as such:
    browser_name = request.config.getoption('--webdriver')
    """
    parser.addoption(
        '--browsername',
        action="store",
        default='firefox',
        choices=['chrome', 'firefox', 'safari', 'Android', 'MicrosoftEdge'],
        help="Specify the web browser to use for the automation."
    )
    parser.addoption(
        '--platform',
        action="store",
        default='macOS 10.12',
        choices=['macOS 10.12', 'Windows 10', 'Linux'],
        help="Specify the platform to use for the automation."
    )


def test_id(fixture_value):
    """Return a human readable ID for a parameterized fixture."""
    return '{browserName}'.format(**fixture_value)


@pytest.fixture(
    params=BROWSERS,
    ids=test_id,
    )
def capabilities(request, capabilities):
    """Used to pass arguments to SauceLabs."""
    capabilities['build'] = (
        'Web UI Automation with Selenium for Beginners'
        )
    capabilities['acceptSslCerts'] = True
    capabilities['javascriptEnabled'] = True
    capabilities.update(request.param)
    return capabilities


@pytest.fixture(
    scope='function',
)
```

(continues on next page)

(continued from previous page)

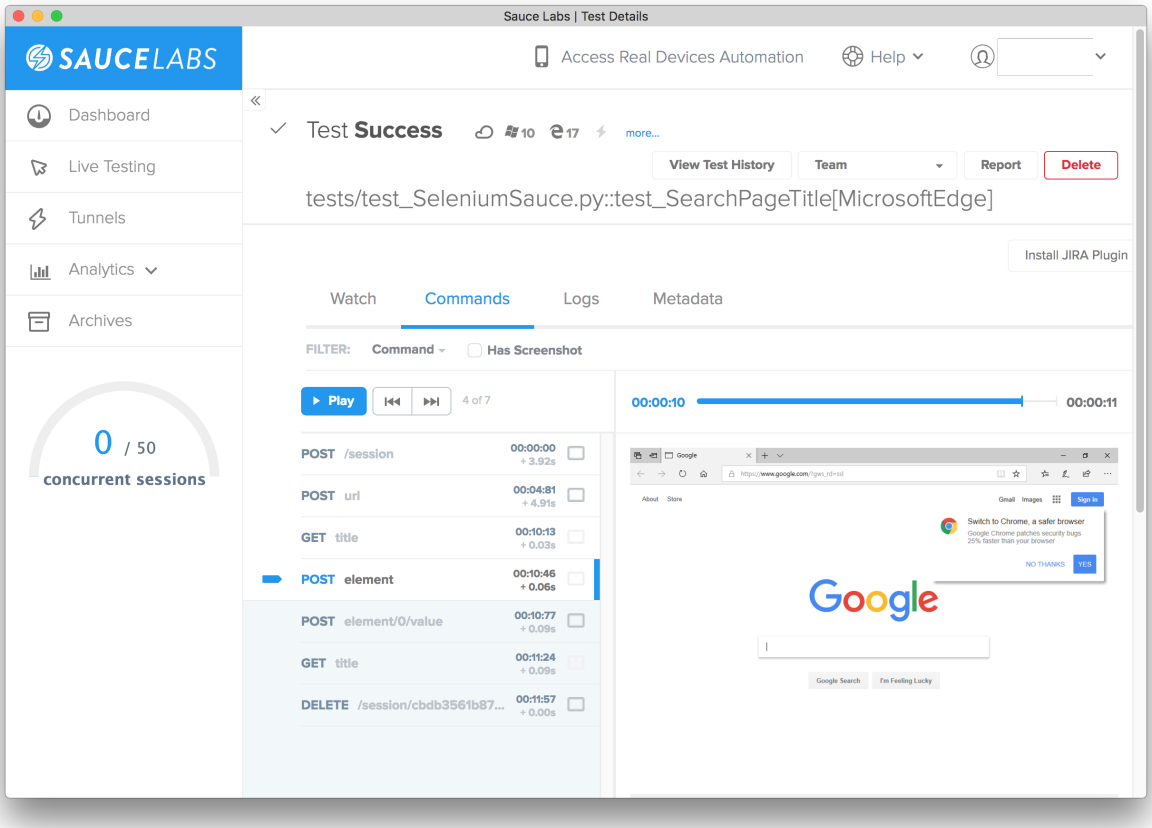
```
def browser(request, selenium):
    """Fixture to create a web browser."""
    def close_browser():
        """Handle closing browser object."""
        selenium.quit()

    def update_saucelabs():
        """Add build info for easy viewing on SauceLabs."""
        selenium.execute_script(
            "sauce:job-result={}".format(
                str(not request.node.rep_call.failed).lower()))
        selenium.execute_script(
            "sauce:job-name={}".format(request.node.rep_call.nodeid))

    request.addfinalizer(update_saucelabs)
    request.addfinalizer(close_browser)

    return selenium


@pytest.hookimpl(tryfirst=True, hookwrapper=True)
def pytest_runtest_makereport(item, call):
    # this sets the result as a test attribute for SauceLabs reporting.
    # execute all other hooks to obtain the report object
    #
    # Borrowed from https://github.com/saucelabs-sample-test-frameworks/
    #                        Python-Pytest-Selenium/blob/master/conftest.py
    outcome = yield
    rep = outcome.get_result()

    # set a report attribute for each phase of a call, which can
    # be "setup", "call", "teardown"
    setattr(item, "rep_" + rep.when, rep)
```

11.2.2 Tests

```
from selenium.webdriver.common.keys import Keys


def test_page_title(browser):
    """Assert that title of page says 'Google'."""
    browser.get('http://www.google.com')
    assert 'Google' in browser.title


def test_search_page_title(browser):
    """Assert that Google search returns data for 'Red Hat'."""
    browser.implicitly_wait(10)
    browser.get('http://www.google.com')
    assert 'Google' in browser.title
    element = browser.find_element_by_id('lst-ib')
    assert element is not None
    element.send_keys('Red Hat' + Keys.RETURN)
    assert browser.title.startswith('Red Hat')
```

Now we can execute these tests:

```
pytest -v -n 2 --driver SauceLabs tests/saucelabs/test_SeleniumSauce.py -k test_page_
↪title
============================================================================================␣
↪test session starts␣
↪============================================================================================
platform darwin -- Python 3.7.0, pytest-3.6.4, py-1.5.4, pluggy-0.6.0 -- /Users/
↪omaciel/.virtualenvs/devconfusa18/bin/python
cachedir: .pytest_cache
driver: SauceLabs
sensitiveurl: .*
metadata: {'Python': '3.7.0', 'Platform': 'Darwin-17.7.0-x86_64-i386-64bit', 'Packages
↪': {'pytest': '3.6.4', 'py': '1.5.4', 'pluggy': '0.6.0'}, 'Plugins': {'xdist': '1.
↪22.5', 'variables': '1.7.1', 'selenium': '1.13.0', 'metadata': '1.7.0', 'html': '1.
↪19.0', 'forked': '0.2', 'base-url': '1.4.1'}, 'Base URL': '', 'Driver': 'SauceLabs',
↪ 'Capabilities': {}}
rootdir: /Users/omaciel/hacking/devconfusa18, inifile:
plugins: xdist-1.22.5, variables-1.7.1, selenium-1.13.0, metadata-1.7.0, html-1.19.0,␣
↪forked-0.2, base-url-1.4.1
[gw0] darwin Python 3.7.0 cwd: /Users/omaciel/hacking/devconfusa18
[gw1] darwin Python 3.7.0 cwd: /Users/omaciel/hacking/devconfusa18
[gw0] Python 3.7.0 (default, Jun 29 2018, 20:13:13)  -- [Clang 9.1.0 (clang-902.0.39.
↪2)]
[gw1] Python 3.7.0 (default, Jun 29 2018, 20:13:13)  -- [Clang 9.1.0 (clang-902.0.39.
↪2)]
gw0 [4] / gw1 [4]
scheduling tests via LoadScheduling

tests/saucelabs/test_SeleniumSauce.py::test_page_title[chrome]
tests/saucelabs/test_SeleniumSauce.py::test_page_title[MicrosoftEdge]
[gw0] [ 25%] PASSED tests/saucelabs/test_SeleniumSauce.py::test_page_title[chrome]
tests/saucelabs/test_SeleniumSauce.py::test_page_title[firefox]
[gw0] [ 50%] PASSED tests/saucelabs/test_SeleniumSauce.py::test_page_title[firefox]
[gw1] [ 75%] PASSED tests/saucelabs/test_SeleniumSauce.py::test_page_
↪title[MicrosoftEdge]
tests/saucelabs/test_SeleniumSauce.py::test_page_title[safari]
[gw1] [100%] PASSED tests/saucelabs/test_SeleniumSauce.py::test_page_title[safari]

============================================================================================␣
↪4 passed in 63.22 seconds␣
↪============================================================================================
```


Sauce Labs | Test Details
SAUCELABS
Access Real Devices Automation
Help
Dashboard
Live Testing
Tunnels
Analytics
Archives
0 / 50
concurrent sessions
Test Success
more...
View Test History
Team
Report
Delete
tests/test_SeleniumSauce.py::test_SearchPageTitle[MicrosoftEdge]
Install JIRA Plugin
Watch
Commands
Logs
Metadata
FILTER:
Command
Has Screenshot
Play
4 of 7
00:00:10
00:00:11
POST /session
POST url
GET title
POST element
POST element/0/value
GET title
DELETE /session/cbdb3561b87...
Google

CHAPTER 12

Gitlab Pipelines

Now that we have successfully automated a couple of Use Cases, how about creating an automated process that can handle re-running our tests whenever changes to the source code happens?

For the remainder of this presentation I will cover how we can leverage Gitlab's Pipelines to perform the following actions:

- Rebuild this presentation
- Execute our automated tests via pytest against
  - Firefox
  - Google Chrome
- Lastly, assuming our tests pass, re-run them against a matrix of web browsers and operating systems on Sauce-Labs

All of these actions will be performed by our Pipeline automatically and without requiring any infrastructure from you, every time code changes get merged into this repository.

12.1 Pipelines

"A pipeline is a group of jobs that get executed in stages(batches). All of the jobs in a stage are executed in parallel (if there are enough concurrent Runners), and if they all succeed, the pipeline moves on to the next stage. If one of the jobs fails, the next stage is not (usually) executed. You can access the pipelines page in your project's Pipelines tab.[1]"

12.2 Defining Pipelines

a pipeline on Gitlab is defined by adding a *.gitlab-ci.yml* file to your repository. The pipeline for this repository can be seen bellow:

[1] Pipelines https://docs.gitlab.com/ee/ci/pipelines.html

```
image: python:3.7

stages:
  - Generate Docs
  - Test Pytest
  - Test SauceLabs

Build Document:
  stage: Generate Docs
  before_script:
    - make install
  script:
    - make html
    - doc8 --ignore D001

UI Pytest Chrome:
  stage: Test Pytest
  before_script:
    - make install
  script:
    - pytest -vvv --driver Remote --capability browserName chrome --host selenium --
↪port 4444 tests/pytest/test_SeleniumPytest.py
  services:
    - name: selenium/standalone-chrome
      alias: selenium

UI Pytest Firefox:
  stage: Test Pytest
  before_script:
    - make install
  script:
    - pytest -vvv --driver Remote --capability browserName firefox --host selenium --
↪port 4444 tests/pytest/test_SeleniumPytest.py -k test_page_title
  services:
    - name: selenium/standalone-firefox
      alias: selenium

UI Pytest SauceLabs Matrix:
  stage: Test SauceLabs
  before_script:
    - make install
  script:
    - pytest -vvv --driver SauceLabs tests/saucelabs/test_SeleniumSauce.py -k test_
↪page_title
```

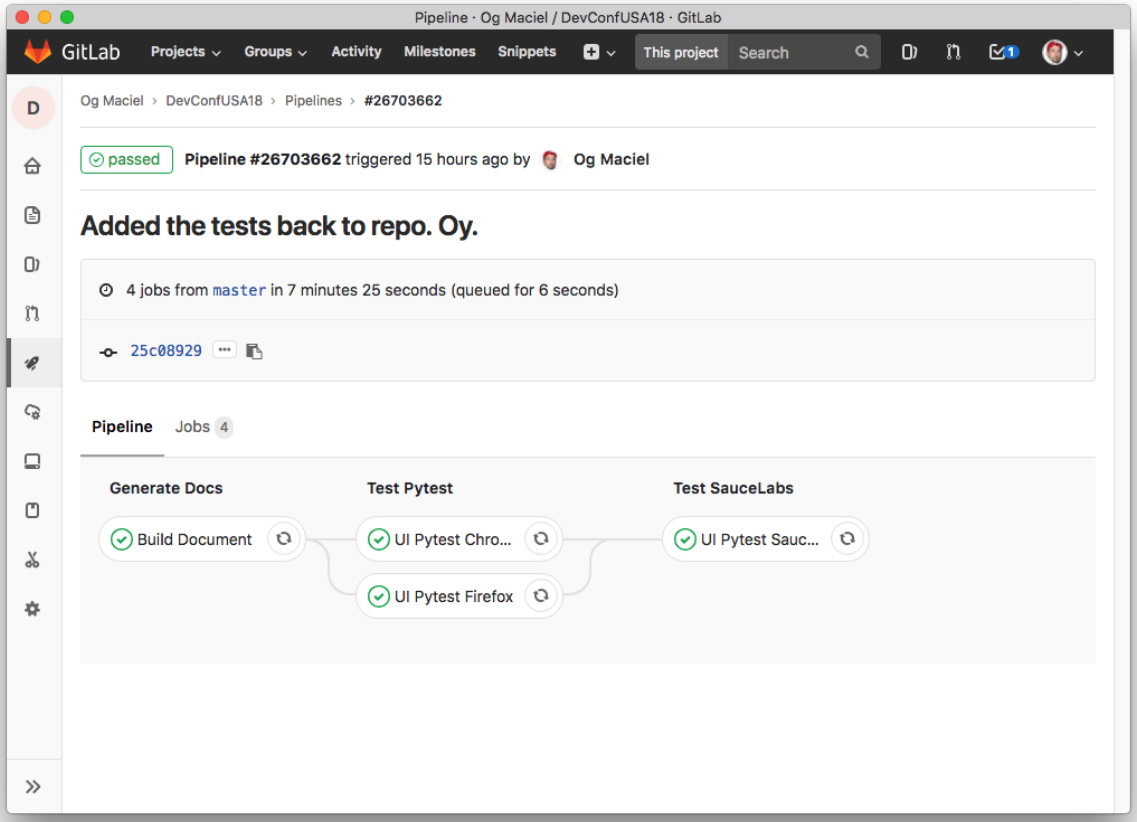
There are a total of 4 unique jobs:

- Build Document: executes *make html* and builds the document for this presentation
- UI Pytest Chrome: Runs automated tests against a Dockerized instance of Google Chrome
- UI Pytest Firefox: Runs a subset of automated tests against a Dockerized instance of Firefox
- UI Pytest SauceLabs Matrix: Runs a subset of automated tests against a matrix of web browsers and operating systems on SauceLabs

Furthermore, these jobs are executed by stages, each stage being a requirement for the next one, as shown below:

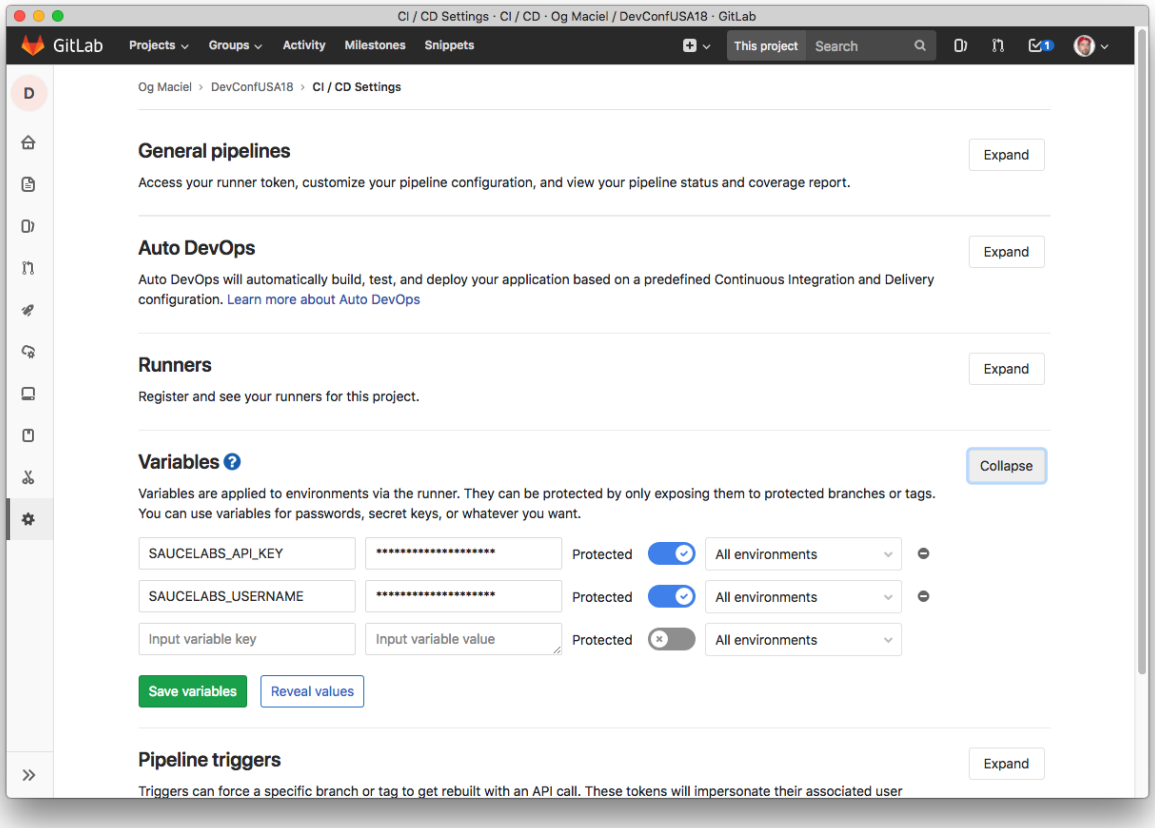
- Generate Docs

- Test Pytest
- Test SauceLabs



Note: In order to run our automated tests against *SauceLabs*, we need to define to environmental variables that will grant the automation access to SauceLabs environment:

- SAUCELABS_API_KEY
- SAUCELABS_USERNAME



Since we don't want to keep this information accessible to the WWW, Gitlab offers a convenient Variables page that let's you store per-project or per-group variables which can then be accessed by your jobs at runtime. Furthermore, you can mark them as **proctected** so that they won't be displayed anywhere on the UI or logs!

Though I won't go into the steps required to create jobs or the syntax, this document is a very good start. The *.gitlab-ci.yml* file provided here should also provide you with the 'seed' for your onw file.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search