2017

INTERNSHIP REPORT

Abhinav Jain

7/15/2017

PREFACE AND ACKNOWLEDGEMENT

For two months from May 2017 to July 2017 I did an Internship at Voylla Fashions Private Limited. It is an Indian e-commerce platform headquartered in Jaipur, India. It was started as a designer apparel selling portal, and later diversified into a full-fledged e-shop that provides jewellery and accessories from designers under categories such as fashion jewellery, silver jewellery and precious metal accessories, to its customers across India. I am pursuing B.Tech from The LNM Institute of Information Technology (Y-15 – Y-19) and done this internship after second year.

This report documents the work done for the DATA ANALYTICS Department at VOYLLA in which different tasks, are included like to Automate the web interaction and save data from different platforms like (Flipkart, Amazon, Facebook, Cretio, Google Analytics) using Python programming language, fetch data from other platforms using their api's , to implement a tool that can monitor file-system and automatically push those files to the company's database by applying selective query without any need of employee, to prepare the scripts that can upload data automatically and to create the visualization of Vendor's performance in Microsoft PowerBI Tool which gives the information about best vendor and worst vendor for particular taxon on the basis of data available for last three years. I have tried my best to keep report simple yet technically correct.

I could not have done this work without the help of whole Data Analytics Department. The work culture really motivates. Everyone is friendly and cheerful companion here that work stress never comes in way. I would like to pay my gratitude to Mr. Parag Jain, Sr. Director at Voylla and my supervisor during the internship. I would also like to thank Mr. Ankur Gupta, Business Analyst who was my mentor, helped in solving every problem, made my concepts clear, regularly monitored my task and assigned me the different ones. I would also like to thank Mr Prakash for helping me and giving innovative ideas.

AUTOMATING THE WEB INTERACTION –

Configuration Required – Python 2.7, Selenium 3, Ubuntu 14.04 or above, Mozilla Firefox 52.02

First task is to automatically download the data from different platforms for Analytics Team so that they can draw the insights from that. The importance of this application is to save the time of employee as now they do not need to manually download the data. They will aleready have data in their database System. For doing this task I used Selenium.

Selenium is a set of tools for automating browsers. It is largely used for testing applications, but its usages are not limited only to testing. It can also be used for screen scraping and automating repetitive tasks in a browser window. Selenium supports automation on all the major browsers including Firefox, Internet Explorer, Google Chrome, Safari, and Opera.

Selenium IDE: This is a Firefox add-in used to record and play back the Selenium scripts with Firefox. It provides a graphical user interface to record user actions using Firefox, But this tool can only be used with Firefox and other browsers are not supported.

Selenium WebDriver: This is a programming interface for developing advanced Selenium scripts using programming languages. Selenium WebDriver offers client libraries in various languages, including Java, C#, Python, Ruby, PHP, and JavaScript, and are more into writing test scripts. I used Python to automate browsers.

We cannot set up test preconditions and post conditions, check the expected and actual output, check the state of the application, report test results, create data-driven tests, and so on with Selenium WebDriver. We can use a unit testing framework or test runners used for unit testing along with Selenium to create a testing framework. The unittest module is used within the Python project to test various standard library modules including unittest itself. A test case is the smallest unit of testing in unittest. It checks for a specifc response to a particular set of actions and inputs using various assert methods provided by the unittest library. The unittest library provides a base class called TestCase that may be used to create new test cases.

A test created with the xUnit framework such as unittest is divided into three parts also known as the 3 A's, as follows:

- **Arrange**: This part sets up the preconditions for tests including the object(s) that need to be tested, related confguration, and dependencies.
- **Act**: This part exercises the functionality
- **Assert**: This part checks the outcome with the expected results

THE TESTCASE CLASS –

Test or group of tests can be created, by inheriting the TestCase class and adding each test as a method to this class. To make a test, we need to either use assert or one of the many variations on assert that are part of the TestCase class. The most important task of each test is a call to assertEqual() to check for an expected result, assertTrue() to verify a condition, or assertRaises() to verify that an expected exception gets raised.

In addition to adding tests, we can add test fxtures: that is the setUp() and tearDown() methods to handle creation and disposition of any objects or conditions that are needed for a test.

The setUp() method

The starting point for test cases is the setUp() method, which we can use to perform some tasks at the start of each test or all the tests that will be defned in the class. These can be test preparation tasks such as creating an instance of a browser driver, navigating to the base URL, loading test data, opening logfles, and so on.
This method takes no arguments and doesn't return anything. When a setUp() method is defined, the test runner will run that method prior to each test method.
In my code, we I use the setUp() method to create an instance of Firefox, to create firefox profile and to set up the properties. Setting up the firefox profile is optional, But I set it so that, at the end after clicking on “Download” Button file can be directly download and windows pop-up do not occour.

Here is setup method used by me-

```
profile = webdriver.FirefoxProfile()
profile.set_preference('browser.download.folderList', 2)
profile.set_preference('browser.download.manager.showWhenStarting', False)
profile.set_preference('browser.download.dir',os.getcwd())
profile.set_preference('browser.helperApps.neverAsk.saveToDisk', 'text/csv/xls')
self.driver = webdriver.Firefox(profile)
self.driver.implicitly_wait(30)
self.base_url = https://seller.flipkart.com/
```

One can take the help for setting up profile from here –
https://stackoverflow.com/questions/1176348/access-to-file-download-dialog-in-firefox

Writing tests

Similar to the setUp() method, test
methods are implemented in the TestCase class. It is important that we name these methods beginning with the word test. This naming convention informs the test runner about which methods represent a test.

FINDING ELEMENTS

There are various method to find elements-

1. find_element_by_id(id)
2. find_element_by_name(name)
3. find_element_by_class_name(name)
4. find_element_by_tag_name(name)
5. find_element_by_xpath(xpath)
6. find_element_by_css_selector(css_selector)
7. find_element_by_link_text(link_text)
8. find_elements_by_partial_link_text(link_text)

Example –

These are some of the lines of my code which I used to interact with elements-

```
driver = self.driver
driver.get(self.base_url + "/")
driver.find_element_by_id("username").clear()
driver.find_element_by_id("username").send_keys("orders-flipkart@voylla.com")
driver.find_element_by_id("userpass").clear()
driver.find_element_by_id("userpass").send_keys("########")
driver.find_element_by_id("edit-submit").click()
driver.get("https://seller.flipkart.com/index.html#dashboard/fa/report")
driver.find_element_by_class_name('caret').click()
driver.find_element_by_link_text("BilaspurWarehouse")
driver.switch_to.frame(driver.find_element_by_tag_name('iframe'))
```

Cleaning up the code

Similar to the setUp() method that is called before each test method, the TestCase class also calls a tearDown() method to clean up any initialized values after the test is executed. Once a test is executed, the values defned in the setUp() method are no longer required; so, it is good practice to clean up the values initialized by the setUp() method after a test is completed.

Example –

```
def tearDown(self):
self.driver.quit()
self.assertEqual([], self.verificationErrors)
```

Webdriver Class –

The WebDriver class provides a number of properties or attributes for browser interaction. We can use the properties and methods of the WebDriver class to interact with the browser window, alerts, frames and pop-up windows. It also provides features to automate browser navigation, access cookies, capture screenshots, and so on.

PROPERTIES OF WEBDRIVER CLASS -

Property/attribute	Description	Example
current_url	This gets the URL of the current page displayed in the browser	driver.current_url
current_window_ handle	This gets the handle of the current window	driver.current_ window_handle
name	This gets the name of the underlying browser for this instance	driver.name
orientation	This gets the current orientation of the device	driver.orientation
page_source	This gets the source of the current page	driver.page_source
title	This gets the title of the current page	driver.title
window_handles	This gets the handles of all windows within the current session	driver.window_handles

METHODS OF WEBDRIVER CLASS –

The WebDriver class implements various methods to interact with the browser window, web pages, and the elements on these pages. Here is a list of some important methods:

Method	Description	Argument	Example
back()	This goes one step backward in the browser history in the current session.		driver.back()
close()	This closes the current browser window.		driver.close()
get(url)	This navigates and loads a web page in the current browser session.	url is the address of the website or web page to navigate	driver. get("http:// www.google. com")
maximize_window()	This maximizes the current browser window.		driver. maximize_ window()
quit()	This quits the driver and closes all the associated windows.		driver.quit()
refresh()	This refreshes the current page displayed in the browser.		driver. refresh()

switch_to.active_ element()	This returns the element with focus or the *body* if nothing else has focus.		driver.switch_ to_active_ element()
Switch.to_alert()	This switches the focus to an alert on the page.		driver.switch_ to_alert()
switch_ to.default_ content()	This switches the focus to the default frame.		driver.switch_ to_default_ content()
switch_ to.frame(frame_ reference)	This switches the focus to the specified frame, by index, name, or web element. This method also works on IFRAMES.	frame_ reference: This is the name of the window to switch to, an integer representing the index, or a web element that is a frame to switch to	driver. switch_to_ frame('frame_ name')
switch_ to.window(window_ name)	This switches focus to the specified window.	window_name is the name or window handle of the window to switch to.	driver. switch_to_ window('main')
implicitly_ wait(time_to_ wait)	This sets a sticky timeout to implicitly wait for an element to be found, or a command to complete. This method only needs to be called one time per session. To set the timeout for calls to execute_ async_script, see set_script_ timeout.	time_to_wait is the amount of time to wait (in seconds).	
set_page_load_ timeout(time_to_ wait)	This sets the amount of time to wait for a page load to complete.	time_to_wait is the amount of time to wait (in seconds).	driver.set_ page_load_ timeout(30)
set_script_ timeout(time_to_ wait)	This sets the amount of time that the script should wait during an execute_async_ script call before throwing an error.	time_to_wait is the amount of time to wait (in seconds).	driver. set_script_ timeout(30

We can interact with elements on a web page using the WebElement class. We can interact with a textbox, text area, button, radio buttons, checkbox, table, table row, table cell, div, and so on using the WebElement class.

Properties of the WebElement class

The WebElement class implements the following properties:

Property/attribute	Description	Example
size	This gets the size of the element	element.size
tag_name	This gets this element's HTML tag name	element.tag_name
text	This gets the text of the element	element.text

Methods of the WebElement class

The WebElement class implements the following methods:

Method	Description	Argument	Example
clear()	This clears the content of the textbox or text area element.	element.clear()	
click()	This clicks the element.	element.click()	
get_ attribute(name)	This gets the attribute value from the element.	name is the name of the attribute.	element.get_ attribute("value") Or element.get_ attribute ("maxlength")
is_displayed()	This checks whether the element is visible to the user.	element.is_ displayed()	
is_enabled()	This checks whether the element is enabled.	element.is_ enabled()	
is_selected()	This checks whether the element is selected. This method is used to check the selection of a radio button or checkbox.	element.is_ selected()	

send_ keys(*value)	This simulates typing into the element.	Value is a string for typing or setting form fields.	element.send_ keys("foo")

Properties of the Select class

The Select class implements the following properties:

Property/attribute	Description	Example
all_selected_options	This gets a list of all the selected options belonging to the dropdown or list	select_element.all_ selected_options
first_selected_ option	This gets the first selected / currently selected option from the dropdown or list	select_element.first_ selected_option
options	This gets a list of all options from the dropdown or list	select_element.options

Methods of the Select class

The Select class implements the following methods:

Method	Description	Argument	Example
deselect_all()	This clears all the selected entries from a multiselect dropdown or list	select_element. deselect_all()	
deselect_by_ index(index)	This deselects the option at the given index from the dropdown or list	index is the index of the option to be deselected	select_element. deselect_by_ index(1)
deselect_by_ value(value)	This deselects all options that have a value matching the argument from the dropdown or list	value is the value attribute of the option to be deselected	select_element. deselect_by_ value("foo")
deselect_ by_visible_ text(text)	This deselects all the options that display text	text is the text value of	select_element. deselect_

	matching the argument from the dropdown or list	the option to be deselected	by_visible_ text("bar")
select_by_ index(index)	This selects an option at the given index from the dropdown or list	index is the index of the option to be selected	select_element. select_by_ index(1)
select_by_ value(value)	This selects all the options that have a value matching the argument from the dropdown or list	value is the value attribute of the option to be selected	select_element. select_by_ value("foo")
select_by_ visible_ text(text)	This selects all the options that display the text matching the argument from the dropdown or list	text is the text value of the option to be selected	select_element. select_by_visible_ text("bar")

Example of Select Class Used in Code –

Select = Select(driver.find_element_by_xpath('/html/body/div[2]/section/div[4]/ui-view/div[2]/div[2]/div/ui-view/div[3]/div/ui-view/div[1]/div[1]/div/form/span[1]/select'))

select.select_by_visible_text("Invoice (CSV)")

To Run the Complete Automation Process in Background –

PhantomJS is a headless webkit which can be used in conjunction with Selenium Webdriver for headless testing and automation task. To implement this following links are pretty nice and helpful.

https://stackoverflow.com/questions/13287490/is-there-a-way-to-use-phantomjs-in-python

https://stackoverflow.com/questions/14699718/how-do-i-set-a-proxy-for-phantomjs-ghostdriver-in-python-webdriver/15699530#15699530

https://dzone.com/articles/python-testing-phantomjs

https://realpython.com/blog/python/headless-selenium-testing-with-python-and-phantomjs/

Incrontab Tool –

Incron is similar to cron, but instead of running commands based on time, it can trigger commands when file or directory events occur (e.g. a file modification, changes of permissions, etc.)

The format of the *incrontab* looks like:

```
<path><mask><command>
```

<path> — This is the path to the directory you want to watch. Do note the Incron is not capable of watching subdirectories. Only files within the path will be monitored. If you need subdirectories monitored, you must give them their own entry.

<mask> — This is one of several options:

IN_ACCESS	File was accessed (read)
IN_ATTRIB	Metadata changed (permissions, timestamps, extended attributes, etc.)
IN_CLOSE_WRITE	File opened for writing was closed
IN_CLOSE_NOWRITE	File not opened for writing was closed
IN_CREATE	File/directory created in watched directory
IN_DELETE	File/directory deleted from watched directory
IN_DELETE_SELF	Watched file/directory was itself deleted
IN_MODIFY	File was modified
IN_MOVE_SELF	Watched file/directory was itself moved
IN_MOVED_FROM	File moved out of watched directory
IN_MOVED_TO	File moved into watched directory
IN_OPEN	File was opened

<command> — This is the command that will run should an event be triggered. In place of a command, you can always use wildcards. The wildcards will report basic information in syslog. The available wildcards are:

$$-----Prints a dollar sign
$@ ---Add the watched filesystem path
$#-----Add the event-related file name
$% ---Add the event flags (textually)
$&----Add the event flags (numerically)

This link is helpful - https://www.howtoforge.com/tutorial/trigger-commands-on-file-or-directory-changes-with-incron/

PYTHON SCRIPT TO QUERY A POSTGRESQL DATABASE

Prepared the script that will work along with Incrontab tool, python script list all the files(csv) present in the folder, applies queries to that file (queries are fixed for each file) and feed it into the database. All the work is dynamic here. And if somehow query do not get executes then it will transfer that file to folder named "Success" otherwise folder named "failure".
Here psycopg2 is used which is Postgresql database adapter for the python programming language. This link - http://initd.org/psycopg/docs/ is useful to understand it. In this python script I prepared 3 different functions –

1. To append data in database table.
2. To insert the data in database table.
3. To update the database table.

psycopg2.sql module is also used which contains objects and functions useful to generate SQL dynamically.
In database there is aleready one table which contains-

File_name	Table_name	Action_name
f1	t1	a1
f2	t2	a2
f3	t3	a3

Python script will list the files present in the directory, is the file_name is ‘f1’ then it will apply action ‘a1’ which can be (Insert, Append or Update) on the table ‘t1’.

```
# Code for connecting to the database and path of directory, Success_folder, Failure_Folder –

conn = psycopg2.connect(database="dwh_20170612", user = "admin", password = "xyz", host =
"127.0.0.1", port = "5555")

path = "/home/localserver/ftp/file"

path1 = "/home/localserver/ftp/Success"

path2 = "/home/localserver/ftp/Failure"

cur = conn.cursor()

# List all the files present in directory and connecting with table to apply action using these files. Here
mst_file_action is table name.

files= os.listdir(path)

cur.execute("SELECT * FROM mst_file_action;")

records = cur.fetchall()

# Function to append the data in table –

def  appendi(file_name, table_name, compare_key):

        file = path+"/"+file_name

        file_success = path1 + "/" + file_name

        file_failure = path2 + "/" + file_name
# Another connection is created for child process.
        conn_tmp = psycopg2.connect(database="dwh_20170612", user = "admin", password = "xyz",
    host = "127.0.0.1", port = "5555")

        cur_tmp = conn_tmp.cursor()
```

```
            try:

# Creating temporary table in which the fields are exactly same as database table.

            cur_tmp.execute(sql.SQL("CREATE TEMP TABLE tmp_t2 (like {}
        );").format(sql.Identifier(table_name)))

            conn_tmp.commit()

            table_name_1 = 'tmp_t2'

# Copying the content of file to temporary table.

            cur_tmp.execute(sql.SQL("COPY {} FROM %s DELIMITER ',' CSV
        HEADER;").format(sql.Identifier(table_name_1)),(file,))

            conn_tmp.commit()

# Appending all that data which is not in the database table from temporary table.

            cur_tmp.execute(sql.SQL("INSERT INTO {0} SELECT * FROM {1} WHERE NOT EXISTS (SELECT 1
            FROM {0} WHERE
            {1}.{2}={0}.{2});").format(sql.Identifier(table_name),sql.Identifier(table_name_1),sql.Identifier(
            compare_key)))

            conn_tmp.commit()

            except Exception as e:

# If somehow the above process unable to get execute then do the same process again by making
another connection.
                        try:

# Again the same procedure but with different connection.

                        except:

# Moving the file to failure_folder.

                                    os.rename(file,file_failure)

            conn_tmp.close()
```

Now the second function to insert -

```
def inserti(file_name, table_name):

        file = path+"/"+file_name

        file_success = path1 + "/" + file_name

        file_failure = path2 + "/" + file_name

#   Another connection is created for child process.

        conn_tmp = psycopg2.connect(database="dwh_20170612", user = "admin", password = “xyz",
    host = "127.0.0.1", port = "5555")

        cur_tmp = conn_tmp.cursor()

#  Truncating the table in database to insert the data into it.

        cur_tmp.execute(sql.SQL("truncate {};").format(sql.Identifier(table_name)))

        conn_tmp.commit()

        try:
# Copy the file to the table in database.

        cur_tmp.execute(sql.SQL("COPY { } FROM %s DELIMITER ',' CSV
    HEADER;").format(sql.Identifier(table_name)),(file,))

        conn_tmp.commit()

# Transferring the file to Success folder.

        os.rename(file,file_success)

        except Exception as e:

# Executing the same above process by making different connection.

                try:

#               Again the same procedure with different connection.

                except:

# If unable to execute the process then transferring the file to failure folder.
                        os.rename(file,file_failure)
```

```
        conn_tmp.close()
```

Function to update the data in table –

```
def updatei(file_name,table_name,compare_key):

        file = path+"/"+file_name

        file_success = path1 + "/" + file_name

        file_failure = path2 + "/" + file_name

#   Creating connection for child process.

        conn_tmp = psycopg2.connect(database="dwh_20170612", user = "admin", password = "xyz",
    host = "127.0.0.1", port = "5555")

        cur_tmp = conn_tmp.cursor()

        try:

# Creating the temporary table exactly same as the permanent table and copyting the file in it.

        cur_tmp.execute(sql.SQL("CREATE TEMP TABLE tmp_t2 (like { }
    );").format(sql.Identifier(table_name)))

        table_name_1 = 'tmp_t2'

        cur_tmp.execute(sql.SQL("COPY { } FROM  %s  DELIMITER ',' CSV
    HEADER;").format(sql.Identifier(table_name_1)),(file,))

        conn_tmp.commit()

        cur_tmp.execute("SELECT * FROM tmp_t2;")

#List having all the column name present in the table of database excluding the column of compare_key.

        colnames = []

        for desc in cur_tmp.description:

        if desc[0] != str(compare_key):

                colnames.append(desc[0])
```

```
# Executing query separately for each column.

        for i in colnames:

                cur_tmp.execute(sql.SQL('''UPDATE {0}
                        SET {1} = tmp_t2.{1}
                        FROM tmp_t2
                        WHERE {2}.{3} = tmp_t2.{3}
                        ;''').format(sql.Identifier(table_name),sql.Identifier(i),sql.Identifier(table_
                        name),sql.Identifier(compare_key)))

        conn_tmp.commit()

# Moving the file to success folder.

        os.rename(file,file_success)

        except:

                try:

# Again same procedure but with different connection.

                except:

# Moving the file to failure folder if the process does not execute.

                        os.rename(file,file_failure)

        conn_tmp.close()
```