Test Documentation

Author: Bartosz Koperek

Contents

Overview	3
Used tools and languages	3
Technical details	3
Assumptions	3
Coding standard	3
Files structure	4
Test.robot	4
Configuration file	4
Exceptions	4
Dealing with timeouts	5
Installation guide	6
Test execution	7
Reports	7

Overview

This test proves that the payment is declined with an invalid payment input.

Used tools and languages

The test is developed in python, selenium and robot framework. Robot Framework is a generic test automation framework for acceptance testing with support for BDD - Gherkin syntax, providing extensive and clear logs and reports.

IDE: Pycharm with IntelliBot plugin.

Technical details

Page object pattern was used. All pages are represented as separate classes and inherit from the BasePage class. The BasePage contains common methods for finding elements and operating on them. All selenium locators occurring on a certain page are stored in correlated class so that when developers change something in the source code of the website, a correction for the test has to be applied only in one class.

Assumptions

- One way flight option is always chosen.
- It is possible to provide only the number of adults and teens as passengers.
- Passengers' details like first and last name are generated automatically as random strings.
- Flight date format: DD/MM/YYYY.
- Card expiration date format: MM/YYYY.
- All other strings required for payment and not passed as parameters are stored in configuration file.

Coding standard

- Following PEP 8 python standard.
- File names lower case with underscore as a separator.
- Class names camel case.
- Function names lower case with underscore as a separator.
- Descriptive function names so that a name reflects action on a website.
- No implicit time sleeps.
- All locators are stored in correlated classes as constant variables. They are upper case with prefix identifying tag type and suffix identifying locating method:
 - TAGTYPE_NAME_LOCATINGMETHOD_LOCATOR, example: INPUT_ONEWAY_ID_LOCATOR

Files structure

```
--Test
-- test.robot
-- config
-- __init__.py
-- configfile.py
-- Resources
-- __init__.py
-- decorator_control.py
-- exceptions.py
-- page_base.py
-- page_base.py
-- page_go_to_checkout.py
-- page_home.py
-- page_payment.py
-- verify_payment.py
```

All files are stored in Test folder with 2 subdirectories: config containing the test configuration file and Resources containing all python libraries.

Test.robot

The acceptance level test is written in robot in Gherkin style.

Example test input:

```
Given I make a booking from 'Bologna' to 'Bucharest' on '16/04/2016' for '2' adults and '1' child When I pay for booking with card details '5555 5555 5557', '10/2018' and '265' Then I should get payment declined message
```

The PaymentVerifier class is an interface between Robot framework and Python. It triggers all selenium actions and checks if a payment declined message was found. It is imported in test.robot file in Settings section:

```
*** Settings ***
Library verify_payment.Payment_Verifier
```

Configuration file

All parameters like url, timeouts and default strings are stored in a separate file in config\configfile.py

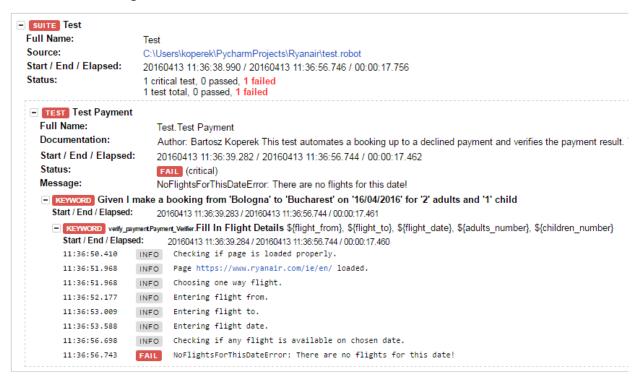
Exceptions

Custom exceptions are defined in file exceptions.py. They are used to provide a clear output to a test owner when a test fails. The output should indicate what actually happened.

Example test input with a date without any flight from Bologna to Bucharest:

```
Given I make a booking from 'Bologna' to 'Bucharest' on '16/04/2016' for '2' adults and '1' child When I pay for booking with card details '5555 5555 5557', '10/2018' and '265' Then I should get payment declined message
```

Test Execution Log



Note: not all exceptions are handled due to a limited time, however this approach can be used to deal with situations where incorrect credit card number, credit card expiration date, email etc. are provided.

Dealing with timeouts

No implicit waits were used as they can drive a test to fail in a certain conditions. Instead WebDriverWait method was used.

```
Example (waiting for an element with a timeout until it appears on a page):
```

```
element = WebDriverWait(self.driver,
timeout).until(EC.presence_of_element_located((By.ID,locator)))
```

Another important method where the WebDriverWait was used is wait_for_change in the BasePage class. This method deals with dynamic objects like ajax widgets and protects the test from failing when an element did not manage to load.

```
source = self.driver.page_source
def compare_source(driver):
    try:
        return source != driver.page_source
    except WebDriverException:
        pass
WebDriverWait(self.driver, timeout).until(compare source)
```

Installation guide

For Windows:

- Install python 2.7.6:
 - o https://www.python.org/download/releases/2.7.6/
- Install setup tools:
 - o Download and execute below file:
 - o https://bootstrap.pypa.io/ez_setup.py

```
Installed c:\python27\lib\site-packages\setuptools-20.7.0-py2.7.egg
Processing dependencies for setuptools==20.7.0
Finished processing dependencies for setuptools==20.7.0
```

- Install pip:
 - https://pypi.python.org/pypi/pip

```
Installed c:\python27\lib\site-packages\pip-8.1.1-py2.7.egg
Processing dependencies for pip==8.1.1
Finished processing dependencies for pip==8.1.1
```

- Install selenium:
 - pip install selenium

```
D:\userdata\koperek\Downloads\python\pip-8.1.1.tar\pip-8.1.1>pip install selenium
D:\userdata\koperek\Downloads\python\pip-8.1.1.tar\pip-8.1.1>pip install selenium

Collecting selenium

C:\Python27\lib\site-packages\pip-8.1.1-py2.7.egg\pip\_vendor\requests\packages\urllib3
ing: An HTTPS request has been made, but the SNI (Subject Name Indication) extension to
tform. This may cause the server to present an incorrect TLS certificate, which can cau
information, see https://urllib3.readthedocs.org/en/latest/security.html#snimissingwarn
SNIMissingWarning

C:\Python27\lib\site-packages\pip-8.1.1-py2.7.egg\pip\_vendor\requests\packages\urllib3
rmWarning: A true SSLContext object is not available. This prevents urllib3 from config
ause certain SSL connections to fail. For more information, see https://urllib3.readthe
insecureplatformwarning.
    InsecurePlatformWarning
    Using cached selenium-2.53.1-py2-none-any.whl
Installing collected packages: selenium
Successfully installed selenium-2.53.1
```

- Install Robot framework:
 - pip install robotframework

```
D:\userdata\koperek\Downloads\python\pip-8.1.1.tar\pip-8.1.1\pip install robotframework
Collecting robotframework
C:\Python27\lib\site-packages\pip-8.1.1-py2.7.egg\pip\_vendor\requests\packages\urllib3\uti
ing: An HTPS request has been made, but the SNI (Subject Name Indication) extension to ILS
tform. This may cause the server to present an incorrect TLS certificate, which can cause v
information, see https://urllib3.readthedocs.org/en/latest/security.html#snimissingwarning.
SNIMissingWarning
C:\Python27\lib\site-packages\pip-8.1.1-py2.7.egg\pip\_vendor\requests\packages\urllib3\uti
rmWarning: A true SSLContext object is not available. This prevents urllib3 from configurin
ause certain SSL connections to fail. For more information, see https://urllib3.readthedocs
insecureplatformwarning.
InsecurePlatformWarning
Using cached robotframework-3.0.tar.gz
Installing collected packages: robotframework
Running setup.py install for robotframework
Running setup.py install for robotframework
Successfully installed robotframework-3.0
```

For Mac OS:

- Install python 2.7.6:
 - o https://www.python.org/download/releases/2.7.6/
- Install selenium:
 - o sudo easy install selenium
- Install Robot framework:
 - https://mitnk.com/wiki/2011/10/install robot framework on mac os x/

Make sure that python is added to the PATH environment variable.

Test execution

To run the test execute the pybot command in a directory where test.robot file is located: pybot --pythonpath Resources --loglevel INFO -- test.robot or for more detailed logs: pybot --pythonpath Resources --loglevel DEBUG -- test.robot

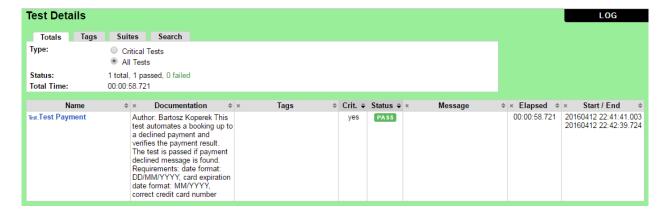
Example console output:

Reports

Pybot automatically generates three files which can be used for reporting:

- log.html the test result details,
- report.html the test result summary,
- output.xml the test results in a portable XML format for integration with other tools.

The report.html file contains all executed tests results with basic info like Test Name, Elapsed time, Start / End date:



It also contains a link to log.html file with more detailed info about the test execution like Test Statistics and Test Execution Log. The execution log is very helpful, especially for debugging. All keywords can be expanded to show python logs.

Test Statistics

	Total Statistics	\$ Total \$	Pass ≑	Fail \$	Elapsed \$	Pass / Fail
Critical Tests		1	1	0	00:00:59	
All Tests		1	1	0	00:00:59	
	Statistics by Tag	\$ Total	Pass \$	Fail	Elapsed \$	Pass / Fail
No Tags						
	Statistics by Suite	\$ Total	Pass ♦	Fail	Elapsed \$	Pass / Fail
Test		1	1	0	00:00:59	

Test Execution Log

