Test

Documentation

Author: Bartosz Koperek

Contents

[Overview 3](#_Toc448347048)

[Used tools and languages 3](#_Toc448347049)

[Technical details 3](#_Toc448347050)

[Assumptions 3](#_Toc448347051)

[Coding standard 3](#_Toc448347052)

[Files structure 4](#_Toc448347053)

[Test.robot 4](#_Toc448347054)

[Configuration file 4](#_Toc448347055)

[Exceptions 4](#_Toc448347056)

[Dealing with timeouts 5](#_Toc448347057)

[Installation guide 6](#_Toc448347058)

[Test execution 7](#_Toc448347059)

[Reports 7](#_Toc448347060)

# 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\_choose\_flight.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 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

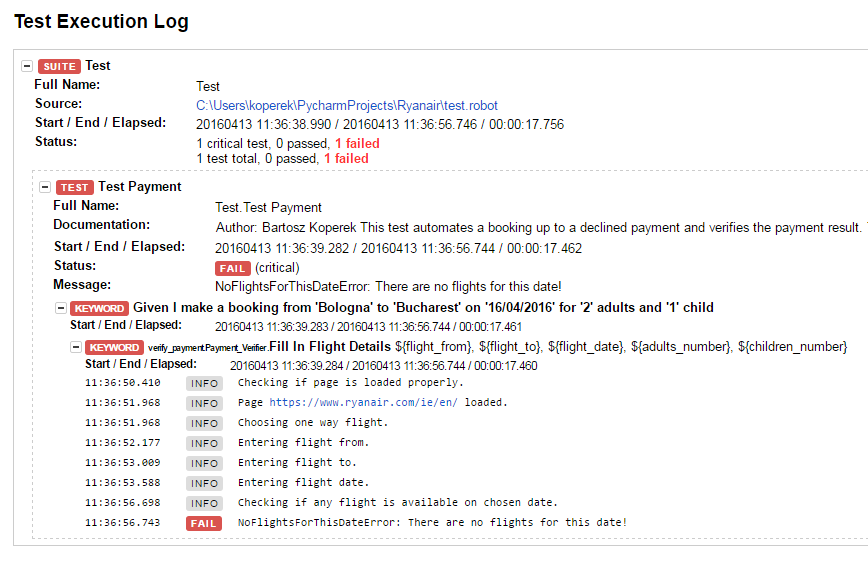
All parameters like url, timeouts and default strings are stored in a separate file in config\configfile.py

## 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 5555 5557', '10/2018' and '265'  
Then I should get payment declined message



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:

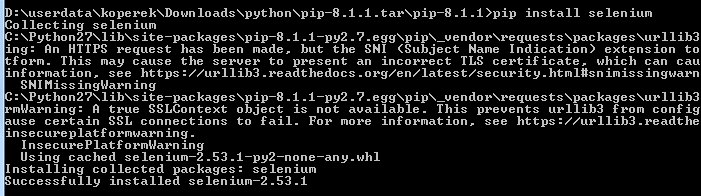
* Install python 2.7.6:
  + <https://www.python.org/download/releases/2.7.6/>
* Install setup tools:
  + Download and execute below file:
  + <https://bootstrap.pypa.io/ez_setup.py>



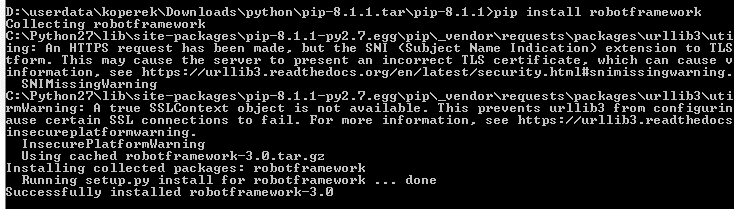
* Install pip:
  + <https://pypi.python.org/pypi/pip>



* Install selenium:
  + pip install selenium



* Install Robot framework:
  + pip install robotframework



For Mac OS:

* Install python 2.7.6:
  + <https://www.python.org/download/releases/2.7.6/>
* Install selenium:
  + 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

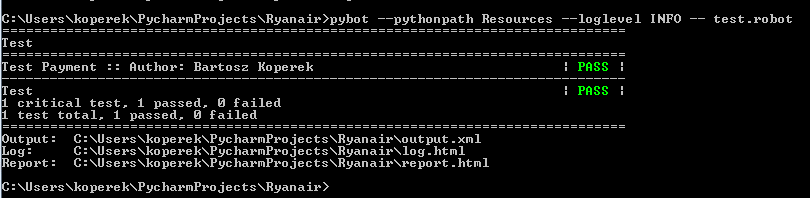
To run the test execute the pybot command in a directory where test.robot file is located:

pybot --pythonpath Resources --loglevel INFO -- test.robot

or for more detailed logs:

pybot --pythonpath Resources --loglevel DEBUG -- test.robot

Example console output:

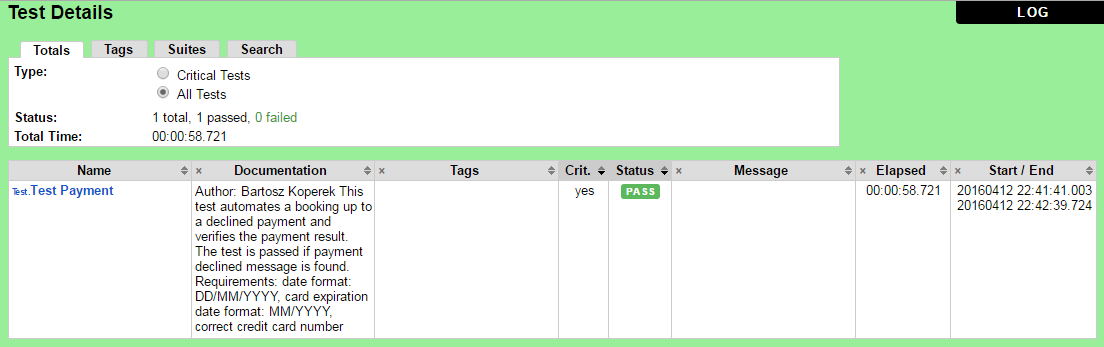


# Reports

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.

