Aplikacje WWW - lab 11

Jan Wróblewski

19 maja 2015

Selenium

Selenium to framework do automatyzacji czynności wykonywanych w przeglądarce internetowej. Strona WWW: `http://www.seleniumhq.org`.

Jest w stanie automatyzować wyszukiwanie elementów, wpisywanie znaków w formularz, klikanie na guziki i inne. Wszystko to dzieje się w prawdziwej przeglądarce z jej natywną implementacją JavaScript i innych technologii.

Wspierane są najpopularniejsze przeglądarki, m. in. Firefox, Chrome, IE, Safari, Opera. Możemy sobie wybrać na czym testujemy.

Selenium

Jest kilka produktów Selenium:

- Selenium WebDriver - służy do automatycznego wykonywania czynności w przeglądarce za pomocą skryptu w języku programowania Python, Java, C#, Ruby, Perl lub PHP;
- Selenium IDE - to add-on do Firefoxa, który nagrywa makra z czynności które ręcznie wykonujemy w przeglądarce, a potem odtwarza je;
- Selenium Grid - gdybyśmy chcieli odpalić testy na wielu serwerach z różnymi konfiguracjami;
- Selenium Remote Control - kontrolowanie przeglądarki uruchomionej na innym serwerze lub lokalnie jako aplikacja klient-serwer.

My będziemy używać Selenium WebDrivera do testów automatycznych.

Selenium przez Pythona

Na początek musimy zainstalować selenium. Na studentsie go nie ma, więc musimy zrobić to w virtualenvie.

```
$ virtualenv ve
$ source ve/bin/activate
(ve)$ pip install selenium
```

Tu jest dobry tutorial/dokumentacja na początek:
`http://selenium-python.readthedocs.org`.

Aplikacja do testowania

Przetestujmy AJAXową aplikację z aukcjami z ostatnich zajęć: `http://www.mimuw.edu.pl/~xi/download/www/auctions2.zip`. Powinna działać na studentsie (Python 2.7, Django 1.7.7). Uruchamiamy ją np. na `localhost:8000`.

Zrobimy dwa testy:

- Sprawdzimy czy poprawnie podbijana jest cena po wpisaniu większej ceny niż obecna i kliknięciu przycisku;
- Sprawdzimy czy poprawnie **nie** jest podbijana cena po wpisaniu mniejszej ceny i czy wyświetla się przy tym jakiś alert.

Test case 1

```
# -*- encoding=utf-8 -*-

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions

driver = webdriver.Firefox()
driver.get("http://localhost:8000")
assert u"Aukcja U" in driver.title
elem = driver.find_element_by_css_selector("span#current-price")
currentPrice = elem.text
newPrice = str(int(currentPrice) + 1)
elem = driver.find_element_by_css_selector("input#price")
elem.send_keys(newPrice)
elem = driver.find_element_by_css_selector("button#bid")
elem.click()
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.text_to_be_present_in_element((By.ID, 'current-price'), newPrice))
driver.close()
```

Prześledźmy wykonanie tego skryptu po kolei.

Test case 1

Otwieranie przeglądarki

```
driver = webdriver.Firefox()
```

Stworzenie obiektu drivera otwiera przeglądarkę podłączoną pod selenium. W tym przypadku otwierany jest Firefox i otwierane jest jego okno.

Otwieranie strony w przeglądarce

```
driver.get("http://localhost:8000")
```

To otwiera konkretną stronę w przeglądarce.

Test case 1

Asercja na tytule

```
assert u"Aukcja U" in driver.title
```

To pierwsza część testów. Sprawdzamy czy tytuł zawiera podstring "Aukcja U". Możemy dać też "Używany kapeć", by sprawdzić czy dobrze pobieramy dane z bazy danych. W Pythonie 2 string musimy poprzedzić literką `u`, aby był w Unicode i przyjmował polskie znaki. Musimy też ustawić kodowanie na utf-8.
Asercje rzucą wyjątkiem, jeżeli wartość logiczna w nich podana będzie `False`.

Test case 1

Wybór elementu i jego zawartości tekstowej

```
elem = driver.find_element_by_css_selector(
    "span#current-price")
currentPrice = int(elem.text)
newPrice = str(currentPrice + 1)
```

Znajdujemy element z obecną ceną za pomocą selektora CSS i pobieramy z niego obecną wartość tekstową. Jeżeli element nie zostanie znaleziony to będzie rzucony wyjątek. Konwertujemy cenę na inta i obliczamy podbitą cenę.

Test case 1

Wpisywanie znaków w input

```
elem = driver.find_element_by_css_selector("input#price")
elem.send_keys(newPrice)
```

Znajdujemy pole do wpisania nowej ceny i symulujemy wpisywanie znaków.

Klikanie w guzik

```
elem = driver.find_element_by_css_selector("button#bid")
elem.click()
```

Znajdujemy guzik do wysłania nowej ceny i symulujemy kliknięcie. W tym momencie przeglądarka zaczyna wykonywać asynchronicznie żądanie AJAX.

Test case 1

Czekanie na zmianę tekstu

```
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.
    text_to_be_present_in_element(
    (By.ID, 'current-price'), newPrice))
```

Potrzebujemy mechanizmu do poczekania na wykonanie się AJAXa lub innej akcji - waita. Czekamy do 10 sekund aż tekst w elemencie o id `current-price` zmieni swoją wartość na nową cenę. W przeciwnym razie rzucamy wyjątek. Jest wiele akcji, na które możemy czekać - patrz dokumentacja.

Test case 1

Zamykanie przeglądarki w przypadku sukcesu

```
driver.close()
```

W ten sposób zamykamy przeglądarkę. Jeżeli po drodze wystąpi wyjątek to przeglądarka pozostanie otwarta.

Zamykanie przeglądarki zawsze

```
try:
    # skrypt testujacy
finally:
    driver.close()
```

W ten sposób zamykamy przeglądarkę po wykonaniu się testów, nawet jeżeli wystąpi wyjątek.

Test case 2

Teraz sprawdzamy czy po wpisaniu za niskiej ceny mamy błąd i brak zmiany ceny.

```
# -*- encoding=utf-8 -*-

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.alert import Alert
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions

driver = webdriver.Firefox()
driver.get("http://localhost:8000")
elem = driver.find_element_by_css_selector("span#current-price")
currentPrice = elem.text
newPrice = str(int(currentPrice) - 1)
elem = driver.find_element_by_css_selector("input#price")
elem.send_keys(newPrice)
elem = driver.find_element_by_css_selector("button#bid")
elem.click()
wait = WebDriverWait(driver, 10)
wait.until(expected_conditions.alert_is_present())
Alert(driver).dismiss()
elem = driver.find_element_by_css_selector("span#current-price")
assert currentPrice == elem.text
driver.close()
```

Test case 2

Czekanie na alerta

```
wait.until(expected_conditions.alert_is_present())
```

W ten sposób czekamy na pojawienie się alerta.

Zamykanie alerta

```
Alert(driver).dismiss()
```

W ten sposób zamykamy alerta. Gdyby alert zawierał opcję tak/nie, moglibyśmy również użyć `accept()`/`dismiss()` odpowiednio.

Materiały

Selenium:
Dobra nieoficjalna dokumentacja-tutorial:
`http://selenium-python.readthedocs.org`.
Pełna dokumentacja dla Pythona: `http://selenium.googlecode.com/svn/trunk/docs/api/py/index.html`.
Pełna dokumentacja narzędzia: `http://www.seleniumhq.org/docs`.

Extra:
Unit testy w Django: `https://realpython.com/blog/python/django-1-6-test-driven-development/`.
Łączenie selenium z testami automatycznymi Django:
`http://agiliq.com/blog/2014/09/selenium-testing/`.