Selenium

- wprowadzenie do testów automatycznych

Marcin Stanilewicz

29.08.2019

Dlaczego zostałem testerem?

...bo BUG tak chciał ;)

Co wyszczególnia testy automatyczne?

- Szybkość działania - np. analiza elementów GUI w wybranej zakładce.
- Spójność i powtarzalność - przejście scenariusza testowego krok po kroku zawsze w tym samym określonym czasie.
- Reużywalność i możliwość udostępniania.
- Cykliczność - duża częstotliwość wykonywania przez 24h/dobę (w nocy itp.)
- Testy regresyjne.

Moda na automatyzację testów w firmach (1)

- Wynik analizy zysków i strat - jeżeli firmę stać na stworzenie dodatkowego działu testerów, którzy będą się zajmować testami automatycznymi w celu utrzymania na najwyższym poziomie jakości produktu, nie spowoduje strat. Dodatkowo przy cyklicznym powiększaniu działu deweloperskiego może przynieść zyski np. przyspieszając czas budowy produktu.
- Bo automaty są modne - wszyscy mają to trzeba wprowadzić. Implementacja testów automatycznych bez wyraźnego celu, bez uprzedniej analizy technologicznej produktu oraz współpracujących z nim narzędzi do automatyzacji testów niestety nie przyczyni się do poprawy jakości a tym bardziej oszczędności - przeciwnie, wygeneruje koszty, które nigdy się nie zwrócą, np. stworzenie odrębnego działu testerów automatycznych.

Moda na automatyzację testów w firmach (2)

- Regresja - celem testów automatycznych jest szybka informacja zwrotna czy stare funkcjonalności wciąż działają w danej wersji aplikacji. Znajdowanie i usuwanie błędów regresyjnych już na wczesnym etapie prac nad nową wersją oprogramowania przyspiesza wytwarzanie oprogramowania i utrzymanie jakości produktu na wysokim poziomie.
- Nie wszystkie obszary aplikacji da się przetestować manualnie np. ręczne wprowadzanie dużej ilości danych jest dużo wolniejsze i narażone na błędy. Dodatkowo automatyczne testy wydajnościowe pozwalają na symulację i przetestowanie środowisk użytkowników końcowych.

Cel automatyzacji:

(1)

- Cel musi być konkretny i możliwy do zrealizowania - testy automatyczne nie zastąpią wszystkich problemów z jakością produktu - nie tędy droga ;)
- Zadaniem testów automatycznych jest uzupełnienie testów manualnych. Automatyzacja nigdy nie zastąpi w pełni pracy testera manualnego. Pierwsze testy funkcjonalne wykonywane są ręcznie, a dopiero po ustabilizowaniu nowych funkcjonalności, testy się automatyzuje i dołącza do testów regresji.
- Celem testowania jest uzyskanie informacji o jakości nowej wersji oprogramowania. Testy automatyczne są w stanie dostarczyć takich danych znacznie szybciej niż ich manualne odpowiedniki, dzięki czemu programista ma szansę szybciej wprowadzić niezbędne poprawki do kodu.

Cel automatyzacji - kiedy automatyzacja się przydaje: (2)

- Testy wydajnościowe - pozwala na symulację aktywności dużej liczby użytkowników jest łatwiej osiągalne poprzez automatyzację.
- Testy obciążenia - automatyzacja pozwala na testowanie systemu w warunkach pracy z dużą ilością danych.
- Testy funkcjonalne - automatyzacja pozwala na symulację szybkich zmian kodu w krótkim czasie - dostarcza deweloperom natychmiastowych wyników z większego zakresu scenariuszy testowych.
- Automatyzowanie testów regresji - nie trzeba skupiać się na ich manualnym wykonywaniu, można ten czas przeznaczyć na testy nowej funkcjonalności.

Kiedy nie opłaca się wprowadzać testów automatycznych

- Niestabilność aplikacji - testy automatyczne są bardzo wrażliwe na wszelkie zmiany w aplikacji oraz destabilizację środowiska. Jeżeli istnieje takie zagrożenie, że aplikacja jest narażona na częste zmiany / poprawki istniejących funkcjonalności, to proces automatyzacji będzie się wiązać z raportowaniem dużej ilości zgłoszeń, które trzeba będzie analizować pod kątem potencjalnego błędu. Kolejny etap to weryfikacja i konsultacja z działem developerów, czy zaraportowany błąd jest wynikiem zmian w aplikacji i należy zaktualizować testy. Automatyzacja takiego przypadku nie ma sensu.
- Weryfikacja graficzna aplikacji, audio czy wydruki - w takich przypadkach cenniejsza jest opinia testera manualnego.
- Testy są rzadko wykonywane.

Od czego zacząć automatyzację?

- Automatyzację testów aplikacji warto uwzględnić już przy samej budowie aplikacji.
- Jeżeli wkraczamy w już istniejący projekt, należy wykonać rozpoznanie dotyczące technologii w której napisana jest aplikacja, na jakich przeglądarkach będą uruchamiane testy, jakich narzędzi będzie trzeba użyć do zbudowania środowiska testowego.
- Analiza kosztów wdrożenia testów automatycznych - w skład wchodzi przygotowanie i eksploatacja środowiska testowego, utrzymanie i rozwój skryptów testowych.

Selenium (1)

Selenium jest to framework służący do automatyzacji testów aplikacji webowych. Obsługuje najbardziej popularne przeglądarki:

- Chrome,
- Firefox.

Testy automatyczne w Selenium można pisać z wykorzystaniem popularnych języków programowania, m.in.:

- Python,
- Java.

Selenium (2)

Chcąc korzystać w pełni z możliwości automatyzowania, polecam wybrać Selenium WebDriver. Jednak, aby rozpocząć w nim pracę, należy znać podstawy programowania w języku wspieranym dla Selenium WebDriver.

Dla osób, które nie mają doświadczenia w programowaniu, można wykorzystać Selenium IDE, które jest wtyczką pod przeglądarkę np. Mozilla Firefox i działa na zasadzie „nagraj-i-odtwórz”.
Testowanie za pomocą narzędzia Selenium IDE można wykorzystać przy powtarzalnej weryfikacji formularzy itp. testów.

Pierwszy projekt z wykorzystaniem Selenium

Do napisania testu automatycznego dla aplikacji webowej wykorzystamy:

- Python.
- Selenium WebDriver - narzędzie, które pozwala na sterowanie przeglądarką z poziomu kodu. Dzięki temu testując stronę mamy pewność, że automatyczna integracja z przeglądarką będzie wiernie odzwierciedlać naturalne użycie.
- IDE PyCharm Community.
- Driver dla przeglądarki Chrome możemy pobrać z: https://sites.google.com/a/chromium.org/chromedriver/downloads

Po zainstalowaniu w/w narzędzi uruchamiamy PyCharm i tworzymy nowy projekt.

Path do WebDriver-a i instalacja pakietu Selenium

Pobrany WebDriver zapisujemy w miejscu, w którym dodaliśmy swój projekt, np.:

```
/Users/administrator/PycharmProjects/chromedriver
```

Gdy mamy ten krok za sobą, należy zainstalować paczkę selenium:

```
pip install selenium
```

PyCharm daje nam tą możliwość, że po wpisaniu w naszym skrypcie main.py:

```
from selenium
```

- automatycznie zainstaluje potrzebną paczkę [Alt+Enter].

Po zakończeniu instalacji z pakietu selenium importujemy moduł webdriver:

```
from selenium import webdriver
```

Uruchomienie przeglądarki za pomocą kodu

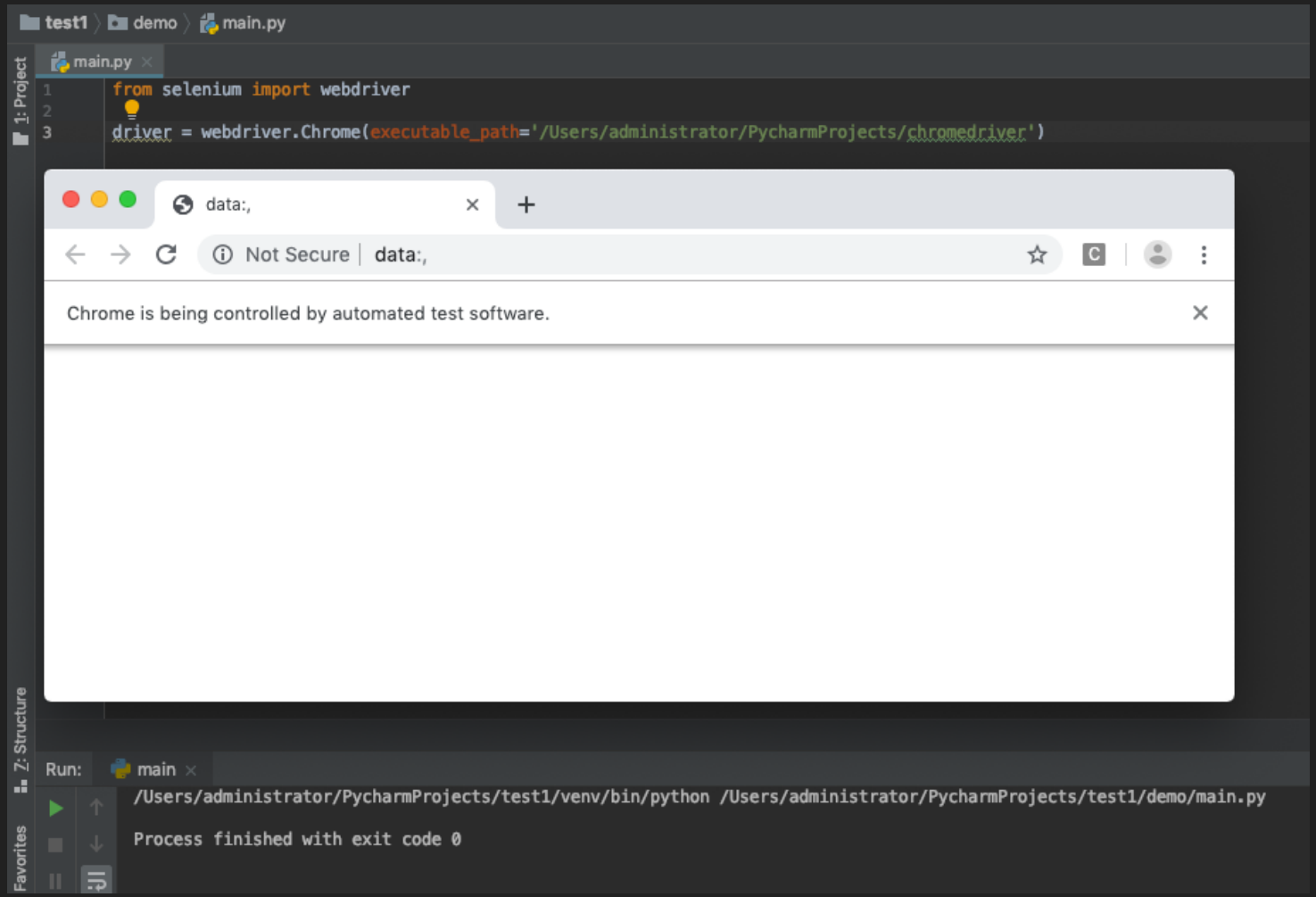
Aby mieć dostęp do modułu webdriver-a przypisujemy do zmiennej driver obiekt, który pozwoli sterować przeglądarką Chrome. Metoda Chrome() wykorzystuje parametr i jego wartość do sterowania przeglądarką:

```
driver = webdriver.Chrome(executable_path='/Users/administrator/PycharmProjects/chromedriver')
```

Efektem uruchomienia dwu-linijkowego skryptu jest otwarcie przeglądarki w nowym oknie. Dodatkowo w konsoli PyCharm powinien pojawić się wpis, e.g.:

```
/Users/administrator/PycharmProjects/test1/venv/bin/python
/Users/administrator/PycharmProjects/test1/demo/main.py

Process finished with exit code 0
```


test1
demo
main.py
main.py
1: Project
1 from selenium import webdriver
2
3 driver = webdriver.Chrome(executable_path='/Users/administrator/PycharmProjects/chromedriver')
data:,
Not Secure | data:,
Chrome is being controlled by automated test software.
Run:
main
/Users/administrator/PycharmProjects/test1/venv/bin/python /Users/administrator/PycharmProjects/test1/demo/main.py
Process finished with exit code 0
Z: Structure
Favorites

Kolejne metody driver-a czyli sterowanie przeglądarką internetową

Kolejnym krokiem jest przejście do wybranego adresu url strony testowej. Wywołujemy na naszej zmiennej driver metodę get(), która jako parametr przyjmuje adres strony, np.: https://www.nbp.pl

Dla wyjaśnienia: zmienna driver przechowuje w sobie reprezentację webdrivera, który posiada swoje metody - dzięki temu możemy je wywoływać używając takiej konstrukcji:

```
driver.get('https://www.nbp.pl')
```

Po uruchomieniu testu z podanym adresem url, przeglądarka powinna otworzyć żądaną stronę.

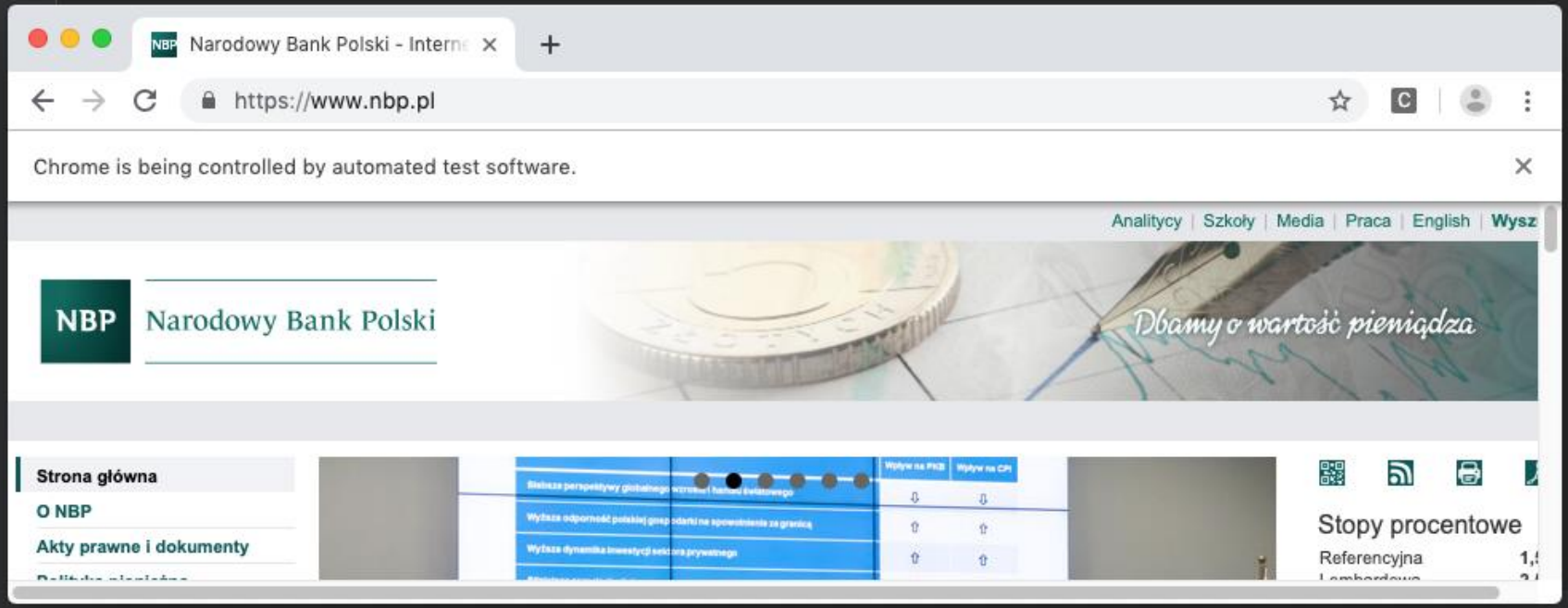
test1 › demo › main.py

main.py

```python
from selenium import webdriver

driver = webdriver.Chrome(executable_path='/Users/administrator/PycharmProjects/chromedriver')

driver.get('https://www.nbp.pl')
```



Run: main

```
/Users/administrator/PycharmProjects/test1/venv/bin/python /Users/administrator/PycharmProjects/test1/demo/main.py

Process finished with exit code 0
```

Pobranie tytułu testowanej strony

W celu sprawdzenia i wypisania w konsoli tytułu strony, na którą został napisany test automatyczny, tworzymy zmienną title. Do pobrania tytułu wykorzystamy kolejny atrybut driver-a:

```
title = driver.title
```

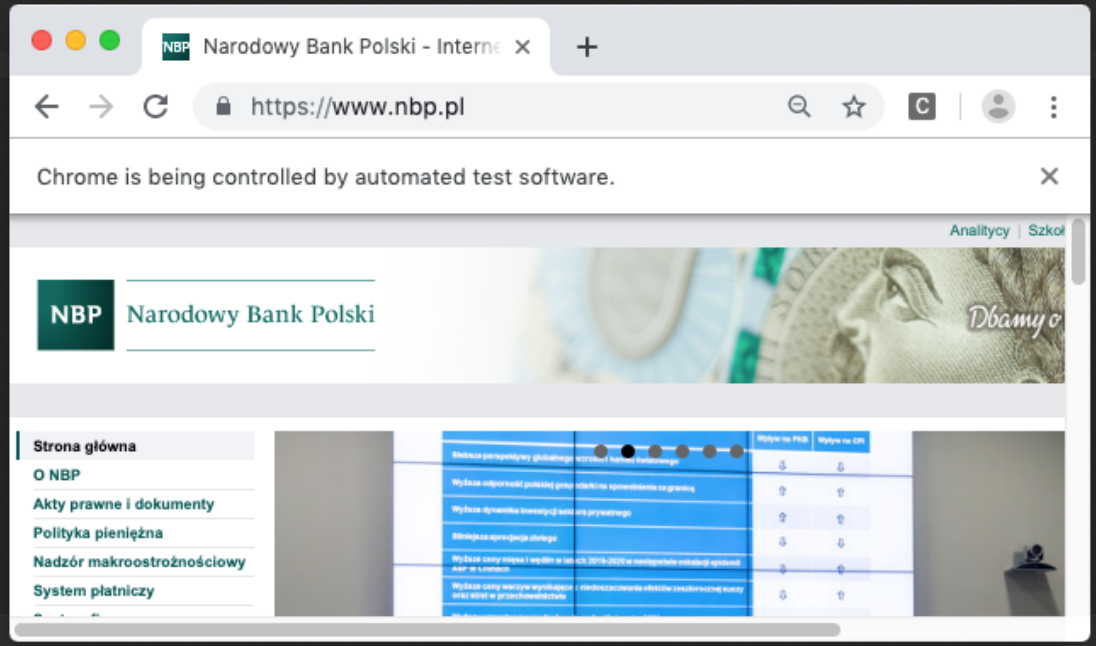
Samo pobranie tytułu nic nam nie wyświetli - musimy jeszcze wypisać wynik w konsoli. W tym celu wykorzystamy wbudowaną funkcję Pythona jaką jest print:

```
print(title)
```

test1 〉 demo 〉 main.py

main.py

```
1  from selenium import webdriver
2
3  driver = webdriver.Chrome(executable_path='/Users/administrator/PycharmProjects/chromedriver')
4
5  driver.get('https://www.nbp.pl')
6
7  title = driver.title
8
9  print(title)
```



Run: main

```
/Users/administrator/PycharmProjects/test1/venv/bin/python /Users/administrator/PycharmProjects/test1/demo/main.py
Narodowy Bank Polski – Internetowy Serwis Informacyjny

Process finished with exit code 0
```

Asercja - sprawdzenie oczekiwanego rezultatu

Asercja - instrukcja, która porównuje dwie zmienne.

Sprawdzenie czy w zmiennej title znajduje się oczekiwana treść:

```
assert title == 'Narodowy Bank Polski - Internetowy Serwis Informacyjny'
```

Ważne: podwójny znak == służy do porównania wartości

Jeżeli po zakończeniu testu nie pojawia się żaden błąd, a w konsoli wyświetli się poniżej wypisana informacja, oznacza to pozytywny wynik testu:

```
Process finished with exit code 0
```

AssertionError - test zakończony niepowodzeniem

W celu kontrolowanego wywołania błędu w teście, zmienimy wartość oczekiwaną tytułu strony poprzez dopisanie kilku znaków, np.:

```
assert title == 'Narodowy Bank Polski - Internetowy Serwis Informacyjny 2019'
```

Otrzymany wynik w konsoli:

```
Narodowy Bank Polski - Internetowy Serwis Informacyjny

Traceback (most recent call last):
  File "/Users/administrator/PycharmProjects/test1/demo/main.py", line 11, in <module>
    assert title == 'Narodowy Bank Polski - Internetowy Serwis Informacyjny 2019'
AssertionError

Process finished with exit code 1
```

Zamknięcie przeglądarki po wykonaniu testu

Do tej pory po każdym uruchomieniu testu, na pulpicie pozostawała otwarta przeglądarka. Aby usprawnić ten proces i za każdym razem posprzątać po teście, możemy użyć metody quit(). Metoda ta służy do automatycznego zamykania przeglądarki po wykonanym teście:

```
from selenium import webdriver

driver = webdriver.Chrome(executable_path='/Users/administrator/PycharmProjects/chromedriver')
driver.get('https://www.nbp.pl')
title = driver.title
print(title)
assert title == 'Narodowy Bank Polski - Internetowy Serwis Informacyjny'
driver.quit()
```

Dziękuję z uwagę ;)