



Magazine

Data-Driven Testing z Selenium

Autor: Jacek Okrojek

O autorze: absolwent Wydziału Fizyki Technicznej, Informatyki i Matematyki Stosowanej Politechniki Łódzkiej, specjalizacja Sieci i Systemy Teleinformatyczne, tester, test leader, freelance developer, od ponad 6 lat zajmuje się testowaniem i tworzeniem oprogramowania w kraju i za granicą w firmie Ericpol Telecom i jako niezależny konsultant, uczestniczył i nadzorował testy na poziomie podstawowym, funkcyjnym, integracyjnym i systemowym, prowadził szkolenia z zakresu testowania oprogramowania, kontakt: jacek.okrojek@gmail.com

Basic

Level

4

Magazine Number

Testowanie oprogramowania

Section in the magazine

Wprowadzenie

W poprzednim numerze wykorzystaliśmy Selenium do automatycznego raportowania błędów. Tym razem wykorzystamy je w typowy sposób, czyli do usprawnienia testowania aplikacji web. Przedstawione rozwiązanie pozwoli przybliżyć koncepcję Data-Driven Testing (DDT).

Automatyzacja testów a Data-Driven Testing

Wyobraźmy sobie aplikację, do której dostęp wymaga podania nazwy użytkownika i hasła. Użytkownikom przypisane są określone role a wraz z nimi zestaw dostępnych funkcji. Naszym zadaniem będzie przetestowanie modułu uwierzytelniającego użytkowników.

Selenium IDE pozwoli nagrywać wprowadzanie danych przez użytkownika i w ten sposób przygotować szkic scenariusza testowego. Test możemy wyeksportować do jednego z obsługiwanych języków programowania. Korzystając z techniki „Copy/Paste” powielimy go odpowiednią ilość razy. Modyfikując dane w kolejnych kopiach pokryjemy interesujące nas przypadki. Dodając asercje będziemy mogli sprawdzić, czy użytkownik dysponuje odpowiednimi funkcjami. Taką kolekcję testów będziemy mogli uruchamiać korzystając z Selenium RC. W tym momencie możemy pochwalić się już pewną automatyzacją testów. Oprócz mało prawdopodobnego uznania kolegów jest tylko jedna korzyść z takiego rozwiązania - przyspieszenie pracy, gdy przyjdzie nam testy powtórzyć.

Co stanie się, gdy pojawi się nowa grupa użytkowników lub powiększy zestaw dostępnych funkcji? Zmuszeni będziemy uaktualnić nasze testy. Wymagać to będzie odnalezienia odpowiednich fragmentów kodu i wprowadzenia potrzebnych zmian lub dodania nowych testów. Nasza praca byłaby prostsza, gdybyśmy dane testowe przechowywali w jednym miejscu. Scenariusz testu jest zawsze ten sam, więc sensowne wydaje się opracowanie jednej sparametryzowanej procedury i wykonywanie jej dla różnych zadanych danych. Taki model przeprowadzania testów nosi nazwę Data-Driven Testing (DDT). Główną cechą DDT jest odseparowanie danych oraz wyników testów od aplikacji/systemu przeprowadzającego testy. Aplikacja przeprowadzająca testy powinna być jak najbardziej uniwersalna i pozwalać się łatwo dostosowywać do aktualnych wymagań. Sposób przechowywania danych do testów jest dowolny. Może to być plik CSV, arkusz kalkulacyjny lub baza danych. Oprócz zalet, jakimi są przyspieszenie implementacji testów i łatwiejsze ich utrzymanie, logiczne odseparowanie danych od scenariusza testu pozwala na to, by dane mogły być przygotowane przez odrębnych ekspertów. Ma to olbrzymie znaczenie, kiedy przygotowanie danych wymaga specjalistycznej wiedzy. Tester może zająć się w tym czasie tym, co lubi najbardziej.

Uogólnienie problemu

Z punktu widzenia testerów aplikacji web, interesują nas przede wszystkim formularze. To dzięki nim użytkownik ma możliwość wprowadzania danych. W ogólnym przypadku testowanie przebiega według schematu:

- Otworzenie wybranej strony
- Wypełnienie pól formularza danymi i zaakceptowanie go
- Odpowiedź prezentowana w odpowiednich elementach strony porównywana jest z oczekiwanym rezultatem

Ponieważ chcemy, by rozwiązanie było jak najbardziej uniwersalne, postaramy się wiernie zaimplementować ten ogólny schemat. Jako źródło danych posłużą nam arkusz kalkulacyjny. Dane proponuję zorganizować jak w tabeli 1.

ID	Tytuł	Url	Uid	Hasło	Action	Result
1	Udane logowanie - administrator	/auth	admin	admin	submit	Witaj admin;Dodaj użytkownika
2	Udane logowanie - księgowi	/auth	ks	ks	submit	Witaj ks;Wyświetl faktury
3	Udane logowanie - sprzedawcy	/auth	sp	sp	submit	Witaj sp;Wyświetl kontrahentów
4	Nieudane logowanie	/auth	ks	sp	submit	Nieprawidłowe hasło

Tabela 1 Przykładowa organizacja danych

Dwie pierwsze kolumny mają charakter informacyjny i można z nich zrezygnować. Kolejna kolumna to adres strony, którą będziemy testować. Ważne jest, by ostatnia kolumna zawierała oczekiwany wynik, a przedostatnia akcje użytkownika. Kolumny pomiędzy adresem strony a akcją użytkownika zawierają dane wprowadzane do formularza na testowanej stronie. Nagłówki kolumny pozwolą na identyfikację pola formularza, do którego będą wpisywane dane. Przez dodanie lub usunięcie kolumny w arkuszu będziemy dostosowywali nasze rozwiązanie do formularzy z inną ilością pól. Usunięcie wszystkich kolumn pomiędzy URL i Action będziemy rozumieli jako stronę bez formularza, tylko z linkami do kolejnych stron. Za szkielet naszej procedury testowej posłuży kod zamieszczony w przykładzie 1.

```
sel = selenium(seleniumHost, seleniumPort, browserStartCommand, browserURL)
sel.start()
sel.open(...)
for k, v in actions.iteritems():
    sel.type(k, v)
sel.click(...)
sel.wait_for_page_to_load(timeout)
results = result.split(";")
for r in results:
    assert sel.is_text_present(...) == true
sel.stop()
```

Przykład 1. Szkielet procedury testowej

Po wystartowaniu przeglądarki przechodzimy na stronę określoną adresem w kolumnie URL arkusza. Następnie w pętli wpisujemy dane do kolejnych pól formularzy metodą type (locator, value). Wartości zmiennej locator będą pochodzić z nagłówków kolumn. Zmienna value będzie odpowiadającą jej wartością testową. Następnie zatwierdzamy formularz metodą click (locator) i czekamy na załadowanie się odpowiedzi. Ostatnią akcją jest sprawdzenie, czy na stronie występują teksty z kolumny Result. Teksty, których obecność chcemy sprawdzić separujemy znakiem „;”.

Pobieranie danych

Dane proponuję pobierać z arkuszy kalkulacyjnych. Przewagą tego rozwiązania nad CSV jest przede wszystkim możliwość czytelnej prezentacji i organizacji danych. W kolejnych zakładkach arkusza możemy umieszczać dane dla testów kolejnych stron. Pozwoli to zachować względny porządek, o który z biegiem czasu może być trudno. Dodatkowo będziemy mogli wykorzystać arkusz do wykonania prostych obliczeń przy testowaniu bardziej złożonych aplikacji.

Pobieranie danych z arkusza Excel zrealizujemy przy pomocy biblioteki xlrd. Poniżej znajduje się fragment skryptu odpowiadający za pobieranie danych. Po otwarciu wybranego pliku i wybraniu arkusza iteracyjnie odczytujemy dane z wszystkich wypełnionych komórek. Do oddzielnych list wpisujemy nagłówki kolumn i odpowiadające im dane.

```

headers = []
book = open_workbook(xlfilename)
sheet = book.sheet_by_index(xlsheetIndex)
cols, rows = sheet.ncols, sheet.nrows
data = [[None] * cols for i in range(rows-1)]
for row_index in range(rows):
    for col_index in range(cols):
        if row_index == 0:
            headers.append(sheet.cell(row_index, col_index).value)
        else:
            data[row_index-1][col_index] =
sheet.cell(row_index, col_index).value

```

Parametryzacja testów

Procedura testowa powinna być wykonywana dla kolejnych zestawów danych. Wykorzystanie biblioteki unittest i pętli ma jednak pewną poważną wadę. Pierwszy nieoczekiwany wynik przerwie cały proces testowania. Dopiero po analizie i poprawieniu błędu będziemy mogli wykonać dalsze przypadki.

Biblioteką, która pozwoli nam ominąć ten mankament i w prosty sposób sparametryzować testy, jest `py.test`. Pakiet instalacyjny i jego dokumentacje można pobrać z [2]. Wszystkie funkcje, których nazwa rozpoczyna się od `test_` są traktowane przez `py.test` jako procedury testowe i automatycznie uruchamiane. Proponuję zatem, aby szkielet z listingu 1. umieścić w funkcji `test_actions`.

Przygotowanie danych do testów standardowo następuje w funkcji `pytest_generate_tests`. W pierwszej jej części pobierzemy dane z arkusza jak na Listingu 2. Dalej połączymy każdy rząd danych z odpowiadającymi im nagłówkami w słownik (ang. dictionary). Wywołując dla każdego tak stworzonego słownika funkcję `metafunc.addcall(funcargs=dict(actions=d))` dodamy kolejny przypadek testowy. Przekazane dane będą dostępne w funkcji `test_actions` pod nazwą `actions`.

```

def pytest_generate_tests(metafunc):
    ...

    for r in range(rows-1):
        d = {}
        n = 2
        for i in headers[2:]:
            d[i] = data[r][n]
            n = n + 1
        metafunc.addcall(funcargs=dict(actions=d))

def test_actions(actions):
    ...

```

Podsumowanie

Pod adresem <http://coremag.eu/> można znaleźć kompletny, gotowy do wykorzystania skrypt. Dołączyłem do niego kod prostego formularza, na którym od razu możemy przeprowadzić testy. Wykorzystanie prezentowanego rozwiązania niesie ze sobą kilka ograniczeń dlatego zachęcam do modyfikacji i wprowadzania własnych poprawek. Celem skryptu jest przede wszystkim zaznajomienie czytelnika z techniką DDT oraz możliwościami Selenium. W kolejnym odcinku pokażę, w jaki sposób

dokładniej badać i analizować wyniki zwracane przez serwer www i testować bardziej skomplikowane aplikacje. Czekam również na Państwa sugestie i komentarze.

Referencje

[1] Xlrd <http://www.python-excel.org/>

[2] Py.test <http://codespeak.net/py/dist/test/index.html>