JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 100–106
Received: 8 March 2021
Accepted: 21 March 2021

Comparative analysis of frameworks used in automated testing on example of TestNG and WebdriverIO

Analiza porównawcza frameworków do automatyzacji testowania aplikacji webowych na przykładzie TestNG i WebdriverIO

Alla Shtokal*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The article presents a comparative analysis of frameworks supporting the development of automated tests for defined test scenarios. The comparative study concerned the TestNG and WebdriverIO frameworks. The overview of the tool has been analyzed both in terms of the test development process as well as the speed and efficiency of their execution. The website github.com was used for the purposes of the work. This application was used to run test scripts written in both frameworks. The results were compared by four defined criteria: the time of running the test scripts with a different maximum number of simultaneously running browser instances, the average time of running all test scripts in headless mode, the average value of memory and CPU usage during the test execution. The summary includes the evaluation of the compared frameworks.

Keywords: Selenium; WebdriverIO; TestNG; framework

Streszczenie
W artykule przedstawiona została analiza porównawcza frameworków wspomagających wytwarzanie testów zautomatyzowanych dla zdefiniowanych scenariuszy testowych. Badanie porównawcze dotyczyło frameworków TestNG oraz WebdriverIO. Omówienie narzędzia zostało przeanalizowane zarówno pod kątem procesu tworzenia testów, jak i szybkości oraz wydajności ich wykonywania. Na potrzeby pracy została wykorzystana strona internetowa github.com. Aplikacja ta posłużyła do przeprowadzania skryptów testowych napisanych w obu frameworkach. Wyniki zostały porównane przez cztery zdefiniowanych kryterium: całkowity czas uruchamiania zbiorów testowych z różną maksymalną liczbą jednocześnie uruchomionych instancji przeglądarki, średni czas uruchamiania wszystkich skryptów testowych w trybie headless, średnia wartość zużycia pamięci oraz CPU podczas wykonania testów. W podsumowaniu zawarta została ocena porównywanych frameworków.

Słowa kluczowe: Selenium; WebdriverIO; TestNG; framework

*Corresponding author
Email address: alla.shtokal@pollub.edu.pl (A. Shtokal)

©Published under Creative Common License (CC BY-SA v4.0)

1. Wstęp

Głównym celem automatyzacji testów jest obniżenie kosztów testowania programu po jego aktualizacji. Okresowo powtarzane kontrole tego samego typu zajmują dużo czasu w cyklu rozwoju. Automatyzacja skraca fazę testów i uwalnia główny zasób firmy - czas pracy specjalistów. Kolejną, nie mniej oczywistą zaletą takich testów jest podniesienie jakości testów, co gwarantuje niezawodność produktu. Przecież straty z tytułu wad ujawnionych dopiero na etapie eksploatacji przemysłowej mogą być bardzo duże. Niezadowolenie ich klientów jest na ogół trudne do oszacowania.

Testowanie ręcznie nie pozwala na kompleksowe testowanie w pełni funkcjonalnych systemów w wyznaczonym przez projekt czasie, co prowadzi do różnego rodzaju negatywnych konsekwencji, dlatego automatyzacja jest niezbędna. Przede wszystkim automatyzacja może poprawić niezawodność oprogramowania i zmniejszyć ryzyko wykrycia usterek na etapie eksploatacji przemysłowej. Lepsza dokładność testowania i możliwość wczesnego znajdowania większej liczby defektów. Możliwe staje się zidentyfikowanie i wyeliminowanie wąskich gardeł wydajności systemu w całym cyklu rozwojowym. Dzięki automatyzacji możesz zobaczyć dokładny obraz wydajności systemu na wszystkich poziomach, w tym podczas uruchamiania. W dużych projektach wybór odpowiedniego frameworku jest kluczowy. Framework musi odpowiadać technologii, na podstawie, której zbudowany jest interfejs graficzny. Na przykład Selenium [1] jest dobre do testowania aplikacji internetowych.

W ramach badania została analiza popularnych na rynku frameworków do automatyzacji testów aplikacji internetowych na przykładzie WebdriverIO oraz TestNG.

2. Przegląd literatury

Analizując problem badawczy można powiedzieć, że istnieje dość dużo badań dotyczących frameworków do automatyzacji testowania. W pracach, które zostały przeanalizowane często poruszanym tematem były zadania i oczekiwania stawiane korzystaniu frameworków do automatyzacji oraz ich użyteczności. W artykule „Test Automation Framework based on WEB” [2] au-

torstwa Fei Wang oraz Wencai Du opisana została zasada działania narzędzia Selenium, który służy do wspierania automatyzacji testów aplikacji internetowych.

W artykule autorzy zaprojektowali framework do automatycznego testowania oprogramowania dla aplikacji internetowej w oparciu o Selenium i JMeter. Do testowania została wybrana aplikacja webowa – translator. Wyniki pokazały, że stworzona struktura oprogramowania poprawia, jakość oprogramowania i zwiększa wydajność.

Kolejna praca, która została uwzględniona w przeglądzie to „Analiza porównawcza rozwiązań wykorzystywanych w testowaniu automatycznym" [3] autorstwa A. Wac oraz T. Watras, G. Kozieł. W tym artykule przedstawiona została analiza porównawcza narzędzi wspomagających wytwarzanie zautomatyzowanych testów. Autorzy zbadali każde narzędzie zarówno pod kątem procesu tworzenia testów, jak i prędkości ich wykonywania. Na podstawie przeprowadzonych badań autorzy podkreślili, że nie można wyznaczyć najlepszego rozwiązania do tworzenia zautomatyzowanych testów.

Podobne podejście prezentowane jest w prace „Comparison analysis of Android GUI testing frameworks by using an experimental study" [4] autorstwa Meiliana, Septian, I., Alianto, R. S. oraz Daniel. W tym badaniu zostały ocenione frameworki testowe systemu Android na podstawie dwóch rodzajów kryteriów. Te pierwsze, podobnie jak w poprzedniej pracy, są bardziej ogólne, kolejne zawierają wyłącznie szczegóły techniczne.

W ramach tego eksperymentalnego badania przez autorów była stworzona prosta aplikacja na Androida, która została używana do oceny wszystkich kryteriów czterech wybranych frameworków. Przez autorów zostały napisane testy do tej aplikacji z użyciem każdego frameworku, wyniki i kryteria zostały zaprezentowane w tabelce. W rezultacie preferowany frameworkiem okazał się Espresso.

Porównania w tych pracach, które zostały przeanalizowane skupiają się na starszych wersjach lub innych frameworkach niż wykorzystane w tej prace, dlatego podejmowanie tego tematu jest aktualne.

3. TestNG

TestNG to platforma testowa zaprojektowana w celu uproszczenia szerokiego zakresu potrzeb testowych, od testowania jednostkowego (testowanie klasy w izolacji od innych) do testów integracyjnych (testowanie całych systemów złożonych z kilku klas, kilku pakietów, a nawet kilku struktur zewnętrznych, takich jak serwery aplikacji). Framework jest przeznaczony i wykorzystany wyłącznie dla języka Java. Eliminując większość ograniczeń starszej platformy, TestNG daje programiście możliwość pisania bardziej elastycznych i wydajnych testów. Ponieważ w dużym stopniu korzysta z adnotacji Java (wprowadzonych z JDK 5.0) do definiowania testów, może również pokazać, jak używać tej nowej funkcji języka Java w rzeczywistym środowisku produkcyjnym.

Dokumentacja dla TestNG jest dość szeroko rozbudowana. Framework posiada bardzo przystępną dla użytkownika dokumentację. Są tam wyszczególnione sekcje. Istnieje 8 głównych sekcji i każda z nich jest podzielona na kilka bloków. Użytkownik może się dowiedzieć z nich jak zaplanować, przeprowadzić oraz przeanalizować wyniki przeprowadzonego testu [5-6].

TestNG zawiera możliwość uruchamiania testów w dowolnie dużych pulach wątków, można też zgrupować wiele przypadków testowych i przekonwertować je na plik XML, a następnie ustawić priorytet przypadku testowego w określonej kolejności. TestNG zapewnia łatwą i elastyczną konfigurację testów, w której nie ma potrzeby pisania kodu, aby modyfikować jakiekolwiek metody testowe.

Raporty TestNG pojawiają się, gdy wykonywane są przypadki testowe przy użyciu TestNG. Po wykonaniu przypadków testowych TestNG wygeneruje domyślny raport HTML. Dodatkowo mogą być wykorzystane inne zewnętrzne frameworki do tworzenia raportów, na przykład Allure Framework. W tym celu wystarczy dodać odpowiednie zależności do pliku pom.xml.

4. WebdriverIO

WebdriverIO to oparty na JavaScript framework do automatyzacji testów zbudowany na Node.js. Jest to projekt typu open source opracowany dla społeczności testujących automatycznie. WebdriverIO jest rozszerzalny, kompatybilny, bogaty w funkcje i łatwy w instalacji. Jest to uważane za platformę automatyzacji testów nowej generacji, która obsługuje zarówno przeglądarki komputerowe, jak i aplikacje mobilne. Co sprawia, że WebdriverIO jest korzystną opcją do testowania automatycznego. Obsługuje koncepcje testowe BDD (Behavior-driven development) i TDD (Test-driven development).

W przeciwieństwie do większości dostępnych obecnie platform automatyzacji przeglądarek, WebdriverIO jest napisane w całości w języku JavaScript. Nawet instalacja Selenium odbywa się za pośrednictwem modułu NPM. Z tego powodu istnieje taka opinia, że programiści front-end powinni pisać testy dla swojego kodu (zarówno jednostkowego, jak i funkcjonalnego). WebdriverIO sprawia, że jest to niezwykle łatwe [7-8].

Framework posiada bardzo przystępną dla użytkownika dokumentację. Strony zawierają materiały referencyjne dla wszystkich zaimplementowanych wiązań i poleceń. WebdriverIO ma zaimplementowane wszystkie polecenia protokołu JSONWire, a także obsługuje specjalne powiązania dla Appium (narzędzia do testów aplikacji mobilnych). WebdriverIO używa Selenium, dlatego możliwe jest uruchamianie testów we wszystkich rodzajach przeglądarek. Selenium to bardzo dobra platforma i lider w branży do uruchamiania automatyzacji przeglądarek [9].

WebdriverIO nie posiada własnego narzędzia do raportowania. Natomiast ma możliwość podłączenia wielu różnych pakietów do tworzenia raportów od in-

nych narzędzi. Na przykład, jest możliwość konfiguracji wtyczki do tworzenia raportów testowych takich jak Allure, HTML-reporter oraz Spec-reporter.

5. Scenariusze testowe

W celach badania do testowania została wybrana dość znana aplikacja webowa, służąca do wspólnego tworzenia oprogramowania: www.github.com. Następnie, na bazie tej strony webowej zostało stworzonych siedem scenariusze testowych, na podstawie, których, zostały wykonane testy automatyczne w analizowanych frameworkach: TestNG (w języku Java) oraz WebdriverIO (w języku JavaScrypt). Wyniki testów zostały porównane pod kontem szybkości uruchomienia oraz zużyciem pamięci komputera. Scenariusze testowe podane w poniższych tabelach (Tabele 1 - 7).

Tabela 1: Scenariusz do zdarzenia Mouse Hover

Nazwa		Zdarzenie Mouse Hover
Cel		Sprawdzenie poprawności działania menu rozwijanego po najechaniu myszki
Warunki wstępne		brak
Wymagania		brak
Kroki		
Opis wykonanych czynności		Oczekiwany wynik
1	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2	Najechanie myszki na element menu:	Pojawia się nowy blok - menu rozwijane pod wskazanym elementem

Tabela 2: Scenariusz zapisywania danych do pliku

Nazwa		Zapisywanie do pliku tekstowego
Cel		Sprawdzenie odczytu elementów ze strony i zapis do pliku
Warunki wstępne		Na stronie jest wyświetlane, co najmniej jedne dostępne stanowisko
Wymagania		Plik do zapisu już istnieje
Kroki		
Opis wykonanych czynności		Oczekiwany wynik
1	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2	Najechanie myszki na element menu:	Pojawia się nowy blok - menu rozwijane pod wskazanym elementem
3	Rozwijanie każdego bloku pozycji i zapis nazw stanowisk do pliku	Nazwy stanowisk z każdego bloku są odczytane ze strony i zapisane do pliku tekstowego

Tabela 3: Scenariusz zapisywania danych do pliku

Nazwa		Pobranie pliku pdf
Cel		Sprawdzenie możliwości pobrania pliku pdf ze strony
Warunki wstępne		Plik istnieje
Wymagania		Plik dostępny do pobrania niezalogowanym użytkownikom
Kroki		
Opis wykonanych czynności		Oczekiwany wynik
1	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2	Przejście do odnośnika „Download Security Guide”	Przeniesienie na witrynę z przyciskiem do pobrania pliku
	Kliknięcie przycisku do pobrania	Plik został pobrany i otworzony w nowej zakładce

Kolejne cztery scenariusze, które są podane w (Tabelach 4-7) będą przeprowadzone za pomocą zbioru danych do testów. W przypadku scenariusza logowania to będą dane użytkowników (nazwa użytkownika oraz hasło), w przypadku scenariusza wyszukiwania – słowa kluczowe, dla scenariusza sortowania – lista dostępnych pól i odpowiednio dla scenariuszy filtrowania – lista języków do pokazania. Te dane zostały wcześniej przygotowane i będą pobierane z pliku .xls podczas uruchamiania testów (Rysunek 3).

Dla każdego scenariusza potrzebującego takiego zbioru danych istnieje osobna zakładka w pliku.

Tabela 4: Scenariusz logowania

Nazwa		Zalogowanie się do aplikacji
Cel		Sprawdzenie poprawności logowania do aplikacji
Warunki wstępne		Wylogowany użytkownik
Wymagania		Brak
Kroki		
Opis wykonanych czynności		Oczekiwany wynik
1	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2	Przejście do odnośnika „SignIn”	Przeniesienie na witrynę logowania
3	Uzupełnienie danych logowania	Uzupełnione wartości
4	Kliknięcie przycisku do logowania	Przejście do witryny konta użytkownika, sprawdzanie pola email/username

Tabela 5: Scenariusz wyszukiwania

Nazwa		Wyszukiwanie przez słowo kluczowe
Cel		Sprawdzenie poprawności wyszukiwania przez słowo kluczowe
Warunki wstępne		Istnieją repozytoria /użytkownicy za podanym słowom kluczowym
Wymagania		Brak
Kroki		
Opis wykonanych czynności		Oczekiwany wynik
1	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2	Kliknięcie w pole wyszukiwania	Kursor myszy został umieszczony w polu wyszukiwania
3	Podanie słowa kluczowego oraz kliknięcie Enter	Przeniesienie na witrynę z wynikami wyszukiwania, sprawdzanie, czy w wynikach wyszukiwania znajduje się słowo kluczowe w nazwie repozytorium lub nazwie użytkownika

Tabela 6: Scenariusz sortowania

Nazwa		Sortowanie
Cel		Sprawdzenie poprawności sortowania wyników wyszukiwania
Warunki wstępne		brak
Wymagania		brak
Kroki		
Opis wykonanych czynności		Oczekiwany Wynik
1.	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2.	Kliknięcie w pole wyszukiwania	Kursor myszy został umieszczony w polu wyszukiwania
3.	Podanie słowa kluczowego oraz kliknięcie Enter	Przeniesienie na witrynę z wynikami wyszukiwania
4.	Kliknięcie w wybrany rodzaj sortowania	Sprawdzanie wybranego rodzaju sortowania

Tabela 7: Scenariusz filtrowania

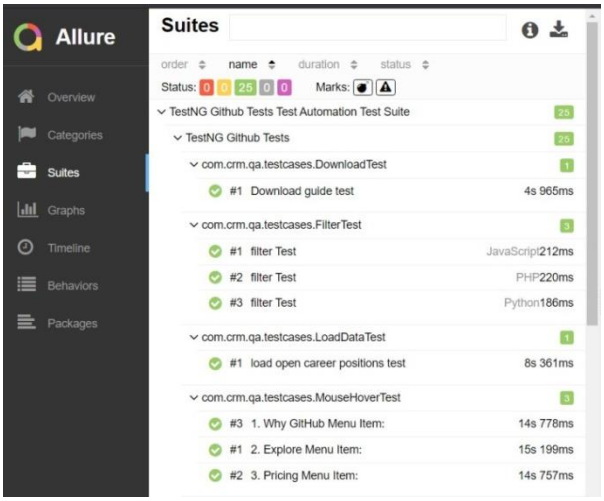
Nazwa		Filtrowanie
Cel		Sprawdzenie poprawności filtrowania wyników wyszukiwania
Warunki wstępne		brak
Wymagania		brak
Kroki		
Opis wykonanych czynności		Oczekiwany Wynik
1.	Wejście na stronę www.github.com	Przeniesienie do witryny portalu github
2.	Kliknięcie w pole wyszukiwania	Kursor myszy został umieszczony w polu wyszukiwania
3.	Podanie słowa kluczowego, kliknięcie Enter	Przeniesienie na witrynę z wynikami wyszukiwania
4.	Kliknięcie w wybrany filtr	Sprawdzanie wyników według wybranego filtru

6. Eksperyment

Skrypty testowe zostały zaimplementowane w obu frameworkach i odpowiadają zaplanowanym wcześniej scenariuszom testowym. One korzystają z tych samych zbiorów danych testowych. Jeden zestaw testów (test suite) składa się z 25 przypadków testowych (test cases). Czas i wyniki uruchamiania testów zostały przedstawione w wygenerowanym w Allure raporcie.

Na poniższym rysunku (Rysunek 1) przedstawione zżuty ekranu raportu wykonania całego zbioru testów (czyli 25 przypadków testowych). Jak widać wszystkie testy przeszły.

W ramach eksperymentu zostały zdefiniowane scenariusze badawcze, które zostały przedstawione w Tabeli 8.



Rysunek 1: Wygląd wygenerowanego Allure Raportu (TestNG).

Tabela 8: Scenariusze badawcze

	Scenariusz badawczy	Liczba wykonań
1.	Pomiar całkowitego czasu uruchamiania wszystkich skryptów testowych:	50
2.	Pomiar całkowitego czasu uruchamiania skryptów testowych w trybie headles (minimalizacja [h,m,s])	50
3.	Pomiar zużycia pamięci podczas wykonania testów (minimalizacja [MB])	50
4.	Pomiar zużycia CPU podczas wykonania testów (%)	50

Dla czystości eksperymentu testy zostały przeprowadzone w tych samych warunkach. Ponieważ do testów została wykorzystana aplikacja działająca w Internecie, zależy tutaj na połączeniu sieciowym. Akurat w ramach badań stosowane było połączenie przez WIFI uczelni – eduroam (EDUcation ROAMing).

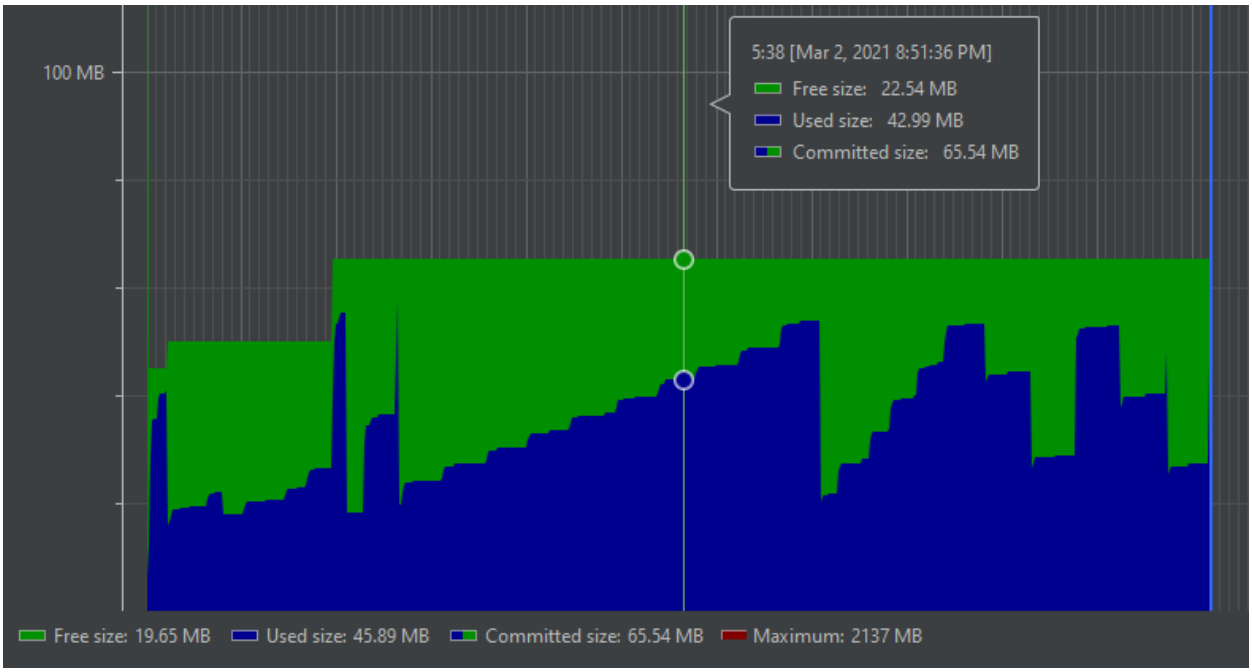
Testy zostały wykonane na komputerze stacjonarnym z następującymi parametrami:

- procesor Intel® Core ™ i7-2760QM (do 3,50 GHz, 4 rdzenie, 8 wątków)
- pamięć RAM 8 GB
- rodzaj dysku SSD
- system operacyjny: Windows 10 Education

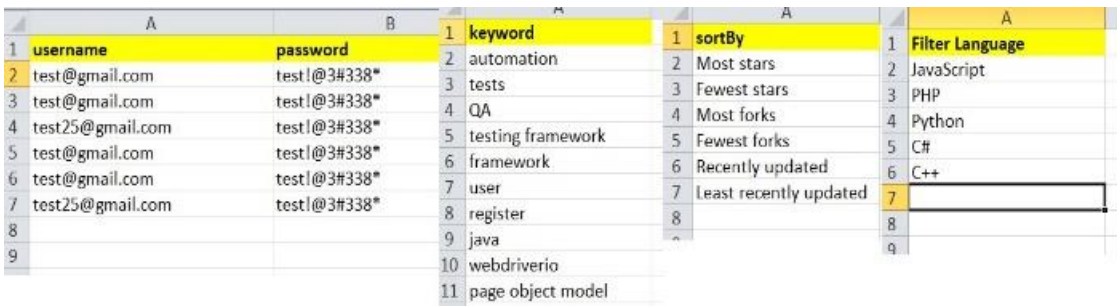
7. Wyniki

Wyniki zostały porównane przez cztery zdefiniowane kryteria: czas uruchamiania zbiorów testowych w różnych trybach działania przeglądarki, średnia wartość zużycia pamięci oraz CPU podczas wykonania testów.

Za pomocą narzędzi pomiarowych zostało zmierzone średnie zużycie pamięci podczas wykonania testów. Dla TestNG ta wartość wynosi 45,89 MB, a dla WebdriverIO 65 MB. Przykładowy wykres dla TestNG podano na Rysunku 2.



Rysunek 2: Pomiar zużycia pamięci podczas wykonania testów (TestNG).



	A	B
1	username	password
2	test@gmail.com	test!@3#338"
3	test@gmail.com	test!@3#338"
4	test25@gmail.com	test!@3#338"
5	test@gmail.com	test!@3#338"
6	test@gmail.com	test!@3#338"
7	test25@gmail.com	test!@3#338"
8		
9		

	A
1	keyword
2	automation
3	tests
4	QA
5	testing framework
6	framework
7	user
8	register
9	java
10	webdriverio
11	page object model

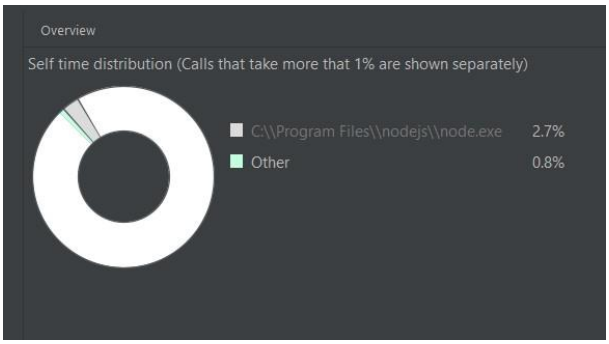
	A
1	sortBy
2	Most stars
3	Fewest stars
4	Most forks
5	Fewest forks
6	Recently updated
7	Least recently updated
8	

	A
1	Filter Language
2	JavaScript
3	PHP
4	Python
5	C#
6	C++
7	
8	

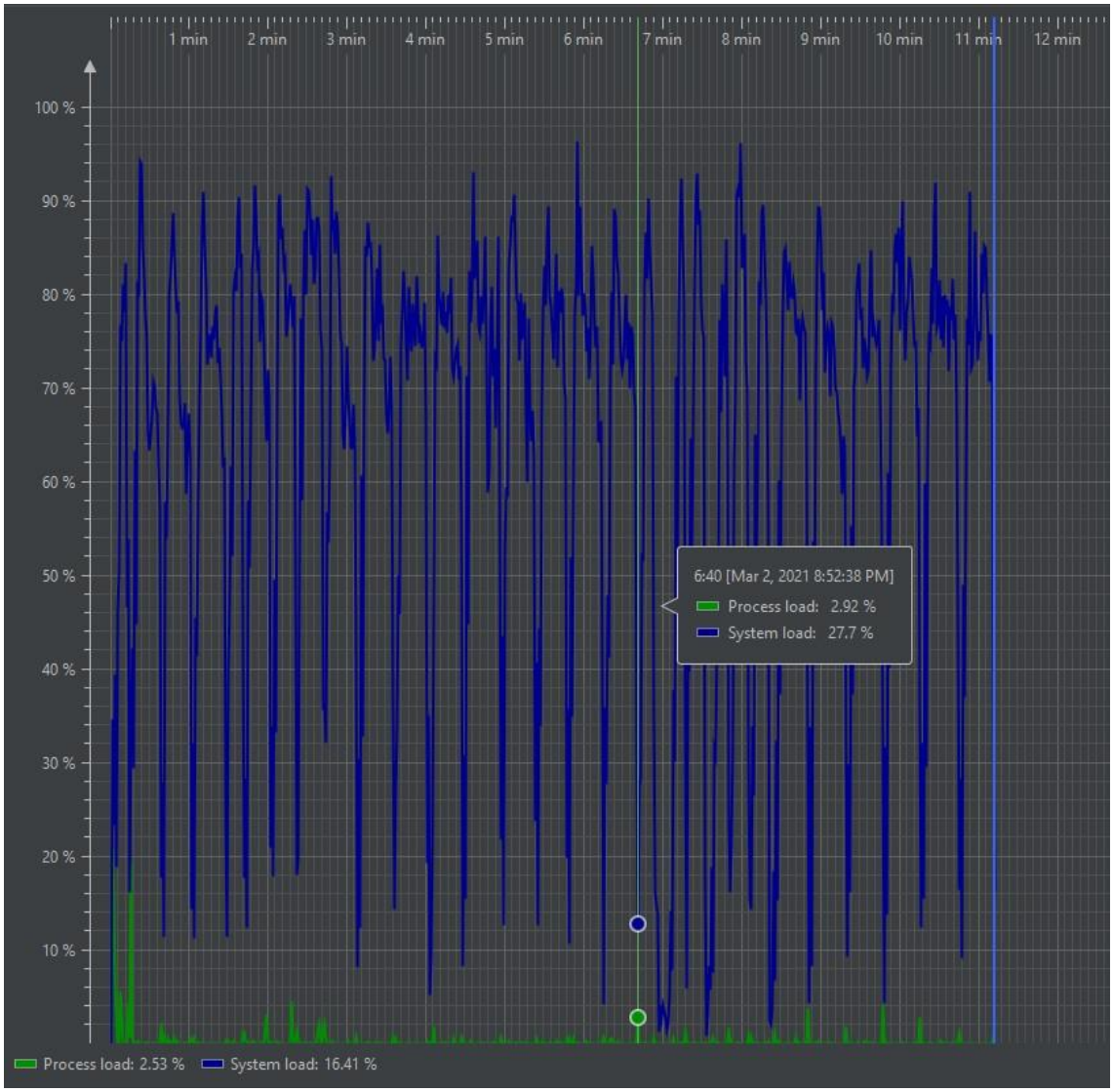
Rysunek 3: Zbiór danych z pliku .xls.

Następnie zmierzono średnie użycie CPU podczas wykonania testów. Dla frameworku TestNG to 2.53%, a dla WebdriverIO ta wartość wynosi 2.7%.

Przykładowy wykres uruchomianego testu podano na rysunkach poniżej, dla WebdriverIO – Rysunek 4, dla TestNG – Rysunek 5.



Rysunek 4: Średnia zużyta CPU dla testów WebdriverIO.



Rysunek 5: Średnia zużyta CPU dla testów TestNG.

Podsumowująca tabela pokazująca wyniki eksperymentu podana poniżej (Tabela 9).

Tabela 9: Wyniki eksperymentu

	Scenariusz badawczy	TestNG	WebdriverIO
1.	Średni czas uruchamiania zbioru skryptów testowych	11m 10s	7m 5s
2.	Średni czas uruchamiania skryptów testowych w trybie headles (minimalizacja [h,m,s])	6m 1s	3m 43s
3.	Pomiar zużycia pamięci podczas wykonania testów (minimalizacja [MB])	45.89 MB	65 MB
4.	Pomiar zużycia CPU podczas wykonania testów (%)	2.53%	2.7 %

Na podstawie przeprowadzonych badań lepszym okazał się framework WebdriverIO, chociaż różnica wyników nie jest dość duża. W pierwszym badaniu uruchamiania zbioru skryptów z jednej jak i z pięciu instancji przeglądarki framework WebdriverIO okazał się 1.61 razy szybszy.

W drugim badaniu, podczas uruchamiania testów w trybie headless WebdriverIO też pokazał lepszy wynik i okazał się 1.58 razy szybszy. Kolejnym badaniem było porównanie użycia pamięci podczas uruchomienia testów, to okazało się, że framework WebdriverIO wymaga więcej pamięci niż TestNG. Nie jest ta różnica zbyt duża i wynosi 19.11 MB. Ostatnie badanie dotyczyło pomiarów zużycia CPU komputera, na którym zostały uruchomione skrypty testowe. Wystąpiła niewielka różnica wynosząca 0.17% na korzyść TestNG.

Pod względem kryteriów ogólnych, warto wyróżnić WebdriverIO z tego powodu, że jest napisany i przeznaczony dla napisania testów w języku JavaScript. Jest to plusem, ponieważ ten język jest często wykorzystywany do napisania stron internetowych i programistom nie ma potrzeby używać innego języka, żeby zacząć pisać testy, innymi słowami WebdriverIO jest łatwiejszy do wdrożenia.

Z przeprowadzonych badań wynika, że jeśli chodzi o wybór lepszego frameworku testowego, to preferowany jest WebdriverIO.

Literatura

[1] T. J. Naidu, N. A. Basri, S. Nagenthram, “Selenium: A comparative analysis,” 2014 Internet Conference Contempt Computer Informatics, IC3I 2014, 2014.

[2] S. Jagannatha, et al, Comparative Study on Automation Testing using Selenium Testing Framework and QTP, International Journal of Computer Science and Mobile Computing, 3(10) (2014) 258-267.

[3] Wac, T. Watras, G. Kozieł, Analiza porównawcza rozwiązań wykorzystywanych w testowaniu automatycznym, 2019.

[4] S. Jagannatha, et al, Comparative Study on Automation Testing using Selenium Testing Framework and QTP International Journal of Computer Science and Mobile Computing, 3(10) (2014) 258-267.

[5] M. Meiliana, I. Septian, R.S. Alianto, Daniel.. Comparison Analysis of Android GUI Testing Frameworks by Using an Experimental Study, Procedia Computer Science, 135 (2018) 736–748. doi: 10.1016/j.procs.2018.08.211.

[6] S.M. Srinivasan, R.S. Sangwan, Web App Security: A Comparison and Categorization of Testing Frameworks, IEEE Software, 34(1) (2017) 99–102.

[7] S. Sharma, “Study and analysis of automation testing techniques,” J. Global Research Computer Science, 3(12) (2012) 36-43.

[8] V. N. Maurya, E. R. Kumar, Analytical Study on Manual vs. Automated Testing Using with Simplistic Cost Model, 2012.

[9] K. Bahl, Software Testing Tools Techniques for Web Applications, 2015.