JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 95–99
Received: 18 February 2021
Accepted: 03 March 2021

# Comparative analysis of performance of ASP.NET Core MVC and Symfony 4 programming frameworks

# Wydajnościowa analiza porównawcza szkieletów programistycznych ASP.NET Core MVC i Symfony 4

Marcin Górski* , Wojciech Andrzej Piwowarski, Mariusz Dzieńkowski

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

**Abstract**
The article presents a comparative analysis of popular ASP.NET Core MVC and Symfony 4 frameworks. Two web applications, containing the same functionalities and acting as a simple system for managing articles, were implemented in these technologies. The applications underwent time performance tests during typical operations performed by means of a simple form such as entering, editing, viewing and deleting data. These actions were performed automatically using commands from the Puppeteer library. The listed operations were repeated 10, 100 and 1,000 times in order to obtain precise mean times. On the basis of the obtained results, it was difficult to clearly state which of the compared programming tools is better. The ASP.NET Core MVC framework coped much better with two time-consuming operations, i.e. entering and editing data. Its results in this regard (the average from 1,000 repetitions) were respectively approximately 28% and 25% better compared to the Symphony 4 framework. However, for the two less time-consuming operations, i.e. displaying and deleting articles, the Symphony 4 framework proved to be considerably better. Its results with regard to displaying and deleting articles (the average for 1,000 measurements) were respectively 15% and 36% lower compared to the other of the tested frameworks.

*Keywords*: performance analysis; automatic testing; framework; ASP.NET Core MVC; Symfony; Puppeteer

**Streszczenie**
Artykuł przedstawia analizę porównawczą popularnych szkieletów programistycznych ASP.NET Core MVC oraz Symfony 4. W technologiach tych zaimplementowano dwie aplikacje internetowe, zawierające te same funkcjonalności, pełniące funkcję prostego systemu do zarządzania artykułami. Te aplikacje zostały poddane testom wydajności czasowej podczas realizacji typowych operacji wykonywanych za pośrednictwem prostego formularza takich jak wprowadzanie, edycja, wyświetlanie i usuwanie danych. Czynności te były wykonywane automatycznie za pomocą poleceń z biblioteki Puppeteer. Wyszczególnione operacje były powtarzane 10, 100 i 1000 razy w celu uzyskania precyzyjnych średnich czasów. Na podstawie otrzymanych wyników trudno było jednoznacznie stwierdzić, które z porównywanych narzędzi programistycznych jest lepsze. Z dwiema czasochłonnymi operacjami tzn. wprowadzaniem i edycją danych, znacznie lepiej radził sobie framework ASP.NET Core. Jego wyniki pod tym względem (średnia z 1000 powtórzeń) były odpowiednio o około 28% i 25% lepsze w stosunku do szkieletu Symfony 4. Natomiast dla dwóch mniej czasochłonnych operacji, czyli wyświetlania i usuwania artykułów, wyraźnie lepszym okazał się szkielet Symfony 4. Jego wyniki dla wyświetlania i usuwania artykułów (średnia dla 1000 pomiarów) były o 15 i 36 procent odpowiednio niższe w stosunku do drugiego badanego szkieletu.

*Słowa kluczowe*: analiza wydajnościowa; testowanie automatyczne; ASP.NET Core MVC; Symfony; Puppeteer

*Corresponding author
*Email address*: marcin.gorski1@pollub.edu.pl (M. Górski)



## 1. Wstęp

Przed przystąpieniem do realizacji projektu programistycznego ważną kwestią, która może nawet decydować o jego powodzeniu lub porażce, jest dobór odpowiednich technologii. Duża liczba języków i opartych na nich szkieletów programistycznych nie ułatwia tego zadania. Ważnym aspektem, który należy uwzględnić przy podejmowaniu decyzji o wyborze technologii jest jej popularność, którą można sprawdzić za pomocą licznych rankingów i narzędzi porównawczych publikowanych na bieżąco w Internecie, np. TIOBE Index [1] czy Google Trends [2]. Za popularnością często idzie wielkość społeczności wspierającej daną technologię. Większa społeczność daje gwarancje, że technologia będzie rozwijana i będzie łatwiejsza w utrzymaniu.

W procesie wytwarzania oprogramowania bardzo ważnym etapem jest jego testowanie, którego celem jest wykrywanie ewentualnych awarii, będących skutkiem występujących w kodzie błędów. Testowanie prowadzi do zwiększania niezawodności oprogramowania i wpływa na jego jakość. Może ono być wykonywane w sposób manualny - przez wykwalifikowanych testerów lub w sposób automatyczny za pomocą specjalnych narzędzi informatycznych oraz bibliotek. Testy automatyczne, nie wymagają angażowania wielu specjalistów, przez co nie są czasochłonne i kosztochłonne. Raz opracowany test może być uruchamiany wielokrotnie [3].

Jednak istnieją pewne przypadki, które wymagają ręcznej weryfikacji. Takim przykładem może być realizacja złożonych scenariuszy testowych, których automatyzacja jest nieopłacalna. Natomiast w przypadku gdy testy będą wielokrotnie powtarzane, gdy będą wykonywane na wielu zestawach danych, na różnych platformach sprzętowych i programistycznych oraz przy różnych konfiguracjach, wówczas automatyzacja jest jak najbardziej wskazana. Także takie względy jak czasochłonność i niemożność ręcznej weryfikacji przemawiają za zautomatyzowaniem procesu testowania [4].

Twórcy systemów internetowych, szukają najlepszych narzędzi do budowy swoich aplikacji. Z takiego właśnie powodu narodził się pomysł przeprowadzenia w ramach tej pracy analizy dwóch popularnych technologii webowych: ASP.NET Core MVC oraz Symfony 4. W tym celu opracowano eksperyment, w którym zbadano dwie aplikacje webowe, jedna zbudowana na bazie szkieletu programistycznego ASP.NET Core MVC, a druga na podstawie Symfony 4. Badania dotyczyły czasowej oceny wydajności obu rozwiązań.

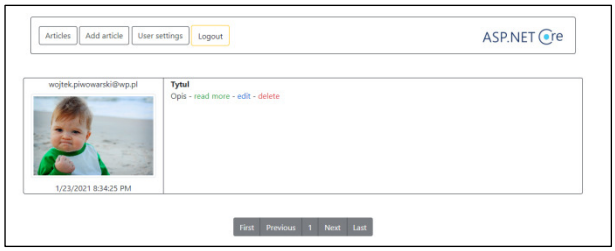
## 2. Porównywane technologie

Symfony jest darmowym i popularnym w Polsce szkieletem programistycznym. Został on utworzony w języku skryptowym PHP i jest w pełni zorientowany obiektowo. Szkielet wspomaga programowanie dynamicznie generowanych aplikacji webowych, dzięki udostępnianiu gotowej architektury opartej na wzorcu projektowym MVC. Symfony nie jest zależne od konkretnego systemu bazy danych i w związku z tym można skonfigurować i podłączyć dowolną bazę. Szkielet ten został stworzony w oparciu o najlepsze standardy oraz wzorce budowania aplikacji www. Wyposażony jest w podstawowe funkcje gotowe do użycia w aplikacjach, np. walidacja formularzy, zarządzanie sesjami czy autoryzacja. Dzięki narzędziu Composer szybko można rozbudować aplikację oraz dołączyć dodatkowe biblioteki. Symfony ma wbudowany silnik szablonów twig, który jest odpowiedzialny za renderowanie widoków [5].

ASP.NET Core jest darmową, wieloplatformową, wysokowydajną i otwartą platformą programistyczną typu „Open Source", umożliwiającą tworzenie nowoczesnych aplikacji z obsługą chmury. Zaletą ASP.NET Core jest to, że praktycznie po każdej aktualizacji pojawiają się nowe funkcje oraz wzbogacane są funkcje już istniejące, które pozwalają na budowę wysoce skalowalnych i wydajnych aplikacji internetowych. W samym szkielecie dostępnych jest bardzo dużo funkcji, które pozwalają na rozwiązywanie typowych problemów stających przed każdym programistą. Powodują one zwiększenie wydajności samej aplikacji oraz umożliwiają szybkie dodawanie różnych nowych funkcjonalności. Wraz z pojawieniem się technologii .NET Core można tworzyć aplikacje i wdrażać je na różnych systemach operacyjnych takich jak Windows, Linux oraz MacOS. Microsoft i cała społeczność programistyczna wykorzystująca narzędzia do tworzenia swoich aplikacji uczynili system Linux idealną platformą do wdrażania aplikacji ASP.NET Core. Wykorzystanie asynchronicznych wzorców programowania opartych na nowych szablonach MVC, sprawia, że aplikacje opracowane na bazie tego szkieletu są coraz szybsze [6].

## 3. Aplikacje testowe

Dla celów porównań szkieletów programistycznych opracowano dwie proste aplikacje webowe. Jedna została zbudowana przy pomocy szkieletu Symfony 4, natomiast drugą oparto na ASP.NET Core MVC. Obie aplikacje są identyczne, ponieważ mają zaimplementowane te same funkcjonalności, pozwalające na zarządzanie artykułami za pomocą prostych operacji CRUD (ang. create, read, update, delete) do przeglądania, dodawania, usuwania oraz aktualizowania artykułów (Rysunek 1).



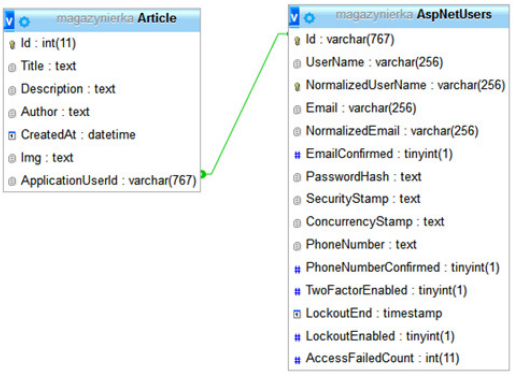

Rysunek 1: Widok strony głównej z artykułami.

Większość działań jest realizowana za pomocą formularza zawierającego trzy pola do uzupełnienia (tytuł artykułu, opis oraz selektor plików do załączania zdjęcia) oraz przycisk potwierdzający daną operację (Rysunek 2). Podczas aktualizacji pola formularza najpierw wypełniane są danymi z bazy, dotyczącymi wybranego artykułu, a następnie są modyfikowane.


Rysunek 2: Tworzenie nowego artykułu.

Jednocześnie na jednej stronie możliwe jest przeglądanie dziesięciu artykułów, co zapewnia pełną kontrolę i swobodę podczas tworzenia nowych treści. Do osiągnięcia tego efektu wykorzystano mechanizm paginacji.

Aplikacje testowe współpracują z prostą, bo składającą się z dwóch tabel, bazą danych (Rysunek 3). Główna tabela służy do przechowywania danych związanych z artykułami. Druga tabela zawiera informacje o użytkownikach, którzy po zalogowaniu się mogą dodawać, edytować i usuwać artykuły. Obie tabele są powiązane ze sobą relacją jeden do wielu.




Rysunek 3: Struktura bazy danych aplikacji.

## 4. Środowisko testowe

Testy aplikacji zostały przeprowadzone na komputerze Lenovo G580, którego parametry przedstawione są w Tabeli 1. Przed uruchomieniem testów, w celu zwiększenia ich wiarygodności, zostały wyłączone wszystkie zbędne procesy.

Tabela 1: Konfiguracja środowiska uruchomieniowego dla przygotowanych testów

| | |
|---|---|
| System operacyjny | Windows 10 |
| Procesor | i3-312M 2.50 GHz |
| Pamięć RAM | 8GB |
| Dysk | SSD |

Na potrzeby badań został wykorzystany profesjonalny serwer VPS (ang. Virtual Private Server) oferowany przez firmę OVHCloud [7]. Wirtualny serwer jest wyodrębnionym środowiskiem utworzonym na serwerze fizycznym przy pomocy technologii wirtualizacji. Rozwiązanie to oferuje wszystkie korzyści standardowego serwera, z wydzielonymi zasobami i pełnym dostępem administratora. Użytkownik może wybrać system operacyjny, a także aplikacje, z których chce korzystać w danej chwili, bez żadnych ograniczeń konfiguracyjnych. W Tabeli 2 znajdują się szczegóły dotyczące wykorzystanego w ramach niniejszej pracy środowiska uruchomieniowego.

Tabela 2: Konfiguracja środowiska uruchomieniowego

| | |
|---|---|
| System Operacyjny | Ubuntu 18.04 Server |
| Pamięć RAM | 2GB |
| Pamięć dyskowa | SSD 20GB |
| Serwer | Apache/2.4.29 (Ubuntu) |
| Serwer MySql | 5.7.32 |
| Wersja PHP | 7.2.24 |
| Środowisko .Net | ASP.Net Core 3.1 |
| EntityFarmework | 3.1 |
| HTTP/SSL | Tak |
| Symfony | Symfony 4.4.10 |

## 5. Metoda badań

### 5.1. Narzędzie badawcze - biblioteka Puppeteer

Do analizy wydajności aplikacji testowych wykorzystano bibliotekę Puppeteer [8]. Została ona napisana w języku TypeScript i w związku z tym do jej uruchomienia niezbędna jest platforma Node.js, która z jednej strony kompiluje kod TypeScript na JavaScript, a z drugiej umożliwia uruchomianie oprogramowania. Puppeteer pozwala zautomatyzować manualne czynności wykonywane przez człowieka. W ten sposób kod aplikacji, uzupełniony odpowiednimi poleceniami pochodzącymi z tej biblioteki, wykonuje się samodzielnie. Oznacza to, że strona www obsługiwana jest automatycznie za pośrednictwem komend, wywołujących kolejne wymagane do jej obsługi czynności. Wśród nich mogą być na przykład kliknięcia myszą komputerową, pisanie na klawiaturze czy robienie zrzutów ekranowych.

Podczas analizy aplikacji testowej wykonanej za pomocą szkieletu ASP.NET jak i Symfony została wykonana ta sama sekwencja poleceń. Najpierw skrypt uruchamia przeglądarkę internetową i autoryzuje użytkownika. Następnie wykonuje wielokrotnie seriami operacje CRUD, mierzy czasy ich trwania oraz na koniec zamyka przeglądarkę. Listing 1 przedstawia metodę *auth*, przyjmującą dwa parametry i zawierającą instrukcje automatyzujące proces autoryzacji użytkownika.

Listing 1: Automatyczna autoryzacja użytkownika

```
async auth(page, model) {

    await page.goto(model.url + model.auth.url,
                    {waitUntil: 'networkidle0'});
    await page.type(model.auth.login, model.login);
    await page.type(model.auth.password, model.password);

    await Promise.all([
        page.click(model.auth.submit),
        page.waitForNavigation({waitUntil: 'networkidle0'}),
    ]);

    run page;
}
```

Pierwszy parametr (*page*) metody *auth* reprezentuje klasę, która jest odpowiedzialna za obsługę przeglądarki, zaś drugi parametr jest modelem, za pomocą którego dostarczane są niezbędne dane do obsługi testowanej aplikacji. W pierwszej kolejności wywołane jest polecenie *goto*, odpowiedzialne za uruchomienie strony o podanym adresie url, przekazywanym w pierwszym parametrze. W drugim parametrze przesyłany jest obiekt, który nakazuje metodzie odczekać do momentu aż się strona do końca wczyta. Dwa następne polecenia uzupełniają formularz danymi. Są one wywoływane z dwom parametrami. Za pomocą pierwszego wskazywane jest id elementu, dla którego będzie wywoływana jakaś akcja. Drugi parametr służy do przekazania wartości, którą będzie wypełniane pole. Następnie uruchamiana jest metoda *Promise.all*, która inicjuje naciśnięcie przycisku *wyślij*, skutkujące wysłaniem formularza. W tym przypadku także, jako jedyny parametr, został wykorzystany obiekt, którego zadaniem jest odczekanie do momentu aż strona zostanie załadowana.

### 5.2. Scenariusze badawcze

Eksperyment badawczy został przeprowadzony przez automat, który samodzielnie zrealizował scenariusze badawcze, podczas których mierzono czasy wykonywania się poszczególnych zadań. Średnie czasy trwania

realizacji poszczególnych operacji przyjęto jako wskaźniki wydajności i zostały użyte podczas późniejszych analiz. Scenariusze badawcze dotyczyły realizacji czterech prostych czynności CRUD na tabeli w bazie danych wykonywanych przez aplikacje testowe. Scenariusze te obejmowały następujące operacje:

- wyświetlanie wybranego artykułu,
- wypełnienie pól formularza danymi i dodanie artykułu do bazy,
- wczytanie danych z bazy do formularza, zmodyfikowanie artykułu i ponowny zapis do bazy,
- usuwanie wybranego artykułu.

### Scenariusz 1: Wyświetlanie artykułów

Metoda odpowiadająca za wyświetlanie artykułów wykorzystuje tylko jedno polecenie *goto*, realizujące przejście do strony (paginy) zawierającej konkretny artykuł wskazany za pomocą adresu url.

Listing 2: Wyświetlanie artykułów realizowane za pomocą poleceń biblioteki Puppeteer

```
async articleSelect(page, pageNumber, model) {
    await page.goto(model.url + model.select.url + pageNumber,
                    {waitUntil: 'networkidle0'});

    run page;
}
```

### Scenariusz 2: Dodawanie artykułów

Skrypt dodawania nowego artykułu (Listing 3) wygląda podobnie jak obsługa formularza przeznaczonego do autoryzacji. W tym przypadku, za pomocą instrukcji *page.type()* uzupełniane są pola z tytułem i opisem artykułu. Inicjowanie kliknięcia na przycisku typu *file* jest realizowane za pomocą polecenia *await page.$(model.new.file),* a następnie za pomocą kolejnej komendy *await input.uploadFile(image)* dodawany jest plik graficzny. Procedurę dodawania artykułu kończy polecenie symulujące naciśnięcie na przycisku *wyślij*.

Listing 3: Dodawanie artykułów realizowane za pomocą poleceń biblioteki Puppeteer

```
async articleNew(page, model) {
    await page.goto(model.url + model.new.url,
                    {waitUntil: 'networkidle0'});
    await page.type(model.new.description, description);
    const input =  await page.$(model.new.file);
    await input.uploadFile(image);
                    {waitUntil: 'networkidle0'});

    await Promise.all([
        page.click(model.new.submit),
        page.waitForNavigation({waitUntil: 'networkidle0'}),
    ]);

    run page;
}
```

### Scenariusz 3: Aktualizowanie artykułów

Edycja artykułu została przeprowadzona w sposób podobny jak dodawanie. Zostały przy tym wykorzystane dodatkowe instrukcje, które umożliwiają najpierw uchwycenie (*document.getElementById*), a następnie wyczyszczenie pól formularza.

Listing 3: Aktualizacja artykułów realizowana za pomocą poleceń biblioteki Puppeteer

```
async articleEdit(page, id, model) {
    await page.goto(model.url + model.edit.url + id
            + model.edit.afterUrl, {waitUntil: 'networkidle0'});
    await page.evaluate((model) => { document.getElementById(
            model.edit.title.substring(1)).value = "", model)
    await page.type(model.edit.title, title);
    await page.evaluate((model) => { document.getElementById(
            model.edit.description.substring(1)).value = "", model)
    await page.type(model.edit.description, description);

    const input =  await page.$(model.edit.file);
    await input.uploadFile(image);

    await Promise.all([
        page.click(model.edit.submit),
        page.waitForNavigation({waitUntil: 'networkidle0'}),
    ]);
    run page;
}
```

### Scenariusz 4: Usuwanie artykułów

W procesie usuwania danego artykułu (Listing 4), po naciśnięciu przycisku *usuń*, wymagane jest jeszcze potwierdzenie tej operacji w okienku dialogowym. Czynności te są realizowane za pomocą poleceń *page.on('dialog')* oraz *dialog.accept()*.

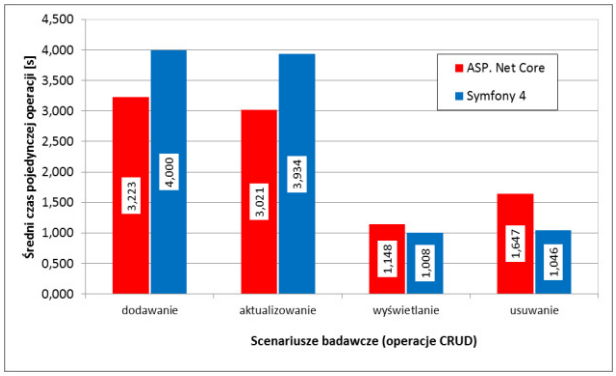
Listing 4: Usuwanie artykułu realizowane za pomocą poleceń biblioteki Puppeteer

```
async articleDelete(page, model) {
    page.on('dialog', async dialog => {
        await dialog.accept();
    });
    await Promise.all([
        page.click(model.delete.submit),
        page.waitForNavigation({waitUntil: 'networkidle0'}),
    ]);

    run page;
}
```

## 6. Wyniki badań

Wykonano po trzy serie testów dla obu badanych aplikacji. W każdej serii po kolei były wykonywane scenariusze, które odpowiadały poszczególnym operacjom CRUD. Dla uzyskania jak największej precyzji wyników każda operacja była powtarzana 10 razy w pierwszej serii, 100 razy w drugiej i 1000 razy w trzeciej serii. Wyniki dla poszczególnych scenariuszy zostały uśrednione i przedstawione na Rysunku 4.

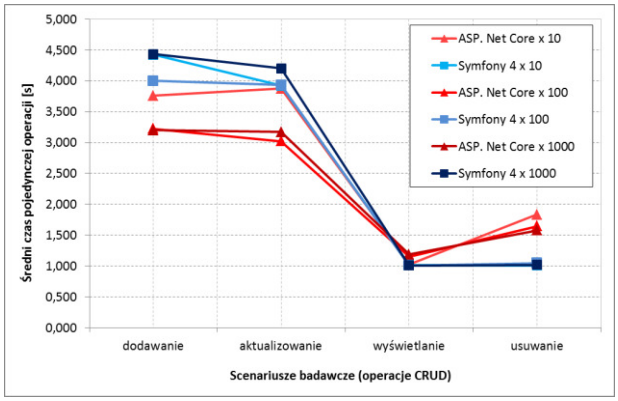



Rysunek 4: Średnie czasy wykonywania operacji przez aplikacje testowe dla 100 powtórzeń.

Powyższy wykres prezentuje średnie czasy ze 100 powtórzeń wykonania poszczególnych scenariuszy, za którymi stoją odpowiednie operacje CRUD. Wynika

z niego, że operacje dodawania i aktualizowania artykułów trwały dłużej dla aplikacji testowej zbudowanej na bazie szkieletu Symfony o odpowiednio 0,777 i 0,913 sekundy w stosunku do aplikacji opartej na platformie programistycznej ASP.NET Core. W przypadku wyświetlania artykułów różnice pomiędzy szkieletami były nieznaczne, gdyż wynosiły 0,14 sekundy na korzyść szkieletu Symfony 4. Także przy operacji usuwania z bazy lepsze czasy uzyskiwała aplikacja oparta na szkielecie Symfony. Dla tego scenariusza różnica była nieco wyższa, wynosiła 0,601 sekundy.

Na Rysunku 5 zestawiono uśrednione wyniki dla obu aplikacji testowych, czterech scenariuszy powtarzanych 10, 100 i 1000 razy.




Rysunek 5: Średnie czasy wykonywania operacji CRUD przez aplikacje testowe dla 10, 100 i 1000 powtórzeń.

W przypadku operacji dodawania dla wszystkich trzech serii (10, 100 i 1000 powtórzeń) średnie wyniki były wyższe dla aplikacji opracowanej na podstawie frameworka Symfony 4. Dla drugiej aplikacji testowej uśrednione wyniki dla serii nr 2 (100 powtórzeń) i serii nr 3 (1000 powtórzeń) były niemal identyczne. Podobna sytuacja była podczas aktualizowania artykułów. Na wykresie wyraźnie widać wyższe średnie poziomy czasów uzyskanych w trzech seriach dla aplikacji zbudowanej za pomocą szkieletu Symfony. W przypadku wyników dla frameworka ASP.NET Core odstające, wyższe wyniki widoczne są dla scenariusza 1, w którym operacja edycji była wykonywana 10 razy. Zwiększenie liczby powtórzeń spowodowało z jednej strony obniżenie średniego czasu oraz bardziej precyzyjne jego przedstawienie. Natomiast dla operacji wyświetlania artykułów, dla obu wersji aplikacji i wszystkich scenariuszy i bez względu na liczbę powtórzeń wyniki były bardzo podobne. W ostatnim scenariuszu, dotyczącym usuwania artykułów, aplikacja bazująca na Symfony 4, niezależnie od liczby powtórzeń operacji, uzyskiwała niemal identyczne średnie wyniki. Znacznie gorsze czasy realizacji zadań na bazie, dla średniej obliczonej z 10 jak i 100 pomiarów, miało oprogramowanie wykonane za pomocą ASP.NET Core.

## 7. Podsumowanie

W pracy porównywano dwa szkielety programistyczne ASP.NET Core i Symfony 4. Do analiz posłużyły specjalnie do tego celu opracowane dwie webowe aplikacje testowe. Przygotowano eksperyment badawczy, w którym proste czynności wypełniania i zapisywania formularza, edycji danych w formularzu i ich aktualizacji, wyświetlenia artykułów oraz usuwania wybranego artykułu zostały zautomatyzowane przez użycie biblioteki Puppeteer wykorzystywanej do realizacji testów automatycznych. Następnie wykonano wielokrotne pomiary czasów trwania poszczególnych czynności a uzyskane wyniki poddano uśrednianiu. Uzyskane średnie wyniki stanowiły podstawę do porównań wybranych szkieletów programistycznych.

Z dwiema czasochłonnymi operacjami, tzn. wprowadzaniem i edycją danych, znacznie lepiej radzi sobie framework ASP.NET Core. Jego wyniki były odpowiednio o około 28% i 25% (średnia z 1000 powtórzeń) lepsze w stosunku do szkieletu Symfony 4. Natomiast w dwóch mniej czasochłonnych operacjach, czyli wyświetlaniu i usuwaniu artykułów, wyraźnie lepszym okazał się szkielet Symfony w wersji 4. Jego wyniki były o 15 i 36 procent niższe (średnia z 1000 pomiarów) w stosunku do drugiego badanego szkieletu.

Z otrzymanych wyników, wykonanych analiz i przeprowadzonego wnioskowania trudno jest jednoznacznie określić, które z dwóch wziętych pod uwagę w tej pracy narzędzi programistycznych jest lepsze. Dobrym kierunkiem byłoby poszerzenie zakresu badań o jeszcze inne aspekty takie jak na przykład pomiar objętość kodu, ilość przesyłania danych czy zużycie zasobów.

Na koniec należy zwrócić uwagę na pewne ograniczenia zrealizowanych badań. Zostały one przeprowadzone w sztucznych warunkach, na aplikacjach testowych, które były bardzo proste, zaopatrzone w mało rozbudowany formularz, za pośrednictwem, którego wykonywano wielokrotnie powtarzane typowe operacje dodawania, edycji, wyświetlania oraz usuwania danych.

## Literatura

[1] Tiobe, https://www.tiobe.com/tiobe-index/, [16.02.2021].

[2] Google Trends, https://trends.google.pl/trends/, [16.02.2021].

[3] A. D. Wac, T. K. Watras, G. Kozieł, Comparative analysis of solutions used in automated testing, Journal of Computer Sciences Institute, 15 (2020) 156-163.

[4] SMARTBEAR, Test Automation Best Practices, https://smartbear.com/learn/automated-testing/best-practices-for-automation/, [16.02.2021].

[5] Symfony, https://symfony.com/, [16.02.2021].

[6] Introduction to ASP.NET Core, https://docs.microsoft.com/pl-pl/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0, [16.02.2021].

[7] OVHCloud, https://www.ovh.pl/, [16.02.2021].

[8] Puppeteer, https://developers.google.com/web/tools/puppeteer, [16.02.2021].