JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 15 (2020) 178-183
Received: 11 March 2020
Accepted: 23 March 2020

Comparative analysis of a selected version of the Symfony framework

Analiza porównawcza wybranych wersji szkieletu programistycznego Symfony

Michał Jusięga*, Mariusz Dzieńkowski

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

Abstract

This article is about research during which selected versions of the Symfony programming framework were compared in terms of their performance. The following versions of the framework were analysed: 3.0, 3.1, 3.2, 3.3, 3.4 LTS, 4.0, 4.1, 4.2, 4.3 and 4.4 LTS. For the purpose of the research, a simple test application in PHP was developed in ten variants corresponding to selected versions of the framework and consisting of 17 fragments of code – methods in the class, each of which operates on one basic component of Symfony. The application prepared in this manner was subject to performance tests in a two-stage experiment. After the experiment, the quantitative analyses were conducted in which the following aspects were taken into consideration: the average values of execution times and the average amounts of memory usage for specific code fragments for individual versions of the Symfony framework components and the average time of execution and demand for memory for the entire tested application. The obtained results for each code fragment representing a given component were visualized in the forms of graphs. The performed analyses showed that the best version of the Symfony programming framework in terms of performance is version 4.1.

Keywords: Symfony; performance; memory usage

Streszczenie

Zrealizowano badania, podczas których porównywano pod kątem wydajności wybrane wersje szkieletu programistycznego Symfony. Analizie poddano następujące wersje tego szkieletu: 3.0, 3.1, 3.2, 3.3, 3.4 LTS, 4.0, 4.1, 4.2, 4.3 oraz 4.4 LTS. Na potrzeby badań, opracowano prostą aplikację testową w języku PHP, w 10-ciu wariantach odpowiadających wybranym wersjom frameworka, składającą się z 17-stu fragmentów kodu - metod w klasie, z których każda operuje na jednym komponencie Symfony. Przygotowaną w ten sposób aplikację poddano testom wydajnościowym, w dwuetapowym eksperymencie. Po zakończeniu eksperymentu przeprowadzono analizy ilościowe, w których wzięto pod uwagę uśrednione wartości czasów wykonywania poszczególnych fragmentów kodu dla określonych wersji komponentów szkieletu Symfony oraz średnie czasy wykonywania się i zapotrzebowania na pamięć fizyczną przez całą aplikację testową. Uzyskane wyniki, dla każdego fragmentu kodu reprezentującego dany komponent, zostały zwizualizowane w formie wykresów. Przeprowadzone analizy wykazały, że najlepszą pod względem wydajnościowym wersją szkieletu programistycznego Symfony jest wersja 4.1.

Słowa kluczowe: Symfony; wydajność; użycie pamięci

*Corresponding author

Email address: **michal.jusiega@gmail.com** (M. Jusięga)

1. Wstęp

Podczas tworzenia aplikacji internetowych, wybór oprogramowania i jego wersji ma ogromny wpływ na optymalne działanie tworzonego systemu oraz ewentualne możliwości jego rozwoju. Decydować o tym może szereg wspieranych standardów kodowania, dostępność aktualizacji, których kolejne wersje zawierają nowe funkcjonalności oraz poprawki związane przede wszystkim ze zwiększaniem bezpieczeństwa. Od początku powstania tego frameworka duży nacisk kładziono na kwestie związane z jego bezpieczeństwem, wydajnością i szybkością. Dzięki temu Symfony jest obecnie jednym z najpopularniejszych, najbezpieczniejszych i najszybszych szkieletów programistycznych PHP [1].

Symfony, oprócz tego, że jest frameworkiem, jest jednocześnie zbiorem komponentów, na bazie których w efektywny sposób można realizować zaawansowane aplikacje internetowe [2]. Komponenty są odpowiedzialne za realizację poszczególnych zadań. Pracując nad określoną aplikacją, programista dobiera i dostosowuje odpowiednie komponenty w taki sposób, aby spełniały jej wymagania.

Na tle innych szkieletów programistycznych PHP, Symfony wyróżnia się silnie prowadzonym i przestrzeganym standardem procesu wprowadzania nowych wersji [3]. Standard ten definiuje wewnętrzne procedury ustalone wewnątrz zespołu doświadczonych programistów pracujących nad rozwojem frameworka, określające w jakim okresie czasowym poprawki lub nowe funkcjonalności mogą zostać dopuszczone do użytku przez społeczność korzystającą z tego szkieletu [4]. Opis całego procesu i plany wydawania oraz utrzymywania kolejnych wersji znajdują się na stronie frameworka [5].

W przypadku poprawek bezpieczeństwa, takie aktualizacje mogą występować średnio co miesiąc, a w zależności od złożoności błędu nawet codziennie. Wykonanie czynności aktualizacji jest zazwyczaj bez-

pieczne dla aplikacji i z reguły programista nie musi modyfikować własnego kodu źródłowego, aby dostosowywać się do wprowadzonych zmian. Takie aktualizacje są bardzo zalecane.

Kolejny przypadek aktualizacji określa standard wprowadzenia zazwyczaj nowych funkcjonalności do aplikacji. Zawierają one również poprawki błędów i wydawane są co pół roku: w maju oraz w listopadzie. Tego typu aktualizacje oprogramowania są dla programisty impulsem do podjęcia działań, polegających na dostosowywaniu swojego kodu do zachodzących zmian. Takie postępowanie uważane jest jako dbałość o bezpieczeństwo.

Ostatni możliwy przypadek aktualizacji komponentów frameworka Symfony, który zachodzi raz na dwa lata dotyczy aktualizacji całego frameworka. Zazwyczaj tego typu aktualizacja zawiera ogromne ilości zmian, które nie są wstecznie kompatybilne z poprzednimi wersjami. W tym przypadku dokonanie aktualizacji wymaga dużej uwagi i determinacji ze strony programisty oraz wgłębienia się w szczegóły wprowadzonych zmian kodu źródłowego aplikacji.

Istotną, a zarazem trudną do rozstrzygnięcia kwestią jest to, czy opłaca się aktualizować komponenty w celu zwiększenia szybkości działania aplikacji i jej optymalnego funkcjonowania, ponieważ to wiąże się często z dużymi kosztami związanymi z wysiłkiem i poświęconym czasem programisty. Z drugiej strony regularne aktualizowanie frameworka może sprawić, że istniejąca aplikacja będzie się skutecznie dostosowywać do ciągle zmieniających się potrzeb biznesu.

2. Cel i teza

Celem pracy jest zbadanie i ocena pod względem wydajności, a także wykorzystania pamięci RAM wybranych wersji frameworka Symfony począwszy od wersji 3.0 do 4.4 LTS włącznie [6].

Za tezę badawczą przyjęto stwierdzenie:

Aktualizacja komponentów szkieletu Symfony ma wpływ na wzrost ich wydajności, a tym samym przyczynia się do wzrostu wydajności całego frameworka.

3. Metoda badawcza

Eksperyment został zrealizowany na bazie dwóch scenariuszy. Pierwszy z nich dotyczył oceny wydajności poszczególnych komponentów wchodzących w skład badanych wersji Symfony. Jako miary do porównań w tym przypadku, użyto średniego czasu wykonania metody operującej na danym komponencie. Drugi scenariusz dotyczył wyznaczenia średnich czasów wykonania całej aplikacji testowej składającej się z 17 metod, z których każda wykorzystuje inny komponent. Dodatkowo w tym scenariuszu dokonano pomiarów zapotrzebowania aplikacji na pamięć fizyczną. Testy odbywały się na opracowanej do tego celu, prostej aplikacji testowej. Narzędziem wykorzystanym do realizacji pomiarów: wyznaczania czasów i użycia pamięci był framework PHPBench.

3.1. Aplikacja testowa

Na potrzeby badań, opracowano prostą aplikację w języku PHP, w dziesięciu wariantach - odpowiadających wybranym wersjom frameworka, składającą się z siedemnastu fragmentów kodu - metod w klasie, z których każda operuje na jednym, podstawowym komponencie Symfony. W tabeli 1 znajduje się lista wszystkich, wykorzystanych w aplikacji, komponentów oraz syntetyczny opis metod wykonujących proste działania na poszczególnych komponentach.

Tabela 1: Opis metod operujących na wybranych komponentach

Komponent	Opis metody
Asset	Utworzenie obiektu klasy Package oraz wywołanie metody getUrl, generującej adres URL zasobu.
Console	Uruchomienie instancji aplikacji, na bazie której wyszukiwane są dwie podstawowe komendy: list i help.
CssSelector	Utworzenie obiektu konwertera, uruchamiającego parser przekształcający wyrażenie CSS do XPath.
Dependency -Injection	Utworzenie i kompilacja fabryki kontenera.
Crawler	Utworzenie obiektu klasy Crawler, na podstawie dokumentu HTML, podanego jako parametr konstruktora.
DotEnv	Inicjalizacja obiektu klasy DotEnv oraz parsowanie treści w schemacie klucz=>wartość.
Expression -Language	Inicjalizacja obiektu klasy ExpressionLanguage oraz wywołanie dwóch poleceń: wykonania i kompilacji podanego wyrażenia.
Finder	Utworzenie obiektu klasy Finder wyszukującego pliki w aktualnym katalogu.
Http -Foundation	Wyodrębnienie superglobalnych zmiennych, przekształcenie ich do postaci obiektowej i przekazanie do zmiennej $request oraz przygotowanie obiektu klasy Response.
HttpKernel	Utworzenie kolekcji dostępnych kontrolerów, utworzenie żądania za pomocą elementów komponentu HttpFoundation, przygotowanie dyspozytora zdarzeń, analizatorów kontrolerów i argumentów i na ich podstawie utworzenie obiektu HttpKernela służącego do analizy żądania zawartego w obiekcie $request.
Options -Resolver	Utworzenie obiektu klasy OptionResolver, przekazanie domyślnych parametrów, analiza danych za pomocą metody resolve.
Process	Utworzenie obiektu klasy Process i przekazanie składowych wiersza poleceń jako zmiennej typu tablicowego.
Routing	Utworzenie kolekcji 1000 unikalnych adresów URL oraz pobranie wylosowanego obiektu klasy Route.
Serializer	Utworzenie obiektu klasy stdClass, normalizacja obiektu do postaci JSON oraz denormalizacja zmiennej tablicowej do klasy stdClass.
Stopwatch	Utworzenie obiektu klasy Stopwatch, uruchomienie stopera, zatrzymanie stopera i wyznaczenie interwału czasowego
Validator	Utworzenie obiektu klasy Validator, przekazanie wartości oraz tablicy ograniczeń, rzutowanie zmiennej do postaci znakowej.
Yaml	Użycie dwóch metod: do przekształcenia postaci tekstowej do postaci zmiennej oraz w drugą stronę.

3.2. Narzędzia badawcze

W podrozdziale 3.1 opisane zostały metody, z których każda operuje na jednym komponencie frameworka Symfony i wykorzystuje dostępne możliwości tych komponentów. Metody te zostały przygotowane do celów badawczych – do przeprowadzenia pomiarów

czasów ich wykonywania się. Do analizy wydajności wykorzystano framework PHPBench [7], który został również napisany w języku PHP. Umożliwia on konstruowanie scenariuszy wydajnościowych w celu testowania całych aplikacji lub wybranych fragmentów kodu. Do podstawowych właściwości narzędzia PHPBench należą:

- przebiegi – cykl (liczba) kolejnych wykonań wybranego fragmentu kodu,
- iteracje – liczba powtórzeń całego testu,
- izolacja procesu – uruchamianie testów w osobnych procesach,
- raportowanie – generowanie raportów w postaci kodu XML i przekształcenie go do czytelnej postaci,
- monitoring użycia pamięci fizycznej.

W przykładzie 1 (Listing 1) przedstawiono dwa polecenia uruchomienia testu wydajnościowego wykorzystującego PHPBench.

Listing 1: Polecenia uruchomienia testów wydajnościowych dla badanej wersji szkieletu Symfony

```
./vendor/bin/phpbench
run .\components\SymfonyXX\benchmarks\ --
bootstrap=.\components\SymfonyXX\vendor/autoload.p
hp --iterations=50 --revs=50 --report=aggregate --
dump-file=raporty/slow/symfonyXX.xml;
./vendor/bin/phpbench report --report=aggregate --
output=html --file=raporty/slow/symfonyXX.xml >
raporty/slow/symfonyXX.html;
```

Pierwsza komenda uruchamia test danej (XX) wersji Symfony, wykorzystującej opisaną w rozdziale 3 aplikację symulującą użycie wybranych komponentów. Każdą metodę podczas jednego testu wywołano 50 razy, co dało w sumie 2500 pomiarów czasów ich wykonań, ponieważ cały test powtarzano 50-krotnie. Dla tak ustawionej liczby powtórzeń, PHPBench wyznaczył średnie czasy trwania pojedynczego wykonania się każdej metody wchodzącej w skład aplikacji testowej. Za pomocą tego narzędzia obliczono również średni czas trwania całego testu dla poszczególnej wersji aplikacji testowej. Dodatkowo framework PHPBench umożliwił wykonanie pomiarów zapotrzebowania aplikacji na pamięć RAM.

Druga komenda, przedstawiona w przykładzie 1 uruchamia generator raportu, który przekształca otrzymane rezultaty z postaci XML do postaci HTML i prezentuje je w formie tabelarycznej.

3.3. Środowisko testowe

Stanowisko badawcze, wykorzystane do przeprowadzenia testów, przedstawia tabela 2.

Tabela 2: Opis stanowiska badawczego

Nazwa	Wartość
System operacyjny	Microsoft Windows 10 Home
Procesor	Intel(R) Core(TM) i7-4700HQ CPU @ 2.40GHz
Dysk	Crucial_CT250MX200SSD3 (250 GB, SATA-III)
Karta graficzna	Intel(R) HD Graphics 4600 (1 GB)
Pamięć RAM	2 x 8 GB DDR3-1600 DDR3 SDRAM
Karta sieciowa	Intel(R) Dual Band Wireless-AC 7260

4. Wyniki

Do badań wyselekcjonowano 17 komponentów dla dziesięciu wersji frameworka Symfony. Po wstępnej analizie otrzymanych wyników dokonano wyboru sześciu komponentów, w których od wersji 3.0 do 4.4 zaszły istotne zmiany, które w dużej mierze wpłynęły na ich wydajność. Wyniki dla tych komponentów są przedstawione w postaci wykresów, na których oś Y reprezentuje średni czas wykonania się jednej z siedemnastu metod (w mikrosekundach), natomiast na osi X znajdują się oznaczenia kolejnych wersji frameworka.

Rysunki 1-6 prezentują wyniki, dla wybranych wersji frameworka, średniego czasu wykonywania się kodu zawartego w pojedynczej metodzie operującej na danym komponencie. Z kolei rysunki 7 i 8 przedstawiają sumaryczne wyniki odnoszące się do:

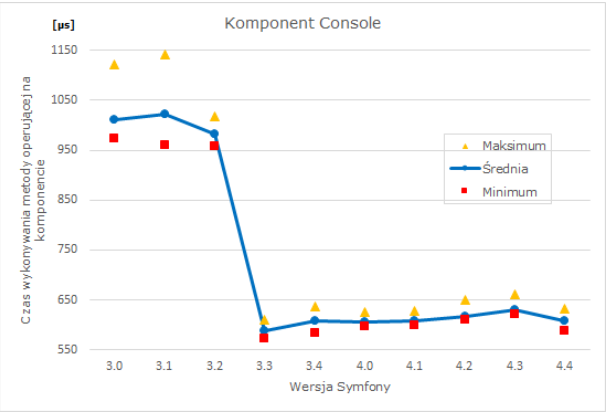
- średniego czasu wykonania się aplikacji testowej (wszystkich fragmentów kodu) dla poszczególnych wersji Symfony,
- średniego zapotrzebowania na pamięć RAM, podczas wykonywania aplikacji testowej zawierającej komponenty z poszczególnych wersji szkieletu programistycznego.

4.1. Wydajność wybranych komponentów

Komponent Console służy do tworzenia aplikacji, które są uruchamiane z poziomu wiersza poleceń. Aplikacje wykorzystujące ten komponent zachowują się tak, jakby były napisane w bash'u lub shell'u, dlatego jest on szczególnie przydatny do tworzenia synchronicznych lub asynchronicznych zadań uruchamianych bez użycia interfejsu graficznego.

Listing 2: Metoda testowa dla komponentu Console

```
final public function benchConsoleComponent(): void
{
  $application = new Application();
  $application->find('list');
  $application->find('help');
}
```



Rysunek 1: Wydajność komponentu Console

Na rysunku 1, od wersji 3.3 frameworka Symfony, widoczny jest znaczny wzrost wydajności komponentu Console. Trudno jest wytłumaczyć tę skokową poprawę, ze względu na brak oficjalnych informacji dotyczących

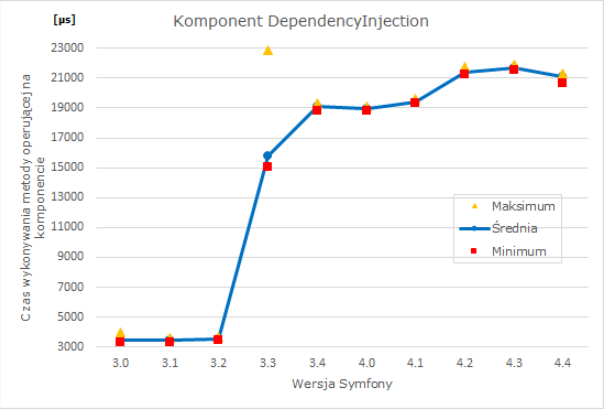
wprowadzanych ulepszeń odnoszących się do kwestii optymalizacyjnych.

Jednym z najważniejszych komponentów frameworka Symfony jest DependencyInjection. Można powiedzieć, że ten komponent jest mózgiem działania każdej aplikacji internetowej wykorzystującej mechanizm wstrzykiwania zależności. Wiąże on dostępne klasy oraz interfejsy (zarówno te napisane przez programistę jak i zewnętrzne biblioteki) i rejestruje je jako serwisy. Dzięki temu komponentowi zamiast wielokrotnego tworzenia obiektu tej samej klasy w różnych miejscach dla danego procesu następuje wstrzykiwanie zależności poprzez konstruktor lub za pośrednictwem dowolnej metody dostępowej, przekazującej argumenty do obiektu. Komponent ten implementuje interfejs PSR-11, który określa standard działania wielu aplikacji.

W związku z tym jest on bardzo ważny w podstawowym funkcjonowaniu każdej opracowanej aplikacji i dlatego jego wydajność ma kluczowe znaczenie.

Listing 3: Metoda testowa dla komponentu DependencyInjection

```
final public function
    benchDependencyInjectionComponent(): void
{
 $containerBuilder = new ContainerBuilder();
 $containerBuilder->register(__CLASS__,
                __CLASS__)->setPublic(true);
 $containerBuilder->setParameter('key', 'value');
 $containerBuilder->compile();
 $containerBuilder->get(__CLASS__);
}
```



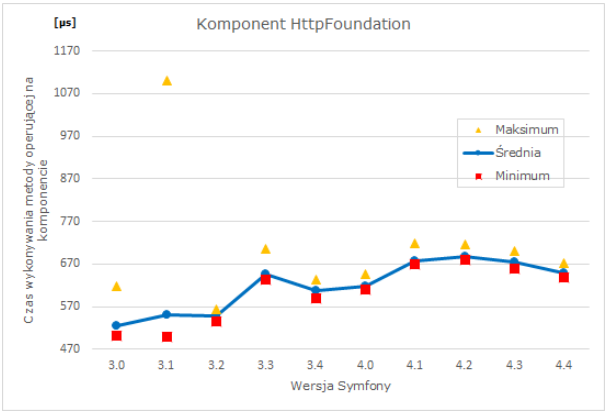
Rysunek 2: Wydajność komponentu DependencyInjection

Widoczny na rysunku 2, od wersji 3.3 duży spadek wydajności tego komponentu, związany jest z trwającymi bez przerwy pracami związanymi z dodawaniem nowych funkcjonalności czy wprowadzaniem usprawnień, których celem jest eliminacja błędów i poprawa bezpieczeństwa.

Kolejnym, komponentem poddanym testom, jest HttpFoundation. Definiuje on w sposób zorientowany obiektowo żądania i odpowiedzi związane z protokołem HTTP. Komponent ten zastępuje obiektami w PHP zmienne superglobalne takie jak: $_GET, $_POST, $_FILES, $_SERVER, $_COOKIE, $_SESSION.

Listing 4: Metoda testowa dla komponentu HttpFoundation

```
final public function benchHFComponent(): void
{
 $request = Request::createFromGlobals();
 $response = new Response(
             'Content',
             Response::HTTP_OK,
             ['content-type' => 'text/html']
             );
 $response->prepare($request);
}
```



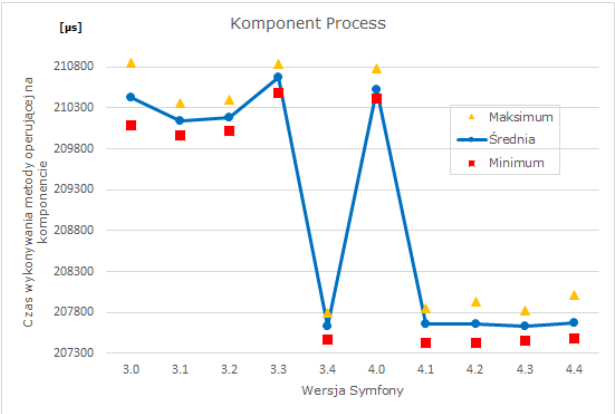
Rysunek 3: Wydajność komponentu HttpFoundation

HttpFoundation to bardzo ważny i w związku z tym stale poprawiany komponent. Usprawnianiu podlegają głownie aspekty bezpieczeństwa, co jednak negatywnie przekłada się na jego wydajność (rysunek 3).

Process to kolejny istotny komponent Symfony, który odpowiada za pośrednictwo pomiędzy językiem PHP, a systemem operacyjnym, na którym został uruchomiony. Zastępuje on typowe funkcje PHP ubogie w funkcjonalności dostępne dla programisty. Ważne jest to, że umożliwia on wykonywanie takich samych poleceń bez względu na system operacyjny, co tym samym daje programiście większą elastyczność pracy. Kolejną kluczową cechą tego komponentu jest możliwość uruchomienia procesu w trybie strumieniowania, dla którego w czasie rzeczywistym można odbierać informacje podczas wykonywania polecenia.

Listing 5: Metoda testowa dla komponentu Process

```
final public function benchProcessComponent(): void
{
 $process = new Process(['ls']);
 $process->run();
}
```



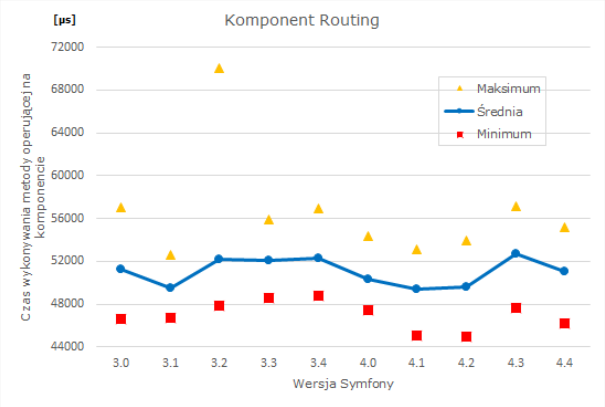
Rysunek 4: Wydajność komponentu Process

Process jest interesującym przykładem komponentu, w którym po skokowym wzroście wydajności w wersji 3.4, a następnie dużym spadku i powrotu do wyników sprzed wersji 3.4, nastąpiła znaczna poprawa i stabilizacja wydajności. Ten wyraźny wzrost wydajności był wynikiem wprowadzenia poważnych zmian, polegających na przykład na rezygnacji z używania jednej z funkcji PHP. Dzięki wsparciu społeczności, w wersji 4.1, zostały wprowadzone kolejne poprawki, które poprawiły działanie i wydajność tego komponentu.

Komponent Routing mapuje żądania HTTP do zestawu odpowiednich zmiennych opisanych przez programistę. Umożliwia on zbudowanie systemu routingu dla aplikacji internetowych, w których każdy unikalny adres URL zostanie powiązany z odpowiednim fragmentem metody kontrolera w architekturze MVC.

Listing 6: Metoda testowa dla komponentu Routing

```
final public function benchRoutingComponent(): void
{
  $routes = new RouteCollection();
  for($i=self::ROUTE_MIN;$i<=self::ROUTE_MAX;
$i++)
  {
    $routes->add('route_'.$i, new Route('/route_'.$i));
  }
  $matcher=new UrlMatcher($routes,
                    new RequestContext('/'));
  $matcher-
>match('/route_'.random_int(self::ROUTE_MIN,
                        self::ROUTE_MAX));
}
```



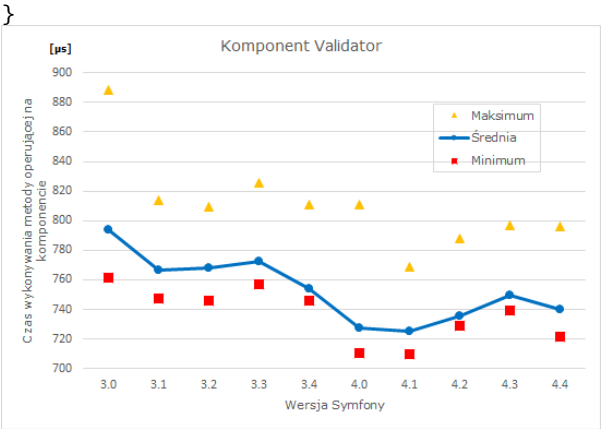
Rys. 5. Wydajność komponentu Routing

Rysunek 5 ukazuje zmiany zachodzące w wydajności komponentu Routing w kolejnych wersjach frameworka. Widoczny wzrost wydajności dla wersji 4.0, 4.1, 4.2 ma związek z zastosowaniem optymalizacji w jednej z bibliotek, która niewłaściwie implementuje część procesu routingu. Koncepcja polegająca na zrezygnowaniu z dopasowywania wyrażeń regularnych jedno po drugim, na skompilowanie ich w jedno duże wyrażenie regularne, wymaga ostatecznie tylko jednego dopasowania [8, 9].

Komponent Validator służy do sprawdzania poprawności właściwości klas, wartości zwracanych przez metody dostępowe oraz całych obiektów, za pomocą zdefiniowanych reguł, które mogą być definiowane za pomocą kodu PHP, adnotacji, plików: YAML, XML lub bezpośrednio przy użyciu prostych obiektów zawierających logikę walidacji.

Listing 7: Metoda testowa dla komponentu Validator

```
final public function benchValidatorComponent(): void
{
 $validator = Validation::createValidator();
 $violations = $validator->validate(
            'Michał',
            [new Length(['min' => 10]),
             new NotBlank(),
            ]);
 (string) $violations;
}
```

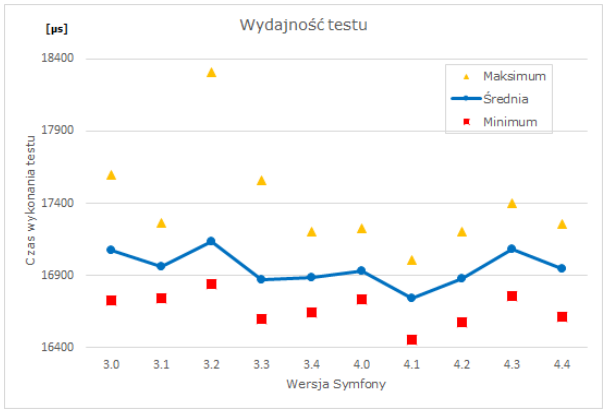


Rys. 6. Wydajność komponentu Validator

Rysunek 6 pokazuje, że wydajność komponentu Validator wzrosła w wersjach 4.0 i 4.1 frameworka, a następnie spadła w kolejnych wersjach. We wprowadzanych wydaniach Symfony, główne zmiany w obrębie tego komponentu koncentrowały się na wprowadzaniu coraz większej liczby nowych reguł, które dawały programistom szeroką bazę możliwości walidacji.

4.2. Wydajność szkieletu programistycznego

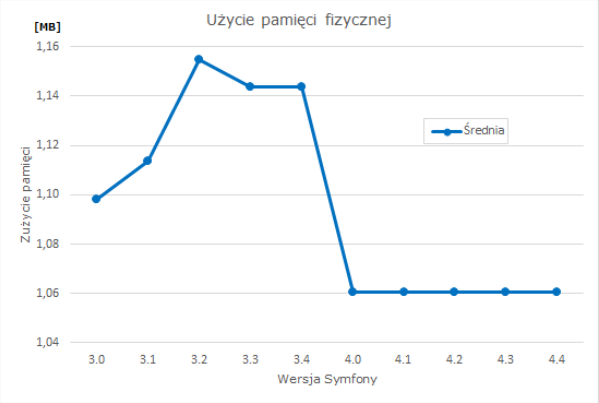
Ocena poszczególnych wersji szkieletu Symfony została dokonana na podstawie dziesięciu aplikacji, z których każda była zbudowana z komponentów danej wersji frameworka. Na tej podstawie wyznaczono sumaryczne średnie czasy wykonań, które zostały pokazane na rysunku 7.



Rys. 7. Średni czas trwania wykonywania się testów dla badanych wersji komponentów (wersji szkieletu programistycznego Symfony)

Dodatkowo wykonano pomiary zapotrzebowania na pamięć fizyczną poszczególnych wersji aplikacji (rysunek 8).

Z rysunku 7 wynika, że spośród dziesięciu wersji szkieletu Symfony najlepszą pod względem wydajności okazała się wersja 4.1. Potwierdza to osiągnięty najmniejszy średni czas wykonania aplikacji testowej. Uzyskany rezultat wynika z wysokiej wydajności komponentów Routing i Validator. Ponadto istotną informacją jest to, że zapotrzebowanie aplikacji na pamięć wyraźnie spadło od wersji 4.0 frameworka i w kolejnych wersjach utrzymywało się na takim samym poziomie (rysunek 8).



Rys. 8. Średnie użycie pamięci RAM przez aplikację testową

5. Wnioski

Przeprowadzone badania wykazały różnice wydajnościowe pomiędzy analizowanymi wersjami. Zmiany wprowadzane w kolejnych wersjach Symfony wiązały się z dodawaniem kolejnych tysięcy, a nawet dziesiątek tysięcy linii kodu. Zwiększanie się objętości frameworka, poprzez dodawanie nowych funkcjonalności, poprawek i zabezpieczeń skutkowało spadkiem jego wydajności. Z drugiej jednak strony wprowadzanie nowych rozwiązań i usprawnień miało pozytywny wpływ na kwestie szybkość działania aplikacji opartych na tym szkielecie. Z rysunku 7 wynika, że do wersji 4.1 wydajność Symfony rosła, a w następnych wersjach spadała. Biorąc pod uwagę zapotrzebowanie na pamięć fizyczną, należy przyznać, że nowe wersje frameworka (od 4.0) są coraz bardziej wydajne (rysunek 8). W związku z tym nie można jednoznacznie potwierdzić, postawionej na początku pracy tezy, że aktualizacja komponentów przyczynia się do wzrostu ich wydajności oraz wzrostu wydajności całego frameworka.

Na podstawie przeprowadzonych badań, okazało się, że najlepszą wersją Symfony, rozpatrywaną w niniejszej pracy, jest ta oznaczona numerem 4.1. To właśnie w tej wersji autorzy wprowadzili nowy mechanizm obsługujący szybszą analizę adresów URL [8, 9]. Późniejsze wersje uzyskały gorsze wyniki, co można powiązać z wprowadzeniem do nich nowych możliwości i poprawek bezpieczeństwa.

Bezpieczeństwo ma fundamentalne znaczenie dla każdej aplikacji internetowej jednakże utrzymywanie jego wysokiego poziomu jest często realizowane kosztem mniejszej wydajności. Pomimo pogorszenia tej wydajności systematyczna aktualizacja komponentów jest konieczna ze względu na potrzebę zapewnienia coraz większego stopnia bezpieczeństwa, ale powinna ona uwzględniać potrzeby i specyfikę danej aplikacji.

Literatura

[1] PHP Benchmarks, http://www.phpbenchmarks.com/en/ [03.12.2019]

[2] Symfony, https://symfony.com/components [04.12.2019]

[3] Zaninotto F., Potencier F. The Definitive Guide to Symfony. Apress, 2007.

[4] Coding Standards, https://symfony.com/doc/current/contributing/code/standards.html [04.12.2019]

[5] Symfony Releases, https://symfony.com/releases [04.12.2019]

[6] https://symfony.com/blog/ [04.12.2019]

[7] PHPBench's documentation, https://phpbench.readthedocs.io/en/latest/ [04.12.2019]

[8] Fast request routing using regular expressions, https://nikic.github.io/2014/02/18/Fast-request-routing-using-regular-expressions.html [15.12.2019]

[9] Making Symfony's Router 77.7x faster, https://medium.com/@nicolas.grekas/making-symfonys-router-77-7x-faster-1-2-958e3754f0e1 [15.12.2019]