JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 17 (2020) 384–389
Received: 25 October 2020
Accepted: 8 November 2020

Performance comparison of web services using Symfony, Spring, and Rails examples

Porównanie wydajności serwisów webowych na przykładzie Symfony, Spring i Rails

Patryk Lubartowicz*, Beata Pańczyk

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

Abstract
The article presents the results of a comparative analysis of web application frameworks for Java, PHP and Ruby. The most popular programming frameworks for each language were used for the research: Spring, Symfony and Ruby on Rails. In each of the frameworks the REST and SOAP web services were prepared and used to measure the request execution time. Measurements were made using Postman and SoapUI tools. The tests results showed that Spring is the fastest way to handle requests.

Keywords: Spring; Symfony; Ruby on Rails; performance

Streszczenie
W artykule przedstawiono rezultaty analizy porównawczej szkieletów aplikacji internetowych dla języków Java, PHP oraz Ruby. Do badań zastosowano, najbardziej popularne dla każdego języka, szkielety programistyczne: Spring, Symfony i Ruby on Rails. W każdym z frameworków przygotowano aplikacje testowe typu REST i SOAP, wykorzystane do testów pomiaru czasu wykonywania żądań. Pomiary wykonano za pomocą narzędzi Postman i SoapUI. Wyniki badań wykazały, że najszybciej żądania obsługiwane są w aplikacjach Spring.

Słowa kluczowe: Spring; Symfony; Ruby on Rails; wydajność

*Corresponding author
Email address: **patryk.lubartowicz@pollub.edu.pl** (P. Lubartowicz)

1. Wstęp

Możliwość uruchomienia tej samej aplikacji z poziomu przeglądarki z dowolnego urządzenia stała się codziennością. Wymiana danych i integracja z zewnętrznymi systemami to obecnie element wszystkich złożonych aplikacji. Aby taka wymiana była możliwa, stosowane są konkretne reguły i formaty komunikacji. Zasady wymiany danych określają protokoły komunikacji, czy odpowiednio usystematyzowane struktury danych. Najbardziej rozpowszechnione jest podejście oparte na SOAP (ang. Simple Object Access Protocol) lub REST (ang. REpresentational State Transfer) [1]. Protokół SOAP umożliwia komunikację pomiędzy aplikacjami w języku XML. Ciągle jest szeroko używany, z uwagi na standaryzację W3C, bezpieczeństwo i prostą kontrolę nad zawartością przekazanych danych. REST jest stylem architektonicznym, który definiuje format przesyłanych danych, jako element standaryzacji protokołu HTTP. Używany jest ze względu na elastyczność, szybkość i prostotę. Nie jest to protokół, ale usługa działająca w oparciu o protokół HTTP.

Dostępność wielu formatów danych zdecydowanie przemawia na korzyść REST. SOAP umożliwia komunikację jedynie z użyciem XML. Ograniczeniem dla usługi REST jest jeden protokół – HTTP, podczas gdy SOAP może działać z protokołami HTTP, SMTP, UDP itp.

Na rynku znajduje się wiele rozwiązań, które usprawniają wytwarzanie usług sieciowych w oparciu o oba wspomniane rozwiązania. Językami najczęściej wybieranymi są tu Java, C#, PHP i Ruby [2]. Każdy z języków posiada narzędzia przyspieszające wytwarzanie aplikacji.

2. Cel i obiekt badań

Celem badań jest analiza wydajności usług sieciowych wykonanych za pomocą trzech popularnych szkieletów programistycznych Spring (Java), Symfony (PHP) i Rails (Ruby) [3].
W artykule podjęto próbę odpowiedzi na pytanie:
W jakim stopniu na czas obsługi żądania wpływa zastosowany język programowania i szkielet programistyczny, w którym usługa została zaimplementowana?

Obiektem badań były trzy aplikacje przygotowane w wybranych frameworkach. Każda aplikacja posiadała dwa typy serwisów: jeden oparty o protokół SOAP [4] i drugi wykorzystujący architekturę REST [5]. Każdy z serwisów został odpowiednio zabezpieczony: protokół SOAP za pomocą rozszerzenia WSS (ang. Web Services Security), a REST przy pomocy JWT (ang. JSON Web Token). Wprowadzone zabezpieczenia miały na celu odwzorowanie rzeczywistych warunków, w których pracują serwisy webowe w Internecie.

3. Przegląd literatury

Po przeanalizowaniu dostępnych źródeł, można wnioskować, że większość prac skupia się na formach wytwarzania serwisów webowych, na dalszym miejscu pozostawiając szybkość działania, która uzależniona jest od obranych metod i stopnia komplikacji wykonywanych zadań.

Artykułem, który najbardziej koncentruje się na szybkości przetwarzania żądań jest *Performacje Evaluation of RESTfull Web Services and SOAP/WSDL Web Services* [6]. Opisuje on badania, które polegały na obliczeniu czasu wykonania żądań za pomocą aplikacji zainstalowanej na smartfonie. Brano tu pod uwagę wielkość oraz czas odpowiedzi z serwera na urządzeniu mobilnym. Wnioski na podstawie przeprowadzonych badań jasno wskazują na przewagę serwisu webowego opartego o REST.

Książka *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services* [7] koncentruje się na opisaniu metod wytwarzania serwisów webowych i podaje rozwiązania znanych problemów. Autor wskazuje kiedy warto korzystać z danego typu serwisu webowego.

Artykuł *Performance comparison of selected web service development technology in web applications* [8] przedstawia problem podobny do przedstawianego w niniejszej pracy. Autor skupia się tam jednak tylko na narzędziach dostosowanych do wytwarzania usług webowych związanych z jednym językiem programowania. Podane tam wnioski również potwierdzają przewagę usługi REST w tworzeniu serwisów webowych.

Praca *Performance and usage comparison between REST and SOAP web services [9]* przedstawia proste badanie wykonane w oparciu o zebranie czasów żądań z API dostarczanego przez Sitatech. Uruchomiony lokalnie serwis webowy był odpytywany przez wcześniej przygotowaną aplikację napisaną w C#, która służyła do zbierania danych o czasie wykonania polecenia. Zliczała ona czas od momentu tworzenia żądania do otrzymania odpowiedzi, bez jej obróbki. Analiza wyników wykonana przez autora pracy wykazała, że wykorzystanie architektury REST jest wydajniejsze w porównaniu do SOAP i zaleca się jej wykorzystanie.

W pracy *Comparing Performance of Web Service Interaction Styles: SOAP vs. REST* [10] skupiono się na zbadaniu wydajności serwisu webowego z uwzględnieniem napływu wielu zapytań jednocześnie. Pod uwagę wzięto czas odpowiedzi żądania oraz przepustowość dla sieci połączonej kablem i bezprzewodowo. Utworzone zostały dwa oddzielne serwisy – REST i SOAP – pobierające dane z tej samej przygotowanej bazy. Dodatkowo utworzono program kliencki komunikujący się obydwoma sposobami, który służył do zbierania czasów z wykonanych żądań. Czasy zliczane były od momentu wywołania metody wykonującej żądanie, do czasu przetworzenia odebranych danych. Wykonana przez autora analiza wyników podkreśla przewagę REST dla szybkości wykonywania żądań i łatwość jego przetwarzania, oraz ma większą przepustowość przesyłania danych.

Artykuł *Analiza możliwości współpracy aplikacji mobilnych z usługami sieciowymi typu REST i Web Service* [11] skupia się na porównaniu szybkości działania bibliotek dla aplikacji mobilnych z systemem Android. Obsługuje ona komunikację z serwisem webowym napisanym przez autora w języku Java, który obsługuje dwa typy komunikacji: REST i SOAP. Przeprowadzone badanie koncentruje się na czasie pracy samych bibliotek. Czas ten obliczono poprzez odjęcie czasu całkowitego wykonania żądania od sumy czasu odpowiedzi usługi oraz czasu odpowiedzi sieci. Wnioski wyciągnięte przez autora skupiają się na wyliczonym przez niego parametrze, jednak dane na temat czasu przetwarzania żądania po przechwyceniu go w ruchu sieciowym przedstawiają też istotne informacje.

4. Metoda badań

Badania zostały przeprowadzone poprzez wykonywanie 100 powtórzeń dla czterech scenariuszy testowych przedstawionych w tabeli 1. Jednakowe scenariusze wykonano dla metody wymiany danych SOAP i REST.

Tabela 1: Scenariusze testowe

Oznaczenie	Scenariusz
1	Pobranie pojedynczego obiektu reprezentującgo dane przetworzone w zapytaniu
2	Pobranie tablicy zawierającej zbiór 100 obiektów
3	Przesłanie obiektu do zapisu
4	Przesłanie obiektu do aktualizacji

Pojedynczą pobieraną w żądaniu daną jest wartość rocznego obrotu. Otrzymano ją z zapytania sumującego wartości faktur z kolumny *Total* w tabeli *Invoice* w podanym w żądaniu roku. Obiektem przesyłanym w żądaniach do zapisu i aktualizacji była encja *Track* o strukturze pokazanej na Rysunku 1.

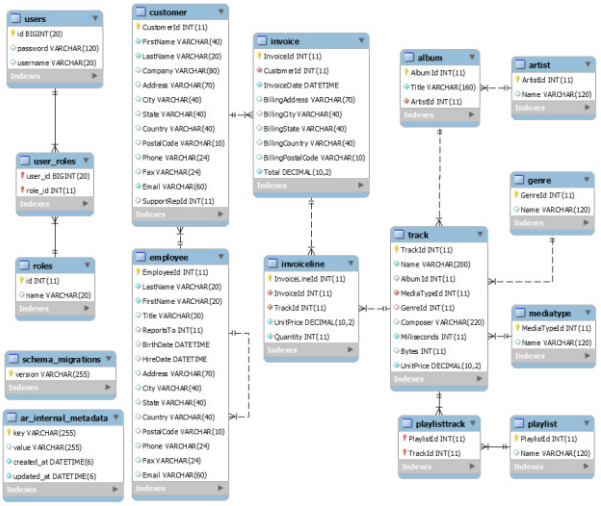
Czasy wykonania 100 żądań dla każdego ze scenariuszy mierzono programami SoapUI dla SOAP i Postman dla REST. Prezentowane wyniki uśredniono (średnia arytmetyczna). Ilość zajmowanej pamięci operacyjnej mierzono za pomocą monitora zasobów systemu Windows. Każda z aplikacji była uruchamiana oddzielnie. Badania prowadzone były na maszynie o następujących parametrach:

- płyta główna: M5A97 R2.0,
- procesor: AMD FX™-6300 4.21GHz,
- RAM: 16GB DDR3 1866MHz,
- system: Windows 10 x64 Pro.

5. Aplikacje testowe

Każda z aplikacji testuje po 4 żądania dla dwóch serwisów webowych: REST i SOAP. Wykonywane są one po uprzedniej autentykacji użytkownika za pomocą tokena JWT (ang. JSON Web Token) w przypadku REST, lub danych logowania przesłanych w sposób jawny w nagłówku WSS (ang. Web Socket Security) według specyfikacji opublikowanych przez OASIS dla SOAP. Aplikacje wykorzystują dane z pojedynczej kopii bazy danych Chinook [13], która jest dostępna na

licencji MIT i reprezentuje sklep z utworami muzycznymi w wersji cyfrowej (Rysunek 1). Na potrzeby projektu dodane zostały do niej tabele wymagane w procesie autentykacji użytkownika i inne niezbędne do pracy z danym frameworkiem.



Rysunek 1: Model bazy danych

5.1. Aplikacja Spring

Aplikacja wykonana została w języku Java v8. Podstawą projektu była paczka Spring Boot w wersji 2.2.4. Dodatkowo wykorzystano wsparcie bibliotek WSDL4J i JAXB2 oraz JJWT. Pierwsze dwie biblioteki wspomagają proces wytwarzania serwisu webowego w oparciu o protokół SOAP, natomiast trzecia służy implementacji zabezpieczeń dla serwisu REST. Aplikacja była uruchamiana z wiersza poleceń ze skompilowanego pliku .jar. Przykładowe fragmenty kodu do realizacji scenariusza 1 dla tej aplikacji w technologii REST i SOAP przedstawiają listingi 1 i 2.

Listing 1. Pobranie pojedynczej danej - REST w Spring

```
Controllers/InvoiceController.java:
@GetMapping("/total-sales-in-year")
@PreAuthorize("hasRole('USER')
 or hasRole('MODERATOR') or hasRole('ADMIN')")
@ResponseBody
public Float getYearSales(
@RequestParam
(value = "year", required = true)
Integer year
) { return invoicesService.getTotalSalesPerYear(year); }
```

Listing 2. Pobranie pojedynczej danej - SOAP w Spring

```
Controllers/Endpoints/InvoicEndpoint.java:
@PayloadRoot(namespace = NAMESPACE_URI, localPart =
"getYearSalesRequest")
@ResponsePayload public GetYearSalesResponse getYearSale(
@RequestPayload GetYearSalesRequest request
){
GetYearSalesResponse response =
new GetYearSalesResponse();
  Float amount =
invoicesService.getTotalSalesPerYear(
request.getYear().intValue()
);
  response.setAmount(amount);
  return response;
}
```

5.2. Aplikacja Symfony

Aplikacja napisana została w PHP v7.2. Projekt opierał się o szkielet projektu Symfony w wersji 5.0.8. W tym przypadku brakuje bibliotek wspomagających tworzenie serwisów webowych SOAP oraz REST. Jedynym dodatkowym pakietem był LexikJWTAuthenticationBundle, wykorzystany do zabezpieczenia wymiany danych metodą REST. Testy wykonywane były z serwera z wersją PHP 7.2.30 x64. Przykładowe fragmenty kodu do realizacji scenariusza 1 dla tej aplikacji przedstawiają listingi 3 i 4.

Listing 3. Pobranie pojedynczej danej - REST w Symfony

```
src/Controller/InvoiceController.php:
/**
   * @param InvoiceRepository $repository
   * @param $year
   * @return JsonResponse
 *                                    @Route("/sales/{year}",
name="invoice_year_sales_get",
methods={"GET"})
   */
public function getTotalSalesPerYear(InvoiceRepository $re-
pository, $year){
$amount = $repository ->
getTotalYearSales($year);
if (!$amount){
$data = [
'status' => 404,
'errors' => "No amounts in this year",
];
return $this->response($data, 404);
}
return new JsonResponse(
$amount,
Response::HTTP_OK
);
}
```

Listing 4. Pobranie pojedynczej danej – SOAP w Symfony

```
src/Service/Soap/TestService.php:
public function getYearSales($year){
     return $this->invoiceRepository
->getTotalYearSales($year);
}
```

5.3. Aplikacja Rails

Ta aplikacja utworzona została w języku Ruby v2.6. Wykorzystano szkielet projektu podstawowej aplikacji Ruby on Rails w wersji 6.0.3.1.

Do wsparcia implementacji dodano biblioteki: JWT, Wash Out oraz Nokogiri. Pierwsza służy zabezpieczeniu komunikacji poprzez REST, dwie następne wspierają wytwarzanie serwisu webowego korzystając z protokołu SOAP. Aplikacja była uruchamiana za pomocą komendy z wiersza poleceń.

Przykładowe fragmenty kodu do realizacji scenariusza 1 dla tej aplikacji w technologii REST i SOAP przedstawiają listingi 5 i 6.

Listing 5. Pobranie pojedynczej danej - REST w Ruby on Rails

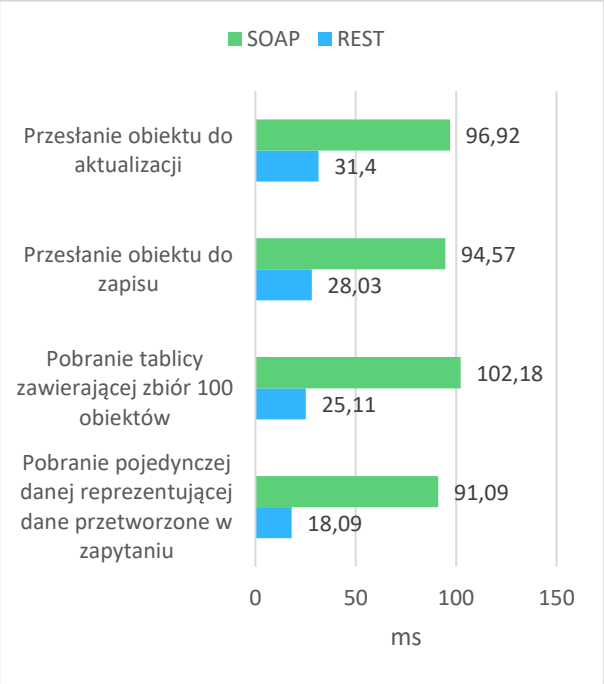
```
app/controllers/invoices_controller.rb:
def totalSalesInYear
  sales =
          Invoice.getYearSales(params[:year])[0]
  render json: sales["amount"].to_f
end
```

Listing 6. Pobranie pojedynczej danej - SOAP w Ruby on Rails

```
app/controllers/soap_controller.rb:
soap_action "getYearSales",
  :args => {:year => :integer},
  :return => :double
def getYearSales
  sales = Invoice.getYearSales(params[:year])[0]
  raise SOAPError, "No sales in this year"
          if sales["amount"] == nil
  render :soap => sales["amount"].to_f
end
```

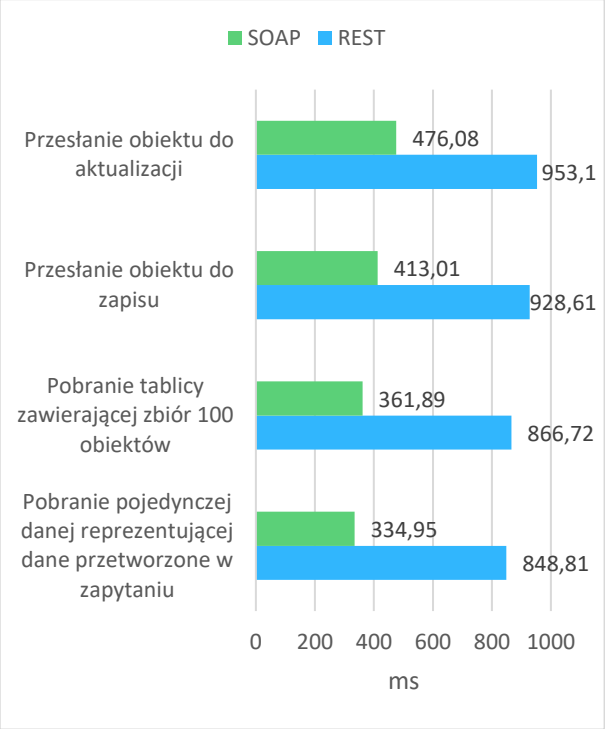
6. Wyniki badań

Na rysunku 2 przedstawiono czasy żądań dla obu serwisów webowych w aplikacji Spring. Widoczna jest duża różnica pomiędzy czasem wykonania żądań w przypadku REST i SOAP - dla SOAP czasy są nawet 3-4 razy dłuższe. Pomimo różnic, oba wyniki są akceptowalne i nie przekraczają 100ms (poza jednym wyjątkiem).



Rysunek 2: Zestawienie średnich czasów wykonania 100 żądań w aplikacji Spring dla REST i SOAP.

Rysunek 3 przedstawia porównanie czasów obsługi żądań dla aplikacji Symfony. W tym przypadku to protokół SOAP ma lepsze rezultaty w odniesieniu do czasu wykonywania żądań. Różnica pomiędzy czasami wynosi 2-krotność w przypadku przesłania obiektu do aktualizacji, w przypadku jego zapisu jest to wartość 2,25. Pobranie tablicy zawierającej 100 obiektów, 2,4 razy szybciej realizuje SOAP, a dla pojedynczej danej wykonuje to aż 2,54 razy szybciej niż REST. Może to być spowodowane brakiem wsparcia bibliotek ze względu na początkowe stadium rozwoju najnowszej wersji frameworka.

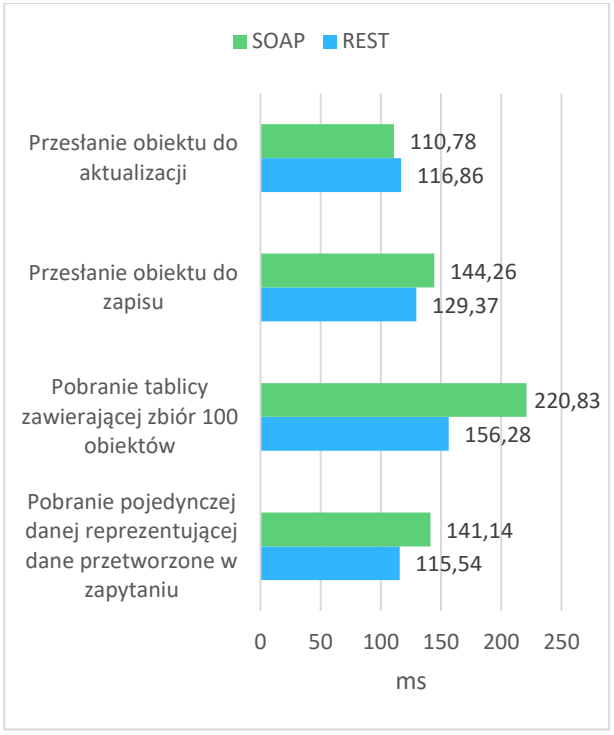


Rysunek 3: Zestawienie średnich czasów wykonania 100 żądań w aplikacji Symfony dla REST i SOAP.

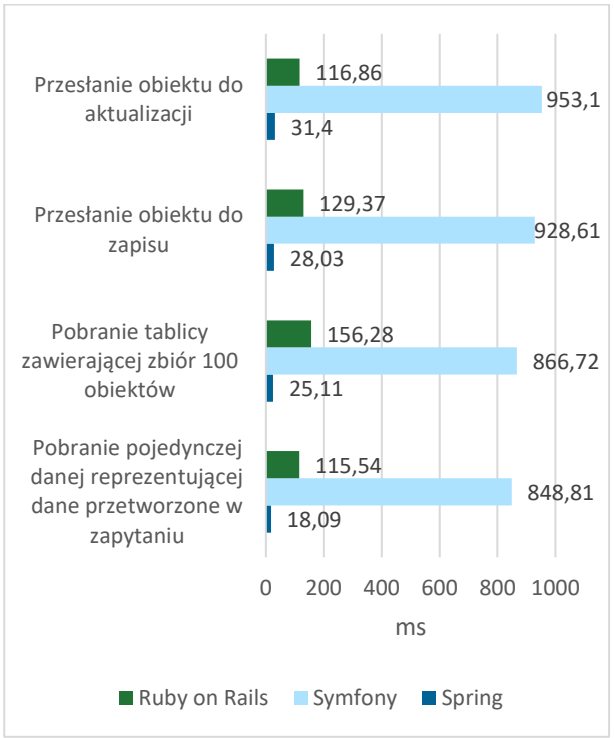
Rysunek 4 pokazuje średnie czasy dla obu serwisów webowych w Ruby on Rails. W tym przypadku czasy żądań (oprócz pobrania 100 obiektów) są zbliżone, ale generalnie wykorzystanie REST daje lepsze rezultaty. Należy zwrócić też uwagę na czas aktualizacji obiektu, który w przypadku REST prawie się nie różni od pobierania pojedynczej danej, a w przypadku SOAP jest znacznie mniejszy. Dzieje się tak z powodu zachowania w aplikacji informacji o wykorzystaniu danego rekordu. Wykonanie po raz kolejny tego samego zapytania o dany rekord, powoduje, że dane nie są pobierane ponownie, a odczytywane z wcześniej przygotowanych ciasteczek.

Rysunki 5 i 6 przedstawiają porównanie czasów wykonania żądań dla każdego scenariusza dla wszystkich aplikacji, odpowiednio dla REST i SOAP.

W przypadku REST wykresy pokazują wyraźną przewagę czasową dla aplikacji Spring. Czas pomiędzy najszybszym, a najwolniejszym wykonaniem żądań jest nawet 50 razy krótszy w porównaniu do aplikacji Symfony. W przypadku Rails wyniki są akceptowalne, na poziomie 115-156ms, odbiegając od rywala od około 4 do 6-krotności różnicy czasu.

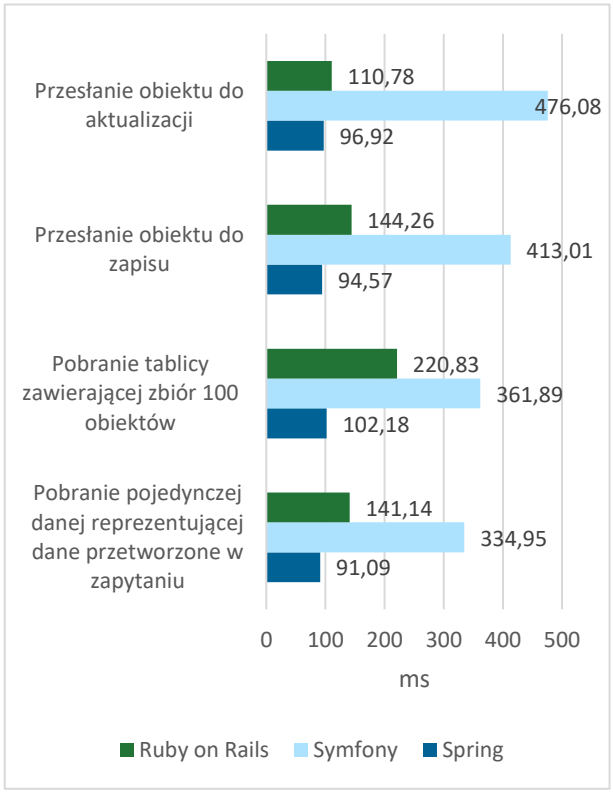


Rysunek 4: Zestawienie średnich czasów wykonania 100 żądań w aplikacji Ruby on Rails dla REST i SOAP.



Rysunek 5: Zestawienie wszystkich scenariuszy testowych we wszystkich badanych aplikacjach dla REST.

W porównaniu do różnic widocznych w technologii REST, w przypadku SOAP różnice nie są już tak bardzo duże. Najniższy czas jest około 5 razy mniejszy od najdłuższego średniego czasu wykonania żądania. Warto zwrócić szczególną uwagę na niski czas wykonania operacji przesłania obiektu do aktualizacji w przypadku Ruby on Rails. Jest on porównywalny z czasem dla aplikacji Spring.



Rysunek 6: Zestawienie wszystkich scenariuszy testowych we wszystkich badanych aplikacjach dla SOAP

Warto też zwrócić uwagę na ilość pamięci zajmowanej przez programy po wykonaniu testów (tabela 2).

Tabela 2: Ilość zajmowanej pamięci operacyjnej przez aplikacje

Aplikacja	Ilość zajmowanej pamięci
Spring	778 160 KB
Symfony	18 844 KB
Ruby on Rails	84 464 KB

Najmniej pamięci zajmuje aplikacja Symfony (niewiele ponad 18MB) co jest bardzo dobrym wynikiem, biorąc pod uwagę ilość zasobów w nowoczesnych serwerach. Podstawowa konfiguracja serwera PHP może zawierać limity dotyczące wykorzystywanej pamięci, choć w tym przypadku wynik nie osiągnął bariery aktualnie ustawianego 128MB. Drugie miejsce zajmuje Rails (około 84MB) a najgorzej wypada aplikacja Spring (aż 778MB, czyli około 43-razy więcej niż w przypadku Symfony). Wybór aplikacji Spring, pomimo najszybszej obsługi żądań, wymaga jednak rozważenia pod kątem dopasowania do posiadanych zasobów sprzętowych.

7. Wnioski

Wykonana analiza wykazała, że wydajność usług sieciowych, na przykładzie REST i SOAP, zależy od wybranego języka programowania i szkieletu programistycznego.

Wykorzystanie aplikacji Spring do utworzenia serwisu webowego pozwala uzyskać największą przepu-

stowość wymiany danych. Niestety, w tym przypadku trzeba liczyć się z większym zapotrzebowaniem na zasoby sprzętowe, potrzebne do uruchomienia i utrzymania serwisu.

W przypadku, gdy zasoby sprzętowe są ograniczone, a trochę dłuższy czas odpowiedzi jest akceptowalny – to zastosowanie Ruby on Rail jest dobrym wyborem. Aplikacja w Ruby potrzebuje 9 razy mniej pamięci operacyjnej w porównaniu do Spring.

Wybór najbardziej optymalnego rozwiązania wiąże się też z wyborem odpowiedniego serwisu webowego. W analizowanych przypadkach, rozwiązanie oparte o REST dało lepsze wyniki. Wynik jest zgodny z oczekiwanym. W publikacji [12] uzyskano analogiczne rezultaty, jednak bez analizy aplikacji Symfony.

Nie zawsze na pierwszym miejscu stawia się szybkość działania, a wykorzystanie SOAP jako ustandaryzowanego rozwiązania nie oznacza, że system będzie dużo wolniejszy, co widać na przykładzie Ruby on Rails.

Literatura

[1] T. Zientarski, M. Miłosz, M. Kamiński, M. Kołodziej: Applicability analysis of REST and SOAP web services, Informatyka, Automatyka, Pomiary W Gospodarce i Ochronie Środowiska, 7 (2017) 28-31.

[2] S. Cass, The 2017 top programming languages, IEEE Spectrum 2018.

[3] Statystyki dotyczące wykonywanych zawodów i wybieranych technologii oraz narzędzi StackOverflow, https://insights.stackoverflow.com/survey/2020, [15.09.2020].

[4] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S.R. Thatte, D. Winer, Simple object access protocol (SOAP) 1.1., W3C Note, 2000.

[5] M. Masse, REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces, O'Reilly Media, Inc. 2011.

[6] R. Digvijaysinh, PERFORMANCE EVALUATION OF RESTFUL WEB SERVICES AND SOAP / WSDL WEB SERVICES, International Journal of Advanced Research in Computer Science, 8 (2017) 415-420.

[7] R. Daigneau, Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services, Addison-Wesley, 2012.

[8] A. Gdula, M. Plechawska-Wójcik, Performance comparison of selected web service development technology in web applications, Journal of Computer Sciences Institute, 1 (2016) 14-19.

[9] J. Makkonen, Performance and usage comparison between REST and SOAP web services, 2017

[10] P. K. Potti, S. Ahuja, K. Umapathy, Z. Prodanoff, Comparing Performance of Web Service Interaction Styles: SOAP vs. REST, Proceedings of the conference on information systems applied research, (2012) 2167.

[11] M. R. Daraż, P. Kopniak, Analysis of the possibilities of cooperation of mobile applications with network services of the type REST and Web Service, Journal of Computer Sciences Institute, 11 (2019) 155-162.

[12] A. Soni, V. Ranga, API Features Individualizing of Web Services: REST and SOAP, International Journal of Innovative Technology and Exploring Engineering, 8 (2019) 664-671.

[13] Baza danych, https://github.com/lerocha/chinook-database, [03.10.2020].