JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 121-125
Received: 29 March 2021
Accepted: 04 April 2021

REST API performance comparison of web applications based on JavaScript programming frameworks

Porównanie wydajności aplikacji internetowych REST API opartych na szkieletach programistycznych JavaScript

Marcin Grudniak* , Mariusz Dzieńkowski

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

Abstract
The aim of the work was to compare two technologies for creating server applications based on the JavaScript programming language. For the purposes of the research, two test applications were created. The first one was built on the basis of the Express programming framework and the second one on the basis of the Hapi framework. The client part of both applications was prepared using the React library. The client and server parts communicated with each other by means of REST API – the universal HTTP interface. The client application sent requests to the server application which then performed basic operations on the MongoDB basis and returned the result. As part of the work, an experiment consisting of four scenarios was developed. In each scenario, a different type of data was taken into consideration: a string of characters, an array, an object and an array of objects. The research focused on the efficiency aspect – measuring the response time of requests during GET, POST, PUT and DELETE operations. The tests were performed on two computers and the measurements were made in two ways: using a single code embedded in test applications and using the Postman tool. The obtained results, after averaging and analyzing them allowed for the conclusion that the Express framework proved to be more efficient than Hapi due to the shorter response time of requests. Only in the scenario where operations with large datasets were performed was the response time of requests at a similar level.

Keywords: Express; Hapi; performance analysis; response time of requests

Streszczenie
Celem pracy było porównanie dwóch technologii do tworzenia aplikacji serwerowych opartych na języku programowania JavaScript. Na potrzeby badań utworzono dwie aplikacje testowe: pierwszą zbudowano na podstawie szkieletu programistycznego Express, a druga została wykonana na bazie szkieletu Hapi. Część kliencką obu aplikacji przygotowano za pomocą biblioteki React. Część kliencka i serwerowa komunikowały się ze sobą za pośrednictwem REST API - uniwersalnego interfejsu HTTP. Aplikacja kliencka wysyłała żądania do aplikacji serwerowej, która następnie wykonywała podstawowe operacje na bazie MongoDB i zwracała rezultat. W ramach pracy opracowano eksperyment składający się z czterech scenariuszy. W każdym scenariuszu operowano na innym typie danych: łańcuchu znaków, tablicy, obiekcie oraz tablicy obiektów. W badaniach skoncentrowano się na aspekcie wydajnościowym - pomiarze czasów obsługi żądań podczas operacji GET, POST, PUT i DELETE. Badania przeprowadzono na dwóch komputerach, a pomiary wykonano dwoma sposobami: za pomocą prostego kodu wbudowanego w aplikacje testowe oraz za pomocą narzędzia Postman. Uzyskane wyniki, po ich uśrednieniu i przeanalizowaniu pozwoliły na sformułowanie wniosku, że szkielet Express okazał się wydajniejszy niż Hapi, ze względu na krótsze czasy obsługi żądań. Tylko w scenariuszu, w którym wykonywano operacje na dużych zbiorach danych, czasy obsługi żądań były na podobnym poziomie.

Słowa kluczowe: Express; Hapi; analiza wydajności; czas obsługi żądań

*Corresponding author
Email address: marcin.grudniak@pollub.edu.pl (M. Grudniak)

1. Wstęp

Współcześnie do budowy aplikacji internetowych najczęściej wykorzystuje się szkielety programistyczne, które definiują strukturę aplikacji i dostarczają narzędzi do realizacji określonych zadań. Upraszczają one także proces programowania, wykorzystując dobre praktyki wypracowane przez społeczność programistów zaangażowaną w ich rozwój. Wybór optymalnej platformy programistycznej, przed którym stoją deweloperzy oprogramowania, dostosowanej do wymagań konkretnego przypadku nie jest łatwym zadaniem. Sytuację komplikuje nie tylko duży wybór szkieletów programistycznych dla danego języka oraz ich szybka ewolucja, ale także ciągłe powstawanie nowych rozwiązań lepiej realizujących podobne zadania. Podjęcie właściwych decyzji, co do wyboru technologii, może zaowocować wytworzeniem bardziej interaktywnych, przyjaznych oraz łatwych w obsłudze aplikacji. Zastosowanie odpowiedniego szkieletu może mieć wpływ na skrócenie czasu powstawania finalnego produktu, zmniejszenie nakładu pracy pracowników oraz obniżenie kosztów przedsięwzięcia informatycznego. Niemniej ważne kwestie związane z wyborem frameworka to także zapewnienie wysokiej jakość kodu oraz odpowiedniego poziomu bezpieczeństwa tworzonego oprogramowania [1]. W przypadku wytwarzania aplikacji internetowych,

z których będzie korzystało jednocześnie wielu użytkowników, również niezwykle ważnym zagadnieniem, które należy uwzględnić dobierając platformę programistyczną, jest odpowiednio wysoka wydajność i wynikająca z niej szybkość transmisji danych.

2. Prezentacja porównywanych technologii

Node jest środowiskiem uruchomieniowym, działającym poza przeglądarką i współpracującym z systemem operacyjnym. Ze względu na swoją szybkość i wykorzystanie języka JavaScript, jest obecnie jedną z wiodących platform do tworzenia aplikacji internetowych po stronie serwera. Aplikacje pracujące na tej platformie mają dostęp przez API do systemu operacyjnego, do jego systemu plików, bibliotek systemowych, uruchomionych procesów, w tym serwerów HTTP [2]. Jądro Node jest mocno ograniczone i tym samym wywołuje potrzebę zastosowania frameworka, niezbędnego do zadań porządkowych oraz zwiększenia produktywności i jakości aplikacji [3].

2.1. Szkielet programistyczny Express

Najpopularniejszym szkieletem dla aplikacji serwerowych operujących na platformie Node jest Express. Potwierdzają to różne serwisy publikujące aktualne rankingi [4-6]. Według raportu opublikowanego w serwisie stackoverflow w lutym 2020 roku szkielet ten był na pierwszym miejscu wśród frameworków serwerowych na platformę Node. Natomiast biorąc pod uwagę inne środowiska wytwarzania aplikacji webowych zajął on piątą pozycję, plasując się za jQuery, React, Angular oraz ASP.NET. Na Express wskazało 21,2% wszystkich respondentów, a wśród nich 29,9% profesjonalnych deweloperów oprogramowania [4]. W serwisie GitHub, biorąc pod uwagę szkielety środowiska Node, Express także lokuje się na pierwszej pozycji, mając przyznanych 52,5 tys. gwiazdek (stan na dzień 22.03.2021) [5].

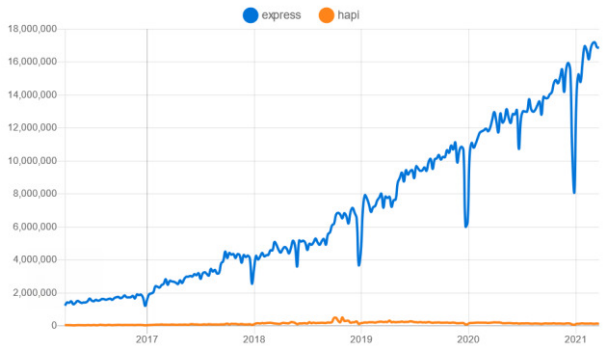
Wysokie notowania w rankingach wynikają z tego, że Express ma wiele zalet. Jest on wydajny, solidny, elastyczny, minimalny, dobrze przetestowany i posiada dużą społeczność. Poza tym Express jest dojrzałym, ciągle aktualizowanym, dobrze udokumentowanym i łatwym w użyciu frameworkiem. Został on zaprojektowany do tworzenia jednostronicowych, wielostronicowych i hybrydowych aplikacji, zapewniając solidny zestaw funkcji dla aplikacji internetowych i mobilnych. Bardzo dobrze nadaje się zarówno do dużych jak i małych projektów [7]. Na bazie tego szkieletu powstało wiele innych szkieletów m.in. KeystoneJS, Blueprint, NestJS i Locomotive [8]. Choć Express nie ma gotowego do użycia mechanizmu mapowania obiektowo-relacyjnego, to ma swobodę w łączeniu się z innymi technologiami, co ułatwia jego błyskawiczna konfiguracja. Express skutecznie rozwija backendowe części stosów MEAN oraz MERN z frontendowym szkieletem Angular lub biblioteką React oraz bazą danych MongoDB.

2.2. Szkielet programistyczny Hapi

Najważniejszą kwestią, która była brana pod uwagę podczas tworzenia frameworka Hapi, było stworzenie narzędzia zapewniającego duże wsparcie programistyczne dla dużych zespołów zarządzających wieloma zadaniami. Niemniej ważnym aspektem było dostarczenie bogatych i niezawodnych funkcji bezpieczeństwa. Z tego względu szkielet Hapi jest wykorzystywany przez programistów do tworzenia serwerów proxy, interfejsów REST API i innych aplikacji, dla których ważne są niezawodność i bezpieczeństwo. Duża liczba wbudowanych w ten szkielet wtyczek eliminuje potrzebę stosowania niepewnego oprogramowania pośredniego (middleware). Hapi powinien być pierwszym wyborem dla deweloperów wytwarzających bezpieczne, skalowalne aplikacje do obsługi mediów społecznościowych w czasie rzeczywistym [9]. Przykładem dużej skali serwisów opartych na tej technologii są PayPal, Disney i Macy's.

Hapi, będąc potężną i wysokowydajną platformą, pracującą w środowisku rozproszonym, charakteryzuje się lekkością i bogactwem funkcji. Zawiera m.in. wbudowaną funkcję buforowania, która ułatwia uruchamianie aplikacji czasu rzeczywistego wymagających dużej ilości danych pochodzących z wielu urządzeń.

Każdy szkielet programistyczny ma swoje mocne i słabe strony. Hapi jest dla Node bardziej abstrakcyjnym frameworkiem niż Express, którego kod wygląda bardziej jak natywny kod Node.js. Zaletami obu szkieletów są rozszerzalność i elastyczność. Pod względem popularności, Express jest zdecydowanie częściej używany niż Hapi i, jak pokazuje rysunek 1, jego popularność ciągle wzrasta. Poniższy wykres przedstawia liczbę pobrań pakietów w ciągu jednego tygodnia (oś Y) za pomocą menadżera npm za okres ostatnich pięciu lat (oś X).



Rysunek 1: Popularność szkieletów programistycznych Express i Hapi wyrażona liczbą pobrań pakietów na tydzień w okresie ostatnich pięciu lat [10].

3. Cel i zakres pracy

W pracy skoncentrowano się na kwestiach wydajnościowych dwóch szkieletów aplikacyjnych opartych na platformie Node. Wybór padł na Express - najpopularniejszy szkielet tej platformy oraz na Hapi – szkielet dedykowany dla dużych, jeśli chodzi o skalę, przedsięwzięć informatycznych.

W związku z tym, celem artykułu jest analiza porównawcza dwóch szkieletów programistycznych Express i Hapi, opartych na architekturze REST, umożliwiającej wygodną wymianę danych pomiędzy częścią serwerową i kliencką aplikacji. Kryterium porównawczym, które zostało wzięte pod uwagę podczas analiz, była wydajność czasowa. Motywem do podjęcia tego tematu była z jednej strony mała liczba artykułów opublikowanych w internecie dotyczących porównania tych dwóch szkieletów programistycznych, a z drugiej strony chęć porównania uzyskanych w ramach tej pracy wyników z rezultatami prezentowanymi przez innych autorów [11-13] .

4. Testowe aplikacje serwerowe

Główną różnicą między obydwoma szkieletami jest ich sposób obsługi żądań. W przypadku Hapi (Listing 1) wszystkie informacje przekazuje się w postaci obiektu, na który składają się metoda, ścieżka i obsługa [14]. W przypadku szkieletu Express (Listing 2), hierarchia jest następująca: najpierw, jako parametr funkcji, podaje się ścieżkę, a następnie metodę HTTP z dwoma parametrami, spośród których drugi zwraca odpowiedź [15].

Listing 1: Fragment kodu aplikacji odpowiedzialny za obsługę żądania dodania nowego dokumentu zrealizowanego w Hapi

```
server. route({
    method: "POST",
    path: "/add/{length}/{size}",
    handler: (req, res) => {
        return obj(req)
            .save()
            .then((err, res) => {
                if(err){
                    return err;
                }
                return res;
            });
    },
});
```

Listing 2: Obsługa żądania dodawania nowego dokumentu w Express

```
router. route("/add/:length/:size").post((req, res) => {
    obj(req)
        .save()
        .then((post) => {
            res.json(post);
        })
        .catch((err) => {
            console.error(err);
            res.status(500).json({ error: err.code });
        });
});
```

5. Metoda badań

5.1. Środowisko badawcze

W celu większej wiarygodności wyników, badania zostały przeprowadzone na dwóch stacjach badawczych, różniących się konfiguracją (Tabela 3).

Tabela 1: Konfiguracja środowisk badawczych

	Komputer 1	Komputer 2
System operacyjny	Windows 10	Ubuntu 18.04
Procesor	Intel Xeon X5690 3.47GHz	Intel i3-4005U
Pamięć RAM	12 GB	4 GB
Dysk	SSD	SSD
Łącze	LAN	WiFi 2.4GHz

Na obu komputerach zostały zainstalowane aplikacje testowe, realizujące obsługujące żądania wykonujące cztery podstawowe operacje na bazie danych, za pośrednictwem metod REST API [16] odpowiedzialnych za tworzenie, odczytywanie, aktualizowanie i usuwanie danych. Żądania te inicjowały działania na dokumentach przechowywanych w bazie MongoDB mających postać rekordów, które zawierają swego rodzaju kontener o strukturze:

- tekstowego typu danych,
- tablicy,
- obiektu,
- tablicy obiektów.

Każdy kontener wypełniany był odpowiednią serią danych: losowym ciągiem znaków o zadanej długości, ustaloną ilością komórek w tablicy, bądź określoną liczbą atrybutów w obiekcie.

W badaniach, oprócz aplikacji testowych, użyto popularnego narzędzia Postman [17], które może być wykorzystywane do prostego wysyłania żądań do API, zapisywania żądań w celu ich późniejszego użycia oraz testowania API.

Do zbierania wyników – czasów obsługi żądań, wykorzystano dwie metody. Takie podejście miało sprawdzić czy wykonywane pomiary dawały takie same lub porównywalne wyniki. Jedną z metod było użycie modułu wewnątrzserwerowego, który mierzy czasy od wysłania żądania do uzyskania odpowiedzi (cykl życia żądania). Druga metoda wykorzystywała aplikację zewnętrzną, która posiadała wbudowany skrypt do rejestracji czasów wykonywanych żądań. W obu przypadkach żądania były wysyłane w sposób iteracyjny, jedno po drugim.

5.2. Scenariusze badawcze

Opracowano eksperyment, w ramach którego testowano podstawowe operacje umożliwiające zarządzanie danymi w bazie danych. W tym celu opracowano cztery scenariusze badawcze. Każdy z nich dotyczył operacji pobierania, wysyłania, modyfikowania i usuwania innego zestawu danych, różniących się strukturą i rozmiarem. W scenariuszu pierwszym operowano na rekordzie, którego elementem był tekstowy typ danych, w postaci ciągu o długości 1000 znaków. W drugim scenariuszu realizowano działania na tablicy zawierającej 100 elementów, którymi były losowe ciągi 100 znaków. Trzeci scenariusz dotyczył obiektu składającego się ze 100 atrybutów, z których każdy zawierał wartość w postaci losowego ciągu o długości 100 znaków. W ostatnim, czwartym scenariuszu operowano na 50 elementowej tablicy składającej się z obiektów o 50

atrybutach, wypełnionych losowymi ciągami zawierającymi po 50 znaków.

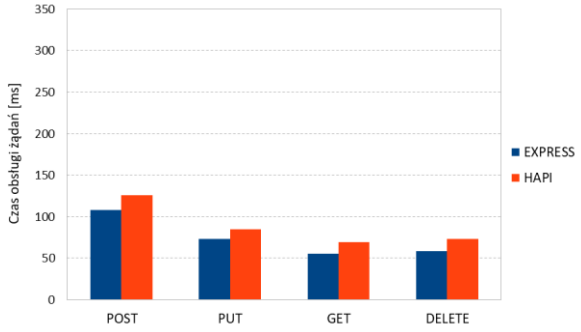
Operację REST API modyfikowania rekordu w bazie danych realizowano z takimi samymi parametrami jak przy tworzeniu kolejnego, nowego dokumentu. Natomiast pobieranie i usuwanie odbywało się po sparametryzowaniu żądania przez podawanie numeru identyfikującego rekord w bazie danych.

Każde żądanie wysyłano zarówno z aplikacji klienckiej jak i z narzędzia Postman po 100 razy. Wyniki eksperymentu były zliczane dwoma sposobami za pomocą modułu natywnego wkomponowanego w kod obsługi cyklu żądania oraz przy użyciu aplikacji Postman. W pojedynczym scenariuszu wykonano cztery typy żądań (pobieranie, aktualizowanie dodawanie, usuwanie), co dawało w sumie 800 wyników. Uwzględniając wszystkie cztery scenariusze – dla różnych typów danych, otrzymano 3200 wyników. Należy jeszcze pamiętać, że testowano dwa szkielety programistyczne Express i Hapi, a badania realizowano równolegle na dwóch komputerach. W ten sposób po przeprowadzeniu eksperymentu uzyskano w sumie 12800 wyników.

6. Wyniki badań

6.1. Scenariusz 1 – pomiar czasu obsługi żądania podczas wykonywania operacji na tekstowym typie danych

Operacje na małych porcjach danych w postaci łańcucha znaków były wykonywane w stosunkowo krótkim czasie. Najdłużej trwała obsługa żądania POST, które miało na celu przesyłanie danych do serwera. Na Rysunku 2 przedstawiono wyniki, które są obliczonymi wartościami średnimi.

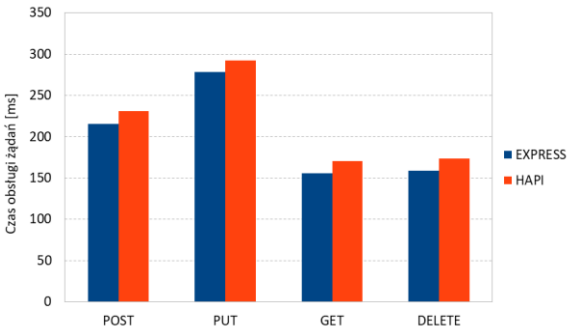


Rysunek 2: Średnie czasy wykonywania cyklu żądań realizujących poszczególne operacje na danych w postaci łańcucha znaków.

Na podstawie uzyskanych wyników można stwierdzić, że dla małych porcji danych, czas obsługi żądań był krótszy, w przypadku aplikacji testowej opracowanej na bazie szkieletu Express, bez względu na typ realizowanej operacji na danych. Czasy realizacji żądań dla metod POST, PUT, GET i DELETE były krótsze o odpowiednio 14%, 14%, 20% oraz 19%.

6.2. Scenariusz 2 – pomiar czasu obsługi żądania podczas wykonywania operacji na tablicy

Operacje na tablicach przechowujących w swych komórkach łańcuchy zawierające po 100 znaków były wykonywane około dwa razy dłużej niż działania na danych w pierwszym scenariuszu. Najbardziej czasochłonna była obsługa żądania PUT, polegająca na aktualizacji danych przechowywanych w określonych rekordach bazy danych. Rysunek 3 prezentuje średnie czasy otrzymane po przeprowadzeniu eksperymentu.

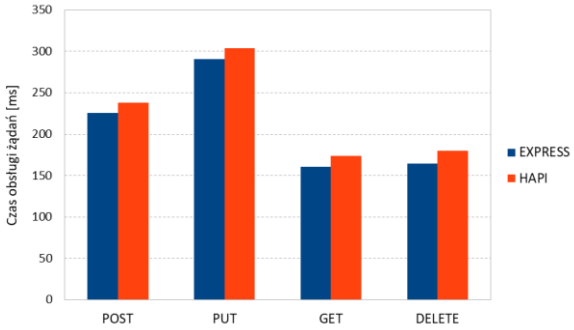


Rysunek 3: Średnie czasy wykonywania cyklu żądań realizujących poszczególne operacje na danych w postaci tablicy składającej się z łańcuchów znaków.

Również w tym przypadku aplikacja testowa zbudowana przy pomocy szkieletu Express pracowała wydajniej, choć jej przewaga, tzn. różnice czasowe były mniejsze niż w przypadku scenariusza 1 i wahały się w zakresie 5% - 9%. W związku z tym, można wysnuć wniosek, że operacje na większych porcjach danych zmniejszały różnice między porównywanymi frameworkami języka JavaScript.

6.3. Scenariusz 3 – pomiar czasu obsługi żądania podczas wykonywania operacji na obiekcie

Działania na średniej wielkości kontenerach na dane, którymi w tym przypadku były duże obiekty, składające się ze 100 pól tekstowych przechowujących stuznakowy tekst, wymagały podobnych jak w scenariuszu 3 czasów obsługi żądania. Również w tym scenariuszu najdłużej trwała obsługa żądania modyfikacji rekordów. Na Rysunku 4 zostały zaprezentowane wyniki - średnie czasy trwania cyklu żądania podczas których realizowano odpowiednie operacje na danych w bazie.

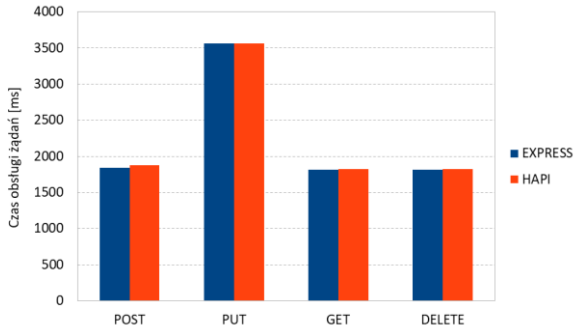


Rysunek 4: Średnie czasy wykonywania cyklu żądań realizujących poszczególne operacje na danych w formie rozbudowanego obiektu.

Także w tej części eksperymentu okazało się, że szkielet Express jest wydajniejszy niż Hapi, choć uzyskiwał on tylko nieznacznie lepsze czasy obsługi żądań (w granicach 1% - 5%).

6.4. Scenariusz 4 – pomiar czasu obsługi żądania podczas wykonywania operacji na tablicy obiektów

W ostatniej fazie badań dokonano pomiarów czasu trwania cyklu żądania podczas realizacji operacji na rekordach bazy danych zawierających duże partie danych, którymi były 50 elementowe tablice obiektów składające się z 50 atrybutów przechowujących łańcuchy znakowe o długości 50 znaków.



Rysunek 5: Średnie czasy wykonywania cyklu żądań realizujących poszczególne operacje na danych w postaci tablicy obiektów.

Wyniki zobrazowane na Rysunku 5 pokazują, że podczas wszystkich operacji na dużych zbiorach danych czasy obsługi żądań aplikacji wytworzonej za pomocą szkieletu Express oraz aplikacji zbudowanej na podstawie szkieletu Hapi zrównały się. Niewątpliwie czasy te były bardzo długie i w przypadku aktualizacji rekordów osiągały wartości powyżej 3500 ms. Dla pozostałych trzech operacji czasy te mieściły się w zakresie 1700 – 1900 ms.

7. Wnioski

W pracy skoncentrowano się na pomiarach i analizie czasów obsługi żądań, które miały za zadanie realizację czterech podstawowych operacji REST na nierelacyjnej bazie danych MongoDB.

Biorąc pod uwagę całość badań, na które złożyły się cztery scenariusze, aplikacja testowa zbudowana na podstawie szkieletu Express miała nieznacznie lepszą wydajność niż aplikacja oparta na frameworku Hapi. Tylko w jednym przypadku, w scenariuszu nr 4, wydajności obu aplikacji testowych były niemal identyczne. Dotyczyło to sytuacji, w której żądania operowały na rekordach zawierających bardzo rozbudowane i duże objętościowo zbiory danych, którymi były w tym przypadku tablice dużych obiektów. Spośród czterech operacji CRUD, aktualizacja danych trwała najdłużej, a pobieranie najkrócej. Wynikało to z tego, że podczas modyfikacji należało najpierw zlokalizować dokument w bazie, odczytać jego zawartość oraz na koniec nadpisać. W przypadku pobierania dokumentu schemat postępowania był prostszy i polegał jedynie na identyfikacji i zwróceniu w odpowiedzi danych.

Zrealizowane w pracy badania mają swoje ograniczenia, ponieważ skupiono się w nich wyłącznie na aspekcie wydajnościowym, a pozostałe kwestie pominięto. Jednak mimo tego, wykonana praca i zaprezentowane wyniki mogą być wykorzystane praktycznie przez programistów tworzących aplikacje webowe.

Literatura

[1] M. Pilar Salas-Zarate, G. Alor-Hernandez, R. Valencia-Garcia, L. Rodriguez-Mazahua, A. Rodriguez-Gonzalez, J. L. Lopez Cuadro, Analyzing best practices on Web development frameworks: The lift approach, Science of Computer Programming, Science of Computer Programming, 102, (2015) 1-19, https://doi.org/10.1016/j.scico.2014.12.004, [16.03.2021].

[2] Mozilla Developer Network, Wprowadzenie do Express/Node, https://developer.mozilla.org/pl/docs/Learn/Server-side/Express_Nodejs/Introduction, [16.03.2021].

[3] Offshore Web Developer, Hapi vs Express: Which NodeJS Frameworks is the Better?, http://www.offshorewebdeveloper.com/blog/hapi-vs-express/, [16.03.2021].

[4] Stackoverflow.com, https://insights.stackoverflow.com/survey/2020#technology, [27.03.2021].

[5] Github.com, https://github.com/showcases/web-application-frameworks, [27.03.2021].

[6] Stateofjs, https://2020.stateofjs.com/en-US/technologies/back-end-frameworks/, [27.03.2021].

[7] RapidAPI Blog, The Best NodeJS Frameworks for 2021, https://rapidapi.com/blog/best-nodejs-frameworks/, [25.03.2021].

[8] Frameworks built on Express, https://expressjs.com/, https://expressjs.com/en/resources/frameworks.html, [25.03.2021].

[9] S. Martin, Top 10 Node.js Frameworks For Web App Development in 2021, https://javascript.plainenglish.io/top-10-node-js-frameworks-for-web-app-development-in-2020-21-38e3ea2a57e5, [25.03.2021].

[10] Npm trends, https://www.npmtrends.com/express-vs-hapi, [27.03.2021].

[11] D. Swerski, Hapi vs. Express in 2019: Node.js framework comparison, https://raygun.com/blog/hapi-vs-express/, [26.10.2020].

[12] Raygun, Node.js performance vs Hapi, Express, Restify, Koa & More, https://raygun.com/blog/nodejs-vs-hapi-express-restify-koa/, [26.10.2020].

[13] B. Miłosierny, M. Dzieńkowski, The comparative analysis of web application frameworks in the Node.js ecosystem, Journal of Computer Science Institute, 18, (2021) 42-48, https://doi.org/10.35784/jcsi.2423, [30.03.2021].

[14] J. Brett, Getting Started with hapi.js, Packt Publishing, 2016.

[15] E. M. Hahn, Express in action. Writing, building, and testing Node.js applications, Manning Publications Co., 2016.

[16] M. Massé, REST API Design Rulebook, O'Reilly, 2012.

[17] Postman, https://www.postman.com/, [27.03.2021].