JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 16 (2020) 309–316
Received: 13 July 2020
Accepted:29 August 2020

Comparison of REST and GraphQL web technology performance

Porównanie wydajności technologii webowych REST i GraphQL

Mateusz Mikuła*, Mariusz Dzieńkowski

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

Abstract

The aim of the study was to compare the performance of two data exchange styles commonly used in web applications, i.e. REST and GraphQL. For the purposes of the study two test applications were developed containing the same functionalities, one of which was REST and the other one was GraphQL. They were used for performance tests done with the help of the JMeter tool, during which measurements of the total processing time of requests and the volume of data downloaded and sent were performed. An experiment was developed that tested the basic operations found in most network services: display, add, update, and delete data. The most attention was devoted to the information display operation in the case of which load tests were done. On the basis of performed studies and obtained results, no differences in performance during the operation of adding, editing and deleting data by applications based on REST API and GraphQL were found. During the display operation under heavy load conditions and while downloading small portions of data, the service using GraphQL had a better performance. When downloading large portions of data, the REST-based service exhibited a higher performance.

Keywords: REST; GraphQL; web service; performance testing

Streszczenie

Zrealizowano badania, których celem było porównanie wydajności dwóch, szeroko stosowanych w aplikacjach webowych stylów wymiany danych REST i GraphQL. Na potrzeby badań opracowano dwie usługi testowe, zawierające te same funkcjonalności, z których jedna była serwisem REST, a druga GraphQL. Posłużyły one do testów wydajnościowych, przeprowadzonych za pomocą narzędzia JMeter, podczas których wykonywano pomiary całkowitego czasu przetworzenia żądań oraz wielkości pobieranych i wysyłanych danych. Opracowano eksperyment, w ramach którego testowano podstawowe operacje występujące w większości usług sieciowych: wyświetlanie, dodawanie, aktualizowanie oraz usuwanie danych. Najwięcej uwagi poświęcono operacji wyświetlania informacji, w przypadku której wykonano testy obciążeniowe. Na podstawie zrealizowanych badań i uzyskanych wyników nie stwierdzono różnic w wydajności podczas realizacji operacji dodawania, edycji i usuwania danych przez aplikacje oparte na REST API i GraphQL. Podczas operacji wyświetlania w warunkach dużego obciążenia i w przypadku pobierania małych porcji danych lepszą wydajność miała usługa wykorzystująca GraphQL. Natomiast w przypadku pobierania dużych porcji danych wyższą wydajność uzyskiwała usługa oparta na REST.

Słowa kluczowe: REST; GraphQL; usługa internetowa; testowanie wydajności

*Corresponding author

Email address: **mateusz.mikula@pollub.edu.pl** (M. Mikuła)

1. Wstęp

1.1. Style wymiany danych REST i GraphQL

Aplikacje webowe są obecnie szeroko stosowane ze względu na popularność Internetu. Działają one w przeglądarce internetowej przez co wielu użytkowników może jednocześnie z nich korzystać. Aplikacje webowe komunikują się z usługami sieciowymi poprzez różne interfejsy programistyczne (API) za pomocą żądań i odpowiedzi używając do tego celu różnych protokołów (HTTP, FTP, POP3, itd.). Do tej pory opracowano wiele standardów, które określają sposób komunikacji klienta z serwerem. Na początku były to protokoły takie jak RPC, COBRA, SOAP, które były złożone, a przez to trudne w użyciu. Problemy te zostały wyeliminowane przez powstały w 2000 roku styl architektoniczny REST (Representational State Transfer) [1], który jest skalowalnym, prostym, rozszerzalnym i zoptymalizowanym wzorcem architektury określającym format przesyłanych informacji [2]. Pomimo wielu zalet i to rozwiązanie ma swoje ograniczenia, które ujawniły się w przypadku popularnych w ostatnim czasie serwisów społecznościowych, w których korzystanie z REST okazało się uciążliwe ze względu na nieakceptowalny przez użytkowników, długi czas wczytywania się wyświetlanych postów na urządzeniach mobilnych [3]. Rozwiązaniem tej kwestii zajęli się programiści firmy Facebook i w 2018 roku opublikowali pierwszą wersję języka zapytań dla interfejsów API o nazwie GraphQL, do komunikowania się z serwerem [4].

Wybór stylu wymiany danych dla tworzonej aplikacji, ma istotne znaczenie w kontekście czasu dostarczania danych. Im więcej informacji aplikacja jest w stanie przetworzyć w określonym przedziale czasu, tym jest ona bardziej wydajna. Należy pamiętać, że wydajność to jeden z najważniejszych aspektów powodzenia funkcjonowania rozwiązania w świecie rzeczywistym [5]. Dotyczy to szczególnie aplikacji internetowych, dających możliwość jednoczesnego dostępu bardzo wielu

użytkownikom, dla których priorytetem jest natychmiastowa i nieprzerwana dostępność usług w sieci [6].

1.2. Testowanie wydajności

Do sprawdzenia czy system, przy obciążeniu zgodnym z planowanym wykorzystaniem, będzie nadal działał prawidłowo, a odpowiedzi będą generowane w akceptowanym czasie, służą testy wydajnościowe [5]. Przeznaczone są one do oceny niezawodności, a co się z tym wiąże, jakości oprogramowania internetowego. Odpowiadają na pytanie, jak duże obciążenie badany system jest w stanie wytrzymać. Głównym celem testów wydajnościowych jest zbadanie systemu pod symulowanym działaniem wirtualnych użytkowników [7]. Najważniejszym parametrem do oceny wydajności jest czas, w którym dany system lub aplikacja musi wykonać żądaną akcję. Aplikacje internetowe są trudne do przetestowania, szczególnie pod względem wydajnościowym ze względu na to, że występują kwestie nieprzewidywalne takie jak wielkość obciążenia czy czas reakcji [8].

Testowanie wydajności najczęściej odbywa się w sposób automatyczny i sprowadza się w większości przypadków do sprawdzenia poprawnego działania badanego systemu pod określonym obciążeniem [7].

Zalety testowania automatycznego to przyspieszenie niektórych czynności, a tym samym skrócenie czasu całego procesu, eliminacja błędów ludzkich, możliwość generowania różnego rodzaju raportów, a także redukcja kosztów na etapie testowania. Narzędziami stosowanymi w automatyzacji testów wydajnościowych są te służące do nagrywania, symulowania i rejestrowania procesu działania aplikacji.

1.3. Przegląd literatury

Na forach i blogach wśród deweloperów oprogramowania trwa dyskusja na temat, który z istniejących modeli architektur (SOAP, REST, GraphQL) będzie najbardziej optymalny dla danego typu aplikacji. Kwestia ta jest ciągle przedmiotem badań, których wyniki pojawiają się w publikacjach naukowych i studenckich pracach dyplomowych.

W pracy [9] porównywano wydajność 3 aplikacji webowych udostępniających usługi internetowe jednocześnie za pomocą dwóch technologii REST i GraphQL. Zaobserwowano, że w dwóch aplikacjach migracja do GraphQL spowodowała wzrost wydajności, biorąc pod uwagę średnią liczbę żądań na sekundę i szybkość przesyłania danych. Jednak dla obciążeń powyżej 3000 żądań usługi oparte na GraphQL działały mniej wydajnie (wyniki w zakresie od 98 do 2159 kB/sek.) od usług typu REST. Natomiast w przypadku typowych obciążeń (100 żądań), usługi oparte na architekturach REST i GraphQL osiągały podobną wydajność z zakresu 6,34 – 7,68 żądań/sek.

Korzyści przejścia z API opracowanego na bazie standardu REST do API zrealizowanego w standardzie GraphQL, zebrane od specjalistów mających doświadczenie w tym zakresie, poddano praktycznej ocenie w artykule [10]. W tym celu przeprowadzono migrację siedmiu systemów z REST do GraphQL i w związku z tym dzięki zastosowaniu GraphQL nastąpiło zmniejszenie rozmiaru dokumentów JSON, zwracanych przez interfejsy REST API o 94% w liczbie pól i 99% w liczbie bajtów (oba wyniki są medianą).

Brito i Valente [3] wykonali doświadczenie w celu porównania wysiłku, jaki trzeba włożyć w generowanie zapytań do GraphQL i implementacje usług webowych REST. Badania, w których wzięło udział 22 studentów, wykazały, że budowanie zapytań GraphQL jest szybsze niż implementacja usług REST.

Analizą porównawczą wydajności rozwiązań wykorzystujących GraphQL i REST zajmował się Cederlund w swojej dysertacji [11]. Jako przypadków testowych używał rzeczywistych aplikacji do oceny opóźnienia, objętości danych i liczby zapytań. Przypadki testowe zostały tak zaprojektowane, aby możliwe było testowanie pojedynczych żądań oraz równoległych i sekwencyjnych przepływów danych. W badaniach skoncentrowano się na pomiarach czasów odpowiedzi zmieniających się w zależności od wielkości przesyłanych danych. Wyniki analiz pokazały, że zastosowanie GraphQL zmniejszyło czas odpowiedzi dla równoległych i sekwencyjnych przepływów danych, jednak użycie pojedynczego, zwykłego endpointa REST gwarantowało najlepszą wydajność.

Taskula w swojej pracy dyplomowej [12] porównywał dwa rozwiązania REST i GraphQL, które wykorzystywał do pobierania danych. W ramach studium przypadku wykonane zostały pomiary wydajności, która zamieniała się wraz ze zmianami rozmiaru transferowanych danych. Okazało się, że w wielu przypadkach testowych GrahpQL lepiej wykorzystywał dane, natomiast REST mógł się równać tylko w przypadku intensywnego filtrowania pól. Dodatkowo w pracy porównywano złożoność i możliwość dalszego rozwijania aplikacji z API wykonanego na bazie REST oraz GraphQL. Analiza jakościowa wykazała, że realizacja API w technologii GraphQL skutkowała mniejszą jego złożonością oraz dawała w przyszłości większe możliwości rozwoju.

W wyniku przeprowadzenia pilotażowego testu [13], porównującego REST z GraphQL okazało się, że pierwsze, starsze rozwiązanie jest szybsze dla prostych ustrukturyzowanych zapytań, takich jak pobieranie informacji tylko z jednego źródła lub tabeli. Ponadto okazało się, że różnica w wydajności w zakresie czasu odpowiedzi rośnie wykładniczo wraz z rozmiarem bazy danych. Pobierając z kolei więcej informacji, GraphQL generuje wolniejsze odpowiedzi w zakresie od 64% do 115%. W przypadku większych baz danych i konstruowaniu bardziej złożonych zapytań GraphQL lepiej się sprawdzał. Różnica pomiędzy GraphQL i REST wynosiła 25%.

W ramach pracy [14], zaproponowano metodologię, która umożliwiła odpowiedź na pytanie, która technologia GraphQL czy REST jest szybsza i lepiej zoptymalizowana. Badaniom poddano start-up - społecznościową aplikację internetową, w której testowano korzyści z migracji z REST do GraphQL. Badania obejmowały

dwie sytuacje: w pierwszej żądania były przesyłane w sposób sekwencyjny, a w drugiej równoległy. Główny wniosek płynący z pierwszej badanej sytuacji jest taki, że GraphQL oferuje krótszy czas odpowiedzi dla każdej sekwencji zapytań. Średnia różnica między obiema technologiami wyniosła 159 ms, co czyni GraphQL o 46% szybszym od REST. Z drugiej badanej sytuacji wynika, że także i w tym przypadku GraphQL był o 35% szybszy niż REST. Ten 11% spadek pomiędzy obiema sytuacjami można wytłumaczyć rodzajem wykonywanych żądań. W przypadku wykonania sekwencyjnego drugiego testu, realizacja za pomocą REST zajęłaby około 3 razy więcej czasu. W związku z tym schemat równoległy sprawia, że REST staje się bardziej konkurencyjny w stosunku do GraphQL. Biorąc pod uwagę rozmiar odpowiedzi, w pierwszej sytuacji GraphQL był o 21% lżejszy, natomiast w drugiej o 71%.

Przywołane prace i ich wyniki nie wskazują jednoznacznie, która technologia jest zdecydowanie lepsza. Odpowiedź na to pytanie jest uwarunkowana kilkoma czynnikami jak na przykład typ i przeznaczenie danego oprogramowania oraz rodzaj i wielkość grupy potencjalnych użytkowników.

1.4. Cel i zakres pracy

W Internecie znajduje się dużo publikacji pokazujących zalety i wady stylów wymiany danych REST i GraphQL. Jednak wciąż niewiele artykułów odpowiada na pytanie, które rozwiązanie jest lepsze, potwierdzając równocześnie to w sposób ilościowy na podstawie wyników z przeprowadzonych badań. Celem tej pracy jest dalsze wypełnienie tej luki poprzez przeprowadzenie analizy porównawczej wydajności dwóch technologii internetowych REST i GraphQL.

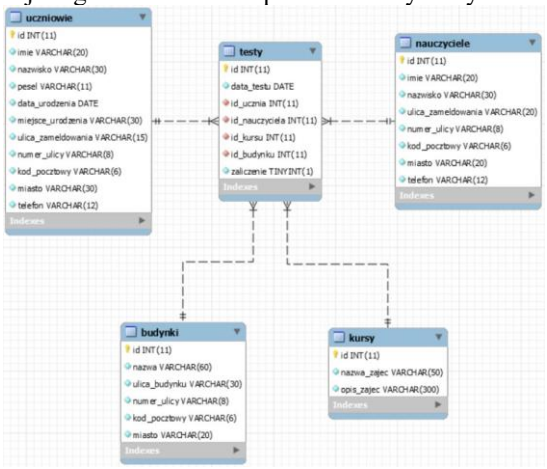
Zakres pracy obejmował następujące elementy: opracowanie aplikacji testowych, przygotowanie eksperymentu, dobór narzędzia diagnostycznego, zrealizowanie zautomatyzowanych testów wydajnościowych oraz opracowanie wyników i ich interpretację.

W pracy postawiono tezę badawczą, którą sformułowano w następujący sposób:
Aplikacje wykorzystujące rozwiązanie GraphQL w sytuacjach, w których klient potrzebuje zestawu danych w określonym formacie i możliwie najmniejszym rozmiarze, mają lepszą wydajność w porównaniu do aplikacji stosujących REST.

2. Metoda badań

2.1. Aplikacje testowe

Na potrzeby badań utworzono dwie usługi - aplikacje webowe zawierające te same funkcjonalności, ale wykorzystujące odmienne style wymiany danych: REST i GraphQL. W związku z tym w obu aplikacjach zaimplementowano odpowiednie API i poddano je testom sprawdzającym ich wydajność. Zadaniem przygotowanego oprogramowania było zarządzanie danymi dotyczącymi przeprowadzonych testów: informacjami na temat uczniów, nauczycieli, kursów oraz budynków, w których odbywały się egzaminy. Warstwy serwerowe aplikacji pobierają określone informacje z bazy danych, której diagram ERD został przedstawiony na rysunku 1.



Rysunek 1: Diagram związków encji

2.2. Środowisko testowe

Do przeprowadzenia badań wykorzystano dwa komputery przenośne. Jeden z nich pełnił rolę serwera, na którym uruchomione zostały dwie usługi webowe: pierwsza zawierająca REST API, druga GraphQL API. Drugi komputer pełnił rolę klienta, na którym zostało zainstalowane narzędzie umożliwiające przeprowadzenie testów - JMeter w wersji 5.2.1. Parametry poszczególnych urządzeń wykorzystanych do badań oraz opis serwera bazy danych, serwera HTTP i parametrów sieci znajdują się w tabelach 1, 2 i 3.

Tabela 1: Opis komputera pełniącego rolę serwera

Procesor	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz, 2601MHZ, Rdzenie: 4, Procesory logiczne: 8
Pamięć RAM	16 GB
Karta sieciowa	Intel(R) Dual Band Wireless-AC 3165
System operacyjny	Microsoft Windows 10 Pro
Serwer bazy danych	MariaDB 10.1.25
Serwer HTTP	Apache Tomcat 9.0.3

Tabela 2: Opis komputera pełniącego rolę klienta

Procesor	Intel(R) Core(TM) i3-2350M CPU @ 2.30 GHz, 2300 MHz, Rdzenie: 2, Procesory logiczne: 4
Pamięć RAM	6 GB
Karta sieciowa	Intel(R) Centrino(R) Wireless-N 130
System operacyjny	Windows 7 Home Premium

Tabela 3: Opis sieci Wi-Fi

Protokół	Wi-Fi 4 (802.11n)
Pasmo sieci	2.4 GHz
Średnia prędkość pobierania danych	16383 kbps
Średnia prędkość wysyłania danych	1052 kbps

2.3. Narzędzie badawcze

Do testów wydajnościowych użyto programu Apache JMeter, który jest narzędziem typu open source napisanym w języku Java z wykorzystaniem biblioteki Swing [15]. Oprogramowanie to umożliwia automatyczne analizowanie wydajności różnych usług (w tym aplikacji internetowych) pod dużym obciążeniem, przy równoczesnej pracy wielu użytkowników. JMeter został wybrany do przeprowadzenia badań z następujących względów [9]:

- program jest dostępny bezpłatnie,
- posiada czytelny i intuicyjny interfejs użytkownika,
- daje możliwość tworzenia symulacji korzystania z aplikacji przez wielu użytkowników jako oddzielnych wątków,
- obsługuje wiele protokołów w tym HTTP, SOAP, POP3, itd.,
- udostępnia możliwość tworzenia zapytań HTTP korzystając z jego metod (GET, POST) oraz pozwala na przetwarzanie wyników odpowiedzi,
- umożliwia opracowanie zaawansowanych scenariuszy testowania,
- wyniki testów można prezentować za pomocą drzew, tabeli i wykresów.

JMeter jest zaawansowanym i kompleksowym rozwiązaniem do wykonywania nie tylko testów wydajnościowo-obciążeniowych, ale również testów jednostkowych, funkcjonalnych i regresyjnych. Osobom używającym tego narzędzia nie stawia wysokich wymagań co do ich umiejętności informatycznych – wymagana jest jedynie znajomość technologii HTML oraz konstruowania wyrażeń regularnych.

2.4. Scenariusze badawcze

Opracowano eksperyment, w ramach którego testowano podstawowe operacje umożliwiające zarządzanie danymi w bazie danych występujące w większości usług sieciowych: wyświetlanie, dodawanie, aktualizowanie oraz usuwanie danych. Najwięcej uwagi poświęcono operacji wyświetlania informacji, w zakresie której rozpatrywano trzy przypadki testowe:

1. nadmiarowego pobierania wszystkich danych przez jeden endpoint REST API oraz pobierania ściśle określonych danych poprzez API oparte na GraphQL;
2. pobierania wszystkich danych przez REST API (1 endpoint) oraz pobierania wszystkich danych poprzez GraphQL API;
3. pobierania kompletu danych z kilku endpointów REST API i jednego endpointa GraphQL API.

Badanie operacji wyświetlania (pierwszy i drugi przypadek) zostało przeprowadzone dla różnej liczby użytkowników generujących jednocześnie żądania (1, 5, 50, 100, 500 i 1000) oraz dla różnej liczby rekordów pobieranych z bazy (5, 50, 100). Przy testowaniu aplikacji podczas wykonywania operacji dodawania, usuwania czy edytowania danych nie zmieniano warunków obciążeniowych: tzn. nie brano pod uwagę przypadków testowania dla różnej liczby testów w bazie danych oraz różnej liczby użytkowników, korzystających równocześnie z aplikacji.

Do porównania wydajności aplikacji, wykorzystujących różne style wymiany danych posłużono się trzema metrykami:

- całkowitym czasem przetworzenia żądań wyrażonym w milisekundach,
- rozmiarem danych pobieranych w bajtach,
- rozmiarem danych wysyłanych w bajtach.

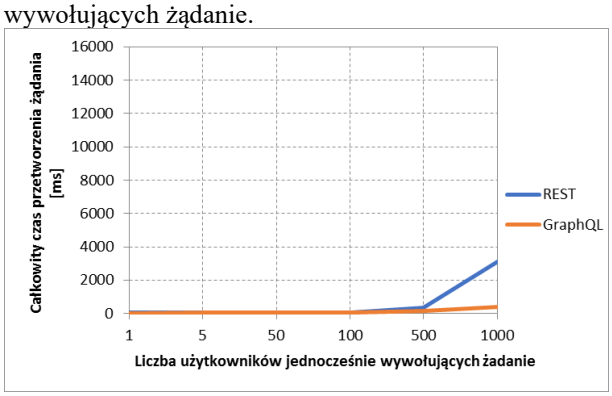
3. Wyniki badań

3.1. Wyświetlanie danych

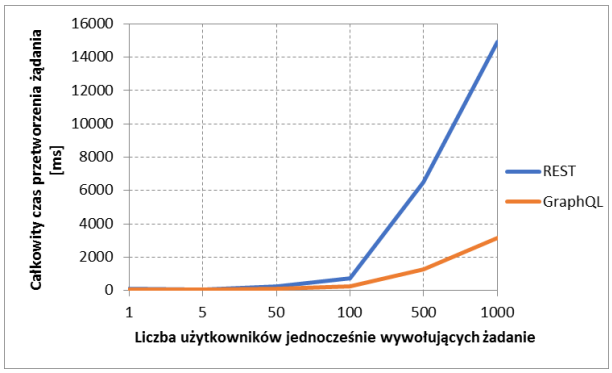
Przypadek 1: *Nadmiarowe pobieranie wszystkich danych poprzez jeden endpoint REST API versus pobieranie ściśle określonych danych poprzez API oparte na GraphQL*

W przygotowanej usłudze internetowej bazującej na architekturze REST został utworzony jeden endpoint /app/testy, poprzez który możliwe było pobranie wszystkich danych dotyczących przeprowadzonych testów wraz z informacjami o uczniu, nauczycielu, kursie oraz budynku, w którym odbywał się egzamin. W wielu sytuacjach klient może nie potrzebować tylu informacji. Ilość otrzymywanych informacji powinna uwzględniać typ (komputer, tablet, smartfon) i specyfikę urządzenia (np. rozdzielczość ekranu, moc obliczeniowa), na którym liczba jednocześnie wyświetlanych informacji może być różna. W części tej zbadano wydajność obu aplikacji w sytuacji, gdy serwis oparty na REST przekazuje często nadmiarowo wszystkie dane, a serwis oparty na GraphQL tylko dane ściśle określone i wymagane przez użytkownika. Ilość pobieranych danych może być również uzależniona i automatycznie dobierana do urządzenia, na którym będą one wyświetlane lub przetwarzane, biorąc pod uwagę rozdzielczość ekranu czy moc obliczeniową.

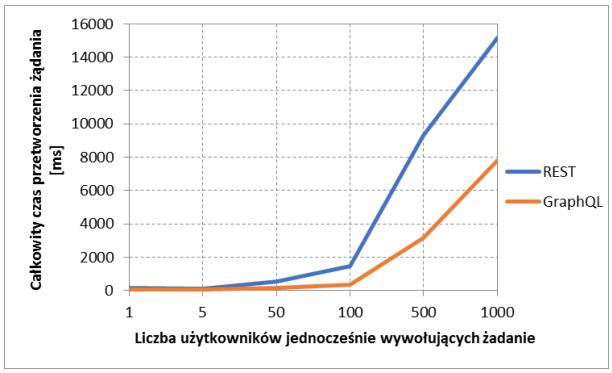
Na rysunkach 2, 3 i 4, w postaci wykresów liniowych, przedstawiono wyniki ukazujące wydajność testowanych aplikacji opartych na dwóch stylach wymiany danych REST i GraphQL. Oś Y reprezentuje całkowity czas przetwarzania żądań wyrażony w milisekundach, natomiast oś X liczbę użytkowników jednocześnie wywołujących żądanie.



Rysunek 2: Wyświetlanie danych (rozmiar: 5 rekordów).

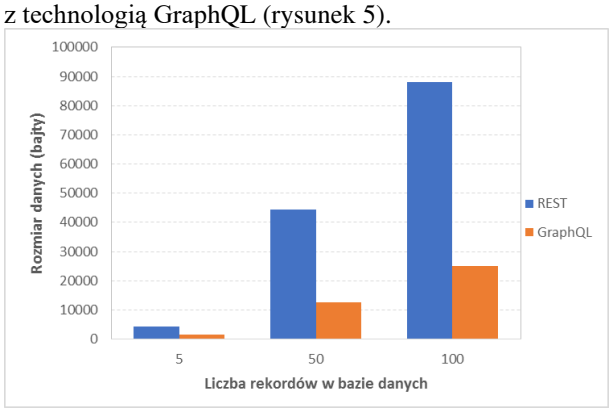


Rysunek 3: Wyświetlanie danych (rozmiar: 50 rekordów).



Rysunek 4: Wyświetlanie danych (rozmiar: 100 rekordów).

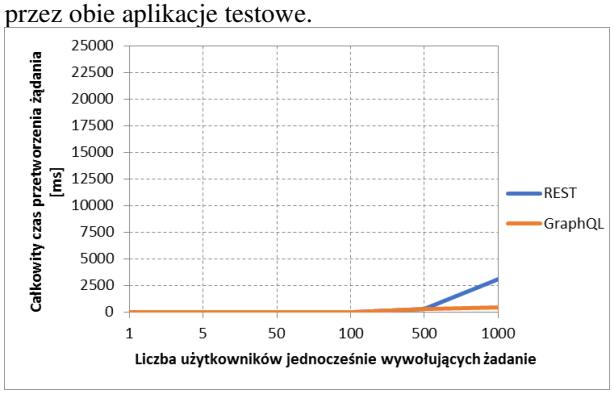
Na wykresach 2, 3 i 4 wyraźnie widać duży spadek wydajności aplikacji wykorzystującej styl REST, przy dużej liczbie użytkowników (>100) wysyłających jednocześnie żądania, zarówno w przypadku gdy otrzymywano 5, 50 czy 100 rekordów. Zaobserwowany duży spadek wydajności był spowodowany tym, że wielkość otrzymywanych danych w przypadku zastosowania architektury REST jest trzy razy większa w porównaniu z technologią GraphQL (rysunek 5).



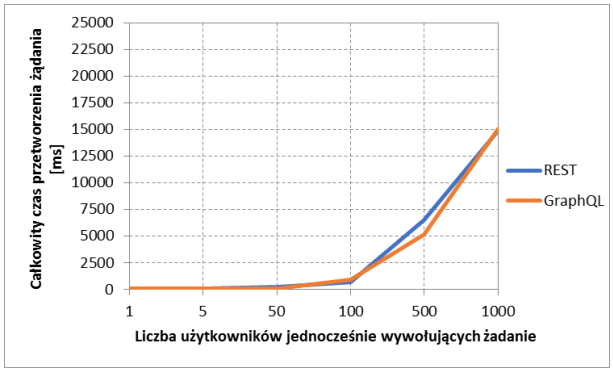
Rysunek 5: Rozmiar otrzymanych danych - REST vs GraphQL.

Przypadek 2: *Jednakowa liczba otrzymywanych (pobieranych) danych - pobieranie wszystkich danych przez REST API (przez 1 endpoint) oraz pobieranie wszystkich danych poprzez GraphQL API*

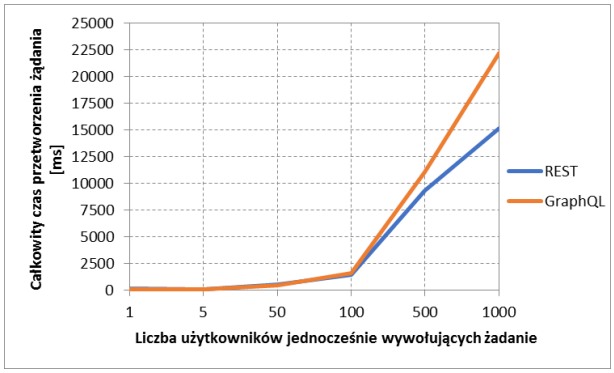
Drugi przypadek polegał na pomiarze tych samych parametrów, tym razem podczas otrzymywania tej samej ilości informacji. W tym celu w aplikacji opartej na GraphQL klient pobierał wszystkie dane dotyczące testu, podobnie jak to miało miejsce w przypadku 1 w aplikacji korzystającej z architektury REST. Rysunki 6, 7 i 8 odzwierciedlają wyniki wydajności, traktowanej jako całkowity czas przetwarzania żądań, uzyskane przez obie aplikacje testowe.



Rysunek 6: Wyświetlanie danych (rozmiar: 5 rekordów).



Rysunek 7: Wyświetlanie danych (rozmiar: 50 rekordów).

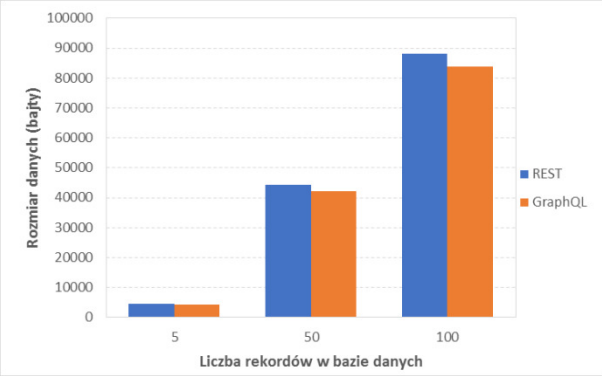


Rysunek 8: Wyświetlanie danych (rozmiar: 100 rekordów).

W przypadku 2, kiedy pobierana jest jednakowa ilość danych, przy dużym obciążeniu (>500 użytkowników), sytuacja staje się bardziej skomplikowana. Zastosowanie GraphQL okazało się bardziej wydajne, gdy pobierana była mała ilość danych: 5 rekordów (rysunek 6) i gdy liczba użytkowników generujących żądania była większa niż 500. W przypadku pobierania 50 rekordów, niezależnie od liczby użytkowników wydajność obu aplikacji była podobna. Usługa oparta na REST okazała się bardziej wydajna podczas jednoczesnego wyświetlania dużych ilości danych (100 rekordów) począwszy od liczby stu użytkowników.

Jeśli chodzi o rozmiar otrzymywanych danych (rysunek 9), to był on w przybliżeniu jednakowy, nato-

miast rozmiar wysłanych danych był zdecydowanie większy po stronie aplikacji wykorzystującej GraphQL (727 bajtów) w porównaniu do REST (133 bajtów).



Rysunek 9: Rozmiar otrzymanych danych - REST vs GraphQL.

Przypadek 3: *Pobieranie kompletu danych z kilku endpointów REST API i jednego endpointa GraphQL API*

Technologie webowe oparte na GraphQL są mniej złożone, gdyż wykorzystują jeden stały endpoint. Z kolei aplikacje bazujące na architekturze REST korzystają z wielu endpointów. W ostatnim przypadku przeanalizowano i porównano wydajność technologii REST i GraphQL podczas pobierania tych samych danych w sytuacji gdy aplikacja z REST korzysta z kilku endpointów, a aplikacja oparta na GraphQL tylko z jednego endpointa.

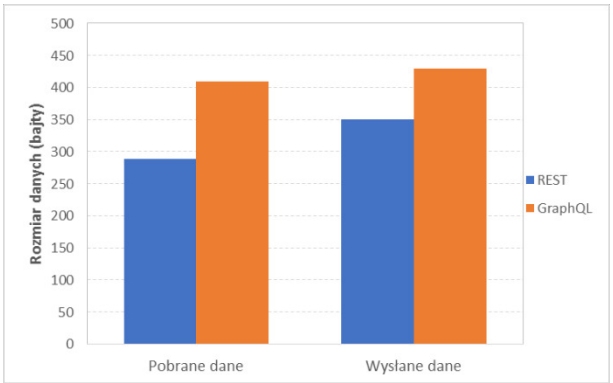
W technologii webowej opartej o REST chcąc uzyskać informacje o określonym kursie, należało użyć odpowiedniego adresu, innego niż na przykład przy otrzymywaniu informacji o danym budynku. Do celów badań została zasymulowana sytuacja, w której klient chce uzyskać pełne informacje o uczniu, nauczycielu i \budynku z określonymi identyfikatorami. W tym przypadku uzyskane czasy przetworzenia żądania były krótsze dla aplikacji wykorzystującej GraphQL. Także rozmiar otrzymanych danych był mniejszy w aplikacji korzystającej z GraphQL, a wysłanych w aplikacji z REST API (tabela 4).

Tabela 4: Porównanie otrzymanych wartości badanych parametrów w aplikacjach z REST i GraphQL

Interfejs (API)	Czas przetworzenia żądań (ms)	Rozmiar wszystkich pobranych danych (bajty)	Rozmiar wszystkich wysłanych danych (bajty)
REST	21	1091	404
GraphQL	15	894	667

3.2. Dodawanie danych

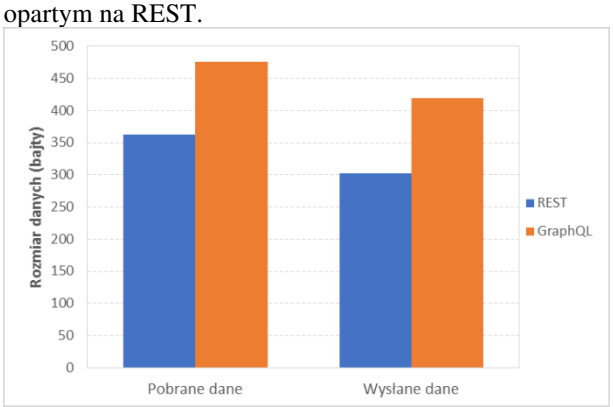
Podczas dodawania rekordu do bazy lepsze rezultaty odnoszące się do rozmiaru otrzymanych jak i wysłanych informacji, dało się zauważyć w usłudze opartej na stylu REST (rysunek 10). Jednak czas przetworzenia żądania był lepszy w przypadku aplikacji z GraphQL. Klient, aby dodać dane o nowym budynku w aplikacji z REST czekał na przetworzenie żądania średnio 144 ms. W GraphQL ta sama operacja była wykonywana w czasie średnio 123 ms.



Rysunek 10: Porównanie rozmiarów danych w REST i GraphQL.

3.3. Aktualizowanie danych

Podczas operacji aktualizowania danych nie zaobserwowano różnicy w czasach przetworzenia żądania w przypadku obu aplikacji testowych. Uzyskane średnie wartości czasu wykonania operacji edycji danych były praktycznie jednakowe (12 milisekund w aplikacji opartej na REST i 13 milisekund dla aplikacji z GraphQL). Natomiast różnice można zauważyć w rozmiarach otrzymanych i wysyłanych danych, podobnie jak to miało miejsce podczas dodawania informacji do bazy danych. Rysunek 11 pokazuje, że w tej operacji lepsze wyniki osiągnął klient, komunikując się z serwisem opartym na REST.



Rysunek 11: Porównanie rozmiarów danych w REST i GraphQL.

3.4. Usuwanie danych

Ostatnią testowaną operacją dla opracowanych usług internetowych było usuwanie danych z bazy. W obu aplikacjach czasy przetworzenia żądań miały podobne wielkości (tabela 5). W usłudze bazującej na interfejsie REST, klient, aby usunąć określony budynek z bazy danych potrzebował przesłać żądanie o rozmiarze 187 bajtów. Natomiast w aplikacji wykorzystującej GraphQL API, klient wysyłał żądanie zawierające 286 bajtów. Z tego wynika, że w tej sytuacji lepiej wypada aplikacja wykorzystująca interfejs REST. Jednak należy pamiętać, że każdy rekord w bazie danych w usłudze z REST API posiada swój odrębny adres, który pozwala na jego usunięcie. W rozwiązaniu stosującym GraphQL klient posługuje się jednym, stały endpointem, co zde-

cydowanie ułatwia komunikację. Kolejną rzeczą, na którą trzeba zwrócić uwagę jest to, że przy próbie usunięcia encji (również przy edycji danych), która nie istnieje w bazie, w podejściu REST klient otrzyma błąd wykonania żądania. W technologii GraphQL klient zaś otrzyma status 200 HTTP, oznaczający poprawne przetworzenie żądania, co może powodować wyświetlanie niewłaściwego komunikatu po stronie klienta (tabela 6).

Tabela 5: Rezultat wykonania żądania usunięcia rekordu z bazy danych w aplikacjach stosujących styl REST i GraphQL

Technologia	Czas przetwo-rzenia żądania (ms)	Status	Rozmiar pobranych danych (bajty)	Rozmiar wysłanych danych (bajty)
REST	535	OK	187	215
GraphQL	495	OK	286	238

Tabela 6: Rezultat wykonania żądania usunięcia rekordu, który nie istnieje w bazie danych w aplikacjach z REST oraz GraphQL

Technologia	Czas przetwo-rzenia żądania (ms)	Status	Rozmiar pobranych danych (bajty)	Rozmiar wysłanych danych (bajty)
REST	56	Błąd	307	215
GraphQL	22	OK	389	238

4. Wnioski

Wyniki badań empirycznych są bardzo pomocne przed podjęciem decyzji o wyborze najbardziej optymalnego rozwiązania w procesie wytwarzania oprogramowania. W związku z tym, w niniejszej pracy przeprowadzono analizę porównawczą pod względem wydajności dwóch, szeroko stosowanych obecnie podejść do budowy interfejsu API: REST i GraphQL. Badania zostały zrealizowane na utworzonych do tego celu dwóch usługach, z których pierwsza do wymiany danych używała architektury REST, natomiast druga wykorzystywała język i silnik GraphQL. Do pomiarów czasów przetwarzania żądań oraz rozmiarów wysyłanych i pobieranych danych posłużyło narzędzie diagnostyczne JMeter.

Zrealizowane badania pokazały, kiedy i jakie różnice występują między obiema technologiami w kontekście ich wydajności. W przypadku operacji wyświetlania, klient korzystający z usługi internetowej opartej na REST, nie może ograniczyć liczby otrzymywanych danych, ponieważ zawsze w odpowiedzi otrzyma pełny zestaw informacji. W tej sytuacji część danych może być dla klienta niepotrzebna. Natomiast w GraphQL to klient określa, posługując się językiem zapytań, jakie informacje chce uzyskać w odpowiedzi od serwera. W tym przypadku, bez względu na liczbę pobieranych rekordów z bazy, usługa internetowa oparta na GraphQL szybciej przetwarzała żądania niż miało to miejsce w aplikacji z interfejsem REST. Wynikało to głównie z ilości otrzymywanych danych, która była ponad trzykrotnie mniejsza w GraphQL niż w REST.

Kluczowym rozpatrywanym przypadkiem, była sytuacja, gdy klient zarówno poprzez jedną jak i drugą usługę otrzymywał tę samą ilość danych (wszystkie dane) i to ta część badań potwierdziła postawioną na początku pracy tezę. W obu technologiach, przy pobieraniu małych porcji danych (5 rekordów) przez 1 endpoint, przy dużym obciążeniu, lepiej wypadała usługa stosująca GraphQL. Jednak w podobnych warunkach, w przypadku pobierania dużych porcji danych (100 rekordów) bardziej wydajna była usługa wykorzystująca interfejs REST. Rozmiar wysyłanych danych był większy w GraphQL, ponieważ klient w przesyłanej wiadomości musiał przy pomocy języka zapytań zawrzeć wszystkie informacje, które chciał uzyskać w odpowiedzi. Należy zatem podkreślić, że zapytania w GraphQL mogą być zawiłe i skomplikowane.

Podczas pobierania kompletu danych z kilku endpointów REST API i jednego endpointa GraphQL API uzyskane czasy przetworzenia żądania były krótsze dla aplikacji wykorzystującej GraphQL. Rozmiar otrzymanych danych był mniejszy w aplikacji korzystającej z GraphQL, a wysłanych w aplikacji z REST API.

W przypadku operacji dodawania, edycji i usuwania danych z bazy, porównywane były czasy przetwarzania pojedynczych żądań. Analiza czasów wykazała, że różnice w wydajności przetwarzania żądań były minimalne. Jeśli chodzi o wielkość wysyłanych i pobieranych informacji, we wszystkich wyżej wymienionych operacjach bardziej wydajny okazywał się interfejs opracowany na bazie technologii REST.

Literatura

[1] R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D, University of California, Irvine, 2000.

[2] R. T. Fielding, R. N. Taylor, Principled design of the modern Web architecture, Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium, Limerick, Ireland, 2000: 407-416, https://doi.org/10.1145/337180.337228, [10.07.2020].

[3] G. Brito, M. T. Valente, REST vs GraphQL: A Controlled Experiment, 2020 IEEE International Conference on Software Architecture (ICSA), 81-91, [10.07.2020].

[4] GraphQL, http://spec.graphql.org/, [10.07.2020].

[5] M. Prywata, Testowanie aplikacji i stron internetowych, Polska Agencja Rozwoju przedsiębiorczości, Warszawa, 2009, https://www.parp.gov.pl/publications/publication/testowanie-aplikacji-i-stron-internetowych, [09.07.2020].

[6] S. Sharmila, E. Ramadevi, Analysis of Performance Testing on Web Application, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 3, Issue 3 (2014), https://ijarcce.com/wp-content/uploads/2012/03/IJARCCE4H-s-sharmila-Analysis-of-Performance-Testing-on-Web-Applications.pdf, [10.07.2020].

[7] P. Marek, Weryfikacja i automatyzacja procesu testowania oprogramowania, CORE Magazine, 2010.

[8] S. Dhiman, P. Sharma, Performance Testing: A Comparative Study and Analysis of Web Service Testing Tools, International Journal of Computer Science

and Mobile Computing, Vol. 5, Issue 6, (2016) 507-512, https://www.ijcsmc.com/docs/papers/June2016/V5I6201697.pdf, [09.07.2020].

[9] M. Seabra, F. Nazario, G. Pinto, REST or GraphQL?: A Performance Comparative Study, In Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '19). Association for Computing Machinery, New York, NY, USA, 123–132. https://doi.org/10.1145/3357141.3357149, [27.08.2020].

[10] G. Brito, T. Mombach, M. T. Valente, Migrating to GraphQL: A Practical Assessment, In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 140-150, https://doi.org/10.1109/SANER.2019.8667986, [27.08.2020].

[11] M. Cederlund, Performance of frameworks for declarative data fetching: An evaluation of Falcor and Realy+GraphQL, Dissertation, KTH, 2016, http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-196058, [27.08.2020].

[12] T. Taskula, Advanced data fetching with graphql: Case bakery service, Dissertation, Aalto University, 2019, https://aaltodoc.aalto.fi/handle/123456789/37147, [27.08.2020].

[13] A. F. Helgason, Performance analysis of Web Services: Comparison between RESTful & GraphQL web services, Dissertation, University of Skövde, 2017, http://his.diva-portal.org/smash/record.jsf?pid=diva2%3A1107850&dswid=-9534, [29.08.2020].

[14] C. Oggier, How fast GraphQL is compared to REST APIs, Dissertation, Haaga-Helia University of Applied Sciences, 2020, http://urn.fi/URN:NBN:fi:amk-2020052714286, [29.08.2020].

[15] Apache JMeter, http://jmeter.apache.org/, [09.07.2020].