JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 19 (2021) 89–94
Received: 06 February 2021
Accepted: 13 February 2021

REST and GraphQL comparative analysis

Analiza porównawcza technologii REST i GraphQL

Piotr Margański*, Beata Pańczyk

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

Abstract

The article presents a comparative analysis of the two most commonly used API web design standards - REST and GraphQL. The time and size of HTTP responses returned by applications were tested. Two applications with the same functionalities, performing CRUD operations, on data stored in the non-relational MongoDB database were used for the research. Both applications were based on NodeJS technology. The JMeter tool was used to collect and analyze the data. On the basis of the obtained results, it was found that there were no significant differences in reading the data with a small number of queries and when removing resources. With the increase in the number of queries, a clear advantage of the REST standard was observed. The advantage of GraphQL, both in response time and size, was demonstrated when retrieving specific data.

Keywords: REST; GraphQL; API; application performance

Streszczenie

W artykule przeprowadzono analizę porównawczą dwóch najczęściej stosowanych standardów projektowania internetowego API – REST oraz GraphQL. Badano czas oraz rozmiar odpowiedzi HTTP zwracanych przez aplikacje. Do badań wykorzystano dwie aplikacje o takich samych funkcjonalnościach, realizujących operacje CRUD, na danych przechowywanych w nierelacyjnej bazie MongoDB. Obie aplikacje stworzono w oparciu o technologię NodeJS. Do zebrania i analizy danych zastosowano narzędzie JMeter. Na podstawie otrzymanych wyników stwierdzono brak znacznych różnic w odczycie danych przy małej liczbie zapytań oraz podczas usuwania zasobów. Wraz ze wzrostem liczby zapytań zaobserwowano wyraźną przewagę standardu REST. Przewagę GraphQL, zarówno w czasie jak i rozmiarze odpowiedzi, wykazano w przypadku pobierania specyficznych danych.

Słowa kluczowe: REST; GraphQL; API; wydajność aplikacji

*Corresponding author

Email address: **piotr.marganski@pollub.edu.pl** (P. Margański)

1. Wstęp

Wraz z coraz większą świadomością użytkowania aplikacji internetowych, wymaga się od nich dwóch podstawowych cech - niezawodności oraz szybkości. Aplikacje te zazwyczaj nie są pojedynczym programem wykonywalnym. Najczęściej obecnie stosowanym podejściem jest budowa aplikacji oparta na trójwarstwowej architekturze - frontend, backend i baza danych [1]. Elementy te muszą się ze sobą komunikować w odpowiedni i stabilny sposób. W tym celu wprowadzono pojęcie API (ang. Application Programming Interface). Jest to pewnego rodzaju zbiór standardów i protokołów umożliwiający wymianę danych pomiędzy wspomnianymi warstwami [2]. Obecnie można znaleźć i użyć, w zależności od decyzji architektonicznych, wielu implementacji tego interfejsu - między innymi REST, SOAP i GraphQL.

Najczęściej stosowanym standardem jest implementacja REST (ang. REpresentational State Transfer). Zaprezentowany w 2000 roku przez Roya Fieldinga był przedmiotem jego rozprawy doktorskiej [3]. REST nie jest technologią, ale zbiorem zasad narzucanych projektantowi API w taki sposób, aby tworzona implementacja opierała się na koncepcie zasobów identyfikowanych poprzez URI (ang. Uniform Resource Identifier).

W 2015 roku pojawiło się konkurencyjne podejście, rozwijana przez zespół deweloperski Facebooka technologia GraphQL [4]. Jej twórcy przedstawiali ją jako elastyczne rozwiązanie charakteryzujące się bardziej nowoczesnym podejściem do pracy z danymi podczas tworzenia usług sieciowych. Największe zainteresowanie budziła możliwość specyfikacji danych, które programista chce otrzymać, ponieważ jest to funkcjonalność, której REST a wcześniej SOAP nie były w stanie zapewnić.

Wobec tak spektakularnego pojawienia się nowego rozwiązania tworzenia API pojawiają się pytania dotyczące efektywności i łatwości implementacji. Zaprezentowane w niniejszym artykule wyniki mają na celu odpowiedzieć na te pytania i ułatwić wybór projektantom API szukającym odpowiedniego dla siebie rozwiązania.

1.1. Cel i obiekt badań

Celem badań jest analiza porównawcza dwóch implementacji API - REST oraz GraphQL. Analiza dotyczy porównania wydajności weryfikowanej czasem oraz rozmiarem odpowiedzi na żądania protokołu HTTP.

W zakres pracy wchodzi przedstawienie podstawowych pojęć, implementacja aplikacji do

przeprowadzenia testów, wykonanie testów, analiza wyników i sformułowanie płynących z niej wniosków.

W artykule podjęto próbę zweryfikowania następujących hipotez:

- Czas odpowiedzi na zapytanie jest krótszy a rozmiar odpowiedzi jest mniejszy w przypadku zorientowanego na zapytania GraphQL niż w powszechnie używanym REST.
- REST jest efektywniejszy od GraphQL w przypadku pobrania wszystkich danych.
- GraphQL będzie wydajniejszy niż REST tylko wtedy, gdy żądanie będzie dotyczyło pobierania konkretnych danych..

1.2. Przegląd literatury

Analiza dostępnej literatury dotyczącej REST, GraphQL i szeroko pojętego projektowania API jednoznacznie wskazuje, że dyskusje na temat najbardziej optymalnego rozwiązania dotyczącego tworzenia usług sieciowych wystawiających API nie są rozstrzygnięte. Przedmiot tej dyskusji jest wciąż poddawany badaniom i pojawia się w coraz bardziej licznych publikacjach naukowych. Jednym z wielu aspektów, który jest istotny przy wyborze odpowiedniego rozwiązania jest doświadczenie programistów, znajomość danej technologii a przede wszystkim stopień trudności zapoznania się z nią.

Zbadanie ostatniego punktu było przedmiotem badań Brito i Valente z Uniwersytetu Minas Gerais w Brazylii [5]. Na próbie 22 studentów (12 magistrantów oraz 10 studentów studiów licencjackich) dowiedli, że GraphQL wymaga mniej wysiłku i czasu do utworzenia usługi sieciowej niż REST nawet w przypadku osób, które w pracy z REST posiadały już doświadczenie.

Pontus Erlandsson oraz Joakim Remes z Uniwersytetu Skovde w Szwecji w swojej pracy licencjackiej dokonali analizy porównawczej GraphQL, REST oraz SOAP względem wydajności [4]. Osiągnięte przez nich rezultaty wykazały, że najsłabszą wydajnością spośród trzech wymienionych charakteryzuje się GraphQL. Jedynym przypadkiem, w którym czas odpowiedzi na zapytanie był najkrótszy, a jej rozmiar najmniejszy było pobieranie wybranych pól zasobu, niemniej jednak wraz ze wzrostem ich ilości czas się wydłużał a rozmiar się zwiększał, co w sposób negatywny zbliżało GraphQL do wyników osiąganych przez REST.

Zaproponowane przez C. Oggier z Haaga-Helia University of Applied Sciences badanie miało na celu zbadanie czy GraphQL jest szybszy i bardziej zoptymalizowany niż REST [6]. Osiągnięte przez nią rezultaty jednoznacznie wskazują na odpowiedź twierdzącą. W obu z przeprowadzonych testów, z których pierwszy badał żądania przesyłane sekwencyjne, a drugi równolegle, GraphQL osiągał krótsze czasy i mniejsze rozmiary odpowiedzi. W pierwszym przypadku GraphQL był szybszy o 46% i lżejszy o 21% niż REST, natomiast w przypadku drugim było to odpowiednio 35% i 71%.

W pracy dyplomowej API Design in Distributed Systems: A comparison between GraphQL and REST Thomas, Eizinger z University of Applied Sciences Technikum w Wiedniu wskazuje na kolejny znaczący obszar porównania [7]. Wykazuje, że GraphQL w zdecydowanym stopniu przenosi odpowiedzialność za pobierane i wyświetlane dane na stronę klienta, co prowadzi do zwiększenia jego złożoności.

Artykuł REST or GraphQL?: A Performance Comparative Study autorstwa M. Seabra, M.F. Nazario oraz G. Pinto, przedstawia badania dotyczące wydajności trzech aplikacji, mierzonej parametrami czasu oraz średniej szybkości transferu między żądaniami [8]. Każda z aplikacji została zbudowana wykorzystując architekturę REST, a następnie po wykonaniu pomiarów dokonano migracji do GraphQL. Badania wykazały wzrost wydajności po migracji w dwóch z trzech testowanych aplikacji. W przypadku liczby do 100 żądań oba standardy osiągały zbliżone wyniki w zakresie 6,34 - 7,68 żądań na jedną sekundę, jednak wraz ze wzrostem liczby żądań, GraphQL tracił przewagę, a po przekroczeniu granicy 3000 wydajność REST była już na zdecydowanie większym poziomie. Przedstawione badania nie rozstrzygają jednoznacznie, która z omawianych metod tworzenia API jest bardziej wydajna. Należy pamiętać, że wydajność nie jest jedynym kryterium decydującym o wyborze technologii do budowy architektury systemu. Ważnymi czynnikami są również doświadczenie zespołu, wysokość tak zwanego progu wejścia i potrzeby systemu.

2. Metoda badań

2.1. Aplikacje testowe

W celu przeprowadzenia badań zaimplementowano dwie aplikacje wystawiające API. Jedna z nich wykorzystuje technologię GraphQL, druga implementuje REST. Obie zostały stworzone w środowisku Node.js i korzystają ze wspólnej, nierelacyjnej bazy danych MongoDB. Dane przechowywane w bazie odzwierciedlały sytuację w tabeli angielskiej ligi piłkarskiej Premier League po każdej z dotychczas rozegranych kolejek spotkań. Listing 1 przedstawia definicję modelu pojedynczej kolekcji w Mongo (w relacyjnych bazach danych zwanej tabelą).

Listing 1: Kod programu - model tabeli bazy danych

```
const eventSchema = new Schema(
  {
    name: String,
    league_id: Number,
    season_id: Number,
    stage_id: Number,
    standings: [
      {
        position: Number,
        team_id: Number,
        team_name: String,
```

```
        overall: {
          games_played: Number,
          wins: Number,
          draws: Number,
          losses: Number,
          goals_scored: Number,
          goals_conceded: Number
        },
        home: {
          games_played: Number,
          wins: Number,
          draws: Number,
          losses: Number,
          goals_scored: Number,
          goals_conceded: Number
        },
        away: {
          games_played: Number,
          wins: Number,
          draws: Number,
          losses: Number,
          goals_scored: Number,
          goals_conceded: Number
        },
        total: {
          goal_difference: String,
          points: Number
        },
        points: Number,
        recent_form: String,
        status: String
      }
    ]
  }
);
```

Połączenie z bazą danych zostało zrealizowane za pomocą pakietu *mongoose* w wersji 5.10.11. Przykładowe fragmenty kodu mające na celu utworzenie zasobu w bazie danych w technologii GraphQL zostały przedstawione na Listingach 2 i 3.

Listing 2: Dodawanie zasobu w GraphQL

```
createStandings: (args) => {
  const standings = new Standings(
   {...args.standings}
  );
  return standings.save();
}
```

Listing 3: Mutacje GraphQL możliwe do wykonania

```
type Mutation {
 createStandings(standings: StandIn):
  Standings
deleteStandings(id: ID!): Standings
}
```

Listingi 4 i 5 przedstawiają tą samą operację zdefiniowaną w usłudze opartej na architekturze REST.

Listing 4: Dodawanie zasobu w REST

```
exports.create = (req, res) => {
  const standings = new Standings(
   {...req.body}
  );
  standings.save()
   .then(data => res.send(data))
   .catch(err => res.status(500)
    .send({ message: err.message }));
}
```

Listing 5: Endpoint dla dodania zasobu w REST

```
app.post('/standings', controller.create)
```

Aby zbadać wydajność obu usług wykorzystano zaimplementowaną w Javie aplikację Apache JMeter w wersji 5.3, która pozwala na zebranie zdefiniowanie, wykonanie i zebranie wyników puli zapytań, które wykonywane są w sposób wielowątkowy [9]. Żądania definiowane są dokładnie w taki sposób jak w standardowej aplikacji klienckiej, to znaczy podając protokół, domenę, numer portu, rodzaj zapytania oraz nazwę dla endpoint, pod który wysyłane jest zapytanie. Opcjonalnie podaje się parametry i ciało zapytania. Aplikacja ta w przeciwieństwie do przeglądarki nie wykonuje kodu JavaScript ani nie renderuje HTML, wobec tego nie trzeba brać pod uwagę ewentualnych zaburzeń w uzyskiwanych czasach spowodowanych tymi właśnie działaniami.

2.2. Środowisko testowe

W celu maksymalnego ograniczenia możliwych opóźnień generowanych przez sieć przeprowadzono badania na jednej maszynie posiadającej następujące parametry:

- Procesor - Intel(R) Core(™) i5-5200U, 2.20 GHz,
- Pamięć RAM - 8 GB,
- Typ systemu: 64-bit,
- System operacyjny - Windows 10 Education.

2.3. Scenariusze testowe

W ramach eksperymentu badawczego zrealizowano cztery scenariusze przedstawione w Tabeli 1.

Tabela 1: Scenariusze testowe

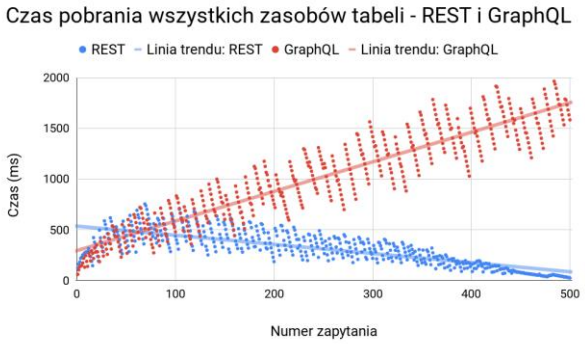
Lp.	Scenariusz
S1.	Zmierzenie czasu oraz rozmiaru odpowiedzi serwera na żądanie wszystkich zasobów danej tabeli.
S2.	Zmierzenie czasu oraz rozmiaru odpowiedzi serwera na żądanie pojedynczego zasobu z wyszczególnieniem konkretnych pól w przypadku GraphQL.
S3.	Zmierzenie jaki czas zajmie serwerowi przetworzenie zapytania POST, które dodaje pojedynczy rekord do bazy danych.
S4.	Zmierzenie jaki czas będzie potrzebował serwer aby usunąć wskazanego za pomocą identyfikatora zasób z bazy.

Jak wyszczególniono na początku artykułu, celem przeprowadzonych badań była analiza porównawcza omawianych technologii, szczególnie względem ich wydajności, z czasem i rozmiarem odpowiedzi jako głównymi jej parametrami. Badanie wydajności aplikacji internetowych jest zadaniem nietrywialnym, ze względu na dużą liczbę czynników, które mogą prowadzić do zaburzenia otrzymywanych wyników, między innymi obciążenia łącza internetowego lub szybkość reakcji. W związku z tym zdecydowano się na symulację działania potencjalnych użytkowników za pomocą narzędzia JMeter w izolowanym środowisku.

3. Wyniki badań

3.1. Wyniki testu pierwszego

W ramach testu pierwszego wykonano 500 zapytań w czasie 5 sekund co przekładało się na wykonanie kolejnego zapytania średnio co 0,01 sekundy. Na Rysunku 1 przedstawiono wyniki w formie wykresu.



Rysunek 1: Czas odpowiedzi REST i GraphQL na żądanie GET wszystkich zasobów.

Wyraźnie widoczna jest różnica czasu przetworzenia żądania oraz tendencja spadkowa wraz ze wzrostem liczby kolejnych zapytań w przypadku REST. Przedstawiono również szczegółowe dane dotyczące minimalnego, maksymalnego oraz średniego czasu i rozmiaru odpowiedzi (Tabela 2).

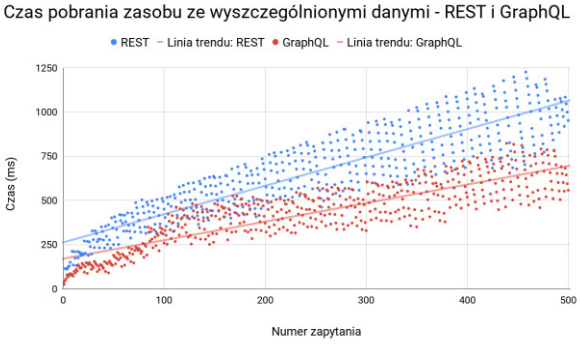
Tabela 2: Wyniki - pobieranie wszystkich zasobów

	REST		GraphQL	
Parametr	czas (ms)	rozmiar (B)	czas (ms)	rozmiar (B)
min	24	9533	59	9554
max	754	9533	1965	9554
średnia	309	9533	1024	9554

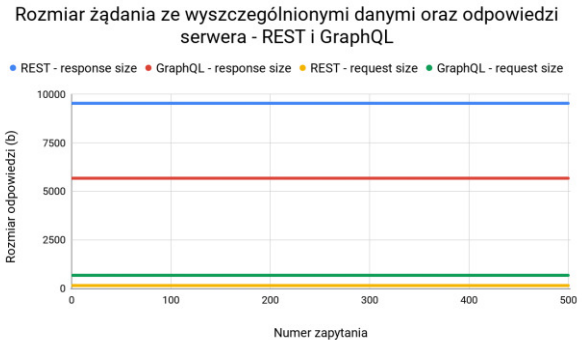
3.2. Wyniki testu drugiego

Drugi test, którego celem było pobranie danych wyszczególnionych w żądaniu, polegał na wysłaniu 500 zapytań w czasie 5 sekund. Rysunek 2 przedstawia wyniki czasu odpowiedzi serwerów, natomiast na Rysunku 3 przedstawiono rozmiar odpowiedzi.

Wyniki tego testu bardzo wyraźnie uwidaczniają powód tak zdecydowanego i dynamicznego wejścia w obszar technologii umożliwiającej projektowanie API. Najbardziej charakterystyczny element GraphQL, a więc pobranie tylko tych danych, które są potrzebne w sposób bezdyskusyjny pozwala na osiąganie krótszych czasów i mniejszych rozmiarów odpowiedzi. Tabela 3 w szczegółowy sposób prezentuje dane dotyczące minimalnego, maksymalnego oraz średniego czasu i rozmiaru odpowiedzi z obydwu usług.



Rysunek 2: Czas przetwarzania żądania konkretnych danych w REST i GraphQL.



Rysunek 3: Rozmiar odpowiedzi serwera REST i serwera GraphQL na żądanie konkretnych danych.

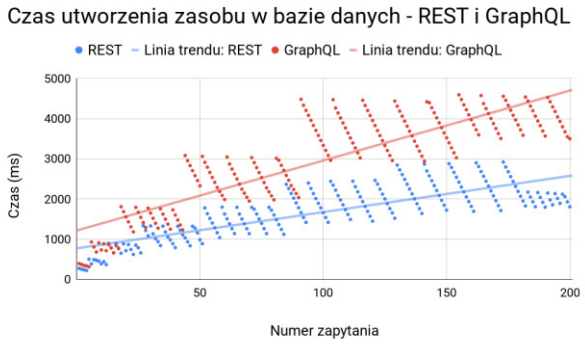
Tabela 3: Wyniki - pobieranie wybranych danych

	REST		GraphQL	
Parametr	czas (ms)	rozmiar (B)	czas (ms)	rozmiar (B)
min	41	9531	25	5674
max	1226	9531	823	5674
średnia	663	9531	432	5674

3.3. Wyniki testu trzeciego

Trzeci test, to wykonanie 200 żądań w czasie 5 sekund, których celem było utworzenie zasobu w bazie danych.

Rysunek 4 przedstawia otrzymane wyniki w formie wykresu, natomiast Tabela 4 prezentuje szczegółowe wyniki tego testu. Zauważono niepodważalną przewagę REST. Czas potrzebny na realizację żądania jest średnio o ponad połowę krótszy niż w przypadku GraphQL (56%).



Rysunek 4: Czas przetwarzania żądania POST w REST i GraphQL.

Tabela 4: Wyniki - dodanie zasobu do bazy danych

	REST		GraphQL	
Parametr	czas (ms)	rozmiar (B)	czas (ms)	rozmiar (B)
min	220	9531	312	275
max	2920	9531	4596	275
średnia	1678	9531	2967	275

3.4. Wyniki testu czwartego

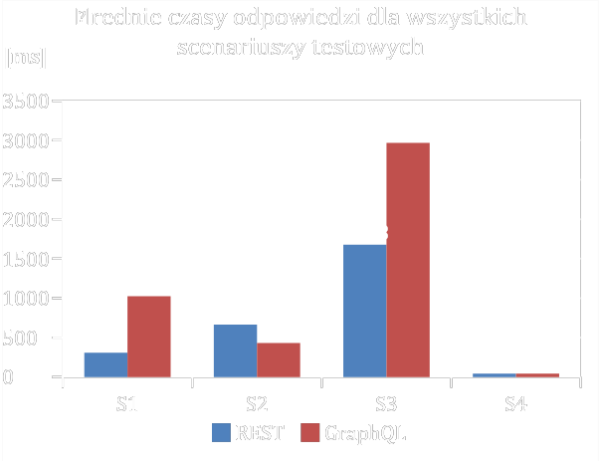
Celem testu czwartego było zmierzenie czasu potrzebnego serwerowi na usunięcie wskazanego zasobu z bazy danych. Aby to osiągnąć wykonano 50 takich żądań, których wyniki zebrano w Tabeli 5. Warto w tym miejscu wspomnieć, że takie żądanie nie jest łatwe do zoptymalizowania ze względu na konieczność podania innego identyfikatora w każdym żądaniu.

Tabela 5: Wyniki - usuwanie zasobu z bazy danych

	REST		GraphQL	
Parametr	czas (ms)	rozmiar (B)	czas (ms)	rozmiar (B)
min	38	244	39	275
max	49	244	46	275
średnia	44	244	43	275

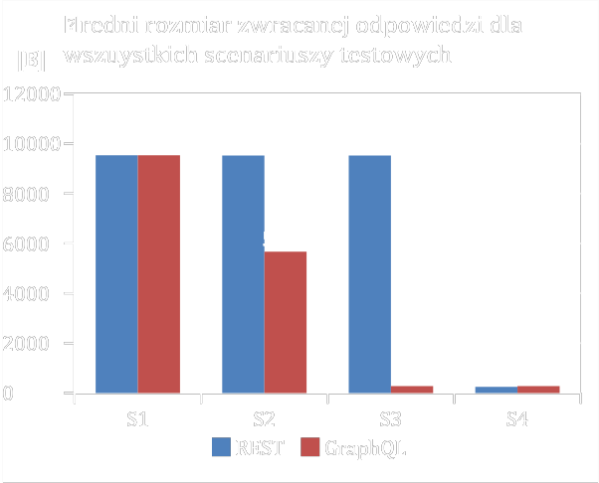
Osiągnięte rezultaty nie wykazują statystycznie istotnych różnic zarówno w czasie, jak i w rozmiarze odpowiedzi serwera. Można więc z całkowitym przekonaniem stwierdzić, że w przypadku usunięcia zasobu z bazy danych REST i GraphQL wykazują różnic w wydajności.

Rysunek 5 przedstawia średnie czasy potrzebne na przetworzenie żądania dla każdego z przeprowadzonych testów. Można zauważyć, że niezależnie od zastosowanego standardu najdłuższym czasem wykonania charakteryzował się przypadek utworzenia zasobu, natomiast najkrótszy czas osiągano w przypadku usunięcia zasobu z bazy danych.



Rysunek 5: Czas przetwarzania żądania POST w REST i GraphQL.

Przedstawiony na Rysunku 6 wykres przedstawia średni rozmiar odpowiedzi zwrotnej z serwera dla każdego z przeprowadzonych scenariuszy testowych.



Rysunek 6: Średni rozmiar odpowiedzi zwrotnej z serwera dla każdego scenariusza testowego.

4. Wnioski

Przeprowadzona analiza miała na celu wskazanie, który z badanych standardów projektowania programistycznego interfejsu aplikacji jest wydajniejszy. Na podstawie uzyskanych wyników można jednoznacznie stwierdzić że w przypadku pobierania wszystkich dostępnych zasobów z danej tabeli oraz tworzenia nowego zasobu w bazie danych REST jest zdecydowanie wydajniejszy od GraphQL. Niemniej jednak należy zaznaczyć że przewaga ta widoczna jest w przypadku przekroczenia liczby 100 żądań. Do wspomnianej granicy obie implementacje wykazują czas oraz rozmiar odpowiedzi na zbliżonym poziomie. Analizując wyniki czwartego testu dotyczącego usunięcia zasobu z bazy danych również zauważalny jest brak przewagi któregoś z serwisów. Różnica w czasie i rozmiarze odpowiedzi zwrotnej jest w tym przypadku statystycznie nieistotna. Ostatnią rzeczą, którą wykazała przeprowadzana analiza jest

zdecydowana przewaga GraphQL nad REST w czasie i rozmiarze odpowiedzi w przypadku ograniczenia ilości pobieranych danych. Wynika to wprost z możliwości i specyfikacji standardów. Stosując rozwiązanie oparte na REST użytkownik musi liczyć się z dodatkowym narzutem danych.

Otrzymane rezultaty pozytywnie weryfikują dwie z trzech postawionych hipotez - REST okazał się wydajniejszy w przypadku pobierania wszystkich zasobów z bazy, natomiast GraphQL był efektywniejszy wyłącznie w przypadku pobierania wyszczególnionych danych. Wyniki te są analogiczne do rezultatów zaprezentowanych w pracy P. Erlandssona i J. Remesa [4]. Hipoteza dotycząca osiągania krótszych czasów i mniejszych rozmiarów odpowiedzi przez GraphQL została zweryfikowana w sposób negatywny. Przeprowadzone badania nie potwierdzają rezultatów otrzymanych przez C. Oggier [6], jednak zbieżne są z wynikami eksperymentu zaprezentowanego w publikacji [8].

Biorąc pod uwagę otrzymane wyniki można stwierdzić, że wybór sposobu i technologii do projektowania API musi spotkać się z wymaganiami stawianymi przez klientów systemu, między innymi rozmiarem aplikacji i liczbą potencjalnych użytkowników. Bardzo istotnym czynnikiem jest również komfort pracy programisty implementującego rozwiązanie - w tym celu, aby ułatwić decyzję dotyczącą wyboru standardu, przedstawiono możliwości i przybliżono specyfikację zarówno REST jak i GraphQL.

Literatura

[1] M. Miłosz, Aplikacje internetowe - od teorii do praktyki, Polskie Towarzystwo Informatyczne, Warszawa, 2008.

[2] What is an API?, Mulesoft, https://www.mulesoft.com/resources/api/what-is-an-api, [22.01.2021].

[3] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Dissertation, University of California, Irvine, 2020, https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, [01.12.2020].

[4] P. Erlandsson, J. Remes, Performance Comparison between GraphQL, REST & SOAP, University of Skovde, Dissertation, http://his.divaportal.org/smash/record.jsf?pid=diva2%3A1449837&dswid=7389, 2020, [19.01.2021].

[5] G. Brito, M. T. Valente, REST vs GraphQL: A Controlled Experiment, IEEE International Conference on Software Architecture (ICSA), Salvador, Brazil, (2020) 81-91, doi: 10.1109/ICSA47634.2020.00016, [21.01.2021].

[6] C. Oggier, How fast GraphQL is compared to REST APIs, Haaga-Helia University of Applied Sciences, 2020, http://urn.fi/URN:NBN:fi:amk-2020052714286, [28.01.2021].

[7] T. Eizinger, API Design in Distributed Systems: A Comparison between GraphQL and REST, Master Thesis, University of Applied Science Technikum Wien, 2017, [20.01.2021].

[8] M. Seabra, M. E. Nazario, G. Pinto, REST or GraphQL?: A Performance Comparative Study, Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse. Association for Computing Machinery, New York, NY, USA, 2019, https://doi.org/10.1145/3357141.3357149, [29.01.2021].

[9] Apache JMeter, https://jmeter.apache.org/, [20.11.2020].