JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 21 (2021) 356–361
Received: 15 September 2021
Accepted: 04 October 2021

Performance comparison of programming interfaces on the example of REST API, GraphQL and gRPC

Porównanie wydajności interfejsów programistycznych na przykładzie REST API, GraphQL i gRPC

Mariusz Śliwa*, Beata Pańczyk

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

Abstract
The article presents a comparison of the performance of three ways of implementing programming interfaces used in web applications - REST, GraphQL and gRPC. For the purposes of the research, three applications were developed, which were made in each of the indicated technologies and with the same functionalities. The applications were used for performance tests carried out with the use of the k6 tool. The applications are used to measure the execution time, performance and volume of processed data during display and adding operations. The obtained results allowed for the conclusion that the best interface in terms of performance (measured as the number of transactions per second) and server response time is REST. However, in terms of the smallest data volume, gRPC is the best choice.

Keywords: REST; gRPC; GraphQL; performance testing

Streszczenie
W artykule przedstawiono porównanie wydajności trzech sposobów realizacji interfejsów programistycznych stosowanych w aplikacjach webowych – REST, GraphQL oraz gRPC. Na potrzeby badań opracowano trzy aplikacje, które zostały wykonane w każdej ze wskazanych technologii i o takich samych funkcjonalnościach. Aplikacje wykorzystano do testów wydajnościowych, przeprowadzonych z użyciem narzędzia k6. Aplikacje zastosowano do zmierzenia czasu wykonania, wydajności i objętości przetwarzanych danych podczas operacji wyświetlania oraz dodawania rekordów. Uzyskane wyniki pozwoliły na sformułowanie wniosku, że najlepszym interfejsem pod względem wydajności (mierzonych jako liczba wykonywanych transakcji na sekundę) oraz czasu odpowiedzi serwera jest REST. Natomiast pod względem najmniejszej objętości danych, najlepszym wyborem jest gRPC.
Słowa kluczowe: REST; gRPC; GraphQL; testy wydajnościowe

*Corresponding author
Email address: mariusz.sliwa@pollub.edu.pl (M. Śliwa)

1. Wstęp

1.1. Interfejsy programistyczne

Najbardziej popularne są obecnie aplikacje, które korzystają z komunikacji przez sieć. Takie aplikacje potrzebują pośredników w celu komunikacji między klientem oraz serwerem. W celu zdefiniowania semantyki i składni komunikatów przekazywanych przez sieć, powstały różne interfejsy programowania aplikacji, realizowane przy użyciu protokołów oraz specyfikacji, szerzej znane jako API (ang. Application Programming Interface) [1]. Zaletą takiego podejścia do projektowania architektury aplikacji jest umożliwienie szerokiej gamie fizycznych urządzeń klienckich i typów aplikacji, interakcji z daną aplikacją. Jedno API może być używane nie tylko do komputerów PC, ale także do telefonów komórkowych i urządzeń IoT (ang. Internet of Things). Komunikacja nie ogranicza się do interakcji między ludźmi a aplikacjami. Przez ostatnie lata pojawiały się różne interfejsy programistyczne, każdy z nich posiadał własne wzorce standaryzacji wymiany danych. Jak dotąd najczęściej używanym API jest REST (ang. Representational state transfer) [2], który zastąpił SOAP (ang. Simple Object Access Protocol). REST posiada wiele zalet, ale mimo to nie nadaje się do każdego rozwiązania. Właśnie dlatego powstały alternatywy, które mają na celu zastąpienie niedoskonałego REST API w różnych zastosowaniach. W 2015 roku pojawiło się konkurencyjne rozwiązanie o nazwie GraphQL, stworzone przez programistów Facebooka, które w krótkim czasie zdobyło bardzo dużą popularność. Kolejnym rozwiązaniem jest gRPC [3] (ang. general-purpose Remote Procedure Calls), które również pojawiło się w 2015 roku, natomiast twórcami tego interfejsu są programiści z Google.

Każde z tych API działa w odmienny sposób oraz posiada swoje zalety w różnych zastosowaniach, niestety w niektórych przypadkach programiści niepoprawnie decydują o wyborze interfejsu obsługującego system złożony z aplikacji, a ma to istotne znaczenie w przypadku stabilności oraz wydajności systemu. Dlatego należy rozpatrzeć mocne i słabe strony każdej z opcji, którą można potencjalnie zastosować w tworzonej aplikacji.

1.2. Przegląd literatury

Celem artykułu "REST vs gRPC vs GraphQL" [4] było porównanie różnic między REST a gRPC i Gra-

phQL. Omówiono zalety i wady każdego podejścia wraz z przypadkami użycia, które pozwoliłyby wybrać, które podejście najlepiej odpowiada potrzebom programistów. Zalety, jakie według autora posiada REST, to łatwa skalowalność, łatwa integralność, wydajność; natomiast wadami są m.in. trudności w trzymaniu się koncepcji HATEOAS (ang. Hypermedia As The Engine Of Application State), czyli ograniczenie architektury aplikacji REST z pomocą hipermediów, brak jednolitego stylu dokumentacji oraz wersjonowanie API. W przypadku gRPC zaletami są silne typowanie, wbudowany generator kodu, używanie protokołu HTTP/2, automatycznie generowanie dokumentacji, natomiast wadami są brak możliwości używania cache HTTP ze względu na używanie jedynie metody POST oraz brak wbudowanego narzędzia do debugowania oraz testowania. Zaletami GraphQL są łatwość w testowaniu, możliwość dokładnego wskazania czego potrzebuje klient od serwera, możliwość pobierania danych z różnych źródeł z użyciem tylko jednego zapytania, natomiast wadami są m.in. brak wbudowanego cache, problemy z zapytaniami cyklicznymi, ryzyko ujawnienia modelu danych. W artykule wskazano, że wybór sposobu realizacji API jest uzależniony od wymagań stawianych przed taką aplikacją i nie jest możliwe jednoznaczne wskazanie, który sposób realizacji API jest najbardziej odpowiedni.

"APIs REST, GraphQL or gRPC – Who wins this game?" [5] - artykuł, w którym autor porównywał różne sposoby realizacji aplikacji definiując wstępnie trzy główne przypadki użycia interfejsów. Były to: "Experience API" - interfejs do użytku przez aplikacje i urządzenia klienckie na potrzeby cyfrowych doświadczeń np. aplikacja mobilna łącząca się z wewnętrznym serwerem odpowiedzialnym za przechowywanie danych , "Open API" - otwarty interfejs służący do integracji z zewnętrznymi aplikacjami oraz "Internal API" - interfejs wewnętrzny służący do komunikacji między mikroserwisami. W celu porównania interfejsów, autor stworzył tabele z kryteriami, w których przyznawał wartości punktowe dla każdego API zależnie od spełniania każdego z kryteriów. W przypadku Experience API największą liczbę punktów uzyskał GraphQL ze względu na szybkość odpowiedzi oraz małą liczbę wykonywanych zapytań, W Open API najwięcej punktów miał REST ze względu na dużą możliwość ponownego użycia części aplikacji oraz łatwość w używaniu. W Internal API najlepiej poradził sobie gRPC dzięki dobrej skalowalności oraz optymalizacji zapytań.

W pracy naukowej "Performance analysis of Web Services: Comparison between RESTful & GraphQL web services" [6] autor przetestował wydajność interfejsu REST API oraz GraphQL mierząc czas odpowiedzi na zapytanie i rozmiar odpowiedzi. W pilotażowym teście okazało się, że REST jest szybszy dla prostszych strukturalnie zapytań, takich jak pobieranie informacji tylko z jednego źródła lub tabeli. Różnica w wydajności w zakresie czasu odpowiedzi rosła wykładniczo wraz z rozmiarem bazy danych. Pobierając więcej informacji, GraphQL generował wolniejsze odpowiedzi. Różnica zmierzona w dwóch eksperymentach przeprowadzonych w tych badaniach wynosiła 64-115%. W przypadku większych baz danych z możliwością tworzenia bardziej złożonych zapytań, GraphQL uzyskało bardzo dobry wynik. W drugim eksperymencie tych badan różnica wynosiła już tylko 25% . Przeprowadzono drugi test w celu potwierdzenia tych ustaleń. Ten test implementował to samo pobieranie danych procedury, jednak zamiast 100 iteracji, test ten powtórzono 1000 razy, co pokazuje, ze GraphQL jest o 51,75% szybszy niż usługa REST. Dzięki możliwości wybierania danych z dużo większa precyzja, całkowite rozmiary pakietów przesyłanych miedzy klientem i serwer można było zmniejszyć, prawdopodobnie skracając czas ładowania. Według autora precyzyjny wybór danych mógł również zmniejszyć prace jaka musi wykonać klient, ponieważ nie wymaga tyle logiki przetwarzającej dużej ilości danych pochodzących z serwera, a raczej przetwarza tylko te dane, które są potrzebne w tym momencie.

1.3. Cel badań

W Internecie można znaleźć wiele artykułów na temat zalet i wad REST API, GraphQL i gRPC. Mimo to wciąż niewiele artykułów pokazuje jaki tak naprawdę wpływ na wydajność mają różne sposoby realizacji komunikacji między aplikacjami. Celem tej pracy jest przeprowadzenie dokładnej analizy porównawczej wydajności tytułowych rozwiązań komunikacji. Zakres badań obejmuje: opracowanie aplikacji testowych, przygotowanie eksperymentu, dobór narzędzia diagnostycznego, zrealizowanie zautomatyzowanych testów wydajnościowych oraz opracowanie i interpretacja wyników.

Postawiono następującą hipotezę badawczą:

„Aplikacje opierające się na gRPC zużywają najmniejszą ilość danych oraz są najbardziej wydajne”.

2. Metody badań

2.1. Przypadki testowe

W celu przetestowania różnic między interfejsami programistycznymi - utworzono aplikacje w REST, GraphQL oraz gRPC z użyciem platformy .NET 5.0 [7]. Każda z tych aplikacji posiadała takie same funkcjonalności i służyła w praktyce do ewidencji autorów oraz ich książek. Każda z aplikacji została przetestowana pod względem czasu przetwarzania operacji, liczby transakcji na sekundę oraz objętości przetwarzanych danych.

2.2. Środowisko testowe

Do przeprowadzenia badań wykorzystano dwa komputery. Ich parametry znajdują się w Tabeli 1.

Tabela 1: Parametry pierwszego komputera

Procesor	AMD Ryzen 7 3800X 8 rdzeni, 16 procesorów logicznych, @ 4,4 GHz
Pamięć RAM	32 GB
Karta sieciowa	TP-LINK 802.11ac Network Adapter
System operacyjny	Windows 10 Education N

Do testów wydajnościowych użyto narzędzia k6 v0.33.0 [8]. Jest to narzędzie typu open source napisane w języku Go. k6 osadza silnik ECMAScript umożliwiając użytkownikom pisanie testów w JavaScript. Oprogramowanie to umożliwia automatyczne analizowanie wydajności różnych usług pod dużym obciążeniem z symulowaniem równoczesnej pracy wielu klientów.

2.3. Scenariusze badawcze

Dla każdej aplikacji opracowano eksperymenty, podczas których serwer przetwarzał zróżnicowaną ilość danych. Do testów użyto operacje pobierania oraz dodawania, ze względu na najczęstsze wykorzystywanie tych operacji w aplikacjach. Badanie pobierania zostało przeprowadzone z użyciem trzech zmiennych:

- liczba symulowanych użytkowników wysyłających zapytania (1, 10, 50),
- liczba zapytań wysłanych przez jednego użytkownika (50),
  liczba rekordów pobieranych z serwera (11, 111, 3112) oraz ich rozmiar.

W przypadku operacji dodawania danych, badanie zostało przeprowadzone z wykorzystaniem różnej liczby użytkowników wykonujących po 50 zapytań do serwera.

Do porównania wydajności aplikacji, wykorzystujących różne interfejsy programistyczne posłużono się kryteriami:

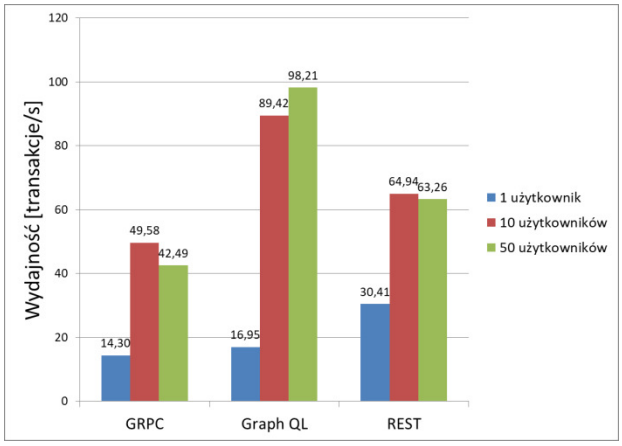
- średni czas przetworzenia żądań wyrażony w milisekundach,
- rozmiar danych w megabajtach (pobieranych lub wysyłanych),
- liczba transakcji na sekundę.

3. Wyniki badań

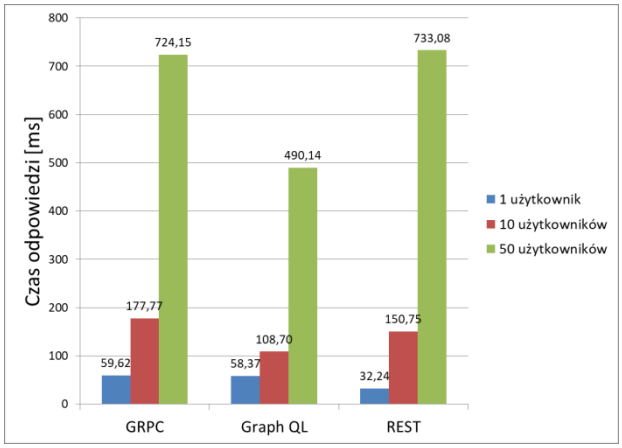
Badanie 1: *Pobranie 3112 rekordów autorów, u których w nazwie znajduje się fraza „Author - 1”*

W każdym z interfejsów, zostały przygotowane odpowiednie punkty końcowe do wysyłania zapytań o autorów, którzy w nazwie posiadali podaną przez użytkownika frazę. Zadaniem napisanych aplikacji było zwrócenie listy autorów z ich pełnymi danymi. Badanie zostało przeprowadzone dla trzech symulowanych grup użytkowników, wykonujących 50 zapytań każdy, a ich wynik został przedstawiony na rysunkach 1-3.

Na Rysunku 1 widoczne jest, że GraphQL najlepiej się sprawdza dla dużej liczby użytkowników wykonujących zapytania w jednym czasie. Porównując interfejs GraphQL z pozostałymi aplikacjami można zauważyć, że wraz ze wzrostem liczby użytkowników, zwiększa się również ilość transakcji na sekundę, co oznacza, że ten sposób realizacji dobrze się sprawdza przy pobieraniu dużej ilości danych równolegle przez wielu klientów. REST oraz gRPC w przeciwieństwie do GraphQL, w przypadku, w którym dane są pobierane przez 50 użytkowników, tracą na pod względem wydajności oraz czasu odpowiedzi.

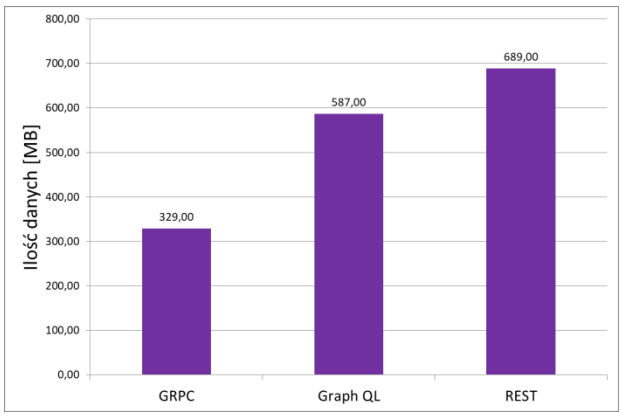


Rysunek 1: Wydajność interfejsów przy 3112 pobieranych rekordach.



Rysunek 2: Średni czas odpowiedzi interfejsów przy 3112 pobieranych rekordach.

Rysunek 3 przedstawia ilość łącznie otrzymanych danych dla 50 użytkowników wykonujących po 50 zapytań i pokazuje, że gRPC jest najbardziej optymalne pod względem jak najmniejszej objętości odebranych danych.

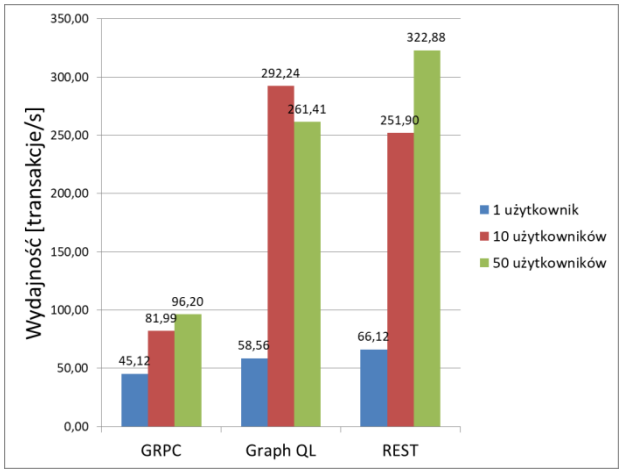


Rysunek 3: Ilość otrzymanych danych z punktów końcowych interfejsów przy 3112 pobieranych rekordach.

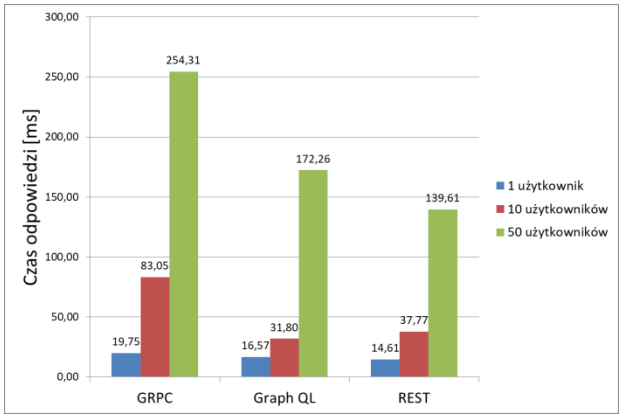
Badanie 2: *Pobranie 111 rekordów autorów, u których w nazwie znajduje się fraza „Author - 100”*

W tym badaniu, tak samo jak w poprzednim, zasymulowano wysyłanie zapytania o pełne dane autorów, których nazwa zawierała frazę „Author - 100”. Badanie zostało przeprowadzone dla trzech grup symulowanych użytkowników, a wyniki przedstawiają Rysunki 4-6.

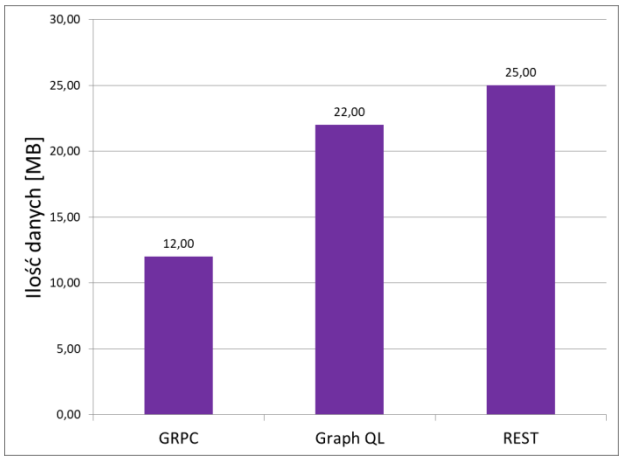
W tym teście widoczny jest wzrost wydajności interfejsu REST API względem poprzedniego badania, dzięki czemu wartości transakcji na sekundę oraz średniego czasu odpowiedzi zbliżyły się do wyników w GraphQL, a w przypadku 50 użytkowników wartość wydajności przekroczyła 300 transakcji na sekundę, czego nie udało się osiągnąć pozostałym aplikacjom. Rysunek dotyczący otrzymywanych danych ponownie wskazuje, że optymalnym rozwiązaniem pod tym względem jest gRPC, natomiast GraphQL oraz REST przesyłają dwukrotnie więcej danych.



Rysunek 4: Wydajność interfejsów przy 111 pobieranych rekordach.



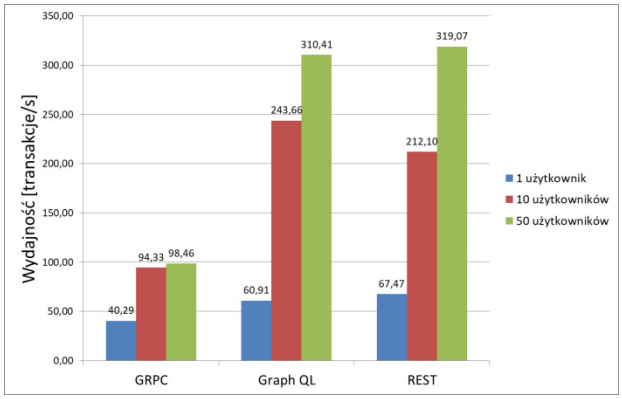
Rysunek 5: Średni czas odpowiedzi interfejsów przy 111 pobieranych rekordach.



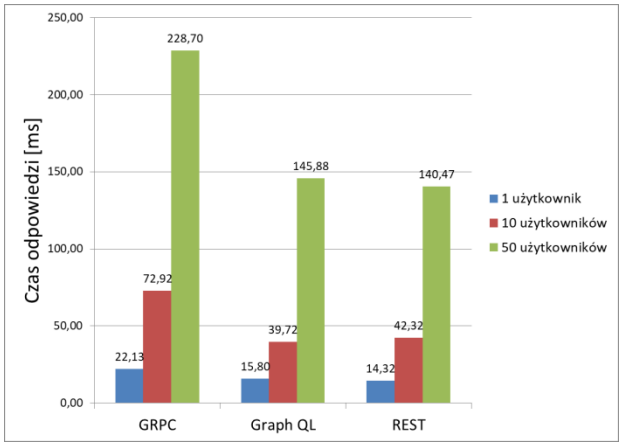
Rysunek 6: Ilość otrzymanych danych z punktów końcowych interfejsów przy 111 pobieranych rekordach.

Badanie 3: *Pobranie 11 rekordów autorów, u których w nazwie znajduje się fraza „Author - 1000”*

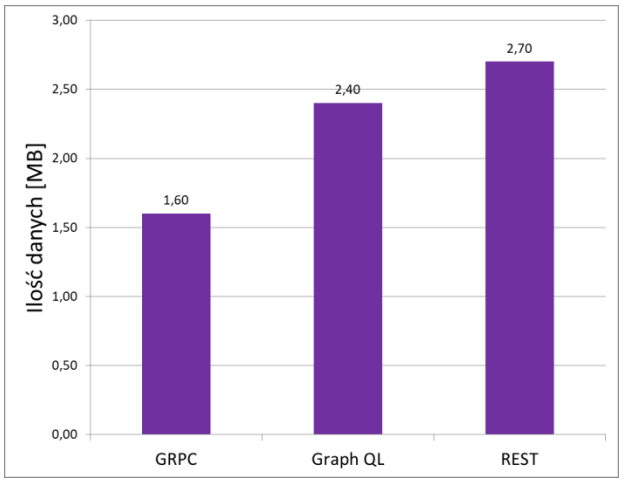
W ostatnim badaniu dotyczącym pobierania danych zasymulowano wysyłanie zapytania o pełne dane autorów, których nazwa zawierała frazę „Author - 1000”. Każda aplikacja miała zwrócić 11 rekordów z bazy danych. Badanie zostało przeprowadzone dla trzech grup symulowanych użytkowników, a wyniki pokazane są na Rysunkach 7-9.



Rysunek 7: Wydajność interfejsów przy 11 pobieranych rekordach.



Rysunek 8: Średni czas odpowiedzi interfejsów przy 11 pobieranych rekordach.

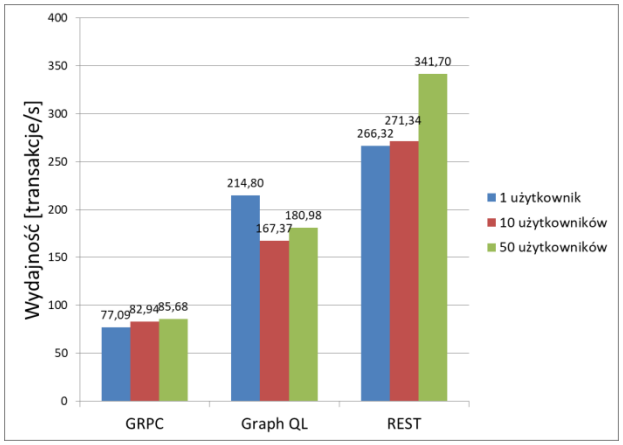


Rysunek 9: Ilość otrzymanych danych z punktów końcowych interfejsów przy 11 pobieranych rekordach.

Przy pobieraniu danych z serwera, po raz kolejny najgorsze wyniki uzyskał interfejs gRPC, który pod względem wydajności w sytuacjach wysyłania zapytania przez 10 oraz 50 użytkowników, jest słabszy 3 krotnie od pozostałych interfejsów. Aplikacje napisane z użyciem GraphQL oraz REST API ponownie zachowują zbliżone wartości wydajności oraz czasu odpowiedzi. Tak jak w poprzednich testach, rysunek 9 dotyczący otrzymanych danych, wskazuje na ciągłą przewagę gRPC nad pozostałymi interfejsami pod względem jak najmniejszą ilości otrzymanych danych.

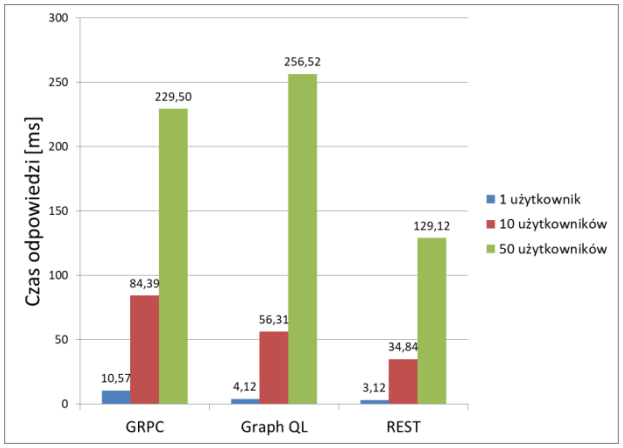
Badanie 4: *Dodanie autora do bazy danych*

Test dotyczący dodania autora do bazy danych zasymulowano poprzez wysyłanie danych zawierających nazwę oraz kraj autora. Każda z aplikacji po pomyślnym dodaniu autora zwracała odpowiedź zawierającą dodanego autora. Badanie zostało przeprowadzone dla trzech grup symulowanych użytkowników, a wyniki zostały przedstawione na rysunkach 10-12.

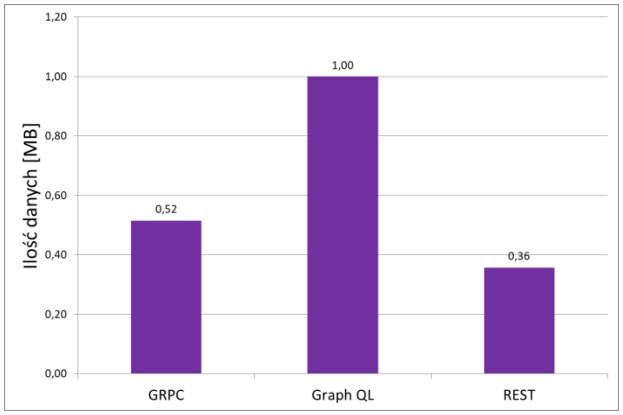


Rysunek 10: Wydajność interfejsów przy dodawaniu autora.

W przypadku dodawania autora do bazy danych, najlepszą wydajność uzyskał REST API, powiększając swoją przewagę nad pozostałymi interfejsami. Wraz ze wzrostem liczby użytkowników, uzyskał on dwukrotną przewagę nad GraphQL oraz czterokrotną nad gRPC. Pod względem czasu odpowiedzi, REST API uzyskał dwukrotnie lepszy wynik. Również w kwestii ilości wysłanych danych, REST API osiągnął wynik lepszy nawet od gRPC, który w testach pobierania danych był najbardziej optymalny.



Rysunek 11: Średni czas odpowiedzi interfejsów przy dodawaniu autora.



Rysunek 12: Ilość otrzymanych danych z punktów końcowych interfejsów przy dodawaniu autora.

4. Wnioski

Badania zrealizowane zostały na podstawie trzech aplikacji utworzonych w REST, GraphQL oraz gRPC, które posiadały maksymalnie zbliżone do siebie funkcjonalności. Pomiary czasu odpowiedzi, wydajności oraz ilości przesyłanych danych umożliwiło oprogramowanie k6. Zrealizowane badania aplikacji wskazały znaczne różnice między wykorzystanymi technologiami. Operacja pobierania danych, której w tym artykule poświęcono najwięcej uwagi wymagała pobrania wielu rekordów z bazy danych, dzięki temu można było sprawdzić jak każda z technologii radzi sobie z takim zadaniem. W przypadku dużej ilości pobieranych danych najlepiej sprawdzał się interfejs GraphQL, natomiast w przypadku średniej i małej wartości pobranych rekordów, wyniki wydajności REST API oraz GraphQL były do siebie bardzo zbliżone z delikatną przewagą REST API. Wyniki testów dotyczące REST oraz GraphQL pokrywają się z częścią pracy dotyczącej mniejszej ilości danych, którą wykonał A. F. Helgason [6], w której również REST miał lekką przewagę nad GraphQL w sytuacji,

gdy zapytania były prostsze strukturalnie. Niestety aplikacja napisana z użyciem interfejsu gRPC nie zdołała dorównać pozostałym przykładom, osiągając wyniki wydajności 2-4 krotnie słabsze pod względem wydajności oraz czasu odpowiedzi. W przypadku ilości danych pobranych podczas zapytań do serwera, gRPC wskazało swoją zaletę, jaką jest dwukrotnie mniejsza objętość danych w porównaniu do GraphQL oraz REST API. Testy poświęcone metodzie dodania autora do bazy danych podkreśliły przewagę REST API nad innymi rozwiązaniami. Interfejs ten osiągnął w kryterium wydajności oraz czasu odpowiedz najlepsze wyniki, osiągając dwukrotnie lepsze rezultaty od pozostałych interfejsów.

W Tabeli 2 podsumowano wyniki badań z wykorzystaniem punktacji w skali od 0 do 2 (im więcej punktów tym lepiej). W tabeli 2 zestawiono kryteria uwzględnione w badaniach, sumę punktów uzyskanych w danym kryterium oraz całkowitą sumę wszystkich punktów.

Tabela 2: Ocena punktowa technologii

Kryterium/Interfejs	gRPC	GraphQL	REST
Wydajność przy dużej ilości danych	0	2	1
Wydajność przy średniej ilości danych	0	1	2
Wydajność przy małej ilości danych	0	1	2
Wydajność przy dodaniu danych	0	1	2
Suma punktów kryterium wydajności	0	5	7
Czas odpowiedzi przy dużej ilości danych	0	1	0
Czas odpowiedzi przy średniej ilości danych	0	1	1
Czas odpowiedzi przy małej ilości danych	0	1	1
Czas odpowiedzi przy dodaniu danych	1	0	2
Suma punktów kryterium czasu odpowiedzi	1	3	4
Rozmiar odpowiedzi z serwera przy dużej ilości danych	2	1	0
Rozmiar odpowiedzi z serwera przy średniej ilości danych	2	1	0
Rozmiar odpowiedzi z serwera przy małej ilości danych	2	1	0
Rozmiar danych przy dodaniu danych	1	0	2
Suma punktów kryterium rozmiaru odpowiedzi	7	3	2
Suma punktów	8	11	13

W efekcie przeanalizowania wyników badań, otrzymane rezultaty udowadniają hipotezę na temat najmniejszego użycia danych przez gRPC, natomiast przypadek największej wydajności nie został potwierdzony.

Uwzględniając otrzymane wyniki badań, można stwierdzić, że ostateczna decyzja co do wyboru interfejsu programistycznego jest bardziej skomplikowana niż mogło by się wydawać. Przy wybieraniu sposobu w jakim będzie realizowana aplikacja musimy kierować się rozmiarem danych przesyłanych między użytkownikiem a aplikacją serwera, wydajnością urządzeń oraz liczbą użytkowników. Istnieją też kryteria, które wiążą się np. z wygodą tworzenia aplikacji przez programistów. Ze względu na swoją popularność i wygodę tworzenia kodu przez programistów, REST API jest najczęściej wybieranym rozwiązaniem, natomiast jak ukazały powyższe badania, nie zawsze jest to najlepsze rozwiązanie. W tym celu przedstawiono możliwości innych interfejsów programistycznych poprzez przetestowanie przykładowej aplikacji do ewidencji autorów oraz ich książek. Według artykułu Rafaela Rocha [5], aplikację tą można sklasyfikować jako "Experience API". Do tego typu aplikacji według autora najbardziej optymalnym rozwiązaniem byłby GraphQL, a najgorszym REST API, natomiast w pracy tej nie tylko aspekt wydajności był brany pod uwagę, ale również takie szczegóły jak możliwości monitorowania aplikacji. W przeciwieństwie do pracy Rocha, w powyższych badaniach wykazano, że REST API jest najlepsze, ale w przypadku dużej ilości danych równie dobrze sprawdza się GraphQL, natomiast w sytuacji wymagania małej objętości danych warto rozważyć użycie gRPC.

Literatura

[1] What is an API? (Application Programming Interface), https://www.mulesoft.com/resources/api/what-is-an-api, [10.09.2021]

[2] B. M. Balachandar, RESTful Java Web Services: A pragmatic guide to designing and building RESTful APIs using Java, 3rd Edition, Packt Publishing, 2017.

[3] Dokumentacja gRPC, https://grpc.io/docs, [01.03.2021]

[4] A. Tuban, REST vs gRPC vs GraphQL, https://technologyrivers.com/blog/rest-vs-grpc-vs-graphql, [01.03.2021]

[5] R. Rocha, APIs REST, GraphQL or gRPC – Who wins this game?, https://www.sensedia.com/post/apis-rest-graphql-or-grpc-who-wins-this-game, [01.03.2021]

[6] A. F. Helgason, Performance analysis of Web Services: Comparison between RESTful & GraphQL web services, University of Skövde, http://his.diva-portal.org/smash/record.jsf?pid=diva2:1107850, 2017, [01.03.2021]

[7] Wprowadzenie do .NET, https://docs.microsoft.com/en-us/dotnet/core/introduction, [10.09.2021]

[8] Dokumentacja JavaScript-owego API biblioteki k6, https://k6.io/docs/javascript-api, [10.09.2021]