UDC: 004.65 Info M str. 49-54

GRAPHQL: PREGLED I PRIMENA TEHNOLOGIJE U MIKROSERVISNOJ ARHITEKTURI
GRAPHQL: OVERVIEW AND APPLICATION OF TECHNOLOGY IN MICROSERVICE ARCHITECTURE

Jelica Stanojević, Miroslav Minović, Dejan Simić
Fakultet organizacionih nauka, Univerzitet u Beogradu
Faculty of Organizational Sciences, University of Belgrade

REZIME: GraphQL tehnologija je interno razvijena od strane Fejsbuka 2012. godine dok je njegova specifikacija javno objavljena 2015. godine i svoju popularnost je stekla kao alternativa, ali i bolji izbor od REST arhitekturalnog stila u određenim slučajevima. GraphQL API je definisan šemom koja opisuje dostupne mogućnosti za manipulaciju podacima kroz tipove. Tema rada je, pored sažetog pregleda ključnih koncepata GraphQL API tehnologije, njena primena u mikroservisnoj arhitekturi kako na samim mikroservisima, tako i u implementaciji moguće komponente ove arhitekture - API Gateway-a. API Gateway predstavlja ulaznu tačku klijentskim aplikacijama za interakciju sa sistemom čija kompleksnost može biti skrivena od njih upotrebom GraphQL-a kao intuitivnog i fleksibilnog interfejsa i u slučajevima implementacije API Composition paterna.
KLJUČNE REČI: GraphQL, API, Mikroservisi, API Gateway
ABSTRACT: GraphQL technology was developed internally by Facebook in 2012 while its specification was publicly released in 2015 and gained its popularity as an alternative, but also a better choice than the REST architectural style in certain cases. The GraphQL API is defined by a schema that describes the available options for manipulating data through types. The topic of the paper is, in addition to a brief overview of key concepts of GraphQL API technology, its application in microservice architecture both on microservices and in the implementation of possible components of this architecture - API Gateway. API Gateway is the entry point for client applications to interact with a system whose complexity can be hidden from them by using GraphQL as an intuitive and flexible interface, also in cases of API Composition pattern implementation.
KEY WORDS: GraphQL, API, Microservices, API Gateway

1. UVOD

S obzirom na to da se poslednjih godina može primeteti stabilan rast upotrebe GraphQL-a, GraphQL i njemu bliske tehnologije će se zadržati na duge staze [1]. U sprovedenoj anketi [1] nad oko 24000 Javaskript programera vidi se spomenut rast popularnosti GraphQL tehnologije iz godine u godinu, gde je 44% ispitanika do 2020. godine koristilo i imaju pozitivno iskustvo sa njom. U kontrolisanom eksperimentu [2] 22 studenta je implementiralo REST i GraphQL upite i došlo se do zaključka da je manje truda potrebno za GraphQL API čak i kod studenata koji su se prvi put susreli sa ovom tehnologijom, ali i kod onih koji su imali iskustva sa REST-om. Do danas postoji oko 200000 projekata koji koriste GraphQL, uz pomoć jednog od najpoznatijih projekata koji implementiraju specifikaciju ove tehnologije – Apollo GraphQL [3]. Na zvaničnom sajtu GraphQL-a mogu se videti široko poznate kompanije koje su prigrlile njegove benefite kao što su AirBnb, Github, Coursera, Meta, Paypal, ResearchGate, Shopify, Twitter i mnoge druge [4].

Pronalazi se sve više primena u kojima predstavlja bolji izbor od standardnog REST arhitekturalnog stila za kreiranje API-ja. GraphQL je dobar izbor ukoliko se zahtevi podataka često menjanju, dok je REST pogodan u slučajevima kada se često dovlače podaci kao i za monolitne informacione sisteme [5]. Takođe se primećuje porast alata i okvira za poboljšanje razvoja distribuiranih sistema [6], a jedna primena GraphQL-a je upravo u ovoj oblasti jer pruža intuitivni interfejs za interakciju sa sistemom koji je dostupan klijentskim aplikacijama i skriva njegovu kompleksnost.

Trend kreiranja informacionih sistema kroz niz međusobno povezanih mikroservisa je u porastu poslednjih godina [7]. Sa pojavom GraphQL-a koji je nastao kako bi promenio protokol razmene podataka u klijent-server aplikacijama usmeravanjem na klijente i njihove potrebe stvara se konkurencija REST-u. Za razliku od REST API-ja kod kog klijent u određenim slučajevima treba da pošalje više zahteva kako bi obezbedio resurse, u GraphQL-u je dovoljan jedan zahtev odnosno upit za dovlačenje povezanih resursa.

Kako je REST (Representational State Transfer) jedan od standarda koji se koristi prilikom kreiranja mikroservisnih sistema, nameće se pitanje da li se GraphQL i njegove osobine mogu primeniti u ovoj arhitekturi. Njegova popularnost raste i u industriji, a i akademskoj zajednici i može predstavljati jedno rešenje komunikacije u distribuiranim sistemima [6].

U nastavku će ukratko biti dat pregled osnovih koncepata GraphQL API tehnologije. Zatim će biti navedeni primeri primene GraphQL-a za koje je prepoznala akademska zajednica poslednjih par godina. Na kraju će ukratko biti objašnjene gradivne komponente sistema koji prati mikroservisnu arhitekturu i pojašnjene moguće primene GraphQL-a u ovoj arhitekturi. Takođe će biti spomenuti GraphQL koncepti koji se uklapaju u njenu osnovnu ideju.

2. OSNOVNI GRAPHQL KONCEPTI

GraphQL predstavlja upitni jezik koji je kreiran u Fejsbuku 2012. godine za opisivanje mogućnosti i zahteva modela podataka u klijent-server aplikacijama [8]. Pruža intuitivnu i fleksibilnu sintaksu i sistem [8] za dovlačenje podataka sa

API-ja, kao i za njihovu izmenu. Usmeren je ka klijentima i vođen je njihovim zahtevima koji se mogu predstaviti u formi hijerarhijski struktuiranih upita kroz koje se definiše i oblik povratnih podataka. Za razliku od servera koji nemaju GraphQL API i kod kojih upravo server određuje format podataka koji vraća kao odgovor, kod GraphQL-a klijenti biraju njima potrebne podatke [8].

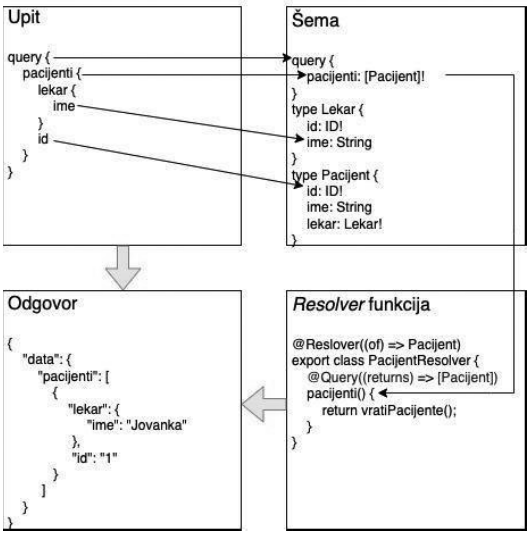
Početna verzija GraphQL specifikacije uz prateći Javascript projekat je javno objavljena 2015. godine i ubrzo je podržana od strane programera koji su krenuli da formiraju zajednicu oko GraphQL-a i da implementiraju izvršna okruženja u raznim programskim jezicima kao što su Go, Ruby, Scala, Java, Python itd. [9] Trenutno je aktuelna šesta verzija GraphQL specifikacije [8] iz oktobra 2021. godine koja predstavlja prvu verziju potvrđenu od strane GraphQL fondacije [10].

Ideja o GraphQL-u se javila u Fejsbuku gde je i razvijen 2012. godine za internu upotrebu nad njihovim mobilnim aplikacijama sa prvobitnom namenom da se izbori sa problemima koje prouzrukuje REST arhitekturalni stil. U početku je Fejsbuk mobilna aplikacija predstavljala verziju sajta za mobilne telefone obgrljenog komponentom koja prikazuje veb stranice u mobilnim aplikacijama što je uticalo na loše performanse i često prekidanje rada aplikacije kako je njena kompleksnost rasla. Tranzicijom na nativne aplikacije se javila potreba za dovlačenje podataka potrebnih za početnu stranu aplikacije preko API-ja, a ne u vidu html-a, u čemu je REST ispoljio mane. [9]

Ključni deo GraphQL servisa predstavlja GraphQL šema. GraphQL šemu opisuju tipovi, direktive koje podržava, kao i dostupne operacije: upiti (čitanje podataka), mutacije (unos, izmena i brisanje podataka) i supskripcije (podaci kao odgovor na događaje) [8]. GraphQL jezik za definisanje šeme (engl. *Schema Definition Language* – SDL) je opisan u specifikaciji [8] i mora biti isti za sve implementacije GraphQL-a u različitim programskim jezicima [11].

U zavisnosti od tipa operacije zahteva postoje razlike u njihovom načinu izvršavanja. Definicija operacije sadrži tip operacije, kao i tražena polja koja se navode u vitičastim zagradama nakon operacije. Svako polje se može posmatrati kao funkcija koja pripada nadtipu i vraća ugnježden tip. Ove funkcije se nazivaju *resolver* funkcijama i programer GraphQL API-ja je zadužen za implementaciju njihove logike. Zahtevana polja mogu biti ugnježdavana sve do skalarnih vrednosti i tada izvršavanje upita prestaje. [8]

Na *Slici 1* je prikazana evaluacija upita od strane GraphQL okvira. Na osnovu šeme se može kreirati upit za koji se poziva odgovarajuća *resolver* funkcija koja formira odgovor. Odgovor se vraća klijentu u JSON formatu [6]. Zbog definisanih tipova u šemi moguće je proveriti sintaksnu validnost upita pre njegovog izvršavanja [8].



Slika 1 GraphQL mehanizam upita [6]

3. MOGUĆA PRIMENA GRAPHQL TEHNOLOGIJE

Sve više se proširuju mogućnosti primene GraphQL-a i po pitanju unapređenja tehničkih rešenja i sistema, ali i sfera oblasti primene. U nastavku su spomenuti doprinosi radova u proteklih par godina vezani za moguće oblasti primene GraphQL-a.

U [12] se prikazuje podobnost GraphQL-a za komunikaciju sa veb servisima koji pružaju bioinformatičke resurse i njegova implementacija na *ZincBind* projektu [13] za koju smatraju da je olakšana prelaskom na GraphQL sa REST-a. GraphQL je jako pogodan za potrebe i karakteristike bionformatičkih resursa. Posebno ističu upravljanje veličinom odgovora kroz definisanje samo potrebnih polja i veza prilikom formiranja upita. Takođe, jako značajna karakteristika GraphQL-a za oblast bioinformatike predstavljaju supskripcije zbog zahtevnih asinhronih zadataka koji ne mogu biti izvršeni kroz jedan ciklus HTTP zahteva i odgovora. [12]

U oblasti zdravstvene nege i elektronskih sistema zdravstvenih kartona u [14] je ponuđena arhitektura koja koristi server resursa koji je zasnovan na GraphQL-u i HL7 FHIR standardu u zdravstvenoj nezi za postizanje interoperabilnosti između heterogenih reprezentacija podataka odnosno uspešne komunikacije između dva heterogena elektronska sistema zdravstvenih kartona. Zapažen je porast arhitektura, dizajn paterna, razvojnih pradigmi, eksperimenata i okvira koji koriste prednosti GraphQL-la i podržavaju interoperabilnost [14]. U [15] je prezentovan algoritam za mapiranje HL7 FHIR resursa u GraphQL šemu i kreiran prototip implementacije koji je upoređen sa REST-om i došlo se do zaključka o visokim performansama, skalabilnosti i fleksibilnosti kombinacije GraphQL-a i HL7 FHIR standarda na veb API-ju za razmenu informacija zdravstvene nege. U [16] gde je blokčejn primenjen u oblasti zdravstva za integraciju podataka o pacijentima GraphQL upiti se koriste za dovlačenje podataka za čuvanje na blokčejn čvorovima.

U oblasti nauke i istraživanja *Open Research Knowledge Graph* (ORKG) [17] implementira servis baziran na *GraphQL Federation* arhitekturi za integraciju više naučnih komunikacionih infrastruktura (npr. DataCite, ORCID, Semantic Scholar itd.) time pružajući širok spektar informacija vezanih za navedenu oblast. Većina API-ja kroz koje se prikupljaju različite vrste informacija npr. o naučnim radovima predstavljaju REST API-je (Altmetric, OpenAIRE, SemanticScholar, Wikidata). DataCite pruža podatke o naučnim radovima i njihovim međusobnim vezama i sa organizacijama, pokroviteljima i projektima preko GraphQL API-ja. ORKG integriše podatke sa spomenutih API-ja i apstrahuje tehnološku heterogenost ovih naučnih infrastruktura i time omogućava izvršavanje kompleksnih distribuiranih upita kroz jednu pristupnu tačku. [18]

U [19] je koristeći GraphQL implementiran prototip modula za upravljanje projektima kako bi se integrisali entiteti podataka dostupni u starim bazama i time omogućio dosledan pristup i isporuka podataka koji se nalaze na heterogenim izvorima. GraphQL, u odnosu na druge servisne arhitekture koje koriste REST, ublažava problem prekomernog ili nedovoljnog dovlačenja podataka za klijentsku aplikaciju [19].

U oblasti IoT-a efikasno korišćenje energije je jako važno jer minorna poboljšanja u očuvanju energije mogu da dovedu do značajnog poboljšanja životnog veka IoT uređaja i cele mreže [20]. U [20] su iskoristili GraphQL i njegove odgovarajuće osobine i na taj način postigli oko 50% poboljšanja u pogledu korišćenja energije i kašnjenja transakcija.

PostGraphile [21] i *Hasura* [22] su alati za automatsko generisanje GraphQL šeme i servera na osnovu šema baza podataka. U [23] su analizirane performanse tako dobijenih GraphQL servera za rad sa *legacy* bazama podataka.

Na osnovu dela spomenutih radova može se zaključiti da GraphQL nalazi primenu u slučajevima kada je potrebno izvršiti spajanje podataka sa različitih izvora. Upravo se u sistemima mikroservisne arhitekture može susresti sa ovakvim slučajem korišćenja, gde je potrebno spajati podatke sa više različitih mikroservisa kako bi npr. korisnička aplikacija dobila sve potrebne podatke koji su distribuirani kroz jedan zahtev. U narednom poglavlju će fokus biti upravo na ovome i biće opisane i dodatne mogućnosti primene GraphQL-a u ovoj arhitekturi.

5. GRAPHQL I MIKROSERVISNA ARHITEKTURA

5.1. *Osnovne komponente mikroservisne arhitekture*

Gruba definicija servisno-orijentisane arhitekture (SOA), kao arhitekturalni stil prema kome se sistem sastoji iz kolekcije servisa koji međusobno sarađuju, može se iskoristiti i kao opis mikroservisne arhitekture. Treba napomenuti da je prisutna diskusija na temu da li je mikroservisna arhitektura upravo to što njen naziv kaže arhitekturalni stil [24] ili predstavlja samo jedan od ispravnih načina implementacije SOA arhitekture [25]. Servisno-orijentisana arhitektura, a samim tim i mikroservisna, se pojavila kako bi se programeri izborili sa izazovima koji dolaze sa velikim monolitnim aplikacijama.

Mikroservisi su glavne komponente sistema koji prati smernice mikroservisne arhitekture. Oni predstavljaju nezavisne servise manjeg obima koji međusobno sarađuju preko mreže i koji su implementirani nad pažljivo određenim poslovnim funkcionalnostima ili poddomenima kako bi bio olakšan rad i agilnost timova. Podržavaju agilne metode i promovišu izolaciju i autonomiju [7].

Karakteristike koje opisuju servise ovog arhitekturalnog stila su [26]:

- održavanje i testiranje na visokom nivou
- labavo spregnuti (engl. *loosely coupled*)
- nezavisno razvijani i raspoloživi (engl. *independently deployable*)
- organizacija oko poslovnih mogućnosti
- vlasništvo malobrojnih timova.

Takođe, *API Gateway* može predstavljati komponentu mikroservisnog sistema i time dosta olakšati komunikaciju klijentskih aplikacija sa distribuiranim sistemom. Ujedno je i dizajn patern koji se najviše susreće u ovoj arhitekturi [7]. Predstavlja ulaznu tačku klijentskim aplikacijama i zadužen je za rutiranje zahteva, implementaciju *API Composition* paterna, autentikaciju itd. [27] Upravo se uspešna primena GraphQL-a može videti njegovom implementacijom u ovom delu sistema o čemu će biti više reči u nastavku rada.

5.2. *Graphql api mikroservisi*

Jedna od mogućih implementacija mikroservisa je u vidu API-ja uz REST specifikaciju, koji pružaju funkcionalnosti na korišćenje kroz krajnje tačke (engl. *endpoints*) preko HTTP-a ostalim mikroservisima i krajnjim korisnicima. Na popularnost implementacije mikroservisa u vidu REST API-ja utiče jednostavnost direktne komunikacije preko HTTP-a bez potrebe za dodatnom infrastrukturom [28]. Osim što REST pruža programerima jednostavno rešenje za implementaciju interprocesne komunikacije, takođe ukoliko se iskoristi na pravi način može povoljno uticati na fleksibilnost, skalabilnost, labavu spregnutost i uklapanje u celinu odnosno kompozitnost samih mikroservisa u distribuiranoj arhitekturi [29].

GraphQL tehnologija može biti iskorišćena u mikroservisnoj arhitekturi za implementaciju API sloja u mikroservisima. Nameće se diskusija da li je u tom slučaju GraphQL alternativa ili mogući dodatak malopre spomenutom REST arhitekturalnom stilu [29].

Zbog osobine GraphQL API-ja da može da sadrži samo jednu krajnju tačku kojoj se kroz POST metodu šalju upiti i mutacije sa zahtevanim poljima, GraphQL se može implementirati nad već postojećim servisima i API-jima. Pored / graphql krajnje tačke mogu biti definisane i standardne REST krajnje tačke tako da se dobiju GraphQL i REST API zajedno. Jedna mogućnost za implementaciju prethodno rečenog je da se smanji ponavljanje koda kroz translaciju REST zahteva u GraphQL upite ili mutacije i na taj način omogući da API podržava i GraphQL i REST način komunikacije. [30]

Takođe, moguće je izvršiti migraciju postojećih REST servisa na GraphQL servise. Vogel i ostali su uspeli da izvrše ovakvu migraciju kućnog IoT sistema koristeći već postojeće servise koje je koristio i REST API i tako dobili oba načina komunikacije koja funkcionišu zajedno bez ograničenja [29].

Što se tiče tehnologija koje mogu biti korišćene za njihovo kreiranje, mikroservisi su tehnološki agnostični. Samim tim implementacioni jezik mikroservisa ne utiče na upotrebu GraphQL-a, jer ga je vrlo ubrzo nakon objavljivanja GraphQL-a zajednica programera podržala i implementirala specifikaciju za mnoge programske jezike u koje spadaju Java, Perl, Python, C#, Ruby, C++ i drugi.

Kako je SOA nastala kako bi se izborila sa izazovima monolitnih aplikacija, a mikroservisna arhitektura kao odgovor na mane SOA arhitekture, upravo jednu od tih mana predstavlja nedovoljno smernica za određivanje granica i opsega samih servisa. Kroz razvoj vođen domenom (engl. *domain-driven design*) i njegove koncepte je moguće olakšati ovaj proces. Dakle, procesu definisanja granica mikroservisa može pomoći Razvoj vođen domenom kao i definisanju domenskog modela na osnovu koga može biti kreirana GraphQL šema. Kod definisanja interfejsa koji klijenti koriste, preporuka je razdvajanje ključnog domena od istog, jer direktno mapiranje ključnog domenskog modela može da dovede do nekompatibilnosti sa klijentima ukoliko dođe do njegove izmene [31].

Iz ovoga se može zaključiti da i kod kreiranja GraphQL šeme treba kreirati interfejs koji ne oslikava u potpunosti ključni domenski model, već zavisi i od samih slučajeva korišćenja. Treba da predstavlja standardizovan interfejs na koji neće uticati promene u ključnom domenu [31].

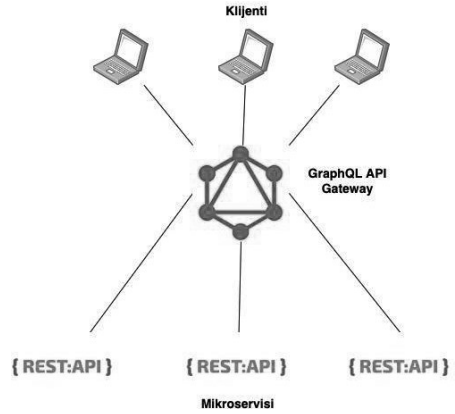
5.3. Graphql api gateway

GraphQL u mikroservisnoj arhitekturi sistema može pronaći svoju primenu i kod implementacije *API Gateway*-a. Upravo je i preporuka tima koji radi na održavanju i razvoju GraphQL-a da u okviru mikroservisne arhitekture postoji jedna GraphQL šema kao API Gateway kako klijenti ne bi morali da komuniciraju sa više (GraphQL) servisa [32]. GraphQL *API Gateway* je zadužen da prima upite i mutacije od strane klijentskih aplikacija koje treba da prosledi u odgovarajućem formatu mikroservisima i na kraju vrati odgovor od strane jednog ili više mikroservisa nazad klijentu. Kod GraphQL upita i mutacija, klijent može sam da izabere polja koja su mu potrebna na osnovu ponuđenih preko povratnih vrednosti operacija u šemi. Za razrešavanje polja, odnosno vraćanje odgovarajućih vrednosti za zahtevanu operaciju, potrebno je implementirati *resolver* funkcije na GraphQL API-ju.

Kao što je spomenuto, ukoliko GraphQL *API Gateway* treba da sublimira odgovore sa više mikroservisa u jedan odgovor za jedan upit moguće je definisati šemu tako da složeni tipovi sadrže odgovarajuća polja sa pratećim *resolver* funkcijama kroz koje komuniciraju sa mikroservisima i koji će biti kontaktirani samo u slučaju navođenja polja kao potrebnog prilikom definisanja upita sa klijentske strane. Naravno, moguće je definisati i odvojene upite u ove svrhe ukoliko se prethodno navedeno ne uklapa u slučaj korišćenja i samu šemu. Ova funkcionalnost predstavlja implementaciju paterna API kompozicije (engl. *API Composition*) u kome ulogu API kompozitora ima *API Gateway* [27].

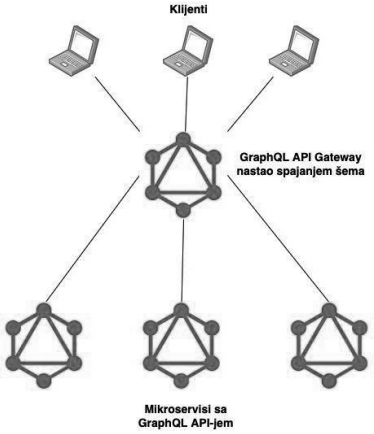
Na osnovu dostupne šeme sistema klijent vidi dostupne strukture podataka i operacije kroz koje interaguje sa sistemom, kao i informacije o potrebnim podacima kako bi to uspešno činio. Interna implementacija sistema je skrivena iza šeme.

Ukoliko mikroservisi pružaju HTTP/REST krajnje tačke šema *API Gateway*-a bi bila kreirana na osnovu njih. Šema bi definisala tipove povratnih vrednosti koje vraćaju krajnje tačke, kao i upite i mutacije u zavisnosti od funkconalnosti koje pružaju, odnosno REST krajnje tačke će biti transformisane u GraphQL šemu koja će biti dostupna klijentima na korišćenje preko *API Gateway*-a. Na taj način klijenti ne moraju biti upoznati sa konkretnom podelom sistema na mikroservise, niti voditi računa o njima, već će preko *API Gateway*-a komunicirati sa sistemom i time smanjiti kompleksnost klijentskih aplikacija.



Slika 2 GraphQL API Gateway nad mikroservisima sa REST API-jem [33]

Spajanje šema (engl. *Schema Stitching*) je proces kreiranja jedinstvene šeme na osnovu više GraphQL šema [34]. Spajanje šema može biti izvršeno nad svim mikroservisima u sistemu ukoliko svi pružaju GraphQL API (Slika 3). Objedinjena šema u tom slučaju predstavlja šemu *API Gateway*-a. Moguće je takođe izvršiti i spajanje šema više logički povezanih mikroservisa pre kombinovanja šema radi kreiranje jedne integralne kroz koju će klijenti komunicirati sa sistemom.



Slika 3 GraphQL API Gateway nad mikroservisima sa GraphQL API-jem [33]

Apollo predstavlja jednu od GraphQL implementacija koje su standardno korišćene u industriji i do danas postoji oko 200000 projekata koji koriste ovu implementaciju [3]. *ApolloServer* predstavlja server otvorenog koda koji odgovara GraphQL specifikaciji i kompatibilan je sa bilo kojim GraphQL klijentom, kao i sa *ApolloClient*-om.

Takođe, Apollo projekat pruža *Apollo Federation* distribuiranu arhitekturu koja se sastoji iz kolekcije servisa koji imaju odvojene GraphQL šeme i *Gateway* čija je uloga spajanje šema u graf podataka i izvršavanje upita kroz servise u grafu. Na taj način je moguće odvojiti odgovornosti po servisima i olakšati razvoj timovima. [35]

Apollo platforma takođe pruža još jednu mogućnost koja može doneti benefite i uticati pozitivno na performanse mikroservisnog sistema, a to je kreiranje perzistentnih upita zbog kojih je moguće definisati dozvoljene upite, a i skratiti vreme potrebno za validaciju upita. Na taj način je moguće smanjiti veličinu zahteva koji može sadržati predugačak upit tako što se šalje jedinstveni identifikator upita koji je prethodno keširan na serverskoj strani sa tim identifikatorom, ka GraphQL API-ju, što može povoljno uticati na performanse. [36]

Takođe, jedna od mogućnosti primene GraphQL-a na koncepte mikroservisne arhitekture jeste da se operacija supskripcije definisana u specifikaciji GraphQL-a iskoristi u implementaciji reagovanja na domenske događaje [33]. Dostupni tipovi, upiti i mutacije su samodokumentovani zbog kreiranja API-ja kroz šemu prema pravilima GraphQL specifikacije. Klijenti mikroservisne arhitekture će uvek imati ažurnu dokumentaciju funkcionalnosti sistema bez korišćenja dodatnih alata za njeno održavanje. Novina u GraphQL-u, koja ne postoji kod REST API-ja zbog osobine da polja koja klijent dobija kao odgovor su predefinisana, dok kod GraphQL-a klijent bira polja odgovora, jeste ta da je moguće voditi računa o konkretnim poljima koja se zahtevaju i na osnovu toga prilagođavati API [33]. Polja koja se ne zahtevaju mogu se označiti zastarelim ključnom rečju *deprecated* koja se navodi u GraphQL specifikaciji [8]. Takođe, preporuka je da autorizaciona logika ne bude smeštena na GraphQL API sloju, već na sloju poslovne logike, što može dodatno olakšati razvoj više API-ja paraleno, kao npr. i REST i GraphQL API-ja, jer se autorizaciona logika neće ponavljati [33].

4. ZAKLJUČAK

U slučajevima spajanja podataka sa različitih izvora, koji mogu biti i heterogeni, GraphQL API može odigrati ulogu posrednika koji će biti zadužen za skrivanje kompleksnosti koja se krije iza sublimiranih podataka. Takođe utiče na smanjenje prekomernog, ali i nedovoljnog, dovlačenja podataka što može uticati povoljno na performanse. Upravo sve ovo dovodi do zaključka da GraphQL upitni jezik može biti uspešno iskorišćen u arhitekturama distribuiranog tipa.

Prednost primene GraphQL tehnologije u mikroservisnoj arhitekturi se može primetiti prilikom implementacije *API Gateway*-a koji može predstavljati gradivnu komponentu mikroservisnog sistema. Samodokumentovan API na osnovu kreirane šeme pruža intuitivan interfejs klijentskim aplikacijama kroz definisane vrste objektnih tipova, upita, mutacija, supskripcija i njihovih povratnih vrednosti, tako da je vođenje računa o dokumentaciji olakšano. Koncept spajanja šema više API-ja u jednu šemu *API Gateway*-a se može iskoristiti kao prednost ukoliko svi mikroservisi poseduju GraphQL interfejs. Čak i da ne poseduju moguće ih je dodatno je definisati jer GraphQL i REST API mogu postojati zajedno, ali ovaj pristup zahteva održavanje oba načina komunikacije. *API Compostion* patern može biti uspešno implementiran u mikroservisnoj arhitekturi uz korišćenje GraphQL-a jer je šemu moguće formulisati tako da sa strane klijenta izgleda kao da se komunicira sa monolitnim sistemom dok je u pozadini GraphQL *API Gateway* zadužen za kompoziciju odgovora kroz komunikaciju sa više mikroservisa.

Sa druge strane GraphQL šema može uticati na visoku zavisnost *API Gateway*-a od modela sistema gde svaka izmena u modelu sistema može uticati na šemu i zato je poželjno uspostaviti standard po kome se tipovi u GraphQL šemi ne bi menjali u velikom obimu iako bi do promena u domenskom modelu dolazilo. Keširanje predstavlja jednu od najčešće spominjanih mana kada je u pitanju GraphQL, odnosno da se mogućnost HTTP keširanja koje je specifično za REST API-je gubi. Ipak, godinama se radi dosta na tome da se prednosti keširanja iskoriste na klijentskim aplikacijama kroz implementacije na toj strani projekata, te tako neke od njih nudi i Apollo projekat. Takođe, treba obratiti pažnju da li postoje cirkularne veze u grafu koje mogu dovesti do kreiranja skupih ugnježdenih upita i sprečiti ih ograničavanjem dubine upita, kao i ograničiti i maksimalnu količinu objekata koji se dovlače ukoliko je tu informaciju moguće definisati prilikom slanja upita.

GraphQL tehnologija, nastala kako bi rešila određene probleme koji se mogu javiti korišćenjem REST-a, vremenom pronalazi primenu u različitim oblastima i može se zapaziti da se pospešuje njeno korišćenje iz godine u godinu u različitim vrstama projekata.

5. LITERATURA

[1] S. Greif and R. Benitte, "GraphQL Experience Over Time," *State of JS 2020*. https://2020.stateofjs.com/en-US/technologies/datalayer/ (Poslednji pristup Apr. 19, 2022).

[2] G. Brito and M. T. Valente, "REST vs GraphQL: A Controlled Experiment," in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil, Mar. 2020, pp. 81–91. doi: 10.1109/ICSA47634.2020.00016.

[3] "Apollo GraphQL Usage Statistics." https://trends.builtwith.com/framework/Apollo-GraphQL (Poslednji pristup Apr. 19, 2022).

[4] The GraphQL Foundation, "Who's using GraphQL?" https://graphql.org/users (Poslednji pristup Apr. 19, 2022).

[5] A. Lawi, B. L. E. Panggabean, and T. Yoshida, "Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System," *Computers*, vol. 10, no. 11, p. 138, Oct. 2021, doi: 10.3390/computers10110138.

[6] P. Stünkel, O. von Bargen, A. Rutle, and Y. Lamo, "GraphQL Federation: A Model-Based Approach.," *JOT*, vol. 19, no. 2, p. 18:1, 2020, doi: 10.5381/jot.2020.19.2.a18.

[7] P. D. Francesco, I. Malavolta, and P. Lago, "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden, Apr. 2017, pp. 21–30. doi: 10.1109/ICSA.2017.24.

[8] "GraphQL Specification 2021," *GraphQL*, Oct. 27, 2021. https://spec.graphql.org/October2021/ (Poslednji pristup Apr. 18, 2022).
[9] L. Byron, "GraphQL: A data query language," Sep. 14, 2015. https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/ (Poslednji pristup Apr. 18, 2022).
[10] L. Byron, "GraphQL Specification 2021 Release Notes," Oct. 27, 2021. https://github.com/graphql/graphql-spec/releases/tag/October2021 (Poslednji pristup Apr. 18, 2022).
[11] E. Porcello and A. Banks, "Learning GraphQL: Declarative Data Fetching for Modern Web Apps," p. 299.
[12] S. M. Ireland and A. C. R. Martin, "GraphQL for the delivery of bioinformatics web APIs and application to ZincBind," *Bioinformatics Advances*, vol. 1, no. 1, p. vbab023, Jun. 2021, doi: 10.1093/bioadv/vbab023.
[13] "ZincBind." https://api.zincbind.net (Poslednji pristup Apr. 12, 2022).
[14] S. K. Mukhiya and Y. Lamo, "An HL7 FHIR and GraphQL approach for interoperability between heterogeneous Electronic Health Record systems," *Health Informatics J*, vol. 27, no. 3, p. 146045822110439, Jul. 2021, doi: 10.1177/14604582211043920.
[15] S. K. Mukhiya, F. Rabbi, V. K. I Pun, A. Rutle, and Y. Lamo, "A GraphQL approach to Healthcare Information Exchange with HL7 FHIR," *Procedia Computer Science*, vol. 160, pp. 338–345, 2019, doi: 10.1016/j.procs.2019.11.082.
[16] A. Cernian, B. Tiganoaia, I. Sacala, A. Pavel, and A. Iftemi, "PatientDataChain: A Blockchain-Based Approach to Integrate Personal Health Records," *Sensors*, vol. 20, no. 22, p. 6538, Nov. 2020, doi: 10.3390/s20226538.
[17] "Open Research Knowledge Graph." https://www.orkg.org/orkg/ (Poslednji pristup Apr. 12, 2022).
[18] M. Haris, M. Stocker, and S. Auer, "Enriching Scholarly Knowledge with Context," *arXiv:2203.14617 [cs]*, Mar. 2022, Poslednji pristup: Apr. 12, 2022. [Online]. Available: http://arxiv.org/abs/2203.14617
[19] D. A. Vasiliev, A.-M. Ghiran, and R. A. Buchmann, "Data Federation for a Project Management Solution through a GraphQL Middleware," p. 12.
[20] R. Khan and A. Noor Mian, "Sustainable IoT Sensing Applications Development through GraphQL-Based Abstraction Layer," *Electronics*, vol. 9, no. 4, p. 564, Mar. 2020, doi: 10.3390/electronics9040564.
[21] B. Gillam, "PostGraphile." https://www.graphile.org/postgraphile/ (Poslednji pristup Apr. 12, 2022).
[22] "Hasura." https://hasura.io/ (Poslednji pristup Apr. 12, 2022).
[23] M. Ismail, "Evaluation of Generic GraphQL Servers for Accessing Legacy Databases," p. 47.
[24] M. Fowler and J. Lewis, "Microservices," Mar. 25, 2014. https://martinfowler.com/articles/microservices.html (Poslednji pristup Apr. 23, 2022).
[25] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.
[26] C. Richardson, "What are microservices?," 2020. https://microservices.io/ (Poslednji pristup Apr. 23, 2022).
[27] C. Richardson, *Microservice Patterns*. Manning, 2018.
[28] C. Williams, "Is REST Best in a Microservices Architecture?," Dec. 18, 2015. https://capgemini.github.io/architecture/is-rest-best-microservices (Poslednji pristup Apr. 23, 2022).
[29] M. Vogel, S. Weber, and C. Zirpins, "Experiences on Migrating RESTful Web Services to GraphQL," in *Service-Oriented Computing – ICSOC 2017 Workshops*, vol. 10797, L. Braubach, J. M. Murillo, N. Kaviani, M. Lama, L. Burgueño, N. Moha, and M. Oriol, Eds. Cham: Springer International Publishing, 2018, pp. 283–295. doi: 10.1007/978-3-319-91764-1_23.
[30] Everett Griffiths, "GraphQL as an API Gateway to Microservices," Feb. 08, 2018. https://www.cloudbees.com/blog/graphql-as-an-api-gateway-to-micro-services/ (Poslednji pristup Apr. 23, 2022).
[31] V. Vernon, *Implementing Domain-driven Design*. Addison-Wesley, 2013.
[32] "GraphQL FAQ - Is GraphQL the right fit for designing a microservice architecture?" https://graphql.org/faq/#is-graphql-the-right-fit-for-designing-a-microservice-architecture (Poslednji pristup Apr. 18, 2022).
[33] V. Touronen, "Microservice architecture patterns with GraphQL," UNIVERSITY OF HELSINKI, 2019.
[34] R. Wawhal, "The Ultimate Guide to Schema Stitching in GraphQL," Aug. 22, 2018. https://hasura.io/blog/the-ultimate-guide-to-schema-stitching-in-graphql-f30178ac0072/#:~:text=Introduction-,Schema%20stitching%20is%20the%20process%20of%20creating%20a%20single%20GraphQL,customise%20an%20existing%20GraphQL%20API. (Poslednji pristup Apr. 23, 2022).
[35] Apollo Graph Inc., "Introduction to Apollo Federation," 2021. https://www.apollographql.com/docs/federation/#architecture (Poslednji pristup Apr. 23, 2022).
[36] Apollo Graph Inc, "Automatic persisted queries," 2021. https://www.apollographql.com/docs/apollo-server/performance/apq/ (Poslednji pristup Apr. 23, 2022).

Jelica Stanojević, asistent na Katedri za Informacione tehnologije, Univerzitet u Beogradu, Fakultet organizacionih nauka.
Kontakt: jelica.stanojevic@fon.bg.ac.rs
Oblasti interesovanja: internet tehnologije, mobilno računarstvo.

Miroslav Minović je redovni profesor pri katedri za Informacione tehnologije, Fakulteta organizacionih nauka. Rukovodilac je laboratorije za Multimedijalne komunikacije.
Oblasti interesovanja: HCI, Multimedija, Blokčejn, Biometrija

Dejan Simić, Univerzitet u Beogradu, Fakultet organizacionih nauka.
Kontakt: simic.dejan@fon.bg.ac.rs
Oblast interesovanja: zaštita informacija u računarskim sistemima, sigurnost podataka, sistemi elektronskog plaćanja, i primena informacionih tehnologija.