Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.738.1:766
DOI: https://doi.org/10.24867/18BE28Urban

IMPLEMENTACIJA GRAPHQL API-IJA ZA DOBAVLJANJE PODATAKA O NEW YORK TIMES ČLANCIMA

IMPLEMENTATION OF GRAPHQL API FOR FETCHING DATA OF NEW YORK TIMES DOCUMENTS

Aleksandra Urban, *Fakultet tehničkih nauka, Novi Sad*

Oblast – PRIMENJENE RAČUNARSKE NAUKE I INFORMATIKA

Kratak sadržaj –*U radu je predstavljen sistem za dobavljanje podataka o New York Times člancima preko GraphQL API-ija (eng. Application Programming Interface). Prvenstveno je implementiran REST API koji dobavlja podatke o New York Times člancima u sistem i koji su korisnicima dostupni preko GraphQL API-ija. U ovom radu može se videti razlika u implementaciji REST i GraphQL API-ija kao i njihove prednosti i mane. Diskutovane su koristi GraphQL API-ija i analizirane određene studije slučaja.*

Ključne reči: *New York Times članci, REST, GraphQL, studije slučaja.*

Abstract – *This paper describes and presents a system for acquiring data regarding New York Times articles using a GraphQL API. Firstly a REST API was implemented which gets New York Times article data from their official database and stores them in our system, which are then made available to users using a GraphQL API. In this paper multiple differences between REST and GraphQL are presented as well as their respectable pros and cons. Many GraphQL use cases were discussed and many case studies over the before mentioned subject were analyzed.*

Keywords: *New York Times documents, REST, GraphQL, case studies.*

1. UVOD

Kako bi sistem bio što optimizovaniji prilikom dobavljanja podataka jedna od relevantnih stvari je brzina dobavljanja istih kao i ekstrahovanje tačno zahtevanih podataka.

Kako bi ispunili ove zahteve i poboljšali performanse svoje zahtevne aplikacije, programeri Facebook-a su 2012. godine razvili GraphQL koji predstavlja jezik upita za rad sa API-jima. GraphQL nudi korisnicima da preciziraju koje podatke žele, uključujući upite podskupa resursa i dozvolu više upita u jednom zahtevu.

Postoji nekoliko načina pomoću kojih se resursi razmenjuju između klijenta i servera.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivanović, red. prof.

Jedan od najzastupljenijih je REST koji funkcioniše veoma dobro dok koncept resursa ostaje nepromenjen, što znači da ako se resurs promeni naš API više nije upotrebljiv. Ovo takođe može da predstavlja problem kada su klijentu potrebni podaci iz više izvora istovremeno koji se rešavaju tako što klijent šalje više različitih zahteva ka serveru ili server isporučuje dodatne podatke koji u većini slučajeva nisu potrebni i čine sam odgovor glomaznim i nepreglednim.

GraphQL omogućava rešenje ovih problema tako što nudi klijentu da odabere koje podatke želi, uključujući upite podskupa resursa i server će mu vratiti odgovor u jednostavnom, sigurnom i predvidivom obliku.

U ovom radu je predstavljen sistem koji u sebe uvlači podatke pomoću REST-a, a korisnicima ih dostavlja uz pomoć GraphQL API-ija pri čemu mogu da se vide njihove suštinske razlike. Kroz nekoliko studija slučaja opisano je zašto najzastupljenije aplikacije prelaze sa REST-a na GraphQL.

2. TEORIJSKE OSNOVE GRAPHQL-A

Jedan od važnijih aspekata GraphQL-a je njegova hijerarhijska arhitektura koja je promenljivog tipa zasnovanog na zahtevima klijenta. Ova hijerarhija podataka slaže se sa graf strukturiranim skladištima podataka. Oblik vraćenih podataka u potpunosti je određen upitom klijenta, tako da je ovim izbegnuto vraćanje bespotrebnih podataka pri čemu su poboljšane performanse. Promene karakteristika resursa na serveru ne zahtevaju promene na klijentskoj strani pa samim tim ne utiču na već postojeće, upotrebljive klijentske pozive. Korišćenje GraphQL API-ija podrazumeva gađanje tačno jednog endpointa sa promenljivim telom zahteva, što podrazumeva promenljiv broj polja koja klijent zahteva. Koje su sve moguće varijacije tela zahteva određuje šema GraphQL-a.

Jezik za definisanje **šeme** (*eng. Schema Definition Language, SDL*) [1] je način za definisanje i opis GraphQL šeme. Glavne komponente šeme su **tipovi** objekta koji sadrže ime i polja. Ta polja su raščlanjena na konkretne podatke tj. skalarne tipove, koji predstavljaju krajnje tačke ili listove upita. Svaki nivo hijerarhijske strukture GraphQL-a odgovara određenom tipu, a svaki tip opisuje skup dostupnih polja. Tip u GrapQL-u se može predstaviti kao pandan klasi u objektno orijentisanom modelu podataka. Pored upita postoje još dva operacijskog tipa a to su **mutacije** i **pretplate**. Klijenti svoj zahtev pišu u vidu upita kada žele da dobavljaju

podatke sa API-ija. Mutacije se koriste za iniciranje upisa ili izmena podataka, a pretplatama se otvara konekcija ka serveru preko koje će u realnom vremenu stizati odgovori vezani za tipove šeme na koje se klijent pretplatio, svaki put kada se neka izmena na serveru dogodi.

GraphQL podržava prosleđivanje argumenata. U svako polje možemo proslediti jedan ili više argumenata. Kao argument dozvoljeno je da se proslede različiti tipovi kao na primer skalari, enumi ili kompleksni input object tipovi. Kao argumente je moguće prosleđivati promenljive. Kada se promenljive prosleđuju kao argumenti u upitu koriste se sa prefiksom $, nakon čega sledi njihov tip. Kao i mnogi drugi tipovi sistema, GraphQL podržava i interfejse. Interfejs je aspektni tip koji uključuje određeni skup polja koja tip mora da uključi da bi implementirao interfejs. Korisni su kada želimo da vratimo objekat ili skup objekata, a oni mogu biti različitih tipova. GraphQL takođe podržava pseudonime. Pošto se polja objekta odgovora podudaraju sa imenima polja u upitu, ali ne uključuju argumente, ne može se direktno postavljati upit za isto polje sa različitim argumentima. Zato su potrebni pseudonimi koji omogućavaju da se preimenuje polje u odgovoru kako ne bi došlo do konflikta. Na listingu 2.1. možemo videti primer najjednostavnijeg upita sa klijentske strane gde želimo da obuhvatimo samo neke atribute dokumenta sa prosleđenim parametrom za tačno određeni id, u našem primeru id sa vrednošću 4.

```
{
    documents(id: 4) {
        id
        source
        headline{
            id
            main
        }
        document_type
        section_name
    }
}
```

Listing 2.1 *Primer GraphQL upita*

GraphQL upiti odražavaju njihov odgovor što se može videti na listingu 2.2. koji predstavlja odgovor na prethodni upit.

```
{
    "errors": [],
    "data": {
        "documents": [
            {
                "id": 4,
                "source": "The New York Times",
                "headline": {
                    "id": 9,
                    "main": "Corrections: Sept. 01, 2021"
                },
                "document_type": "article",
                "section_name": "Corrections"
            }
        ]
    },
    "extensions": null
}
```

Listing 2.2. *Primer GraphQL odgovora*

U upitu mogu da se pojave samo polja koja su definisana u određenom tipu naše šeme. Redosled polja nije bitan i neće uticati na odgovor od servera. Ukoliko se prilikom definisanja polja u šemi nađe uzvičnik (!) to označava da je vrednost polja uvek prisutna (nije null) ako se potražuje. Svako polje za svaki tip je podržano funkcijom *resolver*, koju obezbeđuje programer GraphQL servera. Kada se polje izvrši, odgovarajući *resolver* se poziva da proizvede vrednost. Ako polje proizvodi skalarnu vrednost poput niza ili broja, izvršenje se završava. Međutim ako polje daje vrednost objekta, upit će sadržati drugi izbor polja koja se primenjuju na taj objekat. Ovo se nastavlja sve dok se ne dostignu skalarne vrednosti. GraphQL upiti se uvek završavaju skalarnim vrednostima.

Na najvišem nivou svakog GraphQL servera je tip koji predstavlja sve moguće ulazne tačke u GraphQL API, po konvenciji se naziva *Root* ili *Query*. Iako GraphQL omogućava da se iz grafikona preuzmu tačno ona polja koja su nam potrebna, on ne garantuje automatski jednostavne odgovore. Liste koje dobija kao odgovor mogu da sadrže beskonačno mnogo elemenata, što može dovesti do enormnih odgovora na veoma jednostavne upite. Da bi rešili ovaj problem, GraphQL serveri mogu da paginiraju svoja polja koja predstavljaju liste. Postoji nekoliko načina na koje se može izvršiti paginacija korišćenjem ključnih reči kao što su *first, offset, after, limit* i td.

Još jedna odlika GraphQL-a je korišćenje direktiva. Direktiva se može dodeliti polju ili fragmentu i može uticati na izvršavanje upita na bilo koji način koji server želi. Osnovna GraphQL specifikacija uključuje tačno dve direktive koje mora podržati bilo koja implementacija GraphQL servera, a to su: @include(if: Boolean) – uključiće polje ako je argument tačan i @skip(if: Boolean) – preskočiće polje ako je argument tačan. Direktiva @auth govori GraphQL-u kako da primeni autorizaciju. Može se koristiti za definisanje pravila autorizacije za većinu tipova. Omogućava nam da kontrolišemo koji korisnici mogu da pokreću upite, kao i to koji korisnici mogu da dodaju, ažuriraju i brišu podatke pomoću mutacija.

2.1. REST i GraphQL

REST (*Representational State Transfer*) [2] arhitektura zasniva se na jedinstvenom i unapred definisanom skupu operacija koje korisnicima omogućavaju pristup i manipulaciju veb podacima. Za pristupanje nekom resursu koji se dobija pomoću REST API-ija, klijentu je potrebna putanja do tog resursa kao i da specificira metodu šta želi da uradi sa tim resursom. Najčešće korišćene metode su GET, POST, PUT i DELETE. U REST arhitekturi, API-iji izlažu svoju funkcionalnost kao resurse, koji su bilo koja vrsta usluge, podataka ili objekta kojima klijent može pristupiti. Značajna slabost sistema zasnovanih na REST-u je ta da klijent mora da se prilagodi jedinstvenom interfejsu koji obezbeđuje API. Uopšteno govoreći, podaci se prenose u standardizovanom obliku koji verovatno uključuje identifikator bilo kog pod-resursa ili celog traženog resursa iako je samo mali deo tog resursa potreban klijentskoj aplikaciji.

Kao i REST, GraphQL na neki način predstavlja arhitekturu dizajna API-ija, ali sa drugačijim pristupom koji je mnogo brži i fleksibilniji. Glavna i najvažnija razlika je u tome što se GraphQL ne bavi namenskim resursima [3]. Umesto toga, upit se može prilagoditi klijentskim potrebama, opisujući tačno šta želi da dobije kao odgovor. Moguće je kombinovati različite entitete u jednom upitu i moguće je odrediti koje atribute treba uključiti u odgovor. Ovakav pristup odgovara razvijanju

front-end aplikacija, jer se najveći deo posla obavlja na serverskoj strani. Najčešći problem RESTful aplikacija je problem overfetching-a i underfetching-a jer klijent ne može da izabere koje tačno podatke želi, nego mora da dobavi onakve resurse koji su omogućeni od strane servera.

3. STUDIJE SLUČAJA

U poslednjih nekoliko godina GraphQL je mnogo zastupljeniji za razliku od REST-a i postao je popularan među mnogim velikim organizacijama, koje ga koriste za pravljenje svojih veb aplikacija, kao što su *Facebook, Github, Artsy, Walmart Lab, Coursera, Major League Soccer* i mnoge druge. U nastavku će biti opisano, za neke od njih, na koji način su zamenili REST GraphQL-om [4].

3.1. Facebook

Kako su njihove mobilne aplikacije rasle i postajale sve kompleksnije, patili su od sve većeg degradiranja performansi, kao i čestih padanja aplikacija. Upravo iz tog razloga su odlučili da 2012.godine razviju jedan od najmoćnijih API-ja za dobavljanje podataka. Početna strana ove aplikacije ranije bila je isporučivana samo kao HTML, ali kako su prešli na izvorno implementirane modele i poglede, tako im je bila potrebna API verzija podataka početne stranice. Koristili su RESTful serverske resurse kao i FQL tabele (Facebook-ov API sličan SQL-u) za dobavljanje podataka početne stranice njihovih mobilnih aplikacija. Bili su frustrirani zbog razlika između podataka koje su želeli da koriste u aplikacijama i upita servera koji su im potrebni. Postojala je i značajna količina koda neophodna sa obe strane, kako serverske za pripremu koda tako i klijentske za parsiranje istog. Ova frustracija inspirisala je njihove programere da kreiraju GraphQL, koji bi im omogućio da preuzimaju podatke iz perspektive klijentske aplikacije. Sam GraphQL upit su kreirali tako da ima izgled JSON strukture, a serverska strana vraća identičan oblik, kako bi klijentskoj strani olakšali razumevanje. Ovo rešava problem nedovoljnog ili preteranog dobavljanja podataka, jer klijent dobija tačno ono što zahteva. Takođe klijent više nema problem da prikuplja podatke sa različitih krajnjih tačaka. Neophodno je samo da specificira oblik podataka koji su mu potrebni, pri čemu ne mora da pravi nikakve pretpostavke šta će dobiti nazad jer jasno zna oblik odgovora na osnovu svog upita [5].

3.2. GitHub

Najveća promena GitHub-a u API-iju je bila kada su odbacli XML u korist JSON-a, tačnije kada su GitHub API učinili dostupnim preko GraphQL-a. Kao i većina tehnologija, REST nije savršen i ima neke nedostatke. Njihova ambicija da promene API bila je fokusirana na rešavanje dva problema: problem skalabilnosti i problem fleksibilnosti. REST API je odgovoran za preko 60% zahteva upućenih njihovoj bazi podataka. Ovo je iz razloga što po svojoj prirodi, hipermedijska navigacija zahteva od klijenta da stalno komunicira sa serverom kako bi mogao da dobije sve informacije koje su mu potrebne. Njihovi odgovori su bili ogromni sa mnogim URL nagoveštajima koji bi pomagali ljudima da prolaze kroz API kako bi dobili željene podatke. Ponekad su bila potrebna dva ili više odvojena poziva da bi se sakupio kompletan prikaz resursa što je predstavljalo dooljan razlog da pređu na GraphQL koji im omogućava da uz pomoć samo jednog endpointa dobiju sve željene podatke [6].

3.3. Artsy

Artsy tim je počeo da koristi GraphQL koji povezuje različite API-je sa više front-end aplikacija uključujući iOS aplikacije. Ovo im je pomoglo ne samo da budu dosledniji sa načinom na koji preuzimaju podatke iz aplikacija, već je poboljšalo i zadovoljstvo njihovih programera. Integracija GraphQL-a u njihov API je dovela do povećanja brzine učitavanja stranica. Njihov fokus je tokom prošlih godina najviše bio na tome kako se sadržaj pojavljuje, ali su se trenutno fokusirali na brzinu jer je u današnje vreme bitno prikazati korisnicima sadržaj u najkraćem mogućem roku. Kako bi povećali brzinu stranice refaktorisali su kod kako bi imali manje preuzimanja podataka, ali ovo nije efikasno zato su u svoj kod uveli GraphQL.

Do uvođenja GraphQL-a vraćali su kolekcije članaka sa svim njenim sadržajem, pri čemu je prikazivanje jedne jednostavne stranice sa listom članaka vršila preveliko opterećenje za njihov sajt. Sada im je uz pomoć GraphQL-a omogućeno da dobave samo nekoliko stvari kao što su naslov i slika, bez potrebe da rade bilo kakav dodatni posao. Ovo je posebno važno za korisnike koji se prijavljuju preko mobilnih uređaja koji se susreću sa sporijim brzinama mreže [7].

3.4. Walmart Labs

Timovima koji se bave mobilnim aplikacijama Walmart Labs-a kao i timovi koji razvijaju njihove veb stranice imaju različite zahteve o tome koji podskup podataka žele da dobiju. Iz tog razloga, učiniti podatke lako dostupnim predstavljalo je veliki izazov za njihove programere.

Vremenom su se našli u situaciji da imaju enormnu količinu API-ja, od kojih je svaki napravljen za poseban slučaj korišćenja. Svaki od ovih API-ja je bio drugačije konfigurisan i imao je sopstvene konvencije pisanja URL-a, parametara upita itd. Razvoj bilo kog od postojećih API-ja predstavljalo je veliki problem. Dodavanje novog polja u JSON odgovor je bilo vrlo rizično, jer bi moglo imati nepredviđene posledice za klijente koji su se oslanjali na prethodni odgovor.

Njihov glavni cilj je bio da razviju i poboljšaju API-je bez komplikovanijih migracionih planova. Želeli su da budu u mogućnosti da sa sigurnošću uvode promene u kod, bez straha o potencijalnim nuspojavama. Međutim, postojao je problem jer GraphQL nije bio podržan za programski jezik koji su oni koristili. Pošto GraphQL ima robusnu specifikaciju, uspeli su da izgrade sopstvenu potpuno realizovanu implementaciju – Lacinia.

Nakon što su razvili internu verziju Lacinia-e, bili su u mogućnosti da pregaze skoro sve postojeće API-je sa uslugama koje pokreće GraphQL [8].

4. IMPLEMENTACIJA SISTEMA

U ovom poglavlju je dat opis implementacije aplikacije koja koristi REST arhitekturu za dobavljanje podataka o

New York Times člancima, koji se čuvaju u samoj aplikaciji, a klijentima su dostupni preko GraphQL API-ija. Aplikacija je implementirana u programskom jeziku *Java* (verzija 8), uz korišćenje radnog okvira *Spring Boot* radi razvijanja aplikacije visokih performansi.

4.1. Konfiguracija

Biblioteke koje su se koristile pri izradi aplikacije su ***Spring Boot GraphQL Starter*** koja nudi način za pokretanje GraphQL servera za vrlo kratko vreme i ***GraphQL Java Tools***. Prva pomenuta biblioteka zapravo pretvara Spring Boot aplikaciju u GraphQL server koristeći GraphQL Java Tools biblioteku [9].

GraphQL Java Tools biblioteka omogućava nam da svoju schemu napišemo na jednostavan, prenosiv način, koristeći GraphQL jezik šeme umesto teško čitljivog koda. Ona zapravo služi za mapiranje GraphQL šeme na postojeće Java objekte.

4.2. GraphQL endpoint

New York Times članci su dostupni korisnicima preko GraphQL endpointa. Za komunikaciju sa ovim endpointom bila je neophodna implementacija *Controller*-a koji prima zahtev i prosleđuje ga dalje servisu koji je zadužen za poslovnu logiku. Svaki *Controller* označava se anotacijom *@RestController*. Implementaciju servisa je neophodno anotirati specijalnom anotacijom *@Service* koja govori da ta klasa zapravo izvršava poslovnu logiku. U servisu je implementirana loadSchema() metoda koja ima anotaciju *@PostContruct* što znači da će se izvršiti kada se inicijalizuje servis. Ova metoda služi za učitavanje naše GraphQL šeme u aplikaciju. Ta šema mora da se parsira. Ovo je moguće izvršiti pomoću *SchemaParser*-a koji nam omogućava sam GraphQL koji smo uvukli u našu aplikaciju. Kada se šema parsira ona se učitava u *TypeDefinitionRegistry* koji nam je isto omogućio GraphQL. Tako da će se GraphQL odnositi na taj registar koji ima podatke šeme i pratiće koji se objekti u njoj nalaze, kao i koje objekte mi zahtevamo. *RuntimeWiring* nam omogućava da mapiramo polja iz šeme. Kada je mapiranje završeno i registar spreman, oni se nude GraphQL šemi koju kreiramo da bude izvršiva.

Kako bi se dobavili podaci za svaki upit, neophodno je da imamo dobavljače podataka, a to su *Data Fetcher*-i. *Data Fetcher* je odgovoran za vraćanje vrednosti podataka za dato polje GraphQL-a. GraphQL koristi alat za preuzimanje podataka da dobavi logičko polje u objekat, koji se kreira u vreme izvršavanja, koji će biti vraćen kao deo GraphQL odgovora. U drugim implementacijama oni se ponekad nazivaju "*Resolvers*" ili "*Field Resolvers*" jer oni pretvaraju logičko polje GraphQL-a u stvarnu vrednost podataka. Svako polje u GraphQL-u je povezano sa *graphql.schema.DataFetcher*-om. Neka polja će koristiti specijalizovani kod za preuzimanje podataka koji zna kako da pristupi bazi podataka kako bi dobio informacije o polju, dok većina jednostavno uzima podatke iz vraćenih memorijskih objekata koristeći naziv polja. Svakom *Data Fetcher*-u se prosleđuje objekat *graphql.schema.DataFetchingEnvironment* koji sadrži koje polje se preuzima, koji su argumenti dostavljeni u polje i druge informacije kao što su tip polja i tip nadklase. Definisanje dostupnih tipova za upit postiže se kreiranjem šeme. Tip koji se u ovoj aplikaciji koristi za sve moguće ulazne tačke je *Query* tip. U njemu je naveden tip upita koji može da se koristi a to je *document* uz koji se prosleđuju neobavezni parametri: *id* dokumenta, *offset*, *limit* i *filter*. Da bi se mogao koristiti ovaj upit neophodno je deklarisati tip *Document* kao i svaki drugi objektni tip koji se koristi u šemi.

5. ZAKLJUČAK

U ovom radu opisana je uloga GraphQL-a u dobavljaju članaka iz New York Times-ove baze podataka. Opisane su prednosti i mane ove tehnologije u poređenju sa REST arhitekturom kroz implementaciju samog rada.

U današnje vreme brzina i preciznost vraćenih podataka predstavljaju najznačajnije karakteristike GraphQL-a. Ova tehnologija smanjuje količinu vraćenih podataka zavisno od potreba klijenata. Jedna od najvažnijih karakteristika GraphQL-a koja je opisana u ovom radu je šema koju ova tehnologija koristi radi preciznijeg i lakšeg dobavljanja podataka.

6. LITERATURA

[1] Schema Definition Language https://graphql.org/learn/schema/
[2] Representational State Transfer https://www.ibm.com/cloud/learn/rest-apis
[3] GraphQL vs REST https://blog.api.rakuten.net/graphql-vs-rest/
[4] GraphQL Case studies https://www.graphql.com/case-studies/
[5] Facebook case study https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/
[6] GitHub case study https://github.blog/2016-09-14-the-github-graphql-api/
[7] Artsy case study https://artsy.github.io/blog/2016/11/02/improving-page-speed-with-graphql/
[8] Walmart case study https://medium.com/walmartglobaltech/open-sourcing-lacinia-our-graphql-library-for-clojure-96a4ce5fc7b8
[9] Spring Boot GraphQL Starter https://docs.spring.io/spring-graphql/docs/1.0.0-M5/reference/html/

Kratka biografija:

Aleksandra Urban rođena je 27.12.1997. godine u Sremskoj Mitrovici. Godine 2016. upisala je Fakultet Tehničkih Nauka u Novom Sadu, odsek Računarstvo i automatika. 2018. upisuje usmerenje Primenjene računarske nauke i informatika, a 2019. godine se opredeljuje za modul Internet i elektronsko poslovanje. U septembru 2020. godine je diplomirala i upisala Master akademske studije.

Kontakt: urb.saska@gmail.com