Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/13BE01Mandic

UPOTREBA GRAPHQL TEHNOLOGIJE ZA RAZVOJ VEB APLIKACIJA

USE OF GRAPHQL TECHNOLOGY FOR DEVELOPING WEB APPLICATIONS

Mihailo Mandić, Milan Vidaković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad opisuje implementaciju GraphQL tehnologije, u obliku veb aplikacije za prodaju polovnih automobila. U radu su date osnove GraphQL-a, specifikacija rešenja koja je predstavljena UML dijagramima kao i najvažniji implementacioni delovi sa klijentske i serverske strane. Klijentski deo aplikacije je rađen uz pomoć Angular okruženja i Apollo klijenta. Serverski deo je pisan u programskom jeziku Java i korišćen je Spring framework.*

Ključne reči: *GraphQL, Java, Spring, Veb aplikacija.*

Abstract – *The main goal of this paper is to present the implementation of GraphQL in a web application for selling used cars. The paper covers the basics of GraphQL. The specification of the application is presented with UML diagrams. Paper also contains the most important parts of implementation logic that covers both client and server-side. Client-side is written using the Angular framework and Apollo client. Server-side is written in Java programming language with the Spring framework.*

Keywords: *GraphQL, Java, Spring, Web application.*

1. UVOD

Najvažniji deo veb aplikacija jeste komunikacija između klijenta i servera. Ukoliko je komunikacija spora i/ili neefikasna bilo kakav napor u optimizaciji aplikacionog koda neće rešiti taj problem. Postoji više načina kako klijent može da razmenjuje informacije sa serverom. Jedan takav način je pomoću SOAP-a *(Simple Object Access Protocol).* Drugi način je pomoću REST-a *(Representational State Transfer).* Resursi su podaci i funkcionalnosti sistema koji se razmenjuju između klijenta i servera putem odabranog stateless protokola. GraphQL je razvio Facebook 2012. godine. Prošlo je 3 godine otkako je postao dostupan za sve, pre toga je bio samo interna tehnologija Facebooka [1]. Kako su mobilne aplikacije Facebooka postajale sve kompleksnije, bilo je potrebno pronaći novi način kako izbeći loše performanse i popraviti stabilnost aplikacija. Količina koda koja treba da se napiše za pripremu podataka, kako na serverskoj strani, tako i na klijentskoj strani, nije bila uopšte mala. Zbog toga su inženjeri Facebooka odlučili da sva postojeća rešenja nisu dovoljno dobra i zbog toga je nastao GraphQL.

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

2. TEORIJSKE OSNOVE GRAPHQL-A

Da bismo koristili GraphQL servis, potrebno je podesiti odabranog klijenta i server. Razmena podataka se obavlja tako što klijent šalje zahtev ka serveru i on odgovara sa željenim podacima. Recimo da imamo klasu *User* koja u sebi ima atribute - *username, age, location* i atribut objekat *address*. Klasa *Address* sadrži *street, city* i *zip* atribut. Klase *User* i *Address* se nalaze na našem serveru i definisane su u GraphQL šemi. U listingu 1. možemo videti primer upita sa klijentske strane gde želimo da obuhvatimo samo neke atribute.

```
query getUser {
  user {
    username
    age
    address {
      street
    }
  }
}
```

Listing 1. *Primer GraphQL upita.*

Kao odgovor sa serverske strane dobijamo JSON (Javascript Object Notation) prikazan u listingu 2.

```
{
   "data": {
     "user": {
       "username": "Ricky"
       "age": 19,
       "address": {
           "street": "Liberty Boulevard"
       }
     }
   }
}
```

Listing 2. *Odgovor od strane servera.*

Implementacije GraphQLa variraju u zavisnosti od odabranih klijenata i servera. Osnovni koncepti su šema, upiti, mutacije i ulazna tačka. Ulazna tačka *(endpoint)*, je tačka gde su upućeni svi zahtevi prema GraphQL serverskoj strani. GraphQL upit je zahtev poslat od strane klijenta ka serveru radi dobavljanja podataka. Može se posmatrati kao GET zahtev u REST-u. Postoji više načina kako možemo da napišemo upit. Upit se sastoji od operacijskog tipa, operacijskog imena i tela upita ili samo tela upita. Primer koji sadrži sve elemente upita je u listingu 1. *Query* je operacijski tip, *getUser* je operacijsko ime a *user* i ostatak je telo upita. Postoji još dva

operacijska tipa - *mutation* i *subscription*. Isti upit se može napisati i bez operacijskog imena i tipa. Takav upit se naziva kratki upit. Postoje ograničenja kako se šta piše. Operacijski tip je obavezan osim ako se piše kratki upit. U slučaju kratkog upita, operacijsko ime kao i definicije varijabli ne mogu da se navedu. Praksa pokazuje da je izuzetno poželjno koristiti kompletne upite (tip i ime operacije) zbog debagovanja i beleženja serverskih logova.
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. U većini aplikacija koje koriste GraphQL, prosleđivanje argumenata u samom upitu nije najbolje rešenje. Argumenti koji se prosleđuju u aplikacijama su (uglavnom) dinamični a ne statični. Ukoliko je potrebno da se proslede dinamički argumenti moraju se koristiti varijable GraphQL-a. Na taj način u samom upitu se nalaze varijable koje dobijaju vrednost tako što se prosledi zasebni rečnik (*dictionary* - par ili skup parova ime:vrednost). Prednost korišćenja varijabli umesto statičkog prosleđivanja argumenata jeste da ne moramo svaki put pisati novi upit ukoliko je potrebno da prosledimo drugu varijablu. Definicija varijable je data u formatu *($variableName: type)*. Varijabla se označava sa prefiksom *$* i iza nje sledi njen tip. Definicije varijabli podsećaju na definicije argumenata u funkcijama raznih programskih jezika. Sve deklarisane varijable moraju biti ili skalarnog ili enum ili input object tipa. Ukoliko polje upita prima kompleksan objekat potrebno je znati koji je odgovarajući input tip definisan u šemi koja se nalazi na serveru. Definicije varijabli mogu biti opcione ili obavezne u zavisnosti od toga kako je navedeno u šemi. Obavezne definicije varijabli razlikujemo od opcionih uz pomoć uzvičnika (!) koji stoji iza tipa varijable - *($variableName: type!)*.
Kao što je ranije napomenuto, upiti služe da klijent dobavi potrebne podatke sa servera po nekom kriterijumu. Ako je potrebno da se izvrši neka manipulacija podacima (bilo da je to unos novih podataka, brisanje ili promena postojećih) koji su smešteni u bazi podataka koja je povezana sa našim serverom, GraphQL nudi mutacije *(mutation)* kao rešenje. Sledeći listing pokazuje jedan primer mutacije sa propratnim rečnikom.

```
mutation loginUser($username: String!, $password: String!) {
  userLogin (username: $username, password: $password) {
    id
    age
  }
}
{
  "username": "Ricky",
  "password": "25454sda78931"
}
```

Listing 3. *Primer mutacije sa propratnim rečnikom varijabli.*

Mutacija iz prethodno navedenog listinga prima parametre *username* i *password* tipa *String* koji su obavezni jer je prisutan uzvičnik. Mutacije kao i upiti mogu da vrate polja. Ta polja moraju da se poklapaju sa definicijom tipa objekta na koji se mutacija odnosi u šemi. Kao povratni odgovor servera dobija se *id* i *age* polje koji se mogu iskoristiti na klijentskoj strani. Mutacije ne moraju nužno da primaju bilo kakve argumente. Takođe, mutacije mogu da primaju kompleksne objekte a ne samo skalare. Ako se prosleđuju kompleksni objekti, potrebno je koristiti input object tip. To je specijalna vrsta object tipa.
Svaki GraphQL servis u sebi sadrži šemu. Da bi se zadovoljio neki upit ili mutacija potrebno je taj zahtev validirati i izvršiti u odnosu na šemu. Šema služi da se u potpunosti opiše API servera što omogućava klijentima da znaju koje operacije se mogu izvršiti na serveru. To znači da su u šemi definisani svi tipovi objekata i njihova polja koji reprezentuju podatke. Šema je definisana pomoću GraphQL šematskog jezika *(GraphQL schema language)*. Taj jezik se naziva i SDL *(Schema Definition Language)* [2].
Tip koji se najčešće sreće je object tip. Object tip reprezentuje objekat koji se dobavlja iz servisa i u sebi sadrži sva odgovarajuća polja. Primer object tipa je dat u narednom listingu.

```
type User {
  username: String!
  password: String
  age: Int
  location: String
  buddies: [User]
}
```

Listing 4. *Definicija object tipa u šemi.*

GraphQL polja odgovaraju atributima klase u izabranom programskom jeziku. Samo polja koja su definisana u nekom tipu mogu da se pojave u nekom upitu ili mutaciji upućenom ka serveru. Uzvičnik (!) pored nekog polja označava da je vrednost tog polja uvek pristuna (nije null) ako se potražuje.
Šema u sebi pored definicija svih tipova koje podržava sadrži i *root operation* tipove koje podržava u odeljku *schema*. Ako su korenske operacije *(root operation) query*, *mutation* i *subscription* definisane pomoću tipa koji se zove Query, Mutation i Subscription respektivno nije potrebno definisati *schema* sekciju. Svaka šema mora da podrži *query* root operation tip i on mora biti object tip [3]. Pored korenskih operacionih tipova i object tipova postoje još i skalarni tipovi. Skalarni tipovi u GraphQL-u služe za konkretnu definiciju polja pored ostalih tipova. Na primer u programskom jeziku Java takvi tipovi se nazivaju primitivni tipovi. Postoje predefinisani skalari kao i mogućnost da mi sami definišemo naše skalare. Predefinisani skalari su *Int, Float, Boolean, String* i *ID*. Posebni tip skalara su i enumeracije. Enumeracije su ograničene na određeni set vrednosti.
GraphQL u sebi podržava interfejse. Interfejs je apstraktni tip koji u sebi sadrži određen broj polja. Kada se interfejs nasledi, sva njegova polja moraju biti sadržana u tipu koja ga nasleđuje. Pored interfejsa, GraphQL definiše i unije. Razlika između unije i interfejsa jeste što unije ne specificiraju zajednička polja.
U prethodnim pasusima smo pomenuli input object tip. On služi za plasiranje kompleksnih objekata u upite i mutacije. Input object definiše set ulaznih polja. Umesto ključne reči *type*, koja se koristi pri definiciji object tipa,

za input object tip je potrebno napisati *input*. Važno je napomenuti da input object tip i object tip imaju različite namene iako možda na prvi pogled izgledaju vrlo slično po definiciji u šemi. Neretko se dešava da object tip, iako deli neka ili možda čak i većinu polja sa input object tipom, sadrži druge object tipove i tako se stvara jedna vrlo kompleksna struktura podataka. Object tip može u sebi da sadrži reference prema interfejsima i unijama a njegova polja mogu da definišu argumente u sebi. Interfejsi i unije su apstraktni tipovi za koje nema smisla da se šalju sa klijentske strane.

2.1. REST i GraphQL

REST *(Representational State Transfer)* je stil arhitekture sistema koji smo koristi u izradi našeg projekta. Za pristupanje nekom resursu koji se krije iza REST API-ja, potrebno je da klijent zna putanju do tog resursa kao i da specificira metodu šta želi da uradi s tim resursom. Metode koje klijent može da iskoristi su GET, POST, PUT, DELETE i ostale. U REST API arhitekturi, server definiše koji će podaci biti vraćeni dok kod GraphQL servera, server samo definiše dostupne podatke a klijent specificira koje podatke želi da budu vraćeni. Kada je u pitanju rukovanje greškama, REST se oslanja na podrazumevane status kodove HTTP-a (404, 401, 500, itd.). Kod GraphQL-a, statusni kod je uvek 200 OK bez obzira na odgovor [4]. U slučaju greške klijentu se šalje odgovor sa tekstom greške. U implementaciji GraphQL-a, postoji mogućnost definisanja greški i poruka koje server može da vrati.

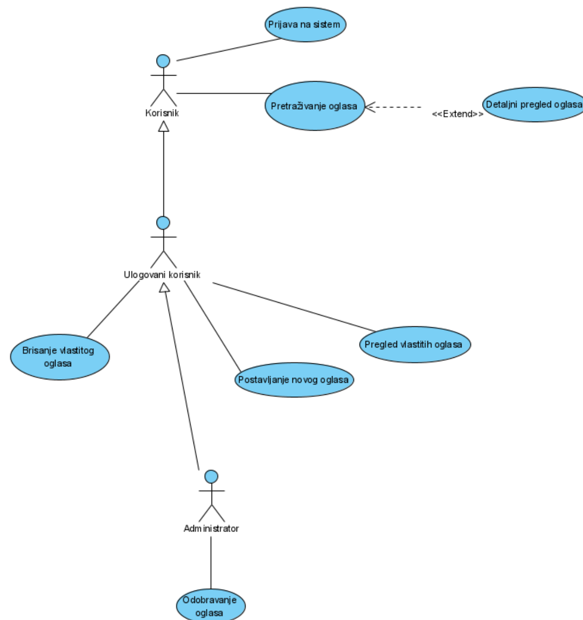
Pored određenih prednosti naspram REST-a, GraphQL donosi i određen broj mana. Na primer, . Ako je potrebno koristiti keširanje podataka u GraphQL-u mora se koristiti eksterni mehanizam na klijentskoj strani dok REST koristi HTTP koji su sebi ima ugrađeno keširanje. Još jedna mana GraphQL-a je format odgovora. Kao odgovor samo podžava JSON format dok REST podržava JSON, XML, HTML i druge [5].

2.2. SOAP i GraphQL

U poređenju sa SOAP-om, GraphQL deli neke elemente poput jedne ulazne tačke, endpoint-a, za pristup podacima. U obe tehnologije potrebno je deklarisati tipove podataka. Takođe, obe tehnologije dele i istu manu a to je nedostatak ugrađenih mehanizama za keširanje. Iako SOAP može da koristi HTTP kao i REST, ograničen je samo na upotrebu POST metoda koja ne kešira. Prednost GraphQL-a u odnosu na SOAP su bolje performanse dok SOAP ima prednost za održavanje starijih sistema koji su bili aktuelni po pojavljivanju SOAP-a.

3. SPECIFIKACIJA ZADATKA

Kao pomoć u opisu specifikacije zadatka sačinjeni su UML dijagram korišćenja i dijagram sekvence. UML *(Unified Modeling Language)* je jezik za modelovanje u softverskom inženjerstvu. Cilj UML-a je da pruži standardan način predstavljanja dizajna nekog softverskog sistema. Dijagram korišćena potpada pod dijagrame ponašanja i opisuje neke od veza između aktera, slučajeva korišćenja i sistema. U navedenom dijagramu korišćenja, koji je dat na narednoj slici, postoje tri tipa korisnika. Neprijavljen korisnik (ili samo korisnik) ima opcije da se prijavi na sistem i da pretražuje oglase. Kada nađe odgovarajuću listu oglasa, omogućeno je da pojedinačan oglas pogleda detaljnije. Kada korisnik izvrši prijavu na sistem, postaje prijavljen korisnik. Neprijavljen korisnik može da vidi svoju listu postavljen oglasa koji su odobreni od strane administratora. Takođe postoji mogućnost da prijavljen korisnik obriše svoj već postavljeni oglas. Da bi se postavio novi oglas, potrebno je da korisnik bude prijavljen. Administrator sistema ima dodatnu funkciju da odobrava oglase pre nego što postanu vidljivi.



Slika 1. *Dijagram slučajeva korišćenja.*

Nakon prijave na sistem, korisnik vidi pregled sopstvenih oglasa i opciju da postavi novi oglas. Nakon popunjavanja svih polja oglasa, rezultat te operacije se šalje aplikaciji radi obrade. Aplikacija kontaktira bazu za upis oglasa i oglas se upisuje kao neodobren. Oglasi koji su označeni kao neodobreni dospevaju na administratorski panel. Administrator ima mogućnost da odbije ili odobri neki oglas. U ovom slučaju administrator odobrava oglas. Posle odobravanja oglasa, aplikacija salje poruku bazi podataka da se status oglasa promeni u odobren. Kao takav, odobren oglas se pojavljuje u listi vlastitih oglasa korisnika kao i u svim pretragama bilo kog korisnika aplikacije.

4. OPIS IMPLEMENTACIJE

Kao što je ranije navedeno, aplikacija se sastoji od klijentskog i serverskog dela.

4.1. Serverski deo

Paketi koji se obično koriste pri izradi serverskog dela su *graphql-spring-boot-starter* i *graphql-java-tools*. Prvi je zadužen da pretvori backend deo aplikacije u GraphQL server a drugi služi za izgradnju GraphQL šeme kao i da olakša rad i implementaciju GraphQL-a. GraphQL server se nalazi na putanji *http://localhost:8080/graphql*. Umesto *graphql*, putanja može da se promeni u proizvoljnu, u *application.properties* fajlu.

Šema projekta podržava upite i mutacije kao što je definisano u *schema* odeljku. Sastoji se iz object tipova i input object tipova.

Object tipovi su *Query, Mutation, Ads, Image, User, CarModel, Extras, Safety, Characteristics* i *Condition*.
Input object tipovi su *AdsInput, CarModelInput, UserInput, ExtrasInput, ConditionInput, CharacteristicsInput* i *SafetyInput*. Input object tipovi sadrže reč *Input* u svom nazivu. Svi nazivi polja svih object tipova izuzev *Query* i *Mutation* odgovaraju nazivima atributa klasa koja su definisana u projektu. To omogućava automatsko mapiranja GraphQL object tipova sa klasama koje su definisane u Javi. Isto važi i za input object tipove. Uz pomoć GraphQL Java alata [6] koji smo uključili u projekat, omogućeno je mapiranje GraphQL objekata na metode i atribute Java objekata. Za većinu polja koja sadrže skalarne tipove dovoljno je napraviti POJO *(Plain Old Java Object)* sa istim poljima, getter-ima i setter-ima. Kompleksnija polja poput ugrađenih objekata zahtevaju u nekim slučajevima pisanje posebnih Resolver-a. Resolveri su funkcije koje daju vrednost za polje u šemi ili obavljaju neke kompleksne kalkulacije ili nešto treće.
U nekim slučajevima potrebno je implementirati *GraphQLResolver* interfejs. Pored Resolver-a za pojedinačna polja, postoje Resolveri za korenske operacije.
U šemi projekta prisutne su mutacije i upiti kao operacije pa samim tim potrebno je implementirati dva korenska resolvera - *GraphQLQueryResolver* i *GraphQLMutationResolver*.
Uslov za implementiranje *GraphQLQueryResolver* interfejsa je da svako polje iz šeme *Query* objekta ima svoju metodu sa istim imenom [7]. Takođe, svako polje *Mutation* objekta u šemi treba da ima svoju metodu sa istim imenom. Jedna od prednosti GraphQL-a jeste definisanje sopstvenih grešaka i poruka. Ukoliko je to potrebno uraditi, mora se implementirati *GraphQLError* interfejs. Kako GraphQL ima nedostataka demonstrirano je upotrebom REST-a u zadatku. Rad sa fajlovima nije jača strana GraphQL-a i ako je moguće taj deo treba izbeći koristeći neku drugu tehnologiju. S obzirom da je ovo aplikacija za prodaju polovnih automobila bilo je potrebno omogućiti postavljanje slike u oglasu. Kada se slika pošalje sa klijentske strane, prvo dolazi do REST kontrolera. Putanja za upload slike je *http://localhost:8080/image/upload* i potrebno je koristiti HTTP POST metodu.

4.2. Klijentski deo

Za izradu klijentskog dela korišćen je Angular i Apollo klijent. Za korišćenje Apollo klijenta potrebno ga je instalirati i podesiti putanju do GraphQL servera. Za pisanje bilo kakvog GraphQL upita ili mutacije potrebno je da tekst bude obmotan *gql* funkcijom koja se nalazi u paketu *apollo-angular*. U konkretnom slučaju, npr. pretrage oglasa, nakon pritiskanja dugmeta za submit svih podataka za pretragu, podaci se beleže i spremaju se za slanje ka GraphQL serveru. Važno je napomenuti da sva imena varijabli na frontendu moraju odgovarati svim poljima definisanim u šemi i klasama na serveru radi tačnog mapiranja. Poziva se apollo *query* metoda koja izvršava zahtev ka serveru i čeka se njen odgovor radi popunjavanja tabele sa rezultatima. Zapravo šta se ovde dešava jeste da Apollo šalje HTTP POST zahtev serveru sa podacima u JSON formatu. Nakon obrade podataka server salje podatke takođe u JSON formatu. Kada je u pitanju slanje slike na server, koristi se http klijent. Po odabiru željene slike se šalje HTTP POST zahtev prema REST kontroleru koji očekuje informacije o slici na *http://localhost:8080/image/upload* putanji. Nakon uspešnog postavljanja slike na server, kao odgovor se uzima jedinstveni identifikator slike da bi bio uključen sa ostatkom podataka koji će biti poslati putem Apollo klijenta na server.

5. ZAKLJUČAK

U ovom radu je dato jedno rešenje kako implementirati GraphQL u programskom jeziku Java. Serverski deo je potpomognut Spring okruženjem, dok klijentski je sačinjen od Angulara, Apollo klijenta i programskog jezika Typescript.
Aplikacija omogućava pregled svih oglasa na sistemu kao i postavke vlastitih oglasa za prodaju polovnih automobila. Takođe sadrži sistem za prijavu/registraciju korisnika u sistemu i administratorski deo za sprečavanje zloupotrebe postavljanjem neprimerenih oglasa.
Prikazano rešenje predstavlja samo studiju slučaja, te nije pogodno za komercijalnu upotrebu u trenutnoj formi. Za komercijalnu upotrebu neophodno bi bilo doraditi makar sledeće funkcionalnosti: zaštita korisnika i cele aplikacije od neovlaštenih upada u sistem, kao i podrška komunikaciji između prodavca i kupca.

6. LITERATURA

[1] Why GraphQL? https://engineering.fb.com/core-data/graphql-a-data-query-language/, Oktobar 2020.
[2] Schema definition language https://graphql.org/learn/schema/#type-language, Oktobar 2020.
[3] Root operation types https://spec.graphql.org/June2018/#sec-Root-Operation-Types, Oktobar 2020.
[4] GraphQL error handling, https://medium.com/better-practices/rest-soap-graphql-gesundheit-6544053f65cf, Oktobar 2020.
[5] REST formats, http://help.arcgis.com/en/businessanalyst/apis/rest/reference/outputFormats.html, Oktobar 2020.
[6] GraphQL Java Kickstart, https://github.com/graphql-java-kickstart/graphql-java-tools, https://www.graphql-java-kickstart.com/tools/, Oktobar 2020.
[7] Naming conventions, https://www.baeldung.com/spring-graphql#3-root-query-resolver, Oktobar 2020.

Kratka biografija

Mihailo Mandić je rođen 1995. godine u Beogradu. Završio je gimnaziju „Isidora Sekulić'' u Novom Sadu, društveno-jezički smer. Fakultet tehničkih nauka, smer Računarstvo i automatika, upisao je 2014. godine. Nakon završenih osnovnih studija, upisao je master studije na istom fakultetu.

Milan Vidaković je rođen u Novom Sadu 1971. godine. Na Fakultetu tehničkih nauka u Novom Sadu završio je doktorske studije 2003. godine. Na istom fakultetu je 2014. godine izabran za redovnog profesora iz oblasti Primenjene računarske nauke i informatika.