Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/13BE07Milivojevic

TESTIRANJE GRAPHQL APLIKACIJA

TESTING GRAPHQL APPLICATIONS

Aleksandra Milivojević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su objašnjeni osnovni koncepti GraphQL specifikacije. Opisani su tipovi i strategije testiranja softvera prema ovom konceptu. Osmišljena je aplikacija za demonstraciju i implementaciju testova specifičnih za sisteme koji koriste GraphQL.*

Ključne reči: *GraphQL, testiranje softvera, integracioni testovi, unit testovi*

Abstract – *This paper explains the basic concepts of GraphQL specification. Types and strategies of software testing according to this concept are given.* The application that demonstrates it is implemented. The tests specific for systems that use GraphQL are written and explained.
Keywords: *GraphQL, software testing, integration testing, unit testing*

1. UVOD

GraphQL [1] je upitni jezik za API-eve čija je speifikacija otvorenog koda. Ovaj novi koncept osmišljen je 2012. godine od strane programera Facebook aplikacije, kao alternativa tradicionalnom RestFULL pristupu radi povećanja efektivnosti i optimalnijeg korištenja resursa koji se dobavljaju sa servera. Ideja ovakvog GraphQL pristupa je da se novim upitnim jezikom specificira koji su atributi neophodni u odgovoru servera.

Testiranje softvera je proces koji se sastoji od svih aktivnosti u životnom ciklusu softvera. Oni bi trebalo da pokažu da li softver zadovoljava specificirane zahteve i ispunjava namenu, kao i da detektuju eventualne greške u sistemu. Kako je GraphQL pristup drastično promenio koncepte slanja zahteva sa klijentske strane i odgovora sa serverske strane, javile su se potrebe za testiranje ovakvih aplikacija na malo drugačiji način.

Za potrebe ovog rada, implementirana je web aplikacija za demonstraciju objašnjenih GraphQL koncepata, a naročito testova specifičnih za njih. Na kraju su prikazani i objašnjeni testovi delova aplikacija specifičnih za GraphQL koncepte.

2. GRAPHQL

GraphQL [1] je upitni jezik i kao nova tehnologija pruža potpun i opšti prikaz strukture podata ka kojim manipuliše API. Samim tim daje klijentima mogućnost da pri dobavljanju istih, jednostavno traže samo ono što im je potrebno i ništa više od toga.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio prof. dr Goran Savić.

2.1 Istorijat i motivacija

Kada je odlučeno da se 2012. godine naprave nove Facebook mobilne iOS i Android aplikacije, bio je potreban novi i drugačiji API, koji bi omogućio dobavljanje sirovih podataka. GraphQL kao novi API standard napravljen je interno od strane razvojnog tima kompanije Facebook 2012. godine, dok je 2015. godine javno publikovana specifikacija, implementacija i oformljena zajednica korisnika.

2.2 Glavni koncepti

Glavni koncepti objašnjavaju na šta se sve odnosi sam termin GraphQL, zatim kako se pristupa ovakvom servisu, kako se definiše struktura podataka i na kraju kako se podaci dobavljaju. Dosadašnji koncept pristupanja API-u zahtevao je da postoje više krajnjih tačaka (*eng. endpoint*) koji vraćaju podatke fiksne strukture. Za GraphQL pristup, neophodan je jedan endpoint koji se gađa, dok se u telu zahteva detaljno specificira zahtev. Jezik za definisanje šeme (*eng. Schema Definition Language, SDL*) je način za definisanje i opis GraphQL šeme. Ovakva sintaksa predstavlja deo zvanične GraphQL specifikacije. Glavne komponente definicije šeme su **tipovi** i njihova **polja**, a pored ovih mogu se naći i dodatne informacije o šemi u vidu direktiva. Tip (*eng. Type*) u GraphQL-u bi predstavljao pandam klasi u objektno orijentisanom modelu podataka. Kada se tip definiše, neophodno je navesti njegovo ime i polja koja mu pripadaju. U kompleksinijim šemama moguće je definisati i koristiti još i enumeracije, interfejse, fragmente i direktive.

Klijent svoj zahtev ka serveru piše u vidu upita, mutacija ili pretplata kojima mogu biti prosleđeni parametri. Upiti služe za dobavljanje podataka sa API-a, mutacije za iniciranje menjanja podataka (njihovim kreiranjem, ažuriranjem ili brisanjem), a pretplatama se otvara konekcija ka serveru preko koje će u realnom vremenu stizati odogovori vezani za tipove šeme na koje se klijent pretplatio. Uvedene su i tzv. ***resolver*** funkcije. One se pišu na serverskoj strani sistema i svaka pojedinačno odgovara jednom GraphQL upitu. Resolver funkcije zadužene su da obrade upit kom odgovaraju i kreiraju konačnu povratnu vrednost, uzimajući u obzir samo polja koja je klijent naveo da su mu potrebna.

2.4 GraphQL i Rest

Ovakav novi pristup izrazito odgovara razvijanju front-end aplikacija, jer je sva kompleksnost prebačena na serversku stranu. Glavni cilj nove GraphQL specifikacije bio je da postigne veću fleksibilnost i efikasnost pri komunikaciji između klijentske i serverske strane. Ono što je viđeno kao najčešći problem RestFULL aplikacija

je problem *overfetching*-a i *underfetching*-a. Takođe, uvođenjem GraphQL pristupa dobija se i detaljniji uvid u podatke koji su zaista potrebni sa serverske strane. Analizirajući upite koji dolaze sa klijentske strane, može se bolje razumeti kako i koji od dostupnih podataka se zaista koriste.

3. TESTIRANJE GRAPHQL APLIKACIJA

GraphQL aplikacije su promenile koncept interakcije i komunikacije između klijentske i serverske strane sistema. To je upravo dovelo do novih aspekata prilikom prolaska kroz neke od faza razvoja softvera, kao što je testiranje.

3.1Testiranje softvera

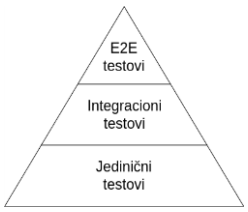
Testiranje softvera je proces koji se sastoji od svih aktivnosti u životnom ciklusu softvera, vezanih za: planiranje, pripremu i izvršavanje zadataka. Oni bi trebalo da pokažu da li softver zadovoljava specificirane zahteve i ispunjava namenu, kao i da detektuju eventualne greške u sistemu. Iz prethodne definicije direktno sledi da proces testiranja veoma utiče na povećanje kvaliteta softvera. Sa druge strane, često se navodi da potpuno testiranje proizvoda nije izvodivo u praksi. Iz tog razloga važno je fokusirati se na kritične delove sistema čije testiranje će poboljšati kvalitet i otkloniti eventualne nedostatke. Kada je reč o veb aplikacijama, testiranjem se teži da ona bude stabilna u produkciji i funkcioniše prema zahtevima kada stvarni krajnji korisnici počnu da je koriste. Pisanje testova takođe utiče na pisanje kvalitetnijeg koda. Kao jedna od važnih svrha testiranja može se navesti i dokumentovanje funkcionalnosti. Dobro testiran kod objašnjava, koje je očekivano ponašanje aplikacije, koji granični slučajevi postoje, kao i koje greške se mogu pojaviti [4].

3.2 Vrste testiranja softvera

Ovaj rad usmeren je na funkcionalne zahteve softvera koji koristi GraphQL specifikaciju, te će samim tim i biti izučavane funkcionalne vrste testova, i to: jedinični (*eng unit tests*), integracioni (*eng. integration tests*) i sistemski testovi (*eng. system tests*).

Jedinični testovi testiraju pojedinačne komponente softvera i one su po obimu, testovi najmanjeg nivoa granulacije. Proveravaju funkcionalnost manjeg dela koda, koji funkcioniše kao celina i uglavnom ne zavisi od ostalih spoljnih komponenti. **Integracioni testovi** namenjeni su otkrivanju nedostataka u interfejsima i interakciji između međusobno povezanih softverskih komponenata. Važno je pri njihovoj integraciji pisati i testove koji će proveriti funkcionalnost celokupnog integrisanog dela sistema. **Sistemski testovi** služe za kompletno testiranje integrisanog sistema da bi se utvrdilo da li sistem u celini ispunjava zahteve.

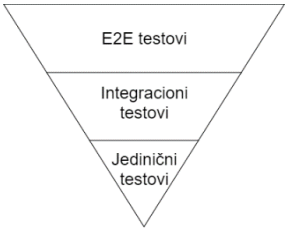
Ovakva vrsta testova poznata je još i po nazivu testiranja "s kraja na kraj" (*eng. end-to-end testing, e2e*), koji bolje naglašava testiranje akcija od samog početka korisnikove interakcije do krajnjeg odgovora sistema na tu akciju. **Piramida testiranja** predstavlja jednu od strategija pisanja testova. Prvi put je spomenuta od strane Majka Kona (Mike Cohn) [2] a zasniva se na konceptu iz psihologije poznatog kao "Piramida potreba" [3] (*eng. Pyramid of Needs*).



SLIKA 3.1 STRATEGIJA PIRAMIDA TESTIRANJA

Ovakav prikaz strategije testiranja pokazuje kako se pojedinačni, integracioni i sistemski testovi oslanjaju jedan na drugi i to iz ugla programera. Na dnu piramide se nalaze pojedinačni testovi, što govori da je važno testirati što više pojedinačnih komponenti, jer se na njima zasniva rad cele aplikacije. Zatim su u sredini integracioni testovi, jer bi trebalo proveriti komunikaciju tih komponenti. Na kraju, na vrhu piramide se nalaze sistemski testovi, jer su oni najteži i najsporiji, a u većoj meri se mogu osloniti na prethodna dva tipa testova kojih bi bilo više.

Sa druge strane, posmatrano iz ugla vođe projekta (*eng. project manager*) sistemski testovi imaju najveću važnost. Zbog toga je nastala i struktura **obrnute piramide testiranja** (*eng. reverse testing pyramid*) koja pokazuje drugu strategiju odnosa ova tri tipa testova (Slika 3.2) [4].



SLIKA 3.2 STRATEGIJA OBRNUTE PIRAMIDE TESTIRANJA

Nakon razmatranja ovakvih strategija testiranja, došlo se do kompromisnog rešenje prikazanog, opet simbolično, figurom trofeja. Nova strategija razvoja i odnosa testova nazvana je **trofej testiranje** (*eng. Testing Trophy*) [5].

SLIKA 3.3 STRATEGIJA TROFEJ TESTIRANJA

3.3 Testovi specifični za GraphQL aplikacije

Poznavajući osnovne koncepte GraphQL specifikacije, može se zaključiti da testiranje ovakvih aplikacija zahteva dodatne testove specifične upravo za te koncepte. Sa serverske strane sistema, očekivani su pojedinačni (*unit*) testovi GraphQL šeme i *resolver* funkcija. Prilikom testiranja šeme proveravaju se da li su polja kreiranih tipova očekivana prema specifikaciji. Ovakvi testovi neće biti od krucijalne važnosti sve dok ne bude porasla kompleksnost šeme, ali oni će svakako doprineti povećanju pokrivenosti aplikacije testovima. Na serverskoj strani pišu se i integracioni testovi specifični za GraphQL koncepte. Nakon imitiranja (*eng. mocking*)

šeme i testiranja *resolver* funkcija, testovima se potvrđuje da li slanje upita, mutacije ili pretplate pogađa očekivanu resolver funkciju i vraća očekivani odgovor na klijentsku stranu [4]. Sa klijentske strane sistema, korisno je pisati testove za komponente koje dobavljaju podatke od GraphQL API-a, iniciraju njihovu promenu ili se pretplaćuju na događaje vezane za te iste podatke.

Da bi se testirala sama konekcija između klijentske i serverske aplikacije, dobro je prethodno testirati funkcionalnost samih *resolver* funkcija pojedinačnim ili integracionim testovima. Za ove potrebe takođe se koriste i razne biblioteke koje pomažu imitiranje podataka kakvi bi mogli doći kao odgovor servera na upit, tzv. mokovanje (*eng. mocking*).

4. STUDIJA SLUČAJA

Za potrebe pisanja ovog rada, osmišljena je aplikacija koja bi svojim funkcionalnostima pokrila i iskoristila sve osnovne koncepte GraphQL specifikacije. Aplikacija nazvana *sci-demo-lib*, omogućava pretragu baze naučnih časopisa i radova u njima, zatim autora radova i naučnih oblasti. Korisnici aplikacije bili bi istraživači, profesori ili naučnici sa određenog univerziteta, koji sa univerzitetskom email adresom kreiraju nalog, a zatim dobijaju pristup za pregled i pretraživanje dostupnog materijala.

4.1 Funkcionalnosti i pregled sistema

Registracija korisnika moguća je svakom ko poseduje univerzitetsku email adresu. Tada se popunjavaju polja sa email adresom, lozinkom, potvrdom lozinke, imenom i prezimenom. Prilikom prijave potrebno je uneti email adresu i lozinku već kreiranog korisnika. Prijavljen korisnik se može i odjaviti sa sistema. Na stranicu za prikaz i ažuriranje sopstvenog profila prijavnjeni korisnik dolazi odmah nakon prijave na sistem ili kroz glavni meni.

Korisniku aplikacije su dostupni lista autora, časopisa, rada i oblasti, a zatim i informacije o svakom pojedinačno. On ima mogućnost da se pretplati na neki od njih, što znači da će ubuduće dobijati obaveštenja od aplikacije kada časopis objavi novo izdanje, zatim kada autor objavi novi rad ili kada se pojavi novi časopis, rad ili autor koji se bavi određenom oblašću.

4.2 Model podataka

Model podataka sistema preslikava se na odgovarajuće tabele u bazi podataka sistema. Njime su modelovani entiteti koje se pojavljuju u aplikaciji kao što su: *User, UserRole, Paper, Journal, ScienceArea, University, City* i *Country*. Ova aplikacije razlikuje tri tipa korisnika: administrator, autor i običan korisnik. Svaki korisnik, rad i časopis označeni su određenim naučnim oblastima kojima se bave.

Naučni rad obavezno mora imati jednog glavnog autora i može imati ni jednog ili više koautora rada, dok svaki časopis referencira na naučne radove koji su u njemu objavljeni. Kao što je već opisano, korisnik ima mogućnost da se pretplati na nekog od autora, naučni rad ili časopis, te zbog toga klasa korisnika takođe ima i N:N vezu ka ovim klasama.

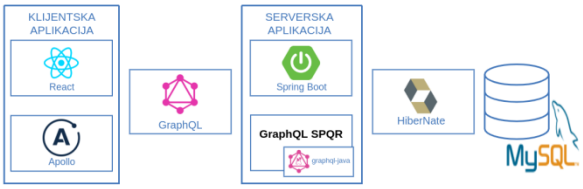
4.3 GraphQL šema

GraphQL šema igra jednu od vodećih uloga u komunikaciji između klijentske i serverske strane sistema. Analizom šeme aplikacije može se uvideti da svaka klasa iz modela podataka odgovara jednom novokreiranom tipu u šemi. Ovo se objašnjava time da se pri dobavljanju podataka iz baze dobijaju objekti tih klasa, koje se dalje prosleđuju na kijentsku stranu sistema, na primer kao odgovor na GraphQL upit.

4.4 Implementacija

Opisani sitetem sastoji se od klijentske i serverske aplikacije. Klijentska aplikacija koristi GraphQL tako što šalje zahteve u vidu GraphQL upita, mutacija i pretplata, a na serverskoj strani se ovi zahtevi obrađuju i odgovore šalju nazad. Na slici 4.1 mogu se videti tehnologije koje su korišćene u ovom sistemu za demonstraciju. Za izradu klijentske aplikacije, korišćen je javascript programski jezik i ReactJS [6] biblioteka za izradu celokupnog korisničkog interfejsa. Za potrebe slanja zahteva i preuzimanja odgovora preko GraphQL-a, korišćena je Apollo biblioteka otvorenog koda [7]**Error! Reference source not found.**. Pomoću ove dve bibilioteke znatno je olakšano dobavljanje, keširanje i menjanje podataka u aplikaciji, dok se korisnički interfejs automatski osvežava. Serverska strana sistema implementirana je kao Spring Boot aplikacija [14]. Za implementaciju GraphQL specifikacije na serverskoj strani, odabrana je GraphQL SPQR biblioteka.

Apollo biblioteka osnovnu, ugrađenu podršku daje React aplikacijama. Na početku neophodno je dodati njenu konfiguraciju. Zatim, da bi se tako kreirani klijent objekat povezao sa React aplikacijom koristi se ApolloProvider koji služi kao omotač i omogućava da mu se pristupi iz bilo kog dela aplikacije.



SLIKA 4.1 TEHNOLOGIJE KORIŠĆENE PRI IZRADI SCI-LIB-DEMO APLIKACIJE ZA DEMONSTRACIJU

Kada je podešena, komponente su spremne za slanje zahteva u vidu upita, mutacija i pretplata. Od verzije 3.0 Apollo biblioteke, zahtevi se pišu u formi React *hook*-ova (*eng. React hooks*) [9]. Komponenta može da prepozna promene statusa odgovora i prikazuje drugačiji korisnički interfejs. Sa druge strane, za implementaciju GraphQL specifikacije serverskog dela sistema, korištena je GraphQL SPQR biblioteka otvorenog koda [10]. Kada se razvijaju serverske GraphQL aplikacije, često se koristi pristup gde se prvo definiše šema, a kasnije se ona spaja sa biznis logikom aplikacije (*schema-first* pristup). Ovakav pristup dovodi do velike količine dupliranog koda. Da bi se takav pristup izbegao, u *sci-lib-demo* aplikaciji korištena je GraphQL SPQR biblioteka, koja koristi pristup poznat kao *code-first*. Dovoljno je prilikom implementacije biznis logike aplikacije anotirati određene

java funkcije i klase kako bi ova biblioteka znala kako da generiše šemu. U listingu 4.1 može se videti primer anotiranja jedne *resolver* funkcije, da bi na kraju pomenuta biblioteka znala kako da izgeneriše *getUser* upit unutar šeme.

```
@GraphQLQuery(name="getUser")
public User
getUser(@GraphQLArgument(name="id")
Long id) {
   return userService.findOne(id);
}
```

LISTING 4.1 PRIMER ANOTIRANJA RESOLVER FUNKCIJE

4.5 Testiranje sci-lib-demo aplikacije

U testove specifične za GraphQL aplikacije spadaju testovi GraphQL šeme i *resolver*-a na serverskoj strani sistema, kao i testovi komponenti na klijentskoj strani. Prilikom testiranja šeme, neophodno je proveriti da li ona sadrži sve tipove koji se očekuju, i da li pojedinačno svaki tip sadrži određena polja određene vrste. Testovi koji proveravaju generisanu šemu implementirani su pomoću standardne JUnit biblioteke [11]. Potrebno je u testu kreirati GraphQLSchema objekat koji će generisati šemu za testiranje, a kasnije se iz tog objekta dobijaju i proveravaju određeni delovi šeme. Posle toga slede provere tipova i polja u tipovima.

Za potrebe pisanja integracionih testova za *resolver* funkcije korištena je takođe JUnit biblioteka [11]. Svakom testu ovog tipa potreban je instanciran *GraphQLSchema* objekat, da bi se pomoću njega kreirao GraphQL objekat koji omogućava izvršavanje upita. Kako se unutar *resolver* funkcije pozivaju servisi i dobavljaju ili menjaju podaci iz baze, njih je potrebno izimitirati (eng. mocking). Na ovaj način proverava se sam poziv upita u odnosu na njegov naziv i eventualno parametre. Za mokovanje *resolver* funkcija korištena je Mockito biblioteka [12]. Ona je iskorištena da bi se prosledila klasa UserResolver-a koja će tada biti instancirana.
Pri testiranju vizuelnih komponenti na klijentskoj aplikaciji, potrebno je proveriti da li ona dobija od servera očekivani odgovor. To bi spadalo u integracioni test, a zatim bi trebalo da se proveri da li se komponenta sa dobijenim podacima i prikazuje onako kako je predviđeno, što se dalje svrstava u jedinične testove. Za potrebe pisanja testova za komponente korisničkog interfejsa korišten je Jest [13] radni okvir za testiranje, dok je za pokretanje testova korištena React Test Renderer biblioteka [14]. Jedinično testiranje komponenti korisničkog interfejsa takođe zahteva mokovanje odgovora servera na poslate zahteve. Apollo biblioteka obezbeđuje MockedProvider komponentu koja upravo maskira pozive GraphQ server.

7. LITERATURA

[1] The Linux Foundation. A query language for your API, https://graphql.org/

[2] Mike Cohn, Succeeding with Agile: Software Development Using Scrum,Signature Book,2009

[3] Anne Mette Hass, Guide to Advanced Software testing, Artech House, 2014

[4] Roy Derks, Testing GraphQl: From Zero to Hundred Percent, NDC Conference

[5] Kent C. Dodds Static vs Unit vs Integration vs E2E Testing for Frontend Apps, www.kentcdodds.com/blog/unit-vs-integration-vs-e2e-tests

[6] Facebook Inc., ReactJS https://reactjs.org/, pristupljeno: 15.10.2020.

[7] Apollo Graph Inc., „Apollo GraphQL“, www.apollographql.com/, pristupljeno: 15.10.2020.

[8] Pivotal „SpringBoot“ https://spring.io/projects/spring-boot, pristupljeno: 21.10.2020.

[9] Introducing React Hooks, https://reactjs.org/docs/hooks-intro.html, pristupljeno: 21.10.2020.

[10] Leangen, GraphQL SPQR Github repozitorijum, https://github.com/leangen/graphql-spqr, pristupljeno: 21.10.2020.

[11] The JUnit Team, JUnit 5, https://junit.org/junit5/ , pristupljeno: 21.10.2020.

[12] Mockito, Mocking framework for unit tests in Java, https://site.mockito.org/ , pristupljeno: 21.10.2020.

[13] JestJS, www.jestjs.io/, pristupljeno: 21.10.2020.

[14] Facebook Inc., React Test Renderer, https://reactjs.org/docs/test-renderer.html

Kratka biografija:

Aleksandra Milivojević rođena je 5. avgusta 1994. godine u Lazarevcu. Završila je Gimnaziju “Jovan Jovanović Zmaj” u Novom Sadu.
Zvanje diplomirani inženjer elektrotehnike i računarstva stiče 13.10.2017. godine, sa odbranom diplomskog rada na temu “Kompleksnost lozinke korisničkog naloga na Windows operativnim sistemima”. Iste godine upisuje master akademske studije na smeru Softversko inženjerstvo i informacione tehnologije, odsek Elektronsko poslovanje.