Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/07BE39Blagojevic

AKTIVNI INKREMENTALNI GENERATOR GRAPHQL SERVERSKE APLIKACIJE

ACTIVE INCREMENTAL CODE GENERATOR FOR GRAPHQL SERVER APPLICATION

Bojan Blagojević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je predstavljen generator koda za server koji podržava GraphQL specifikaciju. Model se zadaje preko GraphQL SDL sheme. Generisanje koda se vrši uz pomoć obrađivača šablona. Podržano je i redefinisanje ponašanja aplikacije integracijom sa ručno pisanim kodom kao i inkrementalne izmjene sheme baze podataka.*

Ključne riječi: *Generator koda, GraphQL, SDL, šablon*

Abstract – *This paper presents the code generator for the web server that satisfies GraphQL API specification. The model is provided using GraphQL SDL schema. Code generation is performed using template engines. Generator also supports modifying application default behaviour by integrating generated code with manually written one. It also supports incremental updates for db schema.*

Keywords: *Code generator, GraphQL, SDL, template*

1. UVOD

Razvojem tehnologije te povećanjem broja aplikacija zasnovanih na mrežnoj komunikaciji, softverski inženjeri se suočavaju sa potrebom da definišu i unificiraju klijent-server komunikaciju. Kao proizvod ove ideje nastaje nekolicina specifikacija komunikacije između dvije udaljene tačke u vidu API-a (*Application programming interface*).

Među specifikacijama API-a, u posljednje vrijeme, najveću popularnost stekla je GraphQL specifikacija [1].

U prilog tome govori i činjenica preuzeta iz istraživanja [2] u kome se prema podacima iz 2017. godine svaki mjesec kreira preko 350 projekata na Github-u, u čijem se imenu ili opisu pominje riječ GraphQL.

Pisanje GraphQL servera se često svodi na pisanje repetitivnog koda vođenog unaprijed definisanom GraphQL shemom. Resursi opisani shemom su uglavnom dovoljni za definisanje interfejsa komunikacije. Nad navedenim resursima najčešće su definisane operacije dobavljanja, kreiranja, kao i njihovog brisanja i izmjene. Sve činjenice ukazuju na to da se softverski inženjeri dovode u situaciju da pišu ponavljajući kod, ali kako je on pisan od strane čovjeka, i dalje ostaje mogućnost pojave grešaka, nekonzistentnosti i bespotrebnog trošenja ljudskih resursa.

U idealnom slučaju GraphQL shema je dobro definisana na početku procesa razrade modela projekta.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Gordana Milosavljević.

Međutim, kako ovaj slučaj nije čest, shema je nerijetko podložna promjenama. Da je to zaista slučaj, potvrđuje i statistika izvučena iz analize evolucije sheme baze podataka za HMS softver koji je korišten u bolnicama u Velikoj Britaniji. Rezultati prikazani u radu [3], analizirani su tokom perioda od 1990. do 1991. godine.

Analizom se došlo do zaključka da je u navedenom periodu, broj relacija između entiteta porastao sa 23 na 55, što je uvećanje za 139%. Takođe, broj kolona se povećao sa 178 na 666, što predstavlja uvećanje za čak 274%. Interesantno je da je od 20 relacija koje su pronađene u novembru 1990. i 1991. godine, samo četiri su ostale nepromijenjene. U drugom iztraživanju [4] vršenom nad MediaWiki softverom koji je *backend* podrška za Wikipedia-u, autori rada su takođe napravili analizu izmena baze podataka. U studiji koja obuhvata četiri godine, došli su do zaključka da se broj tabela uvećao za 100%, a broj kolona za 142 %. Shema je pretrpjela i povećanje broja obrisanih atributa za 41.5 %, dok se broj preimenovanih polja povećao za 25.1 %. Sve ove promjene modela dodatno utiču na usporavanje procesa implementacije.

Jedno od rješenja ovog problema jeste uvođenje generatora koda za GraphQL server. Generatori koda se pišu sa ciljem da se isti generator koda može koristiti više puta za različite ulaze istog tipa. Pisanje generatora koda traje konstantno vrijeme, dok pisanje sličnih aplikacija koje generator može da izgeneriše ima linearnu kompleksnost koja zavisi od toga koliko se puta takva aplikacija implementira. Kada se jednom napiše generator koda za određeni programski jezik, veoma je lako dodati i *plugin* za generisanje iste pozadinske logike za drugi programski jezik. Generator koda rješava i problem konzistentnosti jer za iste programske konstrukcije koristi isti šablon te je na taj način jednoznačnost prilikom generisanja osigurana.

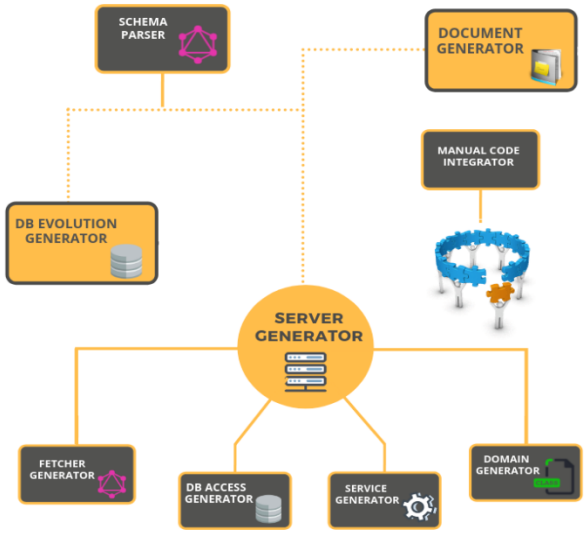
2. SPECIFIKACIJA SISTEMA

Problem kojim se ovaj rad bavi jeste mogućnost generisanja GraphQL servera na osnovu GraphQL sheme. Takođe, generator treba da riješi i problem čestih izmjena nad definisanom shemom, kao i da omogući redefinisanje standardnog ponašanja aplikacije tako što će da dozvoli korisniku da integriše ručno pisani kod. U nastavku teksta predstavljena je arhitektura generatora kroz prikaz njegovih osnovnih komponenti.

Generator GraphQL servera, koji se opisuje u radu, baziran je na pet osnovnih komponenti (Slika 1):

- Parser GraphQL sheme
- Generator evolucija baze podataka
- Generator servera
- Generator dokumentacije

• Komponenta za integrisanje ručno pisanog koda.



Slika 1. *Shema modela generatora*

Parser GraphQL sheme podataka predstavlja prvu komponentu u procesu generisanja. Ova komponenta je zadužena za parsiranje GraphQL sheme prema standardima koje definiše SDL format. Ulaz u parser sheme jeste jedna konkretna instanca SDL sheme preko koje su opisani svi entiteti koji će se koristiti u aplikaciji. Ova komponenta je zadužena i za detekciju grešaka nastalih u definiciji sheme čime se suzbija propagacija greške kroz druge komponente. Kao rezultat dobija se interna reprezentacija modela u vidu klasa kojima generator jednostavnije manipuliše. Interna reprezentacija (Slika 2) zapravo predstavlja međumodel koji služi kao ulaz u ostale komponente sistema.

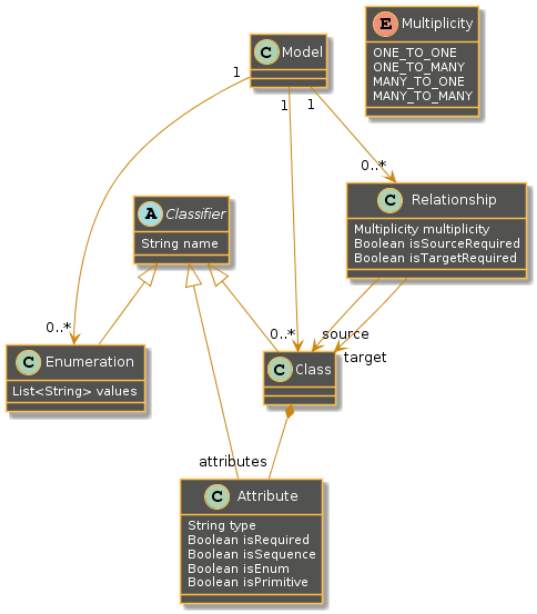
Generator evolucije baze podataka predstavlja komponentu zaduženu za generisanje sheme baze podataka na osnovu interne reprezentcije modela. Osnovne probleme koje treba da riješi ova komponenta jeste mapiranje GraphQL entiteta na entitete neke od relacionih baza podataka (Tabela 1), kao i prilagođavanje sheme baze podataka izmjenama nastalim u GraphQL shemi (Tabela 2).

Tabela 1. *Klase ekvivalencije između GraphQL tipova i relacione baze podataka*

GraphQL tipovi	**Relaciona baza podataka**
Type	*Table*
Primitive field	*Column*
Complex field	*Foreign key / Table*
Enum	*Enum type*

Tabela 2. *Mapiranje izmjena GraphQL sheme na evoluciju sheme baze podataka*

Izmjena GraphQL sheme	**Izmjena relacione sheme baze podataka**
Dodavanje novog tipa	*Dodavanje nove tabele*
Dodavanje novog primitivnog polja	*Izmjena tabele uz dodavanje nove kolone*
Preimenovanje polja	*Preimenovanje kolone tabele*
Promjena kardinaliteta sa oneToMany na manyToMany	*Dodavanje nove tabele uz brisanje kolone koja je predstavljala strani ključ u prethodnoj relaciji*



Slika 2. *Dijagram klasa interne reprezentacije modela*

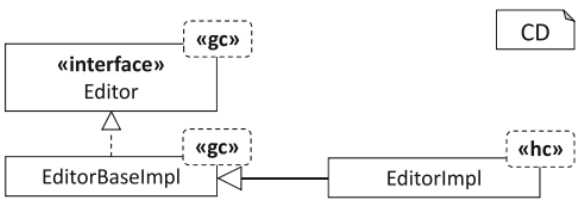
Najkompleksnija komponenta generatora je generator servera. Njegova osnovna funkcionalnost zasniva se na kreiranju funkcionalnog *backend*-a prema GraphQL specifikaciji. Sastoji se iz četiri međusobno zavisne cjeline:

- Generator domenskih klasa
- Generator servisnog sloja
- Generator GraphQL rukovaoca (eng. *handler*)
- Generator sloja za pristup bazi (*database access layer*)

Arhitekturalno gledano, prva tri generatora se mogu svrstati u sloj biznis logike aplikacije, dok se sloj za pristup bazi podataka može okarakterisati kao dio sloja za perzistenciju. Sloj koji je najbliži klijentskoj aplikaciji generiše se pomoću generatora za GraphQL *handler*-e. Za svaki od GraphQL tipova, generator GraphQL *handler*-a treba da omogući podršku za razrješavanje sljedećih zahtjeva:

- GraphQL mutacija za dodavanje entiteta
- GraphQL *query*-a za čitanje podataka

Postoji više tehnika za integraciju generisanog koda sa ručno pisanim kodom. U radu [5] definisane su tehnike za integraciju manuelno pisanog koda za generatore zasnovane na principima objektno-orijentisanog modela. Autori rada opisali su osam osnovnih tehnika za integraciju ručno pisanog koda evaluiranog kroz pet kriterijuma. Mehanizam koji je odabran da se koristi pri integrisanju ručno pisanog koda, autori rada su nazvali *Generation Gap*. Ova tehnika podrazumijeva da se za svaku operaciju generiše interfejs i uobičajeno ponašanje. Ukoliko korisnik želi da redefiniše ponašanje aplikacije, mora da kreira svoju implementaciju interfejsa koju će generator koristiti umjesto uobičajene implementacije. Model ovog mehanizma za integrisanje ručno pisanog koda prikazan je na slici 3.



Slika 3. *Generation Gap* [5] *mehanizam za integraciju ručno pisanog koda*

Generator dokumentacije, predstavlja posljednju komponentu sistema. Zadužen je za generisanje GraphQL IDE-a preko koga se može istraživati GraphQL shema i pregledati dokumentacija vezana za tipove i njihova polja. Takođe, IDE ima direktno integrisani *syntax highlighting* kao i validaciju operacija nad definisanom shemom. Pored toga, u navedenom IDE-u moguće je i izvršiti određene operacije nakon čijeg izvršavanja se prikazuje i rezultat. GraphQL IDE koji se generiše je web aplikacija integrisana sa izgenerisanim GraphQL serverom.

3. IMPLEMENTACIJA RJEŠENJA

3.1 Tehnologije

Generator je implementiran u Python programskom jeziku. Generisanje konstrukcija izvornog koda vrši se uz pomoć Python Jinja *template* jezika [6]. Izgenerisani kod pisan je u Scala programskom jeziku uz pomoć Play [7] radnog okvira. Za parsiraranje GraphQL sheme korištena je Python GraphQL biblioteka, dok se za serversku podršku GraphQL specifikacije u izgenerisanoj aplikaciji koristi Scala biblioteka pod imenom Sangria [8]. Perzistentni sloj prepušten je PostgreSQL bazi podataka, dok se za pristup istoj koristi Scala biblioteka pod imenom Slick [9]. Za pronalaženje razlika između trenutne i nove PostgreSQL sheme koristi se pgdiff [10] alat. Kompletna infrastruktura za lokalno okruženje podignuta je uz pomoć Docker [11] kontejnera.

3.2 Generator koda

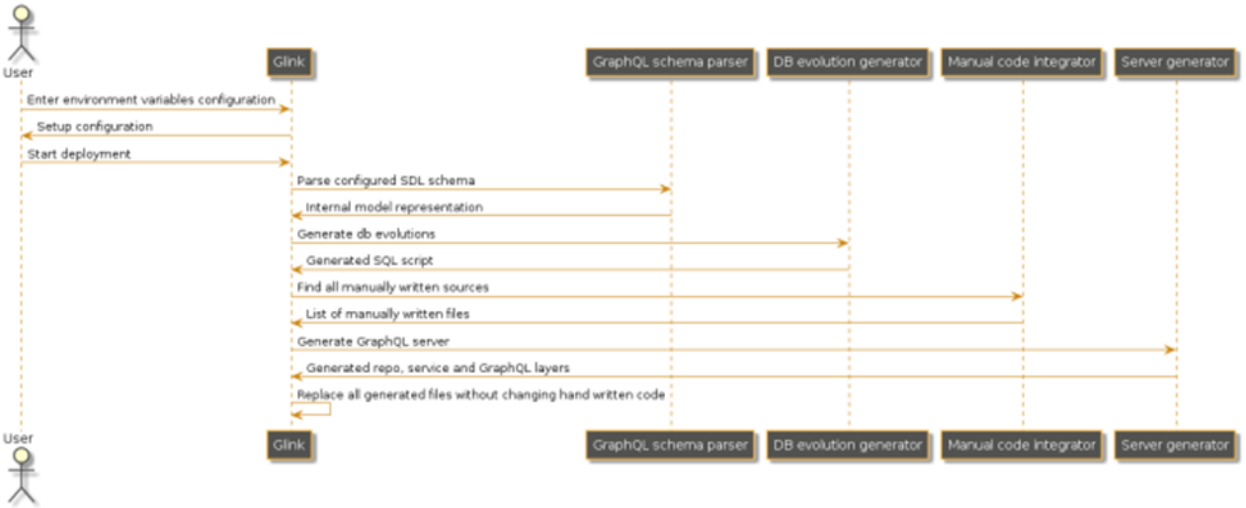
Proces generisanja GraphQL servera, kako je i prikazano na slici 4, sastoji se iz nekoliko koraka. Prvi dio u procesu generisanja predstavlja konfigurisanje putanje do izvorne sheme, foldera u kome će se nalaziti generisani kod i parametara za konekciju na PostgreSQL bazu podataka. Drugi dio u procesu generisanja ne zahtijeva dodatnu interakciju sa korisnikom generatora. Ovaj proces uključuje generisanje servera i svih propratnih sadržaja na osnovu unaprijed definisane konfiguracije. Kako bi se učinilo da čitav ovaj proces bude jednostavan za upotrebu, kreiran je i SDK preko koga se upotreba gorenavedenih instrukcija može olakšati pozivanjem jednostavnih komandi iz sistemske konzole. U skladu sa tim, SDK ima sljedeće komande:

- *init* – inicijalizacija projekta
- configure – podešavanje konfiguracije
- deploy – pokretanje procesa generisanja

Configure komanda služi za podešavanje parametara konfiguracije. Ova komanda može da prima dodatne parametre, za svaku varijablu po jedan. Mogu se konfigurisati putanja do SDL sheme, putanja do foldera u kome će se naći izgenerisani kod, kao i parametri za konekciju ka PostgreSQL bazi podataka.

Nakon što je konfiguracija projekta završena, korisnik generatora mora da pozove *deploy* komandu. Prvi proces koji se nakon toga dešava jeste proces pronalaženja i validiranja specificiranog SDL izvornog fajla. Validacija proslijeđene SDL sheme kao i njeno parsiranje vrši se korištenjem graphql-core Python biblioteke. API ove biblioteke pruža *parse* metodu, kojoj se prosljeđuje SDL shema, te se kao rezultat dobija *Document* objekat. Pimjer jedne ulazne GraphQL sheme dat je na listingu 1.

Generisanje sheme baze podataka u potpunosti je prepušteno Slick biblioteci. Slick biblioteka ima podršku za generisanje SQL iskaza za kreiranje sheme na osnovu opisa iste nad Scala klasama. Na taj način problem generisanja sheme, kao i njenog mapiranja, lokalizovan je na jednom mjestu. Ukoliko dođe do potrebe da generator podržava i neku drugu bazu podataka, neophodno je da se samo zamijeni *driver* baze podataka. Ostatak generisanja sheme je prepušten Slick implementaciji za navedeni *driver*.



Slika 4. *Dijagram sekvence za proces generisanja GraphQL servera*

```
type User {
  firstName: String!
  lastName: String
  email: String!
  age: Int
}
enum TodoState {
  CREATED
  IN_PROGRESS
  FINISHED
}

type Todo {
  title: String!
  description: String
  state: TodoState
  list: TodoList @oneToMany(mappedBy: "items")
}
type TodoList {
  title: String!
  items: [Todo] @manyToOne(mappedBy: "list")
  creator: User @oneToMany
}
```

Listing 1. *Primjer ulazne sheme generatora*

Nakon što se izgeneriše shema baze podataka, generiše se Play server. Generator serverske aplikacije zadužen je za generisanje Play aplikacije koja koristi GraphQL kao specifikaciju za razmjenu resursa. Generator generiše domenske klase aplikacije, sloj repozitorijuma, servisni sloj zajedno sa biznis logikom kao i GraphQL komponente (*GraphQL* tipovi, operacije i *resolver*-i). Kao posljednji proces u generisanju aplikacije izvršava se integracija ručno pisanog koda kao i generisanje GraphQL IDE-a.

4. ZAKLJUČAK

U ovom radu predstavljen je jedan jednostavan generator GraphQL servera namijenjen generisanju manjih aplikacija čiji domen ima uže područje djelovanja. Kao što je ranije navedeno, generišu se isključivo operacije za dobavljanje i kreiranje novih resursa, što za veće aplikacije često nije dovoljno. Velika primjena ovog generatora može se tražiti i u aplikacijama čiji je razvoj baziran na GraphQL shemi. Pod ovim se podrazumijevaju aplikacije u čijem se procesu razvoja prvo specificira struktura sheme, pa se onda implementacija prilagođava istoj. Takođe, generator može naći primjenu i u aplikacijama čiji je domen sklon čestim promjenama. Pošto generator automatizuje primjenu promjena sheme na bazu podataka kao i na izgenerisani kod, razvojni tim se može osloniti da rukovođenje promjenama prepusti generatoru. Na taj način smanjuje se potencijalni izvor grešaka kod ručno pisanog koda. Pošto generator generiše optimalan Scala kod u kombinaciji sa GraphQL konstrukcijama, namijenjen je i svim entuzijastima koji žele da nauče GraphQL tehnologiju.

Kako je navedeno u radu, generator može da generiše isključivo operacije za dobavljanje i kreiranje resursa. U određenim slučajevima ovakva postavka nije dovoljna te bi prvobitno poboljšanje moglo predstavljati dodavanje operacija za izmjenu i brisanje resursa. Pored toga, generator se može proširiti i dodavanjem dodatnih validacija polja kao što su jedinstvenost polja ili zadovoljavanje nekog šablon. Dodatni pravci proširenja mogu se ogledati i u dodavanju mehanizama za autorizaciju za određene operacije kao i generisanju korisničkog interfejsa aplikacije.

5. LITERATURA

[1] „GraphQL specification“, https://graphql.github.io/graphql-spec/June2018, pristupljeno: 11. jul 2019.

[2] „The GraphQL Data Language“, https://thenewstack.io/graphql-data-query-language-resource-guide, pristupljeno: 11. jul 2019.

[3] Dag Sjoberg, Lilybank Gardens, „Quantifying Schema Evolution“, Information and Software Technology, Vol. 35, No. 1, pp. 35-44, januar 1993.

[4] Carlo A. Curino, Hyun J. Moon, Letizia Tanca, Carlo Zaniolo, „Schema evolution in Wikipedia toward a Web Information System Benchmark“, ICEIS 2008 - Proceedings of the Tenth International Conference on Enterprise Information Systems, Volume DISI, 12-16 jun 2008.

[5] P. Desfray, J. Filipe, S. Hammoudi, L. Pires, „Integration of Handwritten and Generated Object-Oriented Code“, Model-Driven Engineering and Software Development, Communications in Computer and Information Science, vol 580. Springer, Cham, 2015.

[6] „Jinja2“, https://jinja.palletsprojects.com/en/2.10.x/, pristupljeno: 17. avgust 2019.

[7] „Play framework“, https://www.playframework.com, pristupljeno: 17. avgust 2019.

[8] „Sangria“, https://sangria-graphql.org, pristupljeno: 17. avgust 2019.

[9] „Slick“, http://slick.lightbend.com/, pristupljeno: 17. avgust 2019.

[10] „pgdiff“, http://pgdiff.sourceforge.net/, pristupljeno: 17. avgust 2019.

[11] „Docker“, https://www.docker.com, pristupljeno: 17. avgust 2019.

Kratka biografija:

Bojan Blagojević rođen je 1. januara 1994. godine u Bijeljini, Republika Srpska, Bosna i Hercegovina. Osnovnu školu „Aleksa Šantić” u Ugljeviku i Gimnaziju „Filip Višnjić” u Bijeljini je završio kao nosilac Vukove diplome. 2013. godine upisao je Fakultet tehničkih nauka u Novom Sadu, smijer Softversko inženjerstvo i informacione tehnologije. 2017. godine završava pomenute osnovne akademske studije sa prosječnom ocjenom 10.00. Iste godine upisuje master akademske studije na istom smijeru. Položio je sve ispite predviđene planom i programom.