Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/22BE24Sarenac

HTML VIZUELIZACIJA GRAPHQL SCHEMA-E U VS CODE EDITORU

HTML VISUALIZATION OF GRAPHQL SCHEMA IN VS CODE EDITOR

Nikolina Šarenac, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Kroz ovaj rad se razmatra mogućnost vizuelnog dokumentovanja interfejsa web servisa definisanog pomoću GraphQL-a. Implementirano je i predstavljeno jedno takvo rešenje, počevši od njegove strukture, načina implementacije, do demonstracije implementiranih funkcionalnosti.*

Ključne reči: *GraphQL šema, SDL, vizuelizacija, API dokumentacija*

Abstract – *This paper contemplates the possibilities of implementing visual documentation of web service APIs that are defined using GraphQL. Alongside, one such solution is implemented and will be presented here, starting with its structure, implementation details, and up to the demonstration of containing functionalities.*

Keywords: *GraphQL schema, SDL, visualization, API documentation*

1. UVOD

U poslovnom svetu, svakodnevno se generišu velike količine podataka u raznim sferama delovanja, što podstiče sve veću zainteresovanost za analizu efikasnosti načina upotrebe i upravljanja tim podacima. Kako su podaci često velike kompleksnosti i međusobno povezani na različite načine, potrebno je da njihova razmena između različitih sistema bude što bolje definisana i optimizovana. Optimalan prenos podataka obuhvata razmenu najužeg potrebnog skupa podataka, obavljenu sa što manje komunikacije između različitih entiteta.

Komunikacija između različitih sistema definisana je specifikacijom aplikativnih interfejsa, odnosno API-ja (eng. *Application Programming Interface)*, a među specifikacijama, imajući u vidu pomenute prednosti, trenutno najveći rast u popularnosti ima *GraphQL*, koji je detaljnije opisan u odeljku 1.1.

Pored specifikacije API-ja veoma je bitan i način na koji je dokumentovan. Na osnovu dokumentacije korisnici, bilo da su klijenti koji koriste funkcionalnosti nekog servisa ili programeri koji takav servis implementiraju, treba da budu u potpunosti upućeni u način na koji se interfejs može koristiti. Velika kompleksnost se često može uprostiti upotrebom vizuelnih prikaza, a više reči o mogućnosti konkretne primene vizuelizacije za dokumentovanje *GraphQL* interfejsa biće u odeljku 1.2.

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

1.1 *GraphQL*

GraphQL opisuje strukture upita koji neki servis pruža korisnicima, kao i tipove podataka koji se koriste. Nastao je kao alternativa REST *(*eng. *Representational State Transfer) web* servisima, a jedna od najvećih prednosti je fleksibilnost u pogledu izbora konkretnih polja povratnih entiteta, gde u okviru upita korisnik dobija tačno one podatke koje traži, ni manje ni više od toga. Fleksibilnost se ogleda i u nezavisnosti od platforme kao i programskog jezika za implementaciju servisa.

GraphQL specifikacija definiše i jezik za opis šeme (eng. *Schema Definition Language - SDL)* koja predstavlja datoteku u kojoj su opisani svi postojeći tipovi, kao i upiti koji su na raspolaganju klijentu (koji se koriste za dobavljanje podataka) i mutacije (koje se koriste za modifikaciju podataka npr.: kreiranje, izmena i brisanje). *GraphQL* šema je pisana jezikom za definiciju šeme i skladištena u datoteci sa *.graphql* ekstenzijom. Na slici 1. je prikazan izgled jednostavne šeme sa definisanim tipovima *User* i *Address* kao i primerima upita i mutacija koji te tipove koriste.

```
type User {
  id: ID
  firstName: String
  lastName: String
  address: Address
}

type Address {
  city: String
  postalCode: Int
}

type Query {
  user(id: ID): User
  userOnAddress(address: Address): [User]
}

type Mutation {
  createAddress(city: String!, postalCode: Int): Address
  deleteUser(id: ID): Boolean
}
```

Slika 1. *Primer GraphQL šeme*

Ovako definisana šema predstavlja i sveobuhvatnu dokumentaciju interfejsa, ona u potpunosti opisuje kompleksne tipove koji se koriste, kao i upite nad podacima koji su korisniku dostupni. Takođe podržava i dodatak opisa za svaki podržani upit. Neki od nedostataka upotrebe šeme kao dokumentacije mogu biti nepreglednost u slučaju kompleksnog interfejsa kao i slabija čitljivost velike količine upita i tipova.

1.2 Vizuelizacija dokumentacije *GraphQL* servisa

Vizuelizacija kao metoda za pojednostavljenje kompleksnih prikaza se dugo koristi za različite tehnologije. Različiti vizuelni prikazi mogu pružiti više pogleda na iste podatke, pa se stoga razlikuju po svojoj svrsi.

Jezici bazirani na *json* notaciji, kao što je i prethodno pomenuti *GraphQL* jezik za definisanje šeme, prednjače po pitanju čitljivosti u odnosu na *xml* bazirane notacije.

Međutim, to ne znači da bi implementacija neke vrste vizuelne reprezentacije za takve dokumente ne bi imala ne prednosti. Iako čitljiviji, ovakvi dokumenti zbog količine podataka lako mogu postati nepregledni.

Upravo u takvim situacijama dolazi do izražaja način prezentovanja podataka korisniku. Grafička predstava ovakvih jezika se uglavnom svodi na upotrebu grafova [1]. Vizuelna reprezentacija po kojoj bi korisnik imao jednostavniji grubi pregled sadržaja, kao i prikaz detalja po potrebi, mogla bi da ima značajan uticaj na efikasnost u radu. Posebno za rad korisnika koji nisu do detalja upoznati sa sintaksama sličnih jezika.

2. PREGLED SLIČNIH SISTEMA

Često je upravo sama definisana šema sve što je potrebno korisniku za uvid u način funkcionisanja servisa, što može biti jedan od razloga za postojanje malog broja ovakvih rešenja. Prilikom istraživanja ove teme pronađeno je nekoliko onih koja pružaju srodne mogućnosti i koja će biti opisana u narednim odeljcima. Uz to će u odeljku 2.4. posebno biti istaknuto rešenje koje je steklo veliku popularnost za dokumentovanje REST servisa i koje je bilo glavna inspiracija za ovaj rad.

2.2. GraphiQL

GraphiQL je interaktivno integrisano razvojno okruženje (eng. *IDE - Integrated Development Environment*) za *GraphQL* koje se pokreće u internet pretraživaču.

Prikaz je podeljen na četiri kolone, gde je dokumentacija API-a prikazana u poslednjoj koloni, sa funkcionalnošću pretraživanja, gde se za odabrane tipove mogu izlistati sva njihova polja [2].

Jedna od prednosti *GraphiQL*-a jeste puna podrška za *GraphQL SDL* kao i ispisivanje predloga u vidu dostupnih polja pri pisanju upita.

Mana ovakvog prikaza dokumentacije je slabija preglednost – kolona u kojoj se nalazi dokumentacija može se uvećati samo na jednu trećinu prozora pretraživača, uz to su sve informacije ispisane u jednom odeljku, što otežava pregled.

2.2. *GraphQL playground*

GraphQL playground takođe predstavlja jedno integrisano razvojno okruženje koje je izgrađeno na osnovama rešenja opisanog u prethodnom odeljku. Vizuelni prikaz čine tri kolone, od kojih poslednja prikazuje dokumentaciju [3].

Prikaz dokumentacije je znatno poboljšan u odnosu na *GraphiQL*, na raspolaganju su dve kartice: za prikaz šeme i za interaktivno istraživanje upita kao i tipova koji se koriste.

GraphQL playground svoj razvojni put neće nastavljati samostalno jer je u planu njegovo integrisanje sa ranije pomenutim *GraphiQL* razvojnim okruženjem.

2.3. *GraphQL Editor*

Za razliku od prethodno pomenutih rešenja, *GraphQL Editor* je najviše fokusiran na vizuelizaciju šeme. Moguće je istovremeno pisati šemu i posmatrati njenu vizuelizaciju u vidu povezanih blokova koji grafički predstavljaju elemente šeme. Podržano je i kreiranje šeme u suprotnom smeru, gde se dodavanjem i povezivanjem grafičkih elemenata generiše odgovarajući kod.

Jedan od nedostataka se uočava kod kompleksnijih šema, mapiranje elemenata iz tekstualnog dela na grafički prikaz se radi ručno, i za svaki element neophodno je tražiti njegovu reprezentaciju i obrnuto.

2.3. *Swagger (OpenAPI)*

Swagger predstavlja kolekciju alata za pomoć pri dizajnu, implementaciji, dokumentovanju i upotrebi REST API-a. Od posebnog značaja za temu ovog rada je *Swagger UI* koji pruža interaktivnu vizuelizaciju resursa web servisa bez implementacionih pojedinosti [5]. Jednostavan vizuelni prikaz čini ga veoma korisnim za različite tipove korisnika.

Operacije koje se mogu pozivati vizuelno su prikazane u vidu kartica, grupisanih prema entitetu na koje se odnose, a na kojima su ispisane osnovne informacije o pozivu. Karakteristična je i upotreba boja za određene vrste operacija što ih vizuelno razdvaja i olakšava pronalazak željene operacije.

3. TEHNOLOGIJE I ALATI

Za razvoj ovog rešenja korišten je *Visual Studio Code* editor, čije jezgro se zasniva na *JavaScript* okruženju, odnosno pokreće ga *Node.js* server.

3.1. TypeScript

Za implementaciju je korišten *TypeScript* programski jezik, strog nadskup *JavaScript*-a, koji dodaje jeziku opcionu statičku tipizaciju i objektnu orijentisanost. Dizajniran je za razvoj velikih aplikacija i prevodi se pre izvršavanja u *JavaScript*.

3.2. Node.js

Node.js je višeplatformsko *JavaScript* radno okruženje otvorenog koda za izvršavanje *JavaScript*-a na serverskoj strani. Iskorišćen je za pokretanje implementiranog programa, kao i za njegovo objavljivanje u vidu NPM (eng. *Node Package Manager*) paketa.

3.3. GraphQL Tools

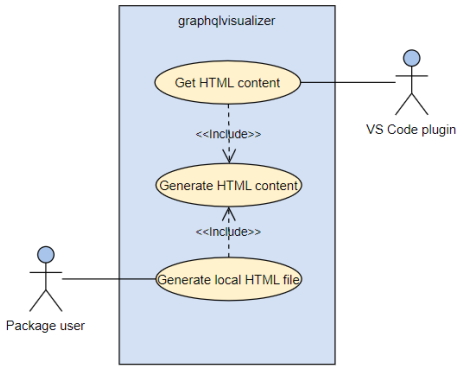
Alat koji predstavlja skup NPM paketa i omogućava lakše učitavanje šema iz raznih izvora, kao i izgradnju servisa definisanih šemom [6].

4. SPECIFIKACIJA

Analizom sličnih sistema, njihovih prednosti i nedostataka kao i uočenih potreba korisnika, definisana je specifikacija ovog rešenja čiji će funkcionalni i nefunkcionalni zahtevi biti obrađeni u nastavku.

4.1. Funkcionalni zahtevi

Osnovna funkcionalnost potrebna za realizovanje programa za vizuelizaciju .graphql šema prikazana je na dijagramu slučaja korištenja na slici 3.



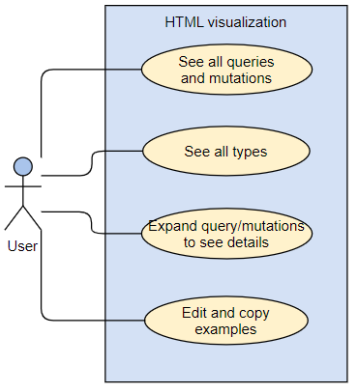
Slika 3. *Dijagram slučaja korišćenja graphqlvisualizer-a*

Rešenje treba da pruži dve osnovne funkcionalnosti, prva se odnosi na običnog korisnika, dok je druga implementirana specifično za ekstenziju *VS Code* editora.

Funkcionalnost koja se odnosi na običnog korisnika treba da omogući da se po potrebi može izvršiti generisanje lokalnog fajla koji sadrži vizuelni reprezentaciju šeme. Stoga, preduslov je da korisnik poseduje definisanu šemu napisanu prema gramatici jezika za definisanje šema. Šema se prosleđuje programu na obradu na dva načina, kao jedan string parametar, ili prosleđivanjem putanje do šeme sačuvane lokalno u okviru direktorijuma projekta koji koristi ovu funkcionalnost. Obrada šeme rezultuje generisanjem *HTML* fajla koji korisnik zatim može koristiti za pregled interfejsa.

Druga funkcionalnost, koja se tiče konkretne ekstenzije *VS Code* editor-a, odnosi se na generisanje delova sadržaja *HTML* stranice, koji se zatim integrišu u samoj ekstenziji. Preduslov za izvršavanje ove funkcionalnosti jeste prosleđivanje šeme u vidu stringa. Izvršavanje rezultuje generisanjem stila, zatim tela tj. komponenti samog prikaza, kao i skriptnog dela stranice koji sadrži *JavaScript* kod potreban za interaktivnost prikaza.

Interakcije koje postoje između korisnika i generisanog vizuelnog prikaza predstavljene su na dijagramu slučaja korišćenja na slici 4.



Slika 4. *Dijagram slučaja korišćenja vizuelnog prikaza*

Korisniku treba izlistati sve postojeće tipove, kao i upite i mutacije. Za svaki od njih, proširenjem prikaza, treba da se prikažu detalji, opis ili primeri. Za upite i mutacije za koje je generisan početni primer, treba da se omogući njihova izmena kao i lako i jednostavno kopiranje sadržaja.

4.2. Nefunkcionalni zahtevi

Pored osnovnih funkcionalnosti definisanih specifikacijom, potrebno je zadovoljiti i nefunkcionalne kriterijume.

Potrebno je sažeto predstaviti sadržaj cele šeme tako da ga korisnik može lako sagledati. Zatim treba da se omogući što lakše snalaženje među elementima, što njihovim raspoređivanjem i grupisanjem, što upotrebom boja za označavanje različitih tipova.

Uz to je neophodno obezbediti i interaktivnost prikaza, kroz proširenje detalja elemenata i njihovo sakrivanje ili uređivanje generisanih primera.

5. IMPLEMENTACIJA I DEMONSTRACIJA

U ovom poglavlju će biti ukratko opisana implementacija sistema specificiranog u prethodnom poglavlju.

Program je osmišljen kao NPM paket i objavljen je pod nazivom *graphqlvisualizer*. Kao takav može biti instaliran i korišten iz CLI-a (eng. *Command Line Interface*), gde je već instaliran *node*. Instalacija se vrši jednostavim pozivom `> npm i graphqlvisualizer.`

Rešenje je implementirano kao *Node.js* projekat u kombinaciji sa *TypeScript* programskim jezikom.

Rešenje se sastoji od pet komponenti: generator tela prikaza (eng. *body*), generator stila, generator skripti, generator *HTML* stranice i *Visualizer* komponenta koja izvozi funkcionalnosti dostupne krajnjem korisniku. Konkretne komponente će biti objašnjene u nastavku.

5.1. *Visualizer* komponenta

Ova komponenta korisnicima daje na raspolaganje tri metode, a to su: `GenerateHtmlFromString`, `GenerateHtmlFromLocalFile` i `Visualize`.

Od navedenih funkcionalnosti plugin implementiran za *VS Code* editor koristi metodu `Visualize` tako što joj prosleđuje šemu u vidu stringa. Preostale dve funkcionalnosti namenjene su korisnicima koji ne koriste ekstenziju editora, nego žele lokalno generisan vizuelni prikaz. `GenerateHtmlFromString` metodi se prosleđuje šema kao string parametar, dok postoji mogućnost učitavanja fajla šeme sačuvane lokalno u okviru projekta, i to upotrebnom metode `GenerateHtmlFromLocalFile`.

Pre prosleđivanja šeme na obradu, upotrebom *GraphQL Tools* alata, radi se učitavanje šeme iz konkretnog izvora, zavisno od metode koja se koristi. Transformisana šema se dalje prosleđuje generatoru *HTML* sadržaja.

5.2. Generator tela sadržaja, stila i skripti

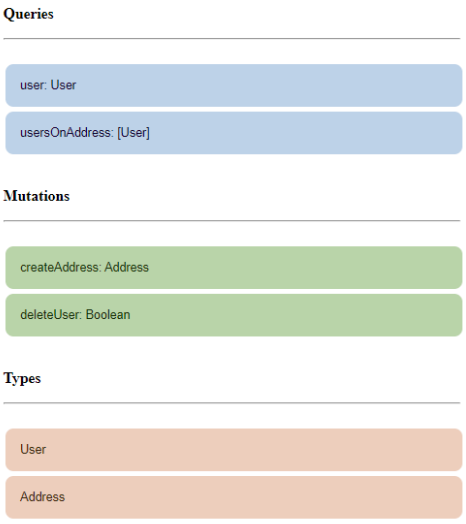
Generator *HTML* stranice kombinuje funkcionalnosti ovih komponenti sačinjavajući funkcionalnu stranicu.

Generator stila gradi sadržaj koju definiše izgled stranice, statički je i ne zavisi od prosleđene šeme.

Generator sadržaja izgrađuje telo stranice na osnovu prosleđene šeme. Prolaskom kroz njen sastav obrađuju se sadržani elementi i generišu prikazi. Za upite i mutacije ispisuje se njihov potpis. Proširenjem prikaza ispisuje se opis pročitan iz šeme, kao i primer poziva. Za korisnički definisane tipove i enumeracije generišu se elementi slični prethodno opisanim, s tim da se proširenjem prikaza ispisuje struktura tipa, njegova polja i tipovi polja, kao i informacija da li su polja obavezna.

Generator skripti omogućava dinamičko ponašanje stranice, i kroz to interakciju korisnika sa prikazom. Izgenerisan je *JavaScript* kod koji se izvršava u pretraživaču. Konkretno se odnosi na mogućnosti interakcije sa karticama koje predstavljaju tipove, proširenje i skrivanje detalja, podrška kopiranja primera klikom na dugme.

Program implementiran na način objašnjen u prethodnom poglavlju rezultuje prikazom na slici 5.



Slika 5. *Primer generisanog prikaza*

Nakon učitavanja generisanog prikaza na prvi pogled uočljiva je podela sadržaja na upite, mutacije i korisničke tipove. Na svakoj kartici se nalazi ili naziv upita i povratni tip ili naziv korisnički definisanog tipa u zavisnosti od toga na šta se kartica odnosi. Oni sačinjavaju osnovu strukture šeme. Za željenu karticu, klikom, otvara se širi prikaz detalja, kao sto je prikazano na slici 6.

Slika 6. *Prikaz detalja upita*

Za odabrani upit se ispisuje opis, ukoliko je definisan, a ispod njega primer poziva. Polje koje sadrži primer korisnik može uređivati i transformisati u oblik koji se zatim može izvršiti na serveru koji podržava datu šemu. Takođe je neophodno parametre date u obliku ‘naziv: tip’ izmeniti tako da se prosleđuje odgovarajuća vrednost. Obavezni parametri su naglašeni podebljanim slovima i ne bi smeli biti izostavljeni.

6. ZAKLJUČAK

Razmatranjem potreba korisnika koji često ulažu vreme u razumevanje i analizu *GraphQL* šema, kao i postojećih sistema koji im taj posao olakšavaju, razvijen je *graphqlvisualizer* koji predstavlja spoj dobrih strana postojećih rešenja i omogućava istraživanje API-a na jednostavan i intuitivan način. Rešenje koje je u te svrhe steklo najveću popularnost kada su u pitanju REST servisi jeste *Swagger UI* koji je imao veliki uticaj na konačni izgled ove vizuelizacije.

Fleksibilnost ovog rešenja se ogleda u dva načina upotrebe, kroz instalirani plugin za *VS Code* editor, kao i direktnom upotrebom NPM paketa gde je omogućeno učitavanje šeme iz stringa, ukoliko je pisana u samom kodu, ili iz posebnog fajla. Funkcionalnosti generisanog prikaza su svedene na jednostavan prikaz sadržaja šeme, što određuje i njegovu osnovnu namenu, a to je upoznavanje API-a na što lakši način.

Ovakva osnova pruža mnogo mogućnosti i za dalji razvoj i proširenja. Koristan dodatak rešenju bila bi i implementirana pretraga kao i filtriranje sadržaja što bi bilo izvodljivo generisanjem dodatnih skripti koje bi se izvršavale nad stranicom.

Nešto drugačiji pristup proširenju bi mogla biti i implementacija funkcionalnog klijenta, gde bi bilo moguće sa iste stranice poslati kreirani upit i prikazati njegov odgovor.

7. LITERATURA

[1] Frisendal, T. “*Visual Design of GraphQL Data*”, Apress, Berkeley, CA, 2018.

[2] https://www.gatsbyjs.com/docs/how-to/querying-data/running-queries-with-graphiql/ (pristupljeno u oktobru 2022.)

[3] https://github.com/graphql/graphql-playground (pristupljeno u oktobru 2022.)

[4] https://graphqleditor.com/features/visual/ (pristupljeno u oktobru 2022.)

[5] De, B. “*API Documentation. In: API Management*”, Apress, Berkeley, CA, 2017.

[6] https://www.the-guild.dev/graphql/tools/docs/introduction#! (pristupljeno u oktobru 2022.)

Kratka biografija:

Nikolina Šarenac rođena je u Zvorniku 1997. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Računarstvo i automatika odbranila je 2022.god.

kontakt: nina.sarenac@hotmail.com