Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/22BE23Vukmirovic

RAZVOJ VS CODE EKSTENZIJE ZA VIZUELIZACIJU I MANIPULACIJU GRAPHQL ŠEMOM

DEVELOPMENT OF VS CODE EXTENSION FOR VISUALIZATION AND MANIPULATION OF GRAPHQL SCHEMA

Tanja Vukmirović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu opisana je implementacija ekstenzija za Visual Studio Code koja treba da omogući dodavanje novih elemenata u GraphQL šemu, kao i vizualizaciju već postojeće šeme*

Ključne reči: *vscode ekstenzija, graphql, šema, vizuelizacija*

Abstract – *This paper describes implementation of an extension for Visual Studio Code that enables users to add new elements to their GraphQL schema, and to visualize already existing schemas*

Keywords: *vscode extension, graphql, schema, visualization*

1. UVOD

Naše današnje društvo svedok je eksplozije informacija, znanja i sve veće složenosti sistema u raznim domenima. Imati informacije „na dohvat ruke“ postalo je ključno pitanje. Da bi se obezbedilo da velike količine informacija budu lako dostupne korisnicima, one moraju biti strukturirane, a sama struktura poznata korisnicima.

Vizuelizacije struktura svojstvenih velikim količinama informacija mogu pomoći u razumevanju odnosa između informacionih elemenata i vizuelnom pretraživanju relevantnih informacija. Stoga, postoji potreba za tehnikama vizuelizacije kako bi strukture informacija u velikim repozitorijumima bile očigledne i od pomoći korisnicima u efikasnom pretraživanju i lociranju elemenata informacija relevantnih za zadatak s kojim se suočavaju [1].

Kako bi se uspostavila komunikacija između dva sistema potrebno je definisati API (Application Programming Inetrface). API opisuje način na koji dva servisa komuniciraju i koje podatke tom prilikom razmenjuju. GraphQL je upitni jezik sa rad sa API-jima.

Za specifikaciju podataka koji se razmenjuju prilikom komunikacije koristi se šema. U slučaju sistema gde kompleksnost raste, raste i kompleksnost šeme, te razumevanje šeme, kao i pretraga i izmena podataka postaju otežani.

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

2. PREGLED SLIČNIH SISTEMA

Kako bi se bolje razumeli i definisali funkcionalni zahtevi poželjno je imati uvid u već postojeća rešenja.

2.1. GraphiQL

GraphiQL [2] je razvojno okruženje integrisano u pretraživač. Podržava kompletnu GraphQL specifikaciju. Pruža isticanje grešaka u realnom vremenu, automatsko kompletiranje koda, a takođe čuva i istoriju pokrenutih upita.

2.2. GraphQL Playground

GraphQL Playground [3] zasnovan je na najboljim delovima GraphiQL-a, uz dodatak novih funkcionalnosti i poboljšanje postojećih, kao što je bolji prikaz istorije pokrenutih upita.

2.3. GraphQL Editor

GraphQL Editor [4] se fokusira na vizuelizaciji šeme. Korisnici su u mogućnosti da definišu šemu i pregledaju blokove dok pišu ili mogu da kreiraju šemu pomoću blokova, a potom da na osnovu njih generišu kod.

2.4. GraphQL Voyager

GraphQL Voyager [5] je alat koji se koristi za vizuelizaciju šeme. Neke od osnovnih funkcionalnosti su brzo pozicioniranje na graf, kao i detaljne informacije o svakom tipu.

2.5. OpenAPI (Swagger) Editor

OpenAPI (Swagger) Editor [6] je VS Code ekstenzija koja daje podršku OpenAPI specifikaciji u JSON i YAML format. Neke od funkcionalnosti koje uključuje su pregled SwaggerUI-a, IntelliSense, navigacija koda, statička sigurnosna analiza i druge.

3. KORIŠĆENE SOFTVERSKE TEHNOLOGIJE

3.1. Visual Studio Code, JavaScript/TypeScript, Node.js

Visual Studio Code editor dolazi sa ugrađenom podrškom za JavaScript, TypeScript i Node.js, kao i bogatim ekosistemom ekstenzija. Za implementaciju rešenja koristio se TypeScript programski jezik koji je tipizirani superset JavaScript-a i koji se kompajlira u čist JavaScript. Node.js je platforma za lako pokretanje brzih, skalabilnih I laganih aplikacija. Može se smatrati serverskim JavaScript-om. Za vizuelizaciju šeme, korišćen je npm paket *graphqlvisualizer*. Ovaj alat omogućuje vizuelizaciju postojećih GraphQL šema. Nudi

dve opcije – vizuelizaciju šeme iz lokalnih fajlova ili vizuelizaciju šeme koja se prosleđuje u tekstualnom formatu.

3.2. Visual Studio Code ekstenzija

Visual Studio Code kreiran je sa idejom da bude proširivo radno okruženje. Od korisničkog interfejsa do funkcija za obradu koda, skoro svaki deo Visual Studio Code-a može biti prilagođen i poboljšan korišćenjem Extension API-ja. Postoje različiti tipovi ekstenzija koje se mogu kreirati. Da bi se kreirala ekstenzija potrebno je razumeti sledeća tri koncepta:

- **Aktivacioni događaji** [7], događaji na koje se ekstenzija aktivira
- **Tačke doprinosa** [8], funkcionalnosti koje se proširuju
- **VS Code API** [9], skup JavaScript API-ja koji mogu da se pozivaju iz koda ekstentije

Svaka ekstenzija treba da sadrži package.json fajl koji predstavlja manifest ekstenzije u kom su definisani aktivacioni događaji i tačke doprinosa, kao i extension.ts koji je ulazni fajl ekstenzije i koji definiše dve funkcije. Funkcija activate() se poziva kada se desi aktivacioni događaj koji je specificiran, a druga nije obavezna i poziva se kad se ekstenzija deaktivira.

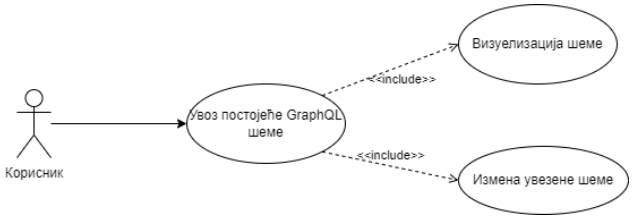
4. SPECIFIKACIJA

4.1. Specifikacija zahteva

U ovom odeljku specificirani su funkcionalni i nefunkcionalni zahtevi.

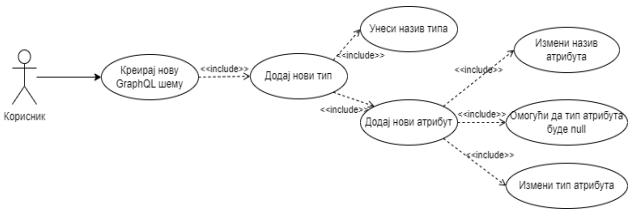
4.1.1. Specifikacija funkcionalnih zahteva

Nakon što se ekstenzija aktivira, korisnik pred sobom ima dve opcije, da uveze već postojeću GraphQL šemu ili da kreira novu šemu ispočetka. Ukoliko se korisnik odluči da uveze postojeću šemu, potrebno je da izabere fajl koji ima ekstenziju .gpl ili .graphql, kako su to jedini podržani formati. Nakon što korisnik uveze šemu, može ručno da pravi promene nad šemom. Ako se desi da korisnik svojim promenama dovede šemu u nevalidno stanje, nakon iniciranja pregleda šeme biće prikazana poruka o grešci. Ukoliko je šema validna, klikom na dugme za pregled šeme prikazaće se vizuelizovana šema. Ovaj scenario prikaza je na Slika 1.



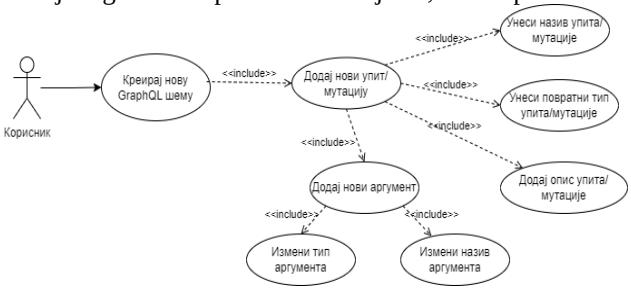
Slika 1. Dijagram slučaja korišćenja postojeće šeme

Korisnik, takođe, može da se odluči da kreira šemu ispočetka i u tom slučaju otvara se prazan tekstualni dokument. Korisnik ima opciju da doda novi tip, enumeraciju, upit ili mutaciju. Na Slika 2. prikazan je dijagram slučaja korišćenja kada korisnik odabere opciju dodavanja novog tipa. Tada korisnik može da menja naziv tog tipa, kao i da dodaje nove atribute, gde takođe može da menja naziv i tip atributa, kao i da specificira da li se dozvoljava da vrednost atributa može biti null.



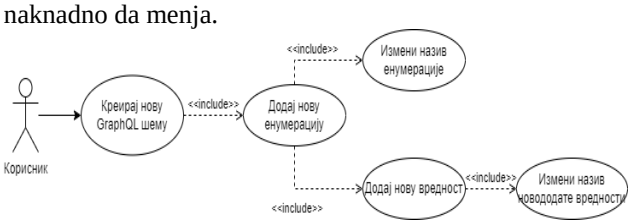
Slika 2. Dijagram slučaja korišćenja kada je odabrano dodavanje novog tipa

Korisnik može da odabere da doda upit ili mutaciju. Ove dve opcije pružaju iste mogućnosti pa su prikazane u sklopu jednog dijagrama slučaja korišćenja prikazanog na Slika 3. Nakon što korisnik izabere da doda novi upit ili mutaciju, može da promeni naziv upita ili mutacije, kao i povratni tip. Ono što korisnik još može da uradi jeste da dodaje argumente upitima i mutacijama, kao i opise.



Slika 3. Dijagram slučaja korišćenja kada je odabrano dodavanje upita ili mutacije

Preostala opcija, koja se pruža korisniku jeste i dodavanje enumeracije, prikazano na Slika 4. Nakon što korisnik doda enumeraciju, može da izmeni naziv enumeracije ili da doda novu vrednost, čiju vrednost takođe može naknadno da menja.



Slika 4. Dijagram slučaja korišćenja kada je odabrano dodavanje enumeracije

4.1.2. Specifikacija nefunkcionalnih zahteva

Upotrebljivost je mera kojom se često iskazuje koliko su korisnički interfejsi laki za korišćenje.
Reč „upotrebljivost“ se, takođe, odnosi i na metode za poboljšanje lakoće korišćenja tokom procesa dizajniranja. Upotrebljivost je definisana sa pet komponenti kvaliteta:

- **Mogućnost učenja –** koliko je korisnicima lako da ostvare osnovne zadatke kada prvi put naiđu na dizajn
- **Efikasnost –** koliko korisnici brzo mogu da izvršavaju zadatke kada nauće dizajn
- **Pamtljivost –** koliko je lako ostvariti efikasnost kada se vratite dizajnu nakon određenog vremena
- **Greške –** koliko često korisnici mogu da naprave greške
- **Zadovoljstvo –** koliko je ugodno koristiti dizajn

Postoji još bitnih atributa kvaliteta, a onaj koje se ističe je korisnost, koji se odnosi na to da li određena funkcionalnost radi ono što korisniku treba. Upotrebljivost i korisnost ne idu jedno bez drugoga. Može se desiti da postoji funkcionalnost koja je jednostavna za korišćenje, ali nije ono što je korisniku potrebno. Isto tako, nije dobro ako sistem radi radi ono što je korisniku potrebno, ali korisnik ne može da to efikasno iskoristi jer je korisnički interfejs prekomplikovan.

4.2. Specifikacija sistema

U ovom poglavlju opisani su model podataka i struktura rešenja.

4.2.1. Model podataka

Kako bi se podržao rad sa osnovnim konceptima GraphQL-a, definisane su klase koje odgovaraju konceptima koji su podržani u sklopu ekstenzije. Koncepti koji su podržani su tipovi, enumeracije, upiti i mutacije.

Bazični koncept GraphQL šeme je objektni tip koji se u ekstenziji modeluje *TypeNode* klasom. Može da sadrži više polja koja se modeluju klasom *PropertyNode*. Upiti su modelovani klasom *QueryNode*. Svaki upit može da ima više argumenata, koji su takođe modelovani *PropertyNode* klasom. Isti slučaj je i sa mutacijama koje se modeluju *MutationNode* klasom. Mutacijama i upitima je moguće dodavati opise, a to je modelovano poljem description u klasama *QueryNode* i *MutationNode*. Poslednji koncept koji je modelovan jeste enumeracija. Enumeracije su modelovane EnumerationNode klasom.

4.2.2. Struktura rešenja

Kako se radi o VS Code ekstenziji, struktura rešenja je jednostavna. Sastoji se iz *TypseScript* fajlova (modula) koji sadrže funkcije koje implemntiraju određenu poslovnu logiku i koji su organizovani u direktorijume prema celini koju zajedno čine.

5. IMPLEMENTACIJA

5.1. Aktivacija ekstenzije

Ulazna tačka u ekstenziju jeste extension.ts fajl, u kome je definisana funkcija *activate()* koja se aktivira kada se desi neki od aktivacionih događaja. Da bi se aktivacioni događaj desio, potrebno ga je prvo definisati u manifestu ekstenzije u *activationEvents* polju. Zadatak funkcije *activate()* jeste da registruje komande i inicijalizuje događaje koji mogu da se dese prilikom obrađivanja komandi.

U ovoj funkciji se definišu i obrađivači događaja

- *onDidChangeActiveTextEditor* i

- *onDidChangeTextDocument*.

Pored aktivacionih događaja u manifestu ekstenzije potrebno je definisati i komande koje su dostupne iz korisničkog interfejsa. Komande se definišu u objektu contibutes.commands. Osnovno prilikom definicije komande jeste navesti identifikator komande, pošto se on kasnije koristi za registraciju komande.

5.2. Korisnički interfejs

Korisnički interfejs sastoji se iz četiri osnovna dela, a to su traka aktivnosti, kontejner pogleda, editor i aktivnoti editora. Traka sa aktivnostima služi za navigaciju u okviru VS Code-a.

Kako bi se dodala nova stavka u traku sa aktivnostima potrebno ju je definisati u manifestu ekstenzije u okviru objekta *contributes.viewsContainer.activitybar*. Na ovaj način se stavka aktivnosti povezuje sa kontejnerom pogleda. Kontejner pogleda je roditeljski kontejner u koji je moguće dodati poglede. Pogledi mogu biti tipa *TreeView*, *WelcomeView* i *Webview*.

WelcomeView je inicijalni pogled i definiše se u sklopu objekta *contributes.viewsWelcome*, a sadrži dve komande, za kreiranje nove šeme i uvoz već postojeće.

Kako bi se kreirao *TreeView* potrebno je u objekat *contributes.views* dodati objekat koji predstavlja listu čvorova u stablu. Naziv ovog objekta treba da odgovara identifikatoru nekog od objekata iz liste *contributes.viewsContainer.activitybar*. Kako bi se prikazali podaci u sklopu ovih pogleda, za svaki pogled potrebno je definisati klasu koja će implementiratiri klasu *TreeDataProvider* i tom prilikom implementirati sve metode: *getChildren* i *getTreeView*. Klasa *getChildren* treba da vrati listu elemenata u slučaju korenskog čvora ili listu pripadajućih elemenata ukoliko se prosledi čvor. Tako, na primer, ukoliko smo pozicionirani u pogledu *Types*, trebali bi dobiti listu svih tipova ili listu svih atributa ako prosledimo određeni tip. Metoda *getTreeItem* treba da vrati UI reprezentaciju čvora, odnosno objekat klase koja nasleđuje TreeItem.

5.3. Manipulacija šemom

Manipulacija šemom omogućava izmene tipova, enumeracija, upita, mutacija i drugih elemenata. Kako bi se ovo uspešno ostvarilo, potrebno je komande koje su definsane u manifestu ekstenzije definisati i nad objektom *contributes.menus* manifesta ekstenzije. Ukoliko se radi o komandama koje se izvršavaju nad pogledom koji je definisan u objektu *contributes.views*, komandu je potrebno definisati u objektu *contributes.menus.view/title* i potrebno je dodati atribut koji govori kada komanda treba da se prikaže. Ukoliko je komandu potrebno dodati nekom od elemenata koji pripadaju definisanim pogledima, onda se to radi u objektu *contributes.menus.view/item/context*, gde je takođe potrebno navesti uslov pod kojim se prikazuje komanda, a koji treba proveriti da li se radi o željenom pogledu i da li kontekstna vrednost odgovara željenoj.

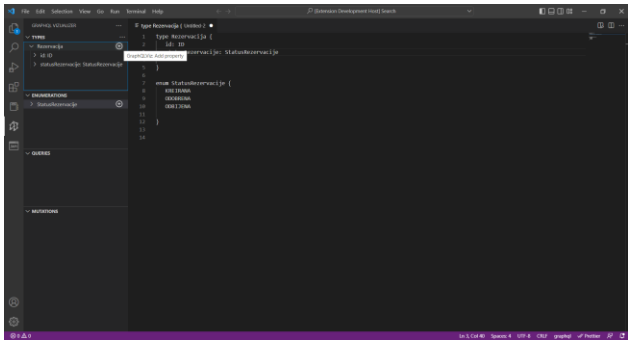
Nakon što se komande dodaju u manifest ekstenzije, za svaku komandu potrebno je obezbediti da postoji odgovarajući obrađivać te komande.

5.4. Vizuelizacija šeme

Za prikaz vizuelizacije šeme koristi se klasa *SchemaPreviewPanel* čija odgovrnost je da kreira novi webview panel za prikaz šeme ili da ažurira postojeći, kao i da prilagodi HTML sadržaj koji je potrebno prikazati u panelu. Kako bi se dobio sadržaj koji je potrebno prikazati u panelu koristi se funkcija *Visualize*, *graphqlvisualizer* npm paketa. Ova funkcija kao parametar prima šemu u tekstualnom formatu, a kao rezultat vraća objekat koji sadrži sam html sadržaj koji je potrebno prikazati, stilove koje je potrebno primeniti na taj sadržaj kao i skripte koje omogućavaju manipulaciju sadržajem.

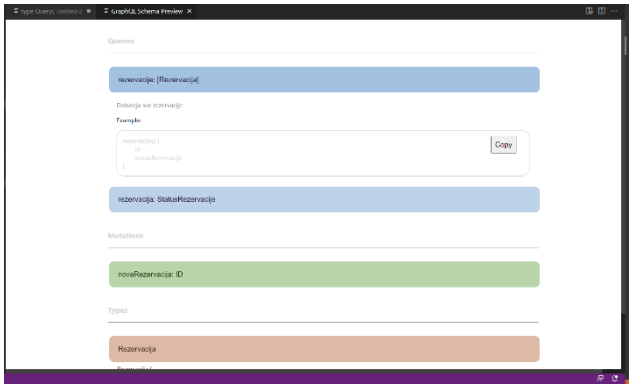
6. DEMONSTRACIJA

U ovom poglavlju biće demonstriran scenario prikazan na Slika 2. Nakon što je korisnik odabrao da kreira šemu ispočetka, može da odabere opciju da doda novi tip, enumeraciju, upit ili mutaciju. U nastavku fokus će biti na dodavanju novog tipa. Da bi korisnik dodao novi tip potrebno je da klikom na „tri tačkice“ u sklopu pogleda *Types* odabere opciju dodavanja novog tipa. Tada se dodaje novi tip sa predefinisanim nazivom, koji je moguće izmeni kao na slici ispod. Nakon što doda novi tip korisnik može da doda i nove atribute, klikom na plus pored naziva odgovarajućeg tipa kome želi da doda novi atribut. Pored ovoga, korisnik dodatom atributu može da postavi da tip bude ili ne bude null vrednost, što čini klikom na olovkicu pored naziva atributa.



Slika 5. Izgled korisničkog interfejsa nakon što se doda tip i enumeracija

Kada korisnik odluči da želi da pregleda šemu, može to učiniti klikom na komandu u editorskom meniju. Tada se korisniku prikazuje panel prikazan na Slika 6.



Slika 6. Prikaz vizuelizovane šeme

7. ZAKLJUČAK

Predmet ovog rada bilo je istraživanje postojećih rešenja kada je u pitanju vizuelizacija GraphQL šema i implementacija novog rešenja koje će omogućiti kako vizuelizaciju šeme, tako i manipulaciju istom u cilju lakšeg rukovanja šemom kada sistemi koji se modeluju naglo porastu, a rukovanje podacima postane otežano.

Ono što ovo rešenje čini različitim od ostalih rešenja koja se bave vizuelizacijom GraphQL šema jeste što se šema vizuelizuje na način sličan SwaggerUI-u koji je poznat kao jedan od popularnijih alata za vizuelizaciju i dokumentaciju OpenAPI specifikacije, čineći je jednostavnom za čitanje i inuitivnom.

Kako je implementacija tek započeta, mesta za napredak ima mnogo. Korisno bi bilo omogućiti povezivanje sa serverom i na taj način omogućiti izvršavanje testnih upita i mutacija i prikaz vrednosti koje se dobijaju kao rezultat izvršavanja API poziva.

4. LITERATURA

[1] Keller, T., Tergan, SO. 2005. Visualizing Knowledge and Information: An Introduction.

[2] GraphiQL https://github.com/graphql/graphiql

[3] GraphQL Playground https://github.com/graphql/graphql-playground

[4] GraphQL Editor https://github.com/graphql-editor/graphql-editor

[5] GraphQL Voyager https://github.com/IvanGoncharov/graphql-voyager

[6] OpenApi (Swagger) Editor https://marketplace.visualstudio.com/items?itemName=42Crunch.vscode-openapi

[7] Aktivacioni događaji https://code.visualstudio.com/api/references/activation-events

[8] Tačke doprinosa https://code.visualstudio.com/api/references/contribution-points

[9] VS Code API https://code.visualstudio.com/api/references/vscode-api

Kratka biografija:

Tanja Vukmirović rođena je 07.09.1997. u Novom Sadu. Srednju školu završila je u Novom Sadu, nakon čega je upisala Fakultet tehničkih nauka u Novom Sadu, studijski program Primenjeno softversko inženjerstvo. Diplomirala je 2020. godine, a iste godine je upisala i master studije. Položila je sve ispite predviđene planom i programom.