Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/05BE39Vojnovic

JEZIK ZA GENERISANJE API TESTOVA ZA GRAPHQL UPITE

A LANGUAGE FOR GENERATING API TESTS FOR GRAPHQL QUERIES

Milorad Vojnović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu opisan je dizajn i implementacija jezika specifičnog za domen, koji se koristi za opis i generisanje programskog koda namenjenih za API testove za GraphQL upite.*

Ključne reči: *GraphQL, textX, DSL, WEB testiranje*

Abstract – *This paper presents design and implementation of a domain-specific language that is used for a description and a generation of GraphQL API tests.*

Keywords: *GraphQL, textX, DSL, WEB testing*

1. UVOD

Ovaj rad obuhvata dizajn i implementaciju jezika specifičnog za domen, koji se koristi za opis i generisanje programskog koda namenjenih za API testove za *GraphQL* upite.

1.1 Jezici specifični za domen

Jezici specifični za domen - JSD (*Domain specific language* - DSL) su računarski jezici ograničene ekspresivnosti, koji su fokusirani na određeni domen [1]. Oni su optimizovani za određenu klasu problema, koji se naziva domen [2]. Dobro dizajniran JSD može da zameni veliku količinu koda, koja je potencijalno i komplikovana za razumevanje. Takođe, ako se u jeziku koriste termini iz domena, to omogućava mnogo lakšu komunikaciju sa domenskim ekspertima.

Jedan od načina klasifikacije JSD-a je po tome kako su implementirani. JSD možemo da klasifikujemo kao interni ili eksterni, u zavisnosti od toga da li su implementirani preko nekog jezika domaćina ili ne [1].

Interni JSD je jezik koji koristi infrastrukturu postojećeg programskog jezika. Jedan on najpopularnijih internih jezika je *Rails*, koji je implementiran preko *Ruby* programskog jezika. U najviše slučajeva, interni JSD je implementiran kao biblioteka preko postojećeg jezika domaćina.

Eksterni JSD je jezik koji se kreira iz početka i za koji je potrebno definisati posebnu infrastrukturu za parsiranje, interpretiranje, kompajliranje i generisanje koda [3]. Razvijanje eksternog JSD-a je slično implementiranju novog jezika.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Igor Dejanović, vanredni prof.

Jezici specifični za domen su namenjeni za rešavanje problema iz određenog domena, dok sa druge strane imamo jezike opšte namene - JON (*General purpose language* - GPL), koji se mogu koristiti za bilo koji domen primene [3]. Osnovne razlike između jezika specifičnih za domen i jezika opšte namene mogu se videti u tabeli 1.1 [2].

Tabela 1.1 *Razlike između JSD i JON*

	JON	**JSD**
Domen	Velik i kompleksan	Mali i dobro definisan
Veličina jezika	Veliki	Mali
Korisnički definisane apstrakcije	Sofisticirane	Ograničene
Tjuring kompletan	Uvek	Uglavnom ne
Životni vek	Dekade	Godine
Dizajniran od	Guru ili komitet	Par inženjera i domenski ekspert
Korisnička podrška	Velika, anonimna i rasprostranjena	Mala, pristupna i lokalna
Evolucija	Spora, često standardizovana	Brzim tempom
Nevalidne promene (deprecation)	Gotovo nemoguće	Vrlo moguće

Postoji nekoliko razloga zašto koristiti JSD. Jedan od razloga je povećanje produktivnosti programera [1]. JSD-ovi su pogodni za lako razumevanje domena i zato se brže pišu, menjaju i postoji manja verovatnoća za greškom. Drugi razlog je poboljšanje komunikacije između programera i klijenata. Ovo je jako teško postići, ali je benefit veoma velik. Upravo komunikacija između programera i klijenata predstavlja jedan od najvećih problema u razvoju softvera i zbog toga JSD-ovi imaju veliki značaj, jer potencijalno mogu da reše jedan od najvećih problema u razvoju softvera [1].

1.2 GraphQL

GraphQL je upitni jezik za API. Takođe je i radno izvršavanje (*runtime*) za popunjavanje upita konkretnim podacima [4]. On pruža kompletan i razumljiv opis podataka, koje je moguće dobiti sa servera i time pruža *web* klijentima mogućnost da traže minimalni skup potrebnih podataka.

Uz pomoć *GraphQL* upita je moguće dobiti samo one podatke koji su potrebni. Omogućeno je kreiranje ugnježdenih upita, koji omogućava da se povezuju podaci i da se dobavljaju podaci koje bi morali dobaviti iz nekoliko HTTP zahteva. Osim toga, moguće je definisati

minimalni skup potrebnih podataka, tako da se ne dobija više od onog što je potrebno. Sa *GraphQL*-om je moguće dobiti sve potrebno podatke u jednom zahtevu, ako je šema dobro definisana. Svaki put kada se izvrši upit, on je validiran od strane sistema tipova. Svaki *GraphQL* servis definiše sistem tipova u *GraphQL* šemi preko koje se svi upiti validiraju [4].

Jezik koji se koristi za *GraphQL* zahteve je direktno povezan sa jezikom koji se koristi za odgovore [5]. Ako se analizira JSON odgovor, može se videti da je to rečnik ključeva i vrednosti. Vrednosti dalje mogu biti ugnježdene i sadržati nove parove ključ – vrednost. Ako se posmatraju samo vrednosti iz odgovora, može se videti da su to podaci koji su traženi preko *GraphQL* upita. Na listingu 1.1 se mogu videti primer *GraphQL* upita i JSON odgovora koji prikazuje sličnost zahteva i odgovora [6].

Zahtev	Odgovor

<pre>{
  hero {
    name
    height
    mass
  }
}</pre>
	<pre>{
	  "hero": {
	  "name": "Luke Skywalker",
	  "height": 1.72,
	  "mass": 77
	  }
	}</pre>

Listing 1.1 *Primeri GraphQL zahteva i JSON odgovora*

GraphQL menja način dizajniranja API-a [4]. Umesto da se API posmatra kao kolekcija pristupnih tačaka, posmatra se kao kolekcija tipova. Kolekcija tipova se naziva šema. Potrebno je pronaći zajednički jezik preko kog će se opisati šema. Da bi to bilo moguće, *GraphQL* dolazi sa jezikom koji služi za opisivanje šema koji se naziva *Schema Definition Language* – SDL [4]. U *GraphQL* dokumentima se definišu tipovi koji se koriste u šemi i koji će biti dostupni u aplikaciji i koji će biti korišćeni u komunikaciji između klijenta i servera. Tipovi predstavljaju gradivne element *GraphQL* šeme.

1.3 WEB testiranje

U svetu testiranja opšte je poznat pojam piramida testiranja [7]. Na vrhu piramide testiranja se nalaze testovi korisničkog interfejsa (UI-E2E). Ovi testovi prolaze kroz čitavu aplikaciju i ponašaju se kao da korisnik vrši neke akcije u sistemu. Osim ovih testova postoje i integracioni (*integration*) testovi. Oni su poput UI testova, ali oni ne testiraju korisnički interfejs. Umesto toga oni kreću od jednog sloja niže i testiraju servise. Na dnu piramide su jedinični (*unit*) testovi. Oni su mali, precizni i na nivou koda. Oni testiraju malu jedinicu, najčešće klasu i daju precizan izveštaj u tome šta nije zadovoljilo kriterijume testiranja.

Kada se pogleda da ima nekoliko vrsta testova postavlja se pitanje koje testove izabrati u kojoj situaciji. Za ovaj odabir se najčešće koristi pravilo palca (*rules of thumb*) [7]:

- Favorizovati jedinične u odnosu na ostale testove
- Pokriti nedostatke jediničnih testova sa integracionim testovima
- Koristiti UI testove štedljivo

UI i integracioni testovi su za testiranje povezanosti komponenti sistema. Ti testovi su sporiji, jer prolaze kroz više slojeva aplikacije. Sa druge strane jedinični testovi su brzi i daju ranu povratnu informaciju. Oni se koriste najčešće tokom razvoja kako bi što pre dobili informaciju o radu neke komponente. Jedinični testovi se koriste kako bi aplikacija brzo iterirala dok se ostale dve vrste koriste da bi se proverilo da li sve funkcioniše. Obe vrste imaju bitnu namenu.

2. SPECIFIKACIJA SISTEMA

2.1 Pregled stanja u oblasti

Karate je alat otvorenog koda (*open-source*) koji kombinuje automatizaciju testova, kreiranje lažnih objekata (*mocking*) i testove performansi u jedan unificirani okvir [8]. Uz pomoć *Karate*-a je moguće pisati jednostavne testove u čitljivoj sintaksi, koja je pažljivo dizajnirana za HTTP, JSON, *GraphQL* i XML. Osim ovoga, moguće je generisanje odgovarajućih izveštaja o uspešnosti izvršavanja testova. Jedan primer testa napisanog preko *Karate* jezika definisan je u listingu 2.1 [8].

```
Scenario Outline:
  * text query =
    """
    {
      hero(name: "<name>") {
        height
        mass
      }
    }
    """
  Given path 'graphql'
  And request { query: '#(query)' }
  And header Accept = 'application/json'
  When method post
  Then status 200

  Examples:
    | name  |
    | John  |
    | Smith |
```

Listing 2.1 *Test za GraphQL API napisan preko Karate jezika*

Karate je odličan alat koji omogućava testiranje *GraphQL* API-a. JSD preko kog se definišu test slučajevi ima mnoštvo mogućnosti. Jedna od mana jezika jeste to što ovo nije jezik samo za *GraphQL* API, pa zbog toga u samom jeziku ima elemenata koji nisu potrebni.

Just-API je deklarativni okvir za testiranje REST, *GraphQL* ili nekih drugih HTTP baziranih servisa [9]. Moguće je testiranje API-a bez pisanja programskog koda. Test specifikacije se čitaju iz YAML fajlova i mogu da se izvršavaju sekvencijalno ili u paraleli. Takođe je moguće generisanje izveštaja o uspešnosti izvršavanja testova u nekoliko formata uključujući HTML i JSON.

Primer test slučaja za *GraphQL* API napisan preko *Just-API* jezika prikazan je u listingu 2.2 [9]:

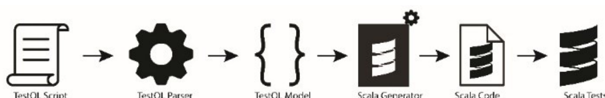
```
meta:
  name: GraphQL location service
configuration:
  host: api.graphloc.com
  scheme: https
specs:
  - name: Get Location of a given ip address
    request:
      method: post
      path: /graphql
      headers:
        - name: content-type
          value: application/json
      payload:
        body:
          type: json
          content:
            query: >
                   {
                    getLoc(ip: "8.8.8.8") {
                      country {
                        iso_code
                      }
                    }
                   }
            variables: null
            operationName: null
    response:
      status_code: 200
      json_data:
        -path: $.data.getLocation
         value: US
```

Listing 2.2 *Test za GraphQL API napisan preko Just-API jezika*

Just-API je sličan alat kao i *Karate*. Ima slične mogućnosti i najveća razlika jeste to što imaju drugačiji JSD za opis testova. *Karate* koristi specifičnu sintaksu dok se u *Just-API-u* testovi definišu preko YAML fajlova. S obzirom da se mogu testirati različite vrste API-a, ovaj alat takođe ima manu, jer nije dovoljno specifičan za *GraphQL* API i da zbog toga ima nepotrebnih stvari u jeziku.

2.2 Arhitektura sistema

TestQL jezik je implementiran kao eksterni JSD. Najpre se napiše TestQL skripta, koja se prosleđuje TestQL parseru. Parser je osnovna komponenta sistema koja na osnovu definisane gramatike parsira skripte koje opisuju test slučajeve. Vrši parsiranje fajlova i proverava da li su programi napisani u skladu sa gramatikom jezika i vrši semantičke provere. Ako je skripta napisana u skladu sa sintaksnim i semantičkim pravilima jezika, kao rezultat se dobila model test slučajeva kao objektna struktura, koja se prosleđuje generatoru test slučajeva. Generator na osnovu datih modela generiše kod za ciljnu platformu, koja je u ovom slučaju *Play* okvir u *Scala* programskom jeziku. Na kraju se *Scala* kod kompajlira i dobijaju se izvršivi testovi. Arhitektura datog sistema je prikazana na slici 2.1.



Slika 2.1 *Arhitektura celog TestQL sistema*

3. IMPLEMENTACIJA SISTEMA

Gramatika TestQL jezika je opisana u *TextX* [10] meta-jeziku za opisivanje jezika specifičnih za domen. Na osnovu opisa meta-jezika je izgenerisan parser za TestQL jezik. Generator u okviru TestQL alata je implementiran u programskom jeziku *Python*, dok je za generisanje koda *GraphQL* testova korišćem šablon-generator *Jinja2*.

3.1 Struktura TestQL skripte

Koreni element u TestQL gramatici je *Test* i predstavlja osnovnu strukturu opisa test slučajeva. *Test* element sadrži *Package*, *Class*, *Scenario*, *Request* i jedan ili više *Case* elemenata. *Package* predstavlja naziv paketa u kom će test slučaj biti generisan. *Class* predstavlja naziv klase gde će test slučaj biti generisan. *Scenario* predstavlja opis funkcionalnosti tj. zahteva koji se testira. *Request* predstavlja definiciju *GraphQL* zahteva koji se treba testirati. *RequestType* definiše tip operacije koji će biti izvršen, i može biti “query“ (upit) ili “mutation“ (mutacija). *Attribute* definiše strukturu koja se očekuje od servera kao odgovor. *Argument* predstavlja argumente operacije, koji se koriste za slanje podataka na server i filtriranje. *ArgumentValue* predstavlja vrednost argumenta i može biti *SimpleValue*, *CompoundValue* ili *TagValue* element. *SimpleValue* predstavlja jednostavnu vrednost i može biti “STRING“, “INT“, “FLOAT“ ili “BOOL“. *CompoundValue* predstavlja složenu vrednost. *TagValue* element predstavlja vrednost koja će biti određena na osnovu vrednosti iz *Examples* elementa. *Case* element predstavlja test slučajeve koji će se testirati. *Response* predstavlja odgovor servera. *JsonObject* predstavlja *JSON* odgovor servera. *JsonMember* predstavlja atribute u *JSON* objektu. *JsonValueWithTags* predstavlja vrednosti *JSON* objekta zajedno sa tag vrednostima i može biti *JsonValue* ili *TagValue*.. *JsonValue* predstavlja vrednost u *JSON* objektu i može biti *JsonObject*, *JsonArray*, *JsonNull* ili *SimpleValue*. *JsonArray* predstavlja *JSON* niz. *Multiplier* predstavlja element koji umnožava vrednosti u nizu. *MultiplierValue* predstavlja broj za koliko puta će biti umnožen element u nizu i može biti “INT“ ili *TagValue* element. *JsonNull* predstavlja *null* vrednost u *JSON* objektu. *Examples* predstavlja vrednosti test slučajeva, koji će biti ubačeni umesto *TagValue* vrednosti tokom generisanja. *ExampleAttributes* predstavlja nazive tagova mesto kojih će vrednosti biti ubačene tokom generisanja. *ExampleAttribute* predstavlja konkretan naziv taga. *Example* element predstavlja konkretne vrednosti koje će biti ubačene umesto tagova tokom generisanja. *RowSeparator* predstavlja granicu između zaglavlja tabele i konkretnih vrednosti tabele. *JsonValueWithEmptyValue* predstavlja vrednosti koje će biti ubačene u tagove tokom generisanja i može biti *JsonValue* ili ’*’.

Pošto su definisani svi elementi gramatike moguće je definisati kompletne primere u TestQL gramatici tj. *Test* element koji je koreni element gramatike. U listingu 3.1 je prikazan primer *Test* elementa uz pomoć TestQL gramatike, koji opisuje test slučaj za dobavljanje kategorija.

```
package:graphql
class:CategoriesSpec
scenario:retrieving categories

request:
```graphql
query findAllCategories {
    categories {
        id
        name
    }
}
```

response:
```json
{
    "data": {
        "categories": [
            [2]{
                "id": <id>,
                "name": <name>
            }
        ]
    }
}
```

examples:
should	id	name
"retrieve all categories"	1	"RAM memory"
*	2	"SSD disk"
```

Listing 3.1 *TestQL primer Test elementa*

4. ZAKLJUČAK

Razvijeni sistem omogućava generisanje test slučajeva za *GraphQL* API. Generišu se testovi za *Play* okvir za programski jezika *Scala* na osnovu specifikacije definisane u jeziku specifičnom za domen. Uz pomoć jezika je moguće brzo i jednostavno definisati test slučajeve. Domen jezika je ograničen na generisanje test slučajeva za *GraphQL* API, što omogućava da delovi jezika budu isključivo iz tog domena što pruža prednost u odnosu na druge alate, koji su nastali kao proširenje za neki drugi tip API-a.

Moguća primena ovog alata je za pisanje jediničnih ili integracionih testova za *GraphQL* API. TestQL alat je namenjen pre svega za programere. Programeri mogu da ga iskoriste i da brže napišu testove za *GraphQL* API. Domen TestQL alata je testiranje *GraphQL* API-a i upravo to predstavlja najveću prednost ovog alata, jer je domen druga dva testiranje API-a i samim tim njihova konciznost je lošija u odnosu na TestQL.

5. LITERATURA

[1] M. Fowler, Domain-Specific Languages, Addison-Wesley Professional, 2010.

[2] M. Voelter, DSL Engineering: Designing, Implementing and Using Domain-Specific Languages, CreateSpace Independent Publishing Platform, 2013.

[3] D. Ghosh, DSLs in action, Manning Publications Co., 2011.

[4] A. B. Eve Porcello, Learning GraphQL: Declarative Data Fetching for Modern Web Apps, O'Reilly Media, 2018.

[5] S. Buna, Learning GraphQL and Relay, Packt Publishing, 2016.

[6] „GraphQL," Dostupno na mreži: https://graphql.org/. [Poslednji pristup 12. 5. 2019.].

[7] J. Rasmusson, The Way of the Web Tester: A Beginner's Guide to Automating Tests, Pragmatic Bookshelf, 2016.

[8] „Karate," Dostupno na mreži: https://intuit.github.io/karate/. [Poslednji pristup 1. 6. 2019.].

[9] „Just-API," Dostupno na mreži: https://kiranz.github.io/just-api/. [Poslednji pristup 1. 6. 2019.].

[10] I. Dejanović, „TextX," Dostupno na mreži: http://textx.github.io/textX. [Poslednji pristup 12. 5. 2019.].

Kratka biografija:

Milorad Vojnović rođen je u Novom Sadu 1994. godine. Master rad na Fakultetu tehničkih nauka u Novom Sadu, iz oblasti Elektrotehnike i računarstva – softversko inženjerstvo, odbranio je 2019. godine.