Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Anže Valher

# Aplikacijski programski vmesnik s poizvedbenim jezikom GraphQL

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2019

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Uporaba ustreznih arhitekturnih slogov lahko olajša razvoj in vzdrževanje specifične programske opreme, ki na ta način postane bolj kakovostna in uporabna. Vendar uporaba teh principov ni zastonj in zahteva čas za vpeljavo v proces razvoja, še zlasti pa je treba paziti na ustrezno izbiro podpornih knjižnic in orodij. V diplomski nalogi celovito predstavite sodoben pristop za razvoj aplikacijskih programskih vmesnikov z uporabo poizvedbenega jezika GraphQL. V ta namen kratko orišite značilnosti samega jezika GraphQL, težišče naloge pa naj bo na njegovi uporabi za razvoj dejanskega aplikacijskega programskega vmesnika za preprosto socialno omrežje. Spletna rešitev naj obsega zaledni del razvit v jeziku PHP in ogrodju Symphony ter začelni del, ki je razvit z uporabo JavaScript-a in ogrodja AngularJS. Okolje za izvajanje naj zagotavlja vsebovalnik Docker z ustreznim spletnim strežnikom. V zaključnem delu ovrednotite opisani pristop.

*Zahvaljujem se mentorju viš. pred. dr. Igorju Rožancu za pomoč pri izdelavi diplomske naloge. Zahvalil bi se družini in prijateljem za izkazano podporo med izdelavo diplomske naloge, kot tudi tekom študija.*

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **DOM** | Document Object Model | objektni model dokumenta |
| **HTML** | Hyper Text Markup Language | jezik za označevanje nadbesedila |
| **HTTP** | Hypertext Transfer Protocol | protokol za prenos podatkov preko spleta |
| **IMAP** | Internet Message Access Protocol | protokol dostopa do internetnih sporočil |
| **JSON** | JavaScript Object Notation | format označevanja javascript objektov |
| **MVC** | Model-View-Controller | model-pogled-krmilnik |
| **MVVM** | Model-View-ViewModel | model-pogled-pogled model |
| **PHP** | Hypertext Preprocessor | skriptni jezik za izdelavo spletnih strani |
| **POP3** | Post Office Protocol version 3 | protokol za prejemanje pošte verzija 3 |
| **REST** | REpresentational State Transfer | predstavitveni prenos stanja |
| **RPC** | Remote Procedure Call | oddaljeni klic podprograma |
| **SMTP** | Simple Mail Transfer Protocol | preprost protokol za prenos e-pošte |
| **SOAP** | Simple Object Access Protocol | protokol za izmenjavo podatkov med spletnimi storitvami |
| **SQL** | Structured Query Language | strukturirani poizvedovalni jezik |
| **TCP** | Transmission Control Protocol | protokol interneta |
| **UDP** | User Datagram Protocol | protokol za uporabniška sporočila |
| **XML** | eXtensible Markup Language | razširljiv označevalni jezik |

# Povzetek

**Naslov:** Aplikacijski programski vmesnik s poizvedbenim jezikom GraphQL

**Avtor:** Anže Valher

V diplomski nalogi je predstavljen poizvedbeni jezik GraphQL za izdelavo aplikacijskih programskih vmesnikov, ki temelji na preverbi podatkov in vgrajeni dokumentaciji za predstavitev aplikacij. Poizvedbeni jezik in njegova implementacija sta podrobneje predstavljeni, ter uporabljeni za razvoj konkretnega aplikacijskega vmesnika.

Zgrajena je spletna stran socialnega omrežja, ki omogoča dodajanje prijateljev, pogovor z njimi in deljenje objav. V spletno stran se je mogoče prijaviti na dva načina: z e-pošto ali uporabo Google prijave. Sestavlja jo zaledna in začelna aplikacija, ki ju povezuje poizvedbeni jezik. Zaledna aplikacija nam z uporabo GraphQL sheme omogoča prikaz dokumentacije in preverja podatke ob izvajanju.

Okolje za izvajanje GraphQL strežnika zagotavlja vsebovalnik Docker s spletnim strežnikom Nginx, podatke pa shranimo v podatkovni bazi MySQL. Zaledna aplikacija je razvita s skriptnim jezikom PHP in ogrodjem Symfony, začelna pa s programskim jezikom JavaScript in ogrodjem AngularJS. Z uporabo poizvedbenega jezika želimo doseči preglednost in višjo kvaliteto aplikacij z zmožnostmi, ki jih jezik ponuja.

**Ključne besede:** aplikacijski programski vmesnik, začelna in zaledna aplikacija, poizvedbeni jezik GraphQL, OpenID Connect.

# Abstract

**Title:** Application programming interface with query language GraphQL

**Author:** Anže Valher

The purpose of this thesis is to present query language GraphQL which is used to create application programming interfaces. It is based on data validation and built-in documentation for applications presentation. Query language and implementation are presented alongside all requirements for running GraphQL server, and a web solution is presented to demonstrate its efficiency.

Social network web application is developed, which makes possible to add friends, chat with them and share articles. Login to the web application is possible by using e-mail or Google login. Web application is built with backend and frontend applications, which are linked with query language GraphQL. Backend application with GraphQL scheme allows generation of documentation and data validation.

Docker container with web server Nginx is used to create environment for running GraphQL server. Data is stored inside database MySQL. Backend application will be developed using PHP script language and Symfony framework. Frontend application is developed with programming language JavaScript and AngularJS framework. By using query language GraphQL we want to achieve transparency and quality of applications using capabilites provided by query language.

**Keywords:** application programming interface, frontend and backend ap-

plication, query language GraphQL, OpenID Connect.

# Poglavje 1

# Uvod

Tehnologije in orodja za razvoj spletnih strani se nenehno izboljšujejo in spreminjajo. Priporočljivo je slediti novim standardom, protokolom in arhitekturnim slogom ter jih po najboljših močeh vpeljati v svoje aplikacije. Pri tem pa je potrebno izluščiti primerna orodja in knjižnice za našo aplikacijo. Ob vsem tem pa moramo biti pozorni na čas, ki je potreben za vpeljavo v razvijalski proces in razvoj aplikacije.

Z izbiro programskega jezika, ogrodja, orodij, knjižnic in standardov za sledenje si omejimo možnosti pri nadaljnjem razvoju. S pravo izbiro si precej poenostavimo tako razvoj, kot dolgoročno vzdrževanje. Z uporabo pravih tehnologij in razvojnih priporočil si zagotovimo kvalitetne aplikacije, ki so pregledne pri razvoju in uporabi.

Arhitekturni slogi nam olajšajo izdelavo in vzdrževanje aplikacijskih programskih vmesnikov. S strukturami določajo programske elemente, povezave med njimi in lastnosti le-teh. Z arhitekturnim slogom se zavežemo k uporabi strukturnih odločitev, ki jih težko spreminjamo po vpeljavi [4].

Diplomska naloga predstavlja enega izmed možnih arhitekturnih slogov za izdelavo zaledne aplikacije. V zgodovini so se pojavili številni protokoli, standardi in arhitekturni slogi za izdelavo aplikacijskih programskih vmesnikov. Med najbolj razširjenimi sta protokol SOAP (ang. Simple Object Access Protocol) [32] in arhitekturni slog REST (ang. REpresentational

State Transfer) [30], vendar so se tekom časa pokazale pomankljivosti, tako pri razvoju kot pri izvajanju aplikacij.

Leta 2012 je podjetje Facebook pripravilo specifikacijo za izdelavo aplikacijskega programskega vmesnika s poizvedbenim jezikom, ki se imenuje GraphQL [34]. V tej diplomski nalogi bo predstavljena izdelava aplikacijskega programskega vmesnika ter poizvedbeni jezik.

Za izvajanje aplikacijskega programskega vmesnika je bil vzpostavljen Docker vsebovalnik [28] z vsemi potrebnimi servisi in odvisnostmi. Vsebuje podatkovno bazo, spletni strežnik in orodja za izvajanje aplikacij. Ker zaledna aplikacija ne prinese dodane vrednosti za uporabnika, je bila razvita tudi začelna aplikacija, ki omogoča grafični prikaz delovanja celotnega sistema.

Izdelava zalednega aplikacijskega programskega vmesnika bo v skriptnem jeziku PHP z ogrodjem Symfony [21], začelna aplikacija pa v JavaScript [23] programskem jeziku z ogrodjem AngularJS [12].

Pri vsaki spletni aplikaciji, ki za delovanje zahteva uporabnike, je potrebno zagotoviti prijavo in registracijo le-teh. Najbolj preprosta in pogosteje uporabljena metoda je registracija uporabnika z e-pošto in geslom. Kot razširitev tega je bil razvit OpenID Connect [39] nad protokolom OAuth 2.0 [35]. Omogoča nam prijavo na spletne strani z uporabo zunanjih ponudnikov. Kot ena izmed možnosti prijave je bil vpeljan tudi v to diplomsko nalogo.

Spletna stran socialnega omrežja bo omogočala dodajanje prijateljev, pogovor z njimi, pošiljanje vsebine in urejanje profila. Spletna stran zaokrožuje celostni razvoj GraphQL strežnika in priprave okolja za izvajanje. Strežnik implementira uporabo poizvedbenega jezika v zaledni in začelni aplikaciji. Zaledna aplikacija bo imela zgrajeno GraphQL shemo, ki jo začelna uporablja za pridobivanje podatkov. Z nanašanjem na shemo bomo prikazali dokumentacijo in preverjali verodostojnost podatkov ob izvajanju.

# Poglavje 2

# Aplikacijski programski vmesnik

Aplikacijski programski vmesnik je programska koda, ki omogoča komunikacijo med dvema programoma. Vsebuje funkcije poimenovane z besedo, ki so predstavljene v dokumentaciji.

Aplikacijski programski vmesniki so zgrajeni iz dveh elementov. Prvi predpisuje na kakšen način se prenašajo informacije med dvema programoma, po principu pošiljanja zahteve in vračanja odgovora. Drugi element sledi tem zahtevam pri pridobivanju podatkov, ki jih prikaže v obliki spletne strani [2].

Sledenje protokolom, standardom in arhitekturnim slogom pri izgradnji vmesnikov je zelo zaželjeno, a ni nujno potrebno. Z njihovo uporabo ohranjamo konsistentnost in preglednost naših aplikacij. Dogovorjena pravila pripomorejo k izboljšanju razvijalskega procesa. S tem zagotovimo, da so vsi razvijalci seznanjeni z njenim delovanjem in s programsko kodo, katero so dolžni pisati v določeni obliki in zagotavljati njena pravila delovanja. Pomemben vidik je tudi kakovost dokumentacije aplikacije, saj nam to izjemno pomaga pri vpeljavi novega razvijalca na projekt in olajša njeno uporabo. Za izdelavo aplikacijskih programskih vmesnikov so najbolj razširjeni protokoli SOAP [32], XML-RPC [29], JSON-RPC [33] in arhitekturni slog REST [30].

## 2.1 Protokol SOAP

SOAP predlaga definicije informacij v XML formatu, ki se lahko uporabljajo za izmenjavanje strukturiranih (tipičnih) informacij med uporabniki v decentraliziranih, porazdeljenih okoljih. SOAP sporočilo je uradno specificirano kot `XML Information Set` [31], ki nam ponuja abstrakten opis vsebine [32]. Slika 2.1 prikazuje primer takega sporočila.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
     <t:transaction
        xmlns:t="http://thirdparty.example.org/transaction"
          env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true">5</t:transaction>
 </env:Header>
 <env:Body>
     <m:chargeReservationResponse
         env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
             xmlns:m="http://travelcompany.example.org/">
       <m:code>FT35ZBQ</m:code>
       <m:viewAt>
         http://travelcompany.example.org/reservations?code=FT35ZBQ
       </m:viewAt>
     </m:chargeReservationResponse>
 </env:Body>
</env:Envelope>
```

Slika 2.1: XML sporočilo.

## 2.2 Protokol XML-RPC

XML-RPC je protokol oddaljenega klica podprograma (ang. Remote Procedure Call), ki deluje preko internetne povezave. XML-RPC sporočilo je HTTP POST [37] zahteva. Vsebina zahteve je definirana v XML formatu. Procedura se izvede na strežniku in vsebina v odgovoru je podana v XML formatu [29]. Slika 2.2 prikazuje zahtevo z XML-RPC sporočilom in njene HTTP [38] spremenljivke.

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181


<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
         </param>
      </params>
   </methodCall>
```

Slika 2.2: XML-RPC zahteva.

## 2.3 Protokol JSON-RPC

JSON-RPC je protokol, ki deluje na principu oddaljenega klica podprograma na način brez stanja (ang. stateless) in je preprost za uporabo. Specifikacija določa več podatkovnih struktur in pravil za njihovo procesiranje. Komunikacija ni določena, kar pomeni, da se lahko ta koncept uporablja znotraj istega procesa preko vtičnic, protokola HTTP [38] ali drugih različnih sporočilnih možnosti. Za komunikacijo uporablja JSON format [33]. Slika 2.3 prikazuje zahtevo in odgovor v JSON formatu.

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"subtrahend": 23, "minuend": 42}, "id": 3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}

--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend": 23}, "id": 4}
<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
```

Slika 2.3: JSON-RPC zahtevi in odgovora.

## 2.4 Arhitekturni slog REST

Arhitekturni slog je sestavljen iz omejitev, ki so uporabljene na elementih znotraj arhitekture. S preizkušanjem vpliva dodanih omejitev v razvijajoč

slog lahko ugotovimo lastnosti, ki so potrebne za spletne omejitve. Dodatne omejitve so potem lahko vpeljane z gradnjo novega arhitekturnega sloga, ki bolje odraža lastnosti moderne spletne arhitekture.
Omejitve arhitekturnega sloga REST so naslednje:

- ničen slog (ang. null style)
- ločitev odjemalca in strežnika (ang. client-server)
- komunikacija brez stanja (ang. stateless)
- predpomnenje (ang. caching)
- večplastni sistem (ang. layered system)
- enoten vmesnik (ang. uniform interface)

Arhitekturni slog REST je abstrakcija arhitekturnih elementov znotraj porazdeljenega nadpredstavnostnega sistema (ang. distributed hypermedia system). Določa komponente, povezave med njimi in interpretacijo podatkovnih elementov. Pri tem se posveča implementaciji komponent in sintaksi protokola. Vsebuje temeljne omejitve na komponentah, priključkih in podatkih, ki so navedeni v osnovah spletne arhitekture [30].

# Poglavje 3

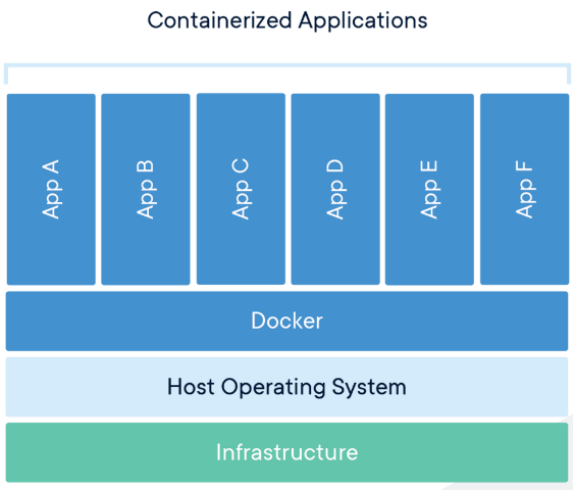
# Orodja in tehnologije

V tem poglavju bodo opisane tehnologije in orodja, ki so uporabljene za izdelavo in izvajanje GraphQL strežnika. Ker je poudarek diplomske naloge na izdelavi in uporabi poizvedbenega jezika, bodo ostala orodja in tehnologije predstavljene le na kratko za lažje razumevanje celostnih potreb pri razvoju aplikacijskega vmesnika. Spletna stran se bo izvajala znotraj vsebovalnika, kar omogoča izvajanje v različnih razvojnih okoljih.

## 3.1 Vsebovalnik Docker

Vsebovalnik je standardna enota programskega paketa, ki vsebuje potrebno programsko kodo in odvisnosti, da se lahko aplikacija izvaja hitro in zanesljivo v različnih računalniških okoljih. Shema Docker vsebovalnik, je preprosta in vsebuje vse potrebno za izvajanje aplikacije: programsko kodo, izvajalnik kode, sistemska orodja, sistemske knjižnice in nastavitve [17].

Slika 3.1 prikazuje izvajanje Docker aplikacij na obstoječi infrastrukturi. Na operacijskem sistemu se izvaja orodje Docker, ki izvaja naše vsebovalnike z aplikacijami.




Slika 3.1: Prikaz delovanja Docker orodja.

## 3.2 Spletni strežnik Nginx

Nginx je odprtokodni programski paket za izvajanje spletnih aplikacij. Vsebuje povratni posrednik (ang. reverse proxying), HTTP predpomnilnik (ang. caching), izenačevanje obremenitve (ang. load balancing) in večpredstavni tok (ang. media streaming). Narejen je bil kot spletni strežnik za učinkovitost in stabilnost. Poleg HTTP strežniških sposobnosti ga lahko uporabljamo kot posredniški strežnik za elektronsko pošto (IMAP, POP3 in SMTP). Uporabljamo pa ga lahko tudi kot povratni posrednik in za izenačevanje obremenitve za HTTP, TCP in UDP protokole [25].

## 3.3 Podatkovna baza MySQL

MySQL je sistem za upravljanje s podatkovnimi bazami. Je odprtokodna implementacija relacijske podatkovne baze, ki za delo s podatki uporablja jezik SQL. Deluje na več različnih operacijskih sistemih, Windows, Linux in še mnogih drugih. Obstajajo programski vmesniki za C, C++, Eiffel, Java, Perl, PHP, Python, Ruby in Tcl [22].

## 3.4 Skriptni jezik PHP

PHP (ang. PHP Hypertext Preprocessor) je razširjen odprtokodni skriptni jezik, ki je namenjen spletnemu razvoju in se lahko vgradi v HTML. Njegova sintaksa izhaja iz programskih jezikov C, Java in Perl, ter je preprost. Primarni cilj jezika je omogočiti spletnim razvijalcem hitro pisanje dinamično generiranih spletnih strani [24].

### 3.4.1 Ogrodje Symfony

Symfony je PHP ogrodje za izdelavo spletnih strani in spletnih aplikacij. Izdelano je nad Symfony komponentami. Komponente so med seboj neodvisne in pripravljene za večkratno uporabo kot PHP knjižnice. Lahko jih uporabljamo v aplikaciji brez uporabe Symfony ogrodja [21]. Ogrodje je bilo uporabljeno za izdelavo zaledne aplikacije.

### 3.4.2 Upravljalnik odvisnosti Composer

Composer je orodje za upravljanje odvisnosti v skriptnem jeziku PHP. Z njim določimo knjižnice, ki so potrebne za učinkovito izvedbo programske rešitve. Namestijo se v za to določen imenik in so aplikaciji dostopne preko imenskega prostora (ang. namespace). Upravljalnik nam omogoča posodabljanje knjižnic z uporabo verzij [27]. Na sliki 3.2 je prikazana preprosta `composer.json` datoteka, ki namesti knjižnico Monolog [14].

```
{
    "name": "acme/hello-world",
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```

Slika 3.2: `Composer.json` datoteka.

### 3.4.3 Doctrine

Knjižnica Doctrine je namenjena objektno-relacijski preslikavi za skriptni jezik PHP in nam omogoča transparentno shranjevanje PHP objektov. Uporablja vzorec preslikave podatkov (ang. object relational mapping) in si prizadeva za popolno ločevanje poslovne programske logike od shranjevanja podatkov v relacijski podatkovni bazi [16].

## 3.5 Ogrodje AngularJS

AngularJS je odprtokodno JavaScript ogrodje za izdelavo spletnih aplikacij na strani odjemalca. Vzdržuje ga Google s skupino posameznikov in podjetij, s katerimi se srečujejo z izzivi pri izdelavi dinamičnih spletnih strani. JavaScript komponente dopolnjujejo ogrodje Apache Cordova [1]. Uporablja se za izdelavo mobilnih aplikacij, ki se izvajajo na različnih računalniških okoljih. Stremi k poenostavitvi razvoja in testiranja aplikacij z ogrodjem, ki je sestavljen z MVC in MVVM arhitekturo, kot tudi z že uveljavljenimi komponentami.

V prvem koraku AngularJS ogrodje prebere HTML stran, katera vsebuje oznake, ki so namenjene ogrodju. Angular prepozna te oznake kot ukaze in jih poveže z modelom, ki je sestavljen iz standardnih JavaScript spremenljivk. Vrednost spremenljivk se lahko ročno nastavi v kodi ali pa se prebere iz statičnih ali dinamičnih JSON virov.

AngularJS temelji na prepričanju, da se deklarativno programiranje uporablja za izdelavo uporabniškega vmesnika, medtem ko je ukazno programiranje boljše za izdelavo aplikacij s poslovno logiko. Ogrodje razširja HTML za prikaz dinamične vsebine. Povezuje modele ogrodja in poglede s katerim omogoča avtomatsko posodabljanje podatkov [15].
Cilji načrtovanja ogrodja so:

- ločevanje DOM manipulacij in aplikacijske logike
- ločevanje aplikacij za odjemalca od strežniških aplikacij

- zagotavlja strukturo za izdelavo aplikacij

## 3.6 Avtentikacija z OpenID Connect

`OpenID Connect` je plast narejena nad OAuth 2.0 protokolom [35]. Odjemalcu omogoča prepoznavo identitete uporabnika, ki jo sporoči avtentikacijski strežnik. Pridobi tudi osnovne informacije uporabnika v preprosti obliki. Odjemalcem vseh tipov (vključno s spletnimi stranmi, mobilnimi aplikacijami in JavaScript odjemalcem) omogoča pridobivanje informacij o obstoječih sejah in uporabnikih [39].

Za diplomsko nalogo je uporabljen Google kot zaupanja vreden ponudnik. Spletna stran pošlje zahtevo na aplikacijski programski vmesnik, ki je končna točka za `OpenID Connect` zahtevo. Poleg zahteve je potrebno še določiti, katere uporabnikove podatke želimo. Po potrditvi identifikacije uporabnika nam končna točka pošlje ID žeton v katerem so shranjeni uporabnikovi podatki in je prikazan na sliki 3.3.


```
eyJhbGciOiJSUzI1NiIsImtpZCI6IjcyOGY0MDE2NjUyMDc5YjllZDk5ODYxYmIwOWJh
ZmM1YTQ1YmFhODYiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJhY2NvdW50cy5nb29nbGUuY
29tIiwiYXpwIjoiODQ1Mzc2MjkyNTExLWdzaXYwOTE4aThwczBuZTBoOGp1Y2pqc3FlO
HI3bWtyLmFwcHMuZ29vZ2xldXNlcmNvbnRlbnQuY29tIiwiYXVkIjoiODQ1Mzc2MjkyN
TExLWdzaXYwOTE4aThwczBuZTBoOGp1Y2pqc3FlOHI3bWtyLmFwcHMuZ29vZ2xldXNlc
mNvbnRlbnQuY29tIiwic3ViIjoiMTEyMzE3MTI4NTc1MTk3OTIzNzExIiwiZW1haWwiO
iJhbnplLnZhbGhlckBnbWFpbC5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiYXRfa
GFzaCI6IkhwN0Q3VFJRNEdnWm8wNzlWWUtVMWciLCJuYW1lIjoiQW7FvmUgVmFsaGVyI
iwicGljdHVyZSI6Imh0dHBzOi8vbGg0Lmdvb2dsZXVzZXJjb250ZW50LmNvbS8tZktrQ
nVOdzZkaEUvQUFBQUFBQUFBQUkvQUFBQUFBQUFCZkkvYU9mZUo2T04xWEEvczk2LWMvc
GhvdG8uanBnIiwiZ2l2ZW5fbmFtZSI6IkFuxb5lIiwiZmFtaWx5X25hbWUiOiJWYWxoZ
XIiLCJsb2NhbGUiOiJlbiIsImlhdCI6MTUzOTg4NjYzNCwiZXhwIjoxNTM5ODkwMjM0L
CJqdGkiOiIzMTU2NTUwZmQ3NWU3MTc3N2E0N2E3OTMxYzk1MjFkODJkNjIyMmFmIn0.E
zuvi2FH5rptwWUCh1DtMJa3HsNcQdhYItDpK1McWn6EZlsCylO3XbaLsD_No_Tqk0EC8
FsiMEEw-LtU6Rju38xoso5jDKFdDJAs_icI6ax639h4dgbaw_cC9yIKqJ4ShsfT928M-
ErR3LpFoyj-aTlvUDb91Bm1v0V0zG6pWgA2D4dMOmPoNzc3bKyXAdSCBne6-6jTG1ci0
ETpYtSEbvVrFpS2JY-NpMqJplXMfbeRBb0XhUSbKIaDhTRHP2vka5eqTaN1uAq9vtlNU
-KBHngCgFWxOsMAEnd7JAKY3vEgmdZnqt8A3XldvqZ3pci7ZD_fOlNIQVtYt_i5hGGnRA
```


Slika 3.3: ID žeton.

ID žeton je JSON Web Token [11], ki vsebuje podatke uporabnika. Vrsta podatkov je odvisna od podane zahteve za le te. Čeprav podatki pripadajo uporabniku, so ti dostavljeni le v primeru, ko uporabnik lahko preveri svoj

podpis. ID žeton je sestavljen iz treh delov: glava, vsebina in podpis [10].

- Glava vsebuje tip žetona in uporablja razpršilni algoritem.
- Vsebina vsebuje zahtevane uporabnikove podatke.
- Podpis je namenjen prejemniku, ki preveri pošiljatelja žetona in zagotovi, da je sporočilo ostalo nespremenjeno.


```
{
    "iss": "accounts.google.com",
    "azp": "84920327483-apps.googleusercontent.com",
    "aud": "84920327483-apps.googleusercontent.com",
    "sub": "18260523045439272129",
    "email": "anze.valher@diploma",
    "email_verified": "true",
    "at_hash": "Hp32SDEFRCf23oV4ED",
    "name": "Anže Valher",
    "picture": "https://lh4.googleusercontent.com/cphoto.jpg",
    "given_name": "Anže",
    "family_name": "Valher"
    "locale": "en",
    "iat": "153988633",
    "exp": "153988633",
    "jti": "dffeb9db8464c049144885dda4a933ac",
    "alg": "RS256",
    "kid": "a0611c9f8286c489176389a8ce51cd84",
    "typ": "JWT"
}
```


Slika 3.4: Odkriptiran ID žeton.

Na sliki 3.4 je prikazan odkriptiran ID žeton in podatki, ki jih vsebuje. Podatki kot so e-poštni naslov, ime, priimek in slika bodo shranjeni in uporabljeni za prikaz v socialnem omrežju, ter tudi za prijavo.

# Poglavje 4

# Poizvedbeni jezik GraphQL

GraphQL je poizvedbeni jezik, ki je načrtovan za gradnjo aplikacij za odjemalca. Ponuja nam intuitivno, prilagodljivo sintakso in sistem za opisovanje podatkovnih zahtev in medsebojnega sodelovanja. Poizvedbeni jezik GraphQL ni splošno namenski programski jezik, ampak je uporabljen le za poizvedbe na aplikacijskem strežniku. V GraphQL specifikaciji so določene vse možnosti, ki nam jih poizvedbeni jezik ponuja. Za implementacijo ne zahteva določenega programskega jezika ali podatkovne baze. Namesto tega aplikacijski strežnik s svojimi sposobnostmi preslika podatke v enoten jezik, tip sistema in filozofijo, ki jo GraphQL zahteva. To zagotavlja enoten, prijazen vmesnik za razvoj produkta in platformo za gradnjo orodij.
Principi poizvedbenega jezika GraphQL so:

- hierarhičnost (ang. hierarchical)
- osredotočenost na izdelek (ang. product-centric)
- strogo tipiziranje (ang. strong-typing)
- odjemalec opisuje poizvedbe (ang. client-specified queries)
- izvajanje po pravilih vgrajene sheme (ang. introspective)

Zaradi teh principov nam GraphQL omogoča produktivno okolje za gradnjo aplikacij odjemalcev [34].

Na slikah 4.1 in 4.2 je prikazan primer GraphQL poizvedbe v obliki povpraševanja in odgovora. Poizvedba se nanaša na pridobivanje imena uporabnika z identifikacijsko številko. Odgovor je v JSON formatu in vsebuje ime zahtevanega vira, ime elementa, ter vrednost elementa.

*Example № 3*

```
{
  user(id: 4) {
    name
  }
}
```

Slika 4.1: GraphQL poizvedba.

*Example № 4*

```
{
  "user": {
    "name": "Mark Zuckerberg"
  }
}
```

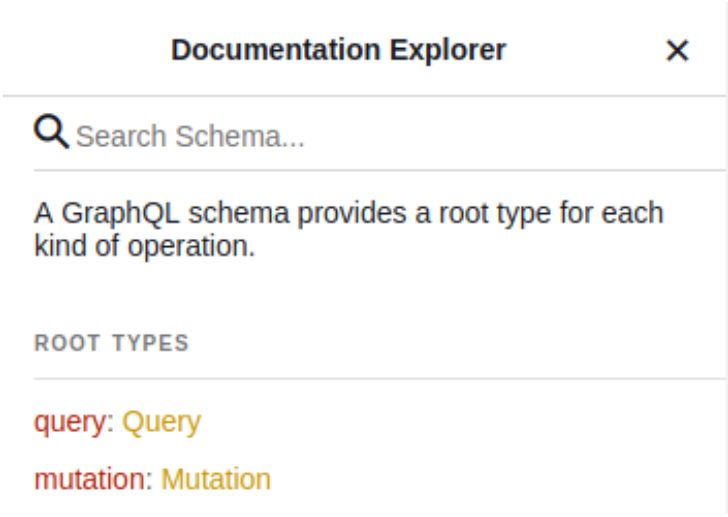
Slika 4.2: GraphQL odgovor.

GraphQL strežnik največkrat uporablja komunikacijski protokol HTTP [38]. Za razliko od arhitekturnega sloga REST poteka GraphQL komunikacija preko ene končne točke (`/graphql`). Za komunikacijo se uporabljata metodi HTTP GET [36] in HTTP POST [37], format pa je JSON [9]. Poizvedbeni jezik že v zasnovi ponuja dokumentacijo in strogo tipiziranje. S strogim tipiziranjem zagotavlja, da so vrnjeni podatki pravega tipa. Če se vrnjeni podatki ne ujemajo, se vrne napaka s primernim sporočilom.

## 4.1 Dokumentacija

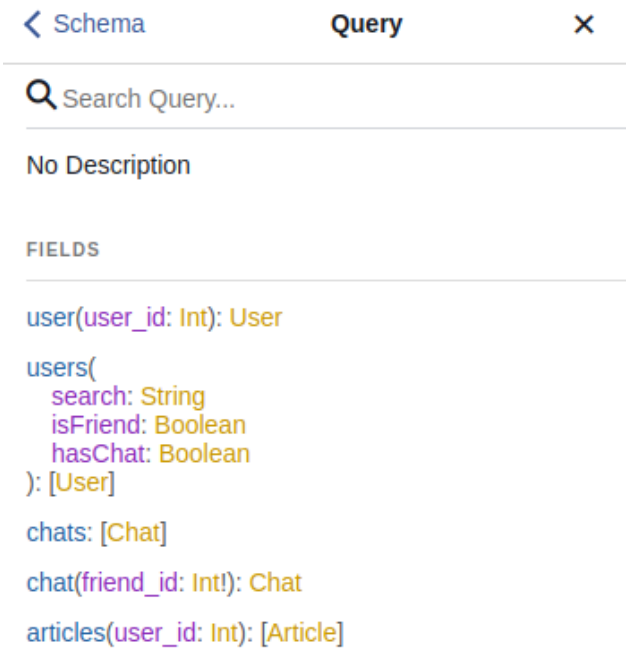
Podpora za dokumentiranje je največja prednost tipiziranega sistema. Za zagotavljanje usklajenosti dokumentacije z zmožnostmi GraphQL aplikacije so lastnosti definicij določene v shemi in dostopne preko introspekcije. Opisi uporabljajo sintakso Markdown [13].

GraphQL tipi, polja, argumenti in ostali elementi, naj bi vsebovali podroben opis za lažje razumevanje [7]. Pregled nad zmožnostmi GraphQL strežnika imamo z dokumentacijo. Na sliki 4.3 je prikazana začetna stran dokumentacije in osnovni operaciji, ki jih strežnik omogoča. To sta poizvedba in mutacija za pridobivanje in spreminjanje podatkov.




Slika 4.3: Dokumentacija in osnovni operaciji.

Slika 4.4 prikazuje vire, ki so na voljo za pridobivanje. Nekatere poizvedbe zahtevajo vhodne podatke, ki so določenega tipa. Poizvedbe, kot tudi mutacije, v odgovoru vračajo vir ali pa množico virov, ki so določeni v GraphQL shemi.




Slika 4.4: Prikaz poizvedb v aplikaciji.

### 4.1.1 Grafični vmesnik GraphiQL

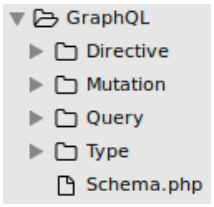
Za lažje preizkušanje in delo z zaledno aplikacijo, ki podpira GraphQL poizvedbeni jezik, je bil razvit grafični vmesnik GraphiQL [5]. Uporabljamo ga preko brskalnika, ponuja pa nam možnosti izvajanja poizvedb, mutacij in prikaz dokumentacije.

Lahko ga namestimo kot samostojno aplikacijo na strežnik in dostopamo do njega, druga možnost pa je namestitev v obliki razširitve za brskalnik Chrome, katerega uporaba je v nadaljevanju prikazana na slikah 4.7 in 4.8.

## 4.2 Sistem za strogo tipiziranje

Sistem za strogo tipiziranje nam omogoča, da GraphQL strežnik določa pravilnost poslanih poizvedb. Vhodni podatki in njihovi tipi so določeni v shemi, ter se preverjajo ob vsaki poizvedbi [8]. Tipi, ukazi in osnovne operacije tvo-

rijo GraphQL shemo, ki je v programski kodi smiselno ločena po imenikih, kot je prikazano na sliki 4.5.



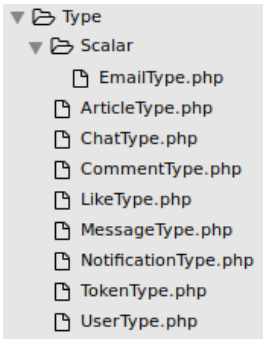

Slika 4.5: Struktura imenikov.

### 4.2.1 Tipi

Osnovna enota GraphQL sheme je tip. Obstaja šest določenih tipov in dva tipa, ki sta sestavljena iz drugih. Gradimo in povezujemo jih med seboj po meri. GraphQL tipi so:

- skalaren (ang. scalar)
- naštevalni (ang. enums)
- objektni (ang. object)
- vmesnik (ang. interface)
- unija (ang. union)
- vhodni objekt (ang. input object)
- seznam (ang. list)
- ne-nični (ang. non-null)

Tipi so smiselno ločeni po datotekah in imenskem prostoru, vsebujejo pa PHP objekte z elementi, kot prikazuje slika 4.6.




Slika 4.6: GraphQL tipi.

### 4.2.2 Poizvedbe

Poizvedbe so hierarhične, kjer zahtevane podatke opišemo z drevesno strukturo, kot je prikazano na sliki 4.7. Za razliko od arhitekturnega sloga REST lahko s poizvedbenim jezikom GraphQL zahtevamo več virov z enim klicem. Pridobljene podatke nato uporabimo za prikaz dinamične spletne strani. Z enim klicem znižamo zakasnitve, ki nastanejo ob večkratnem klicanju zaledne aplikacije.


Slika 4.7: Primer poizvedbe.

### 4.2.3 Mutacije

Mutacije so namenjene shranjevanju in posodabljanju podatkov. Operacije določimo sami glede na potrebe, ki jih imajo naši odjemalci. Kot je prikazano na sliki 4.8, lahko z eno zahtevo opravimo več operacij hkrati.


Slika 4.8: Primer mutacij.

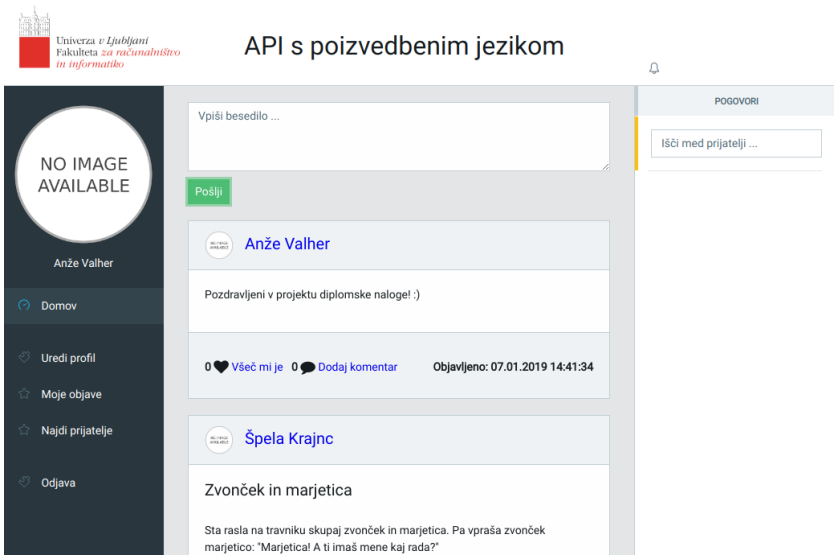
### 4.2.4 Ukazi

Ukazi nam omogočajo spreminjanje trenutnega izvajanja in preverjanja tipov v GraphQL dokumentu. Ko nam za točno določeno izvajanje GraphQL operacij ne zadostujejo vhodni podatki, nam specifikacija omogoča uporabo ukazov. Uporabimo jih, če recimo želimo dodati ali preskočiti polje. Postavimo jih za ime polja na katerem želimo, da se izvaja, in pred ime ukaza dodamo predpono @. Z njimi lahko opišemo dodatne informacije tipom, poljem, operacijam in poizvedbam, ki so narejene po meri [6].

## 4.3 Implementacija in integracija

Zgrajena bo dinamično generirana spletna stran socialnega omrežja. Za komunikacijo med zaledno in začelno aplikacijo bo implementiran GraphQL strežnik. Z njim bo socialno omrežje omogočalo dodajanje prijateljev, pogovor z njimi, pošiljanje vsebine in urejanje profila. Zaledna aplikacija nam bo omogočala prikaz dokumentacije in preverjanje prenešenih podatkov za

zagotavljanje konsistentnosti. Potrebno bo vzpostaviti okolje za izvajanje spletne strani in zgraditi aplikacije s podporo poizvedbenega jezika. Končni produkt bo delujoča spletna stran, ki bo pripravljena za izvajanje v lokalnem ali produkcijskem okolju. Slika 4.9 prikazuje začetno stran po prijavi v spletno aplikacijo.




Slika 4.9: Spletna stran socialnega omrežja.

### 4.3.1 Priprava okolja z Docker orodjem

Okolje za izvajanje aplikacij je zgrajeno znotraj vsebovalnika Docker, ki nam omogoča lahko postavitev in zagon okolja. Uporablja operacijski sistem Alpine linux z vsemi zahtevanimi servisi in knjižnicami. Pravila za izdelavo Docker vsebovalnika se nahajajo v datoteki `Dockerfile`. Z uporabo `FROM` besede določimo operacijski sistem, ki ga želimo uporabljati.

```
FROM alpine:latest
```

Z uporabo `RUN` izvajamo ukaze znotraj vsebovalnika za izdelavo slike, ki jo bomo zagnali in uporabljali. V spodnjem primeru je prikazana priprava

okolja z namestitvijo spletnega strežnika, podatkovne baze, skriptnega jezika PHP in NodeJs.

```
RUN apk add --no-cache nginx php7 php7-fpm nodejs mariadb \
 && apk add --no-cache npm bash mariadb-client php7-phar \
 && apk add --no-cache php7-iconv php7-json php7-openssl \
 && apk add --no-cache php7-pdo_mysql php7-xml php7-yaml \
 && apk add --no-cache php7-ctype php7-dom php7-mbstring \
 && apk add --no-cache php7-pdo php7-session php7-tokenizer
```

`COPY` možnost nam omogoča kopiranje podatkov iz gostitelja v vsebovalnik. Z njo lahko kopiramo vsebino imenikov ali pa posamezno datoteko.

Z `ENTRYPOINT` možnostjo določimo servis ali pa pogonsko datoteko s katero zaženemo pripravljene servise, ki jih potrebujemo ob zagonu vsebovalnika.

```
COPY services.sh /services.sh
COPY app /opt/app/

ENTRYPOINT ["./services.sh"]
```

Po pripravi datoteke `Dockerfile` zgradimo sliko, ki jo smiselno poimenujemo in zaženemo. Ker želimo dostopati do naše aplikacije tudi iz računalnika na katerem poganjamo Docker vsebovalnik, moramo vrata spletnega strežnika narediti dosegljiva tudi izven vsebovalnika. To opravimo z uporabo opcije `--publish list` ob zagonu naše slike, kot je prikazano v primeru spodaj.

```
docker build -t diploma .
docker run -it -p 80:80 diploma
```

Za lažjo uporabo in predstavo aplikacije se bo podatkovna baza napolnila s testnimi podatki ob zagonu vsebovalnika. Docker orodje nam omogoča izgradnjo ali prenos slik na različne računalnike, kar omogoča izvajanje v različnih okoljih.

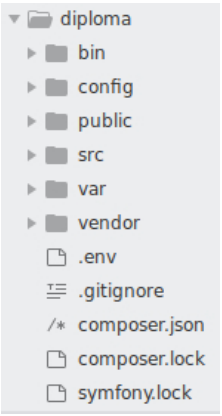
### 4.3.2 Izdelava zaledne aplikacije

#### Priprava ogrodja

Namestitev in priprava ogrodja Symfony [21] je narejena z uporabo upravljalnika odvisnosti Composer [27].

```
composer create-project symfony/skeleton diploma
```

Po izvedbi ukaza se v imeniku `diploma` nahaja delujoča Symfony aplikacija. Vsebuje drevesno strukturo imenikov in datoteke, ki so potrebne za delovanje, kot prikazuje slika 4.10.




Slika 4.10: Skeleton Symfony aplikacije.

#### Vgradnja knjižnic

Po GraphQL specifikaciji je narejenih mnogo knjižnic za več različnih programskih jezikov. Med njimi najdemo podporo za jezike Java, Go, Clojure, JavaScript, PHP, Python in mnoge druge. Za skriptni jezik PHP in njegova ogrodja obstaja več različnih knjižnic, ki so primerne za izdelavo GraphQL strežnika. Vse popolnoma podpirajo specifikacijo poizvedbenega jezika.
PHP knjižnice so:

- Webonyx [19]

- Folklore [18]
- Youshido [20]

Za izdelavo diplomske naloge je bila izbrana knjižnica uporabnika Youshido. Podpira izgradnjo GraphQL sheme s PHP objekti in je tesno povezana z ogrodjem Symfony [21]. V ogrodje je bila dodana s pomočjo upravljalnika odvisnosti Composer [27], tako da sta bili v datoteko `composer.json` dodani knjižnici.

```
"youshido/graphql-bundle": "^1.2",
"youshido/graphql-files-bundle": "^0.0.1"
```

Z uporabo ukaza `composer install` namestimo knjižnici v za to namenjen imenik in do njiju dostopamo preko imenskega prostora (ang. namespace). Youshido GraphQL knjižnica se nahaja pod imenskim prostorom `Youshido\GraphQL`. Nato je potrebno povezati naše Symfony ogrodje in GraphQL knjižnico. To storimo v `config.yml` datoteki, v kateri nastavimo pot imenika z objekti GraphQL sheme.

```
graphql:
    schema_class: "AppBundle\\GraphQL\\Schema"
    security:
        guard:
            operation: true
            field: false
```

V `routing.yml` datoteki nastavimo pot, kjer se nahaja PHP GraphQL knjižnica za ogrodje Symfony. To storimo s prikazanimi nastavitvami.

```
graphql:
    resource: "@GraphQLBundle/Controller/"
graphql_file.image_resizer:
    resource: "@GraphQLFilesBundle/Resources/config/route.yml"
```

Nato lahko začnemo razvijati shemo in izdelovati objekte po meri.

### Komunikacija s podatkovno bazo

Za povezavo s podatkovno bazo se uporablja knjižnica Doctrine 2 [16], ki je bila dodana z upravljalnikom odvisnosti Composer [27].

```
"doctrine/doctrine-bundle": "^1.6",
"doctrine/orm": "^2.5",
```

V datoteko `parameters.yml` dodamo podatke za povezavo na podatkovno bazo.

```
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: diploma
    database_user: root
    database_password:
```

Ob pravilnih nastavitvah imamo delujočo povezavo s podatkovno bazo znotraj Symfony ogrodja. Vire v podatkovni bazi ustvarimo z uporabo entitet, ki jih gradimo znotraj imenika `src/AppBundle/Entity`.

Entitete so določene kot PHP objekti in nastavitve v njih podane preko anotacij. Uporabljajo se za določanje pripadajočih nastavitev tabel v podatkovni bazi. Z anotacijo `Table` nastavimo tabelo v podatkovni bazi, ki jo bo naša entiteta uporabljala. `Entity` pa nam omogoča nastavitev objekta, ki upravlja z našo entiteto.

```
@ORM\Table(name="user")
@ORM\Entity(repositoryClass="ApBndle\Repository\UsrRepository")
```

Z anotacijami nad objektnimi spremenljivkami določamo nastavitve enega polja v podatkovni bazi. S `Column` določimo ime polja in njegov tip. Z `Id` povemo, da je objektna spremenljivka enolični identifikator. `GeneratedValue` pa nam omogoča avtomatsko vstavljanje vrednosti v polje ob vsakem novem vnosu v tabelo.

```
@ORM\Column(name="id", type="integer")
@ORM\Id
@ORM\GeneratedValue(strategy="AUTO")
```

Z uporabo anotacij `JoinColumn`, `ManyToOne` in `OneToMany` ustvarimo povezave tabel v podatkovni bazi. Na primeru spodaj je prikazana povezava med tabelama, uporabniki in objave. Vsaka objava ima avtorja, avtor pa ima lahko več objav. Avtor je povezan z entiteto uporabnikov preko enoličnega identifikatorja uporabnika.
Entiteta objav:

```
/**
 * Many Posts have One User.
 *
 * @ORM\ManyToOne(targetEntity="User", inversedBy="articles")
 * @ORM\JoinColumn(name="user_id", referencedColumnName="id")
 */
public $user;
```

Entiteta uporabnikov:

```
/**
 * One User has Many Posts.
 *
 * @ORM\OneToMany(targetEntity="Article", mappedBy="User")
 */
public $articles;
```

Delujoča povezava s podatkovno bazo v ogrodju nam omogoča shranjevanje in spreminjanje podatkov. Z uporabo zgrajenih funkcij in entitet nam knjižnica omogoča branje podatkov in njihovih povezav ter jih pripravi za uporabo v obliki objektov.

### Izdelava sheme

V sklopu izgradnje sheme je potrebno izvesti več korakov s katerimi povežemo podatkovno bazo in sistem za strogo tipiziranje.
Določiti je potrebno:

- tipe
- mutacije
- poizvedbe
- razreševalnike

Vsak korak je smiselno postavljen v pripadajoči imenik in je določen kot PHP objekt. Osnovne in zahtevnejše tipe se določi po potrebi in povezuje med seboj za izdelavo željene sheme.

Objekte gradimo v imeniku `tipi` in znotraj njih določimo vrednost elementov, ki jih tip predstavlja. To storimo v funkciji `build`, kjer tudi povezujemo vire med seboj. Vire povežemo med seboj z uporabo poizvedbenega objekta vira namesto objekta tip. Na sliki 4.11 je prikazan objektni tip objav. Sestavljen je iz različnih tipov in virov.

```php
namespace AppBundle\GraphQL\Type;

use AppBundle\GraphQL\Query\User\{UserField, CommentsField, LikesField};
use Youshido\GraphQL\Type\NonNullType;
use Youshido\GraphQL\Type\Scalar\{IntType, StringType, DateTimeType};
use Youshido\GraphQL\Type\Object\AbstractObjectType;
use Youshido\GraphQL\Config\Object\ObjectTypeConfig;

class ArticleType extends AbstractObjectType {
    public function build(ObjectTypeConfig $config) {
        $config->addFields([
            'id'      => new NonNullType(new IntType()),
            'body'    => new NonNullType(new StringType()),
            'likes' => new NonNullType(new IntType()),
            'user' => new UserField(),
            'comments' => new CommentsField(),
            'likes' => new LikesField(),
            'createdAt' => new DateTimeType('d.m.Y H:i:s')
        ]);
    }
}
```

Slika 4.11: Primer objektnega tipa.

Z mutacijami in poizvedbami določimo, na kakšen način bo odjemalec upravljal s podatki v naši aplikaciji. Prav tako lahko določimo vhodne podatke, ki so na voljo za uporabo pri operacijah in so strogo tipizirani.

Imeniki in datoteke za poizvedbe so smiselno poimenovani glede na vir kateremu pripadajo. Vhodni podatki za pridobivanje določenih podatkov lahko prihajajo iz različnih virov. Dobimo jih lahko od odjemalca, smiselne povezave na že zahtevani objekt ali pa od drugod. Programsko logiko sprogramiramo v za ta vir namenjen objekt. Vrnjeni podatki poizvedbenega objekta se morajo skladati s tipom, ki ga določa. Na sliki 4.12 je prikazan PHP poizvedbeni objekt za pridobivanje uporabnika z enim vhodnim podatkom, tremi različnimi načini za pridobivanje uporabnika in nastavitvijo, da vrača tip uporabnika.

```php
namespace AppBundle\GraphQL\Query\User;

use AppBundle\GraphQL\Type\UserType;
use Youshido\GraphQL\Type\Scalar\IntType;
use Youshido\GraphQL\Execution\ResolveInfo;
use Youshido\GraphQLBundle\Field\AbstractContainerAwareField;
use Youshido\GraphQL\Config\Field\FieldConfig;

class UserField extends AbstractContainerAwareField {
    public function build(FieldConfig $config) {
        $config->addArguments([
            'user_id' => new IntType()
        ]);
    }

    public function resolve($value, array $args, ResolveInfo $info) {
        if (is_object($value))
        {
            $user = $this->container->get('resolver.user')->findUserById($value->user->id);
        } elseif(isset($args['user_id'])) {
            $user = $this->container->get('resolver.user')->findUserById($args['user_id']);
        } else {
            $user = $this->container->get('resolver.user')->findCurrentUser();
        }

        return $user;
    }

    public function getType() {
        return new UserType();
    }
}
```

Slika 4.12: Primer poizvedbenga objekta.

Za izdelavo mutacij in poizvedb je potrebno določiti tri funkcije. S funkcijo `build` določimo argumente, ki jih operacija sprejme iz strani odjemalca. S funkcijo `resolve` določimo izvedbo operacije oziroma kje se nahaja poslovna logika za uspešno izvedbo operacije. Spremenljivka `container`, ki jo pridobimo z abstraktnim razredom, omogoča uporabo Symfony aplikacije in njenih servisov. V funkciji `getType` nastavimo tip, ki ga bo operacija vrnila. Za popolno delovanje je potrebno zgraditi še objekte `resolve`, ki se uporabljajo kot je prikazano na primeru ob klicanju razreševalnika uporabnika na sliki 4.12.

GraphQL shema in Symfony ogrodje sta povezana z uporabo servisov in skupne spremenljivke, ki nam omogoča dostop do njih. Servisi so določeni v ogrodju in sicer v nastavitveni datoteki `services.yml`.

```
resolver.user:
    class: AppBundle\Resolver\UserResolver
    autowire: true
    autoconfigure: false
    public: true
    parent: 'resolver.base'
```

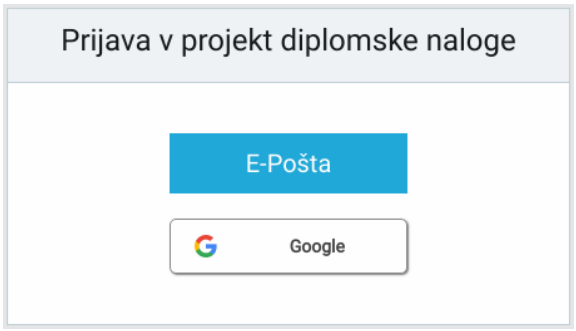
Znotraj razreševalnih objektov sprogramiramo željene funkcije, ki lahko vsebujejo poslovno logiko ali pa upravljajo s podatki v podatkovni bazi.

```
public function updatePassword($password) {
    $user = $this->currUser;
    $user->password = $password;

    $this->em->persist($user);
    $this->em->flush();

    return $user;
}
```

Po izgradnji potrebnih korakov imamo zgrajeno GraphQL shemo povezano z ogrodjem Symfony. Ker je povezava s podatkovno bazo že vzpostavljena in uporabljena v razreševalnih objektih, je aplikacijski programski vmesnik pripravljen za uporabo. Podpira poizvedbeni jezik GraphQL in ga lahko razširimo po potrebi.

### Avtentikacija

V socialno omrežje se lahko prijavimo na dva načina. Prva možnost je prijava z uporabniškim imenom in geslom, druga pa uporaba Google prijave, ki uporablja `OpenID Connect`. Le-ta je primernejša za uporabnika in ne uporablja gesla. Slika 4.13 prikazuje prijavno stran izdelane spletne strani.




Slika 4.13: Prijava v omrežje.

Za potrebe avtentikacije je bil uporabljen objekt Symfony Voter [26]. Izvede se ob vsakem klicu na aplikacijski programski vmesnik. V ta namen je bil narejen po meri zgrajen objekt `GraphQLVoter` in dodan v ogrodje kot servis.

```
AppBundle\Security\GraphQLVoter:
    public: false
    arguments:
      $request: '@request_stack'
      $em: '@em'
    tags: ['security.voter']
```

Servis kot vhodne podatke dobi spremenljivke zahteve in upravljalca entitet, kar nam omogoči dostop do podatkovne baze. Ob klicu se izvede funkcija `voteOnAttribute`. V njej se preveri ID žeton, veljavnost in pripadnost uporabniku.

Dodana je bila `unprotected` objektna spremenljivka, ker avtentikacija ni potrebna pri vseh operacijah. V njej so vse operacije, ki jih vedno izvedemo, kot recimo prijava in registracija uporabnika. Na sliki 4.14 je prikazano preverjanje žetona in nezaščitenih operacij.

```
protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
{
    if (in_array($subject->getName(), $this->unprotected)) {
        return true;
    } else if ($this->request->headers->has('access-token')) {
        return $this->validateToken($this->request->headers->get('access-token'));
    }

    return false;
}
```

Slika 4.14: Preverjanje veljavnosti žetona.

Žetoni se shranjujejo v podatkovni bazi in do njih dostopamo preko knjižnice Doctrine. Žeton ima v tabeli polje z enoličnim identifikatorjem uporabnika. V kolikor žeton obstaja v tabeli, se za nadaljnje poizvedbe uporablja pripadajoči uporabnik, sicer pa se zahteva ponovna prijava, kar prikazuje slika 4.15.

```
private function validateToken($token) {
    $token = $this->em
                  ->getRepository('AppBundle:Token')
                  ->findByAccessToken($token);

    return !$token ? false : true;
}
```

Slika 4.15: Iskanje žetona v podatkovni bazi.

Z uporabe avtentikacije poiščemo uporabnika, ki uporablja spletno stran, in podatke zanj prikazujemo dinamično. To pomeni, da lahko podatke skri-

jemo pred uporabniki in jih ne prikažemo ali pa prikazujemo samo podatke, ki pripadajo uporabniku.

### 4.3.3 Uporaba v odjemalcu

Za izdelavo odjemalčeve aplikacije je uporabljen Apollo odjemalec za Angular. Je zelo prilagodljiv, narejen s strani skupnosti za Angular, JavaScript in izvorna okolja. Načrtovan je za gradnjo od spodaj navzgor (ang. bottom up) kar nam olajša izdelavo komponent, ki pridobivajo podatke iz GraphQL strežnika. Lahko se uporablja v vsakem JavaScript začelnem sistemu, kjer se uporablja GraphQL za pridobivanje podatkov [3]. Apollo odjemalec je:

- postopoma prilagodljiv
- splošno usklajen
- preprost
- preverljiv in razumljiv
- narejen za interaktivne aplikacije
- majhen in prilagodljiv
- vzdrževan s strani skupnosti

Za izdelavo začelne aplikacije je potrebno namestiti AngularJS ogrodje ter implementirati komponente in odvisnosti. Za uporabo poizvedbenega jezika GraphQL je potrebno namestiti Apollo odjemalca.

```
ng add apollo-angular
```

V AngularJS komponentah je potrebno uvoziti dve knjižnici za komunikacijo s poizvedbenim jezikom GraphQL.

```
import { Apollo } from 'apollo-angular';
import gql from 'graphql-tag';
```

Komponente pripadajo določeni podstrani našega socialnega omrežja. V njih nastavimo poizvedbe za vsako podstran s podatki, ki jih potrebujemo za izgradnjo. V strukturi poizvedbe imamo pregled nad podatki, ki jih pridobiva, ter nad zahtevanimi argumenti. Podatki se v odgovoru vračajo v obliki, kot jo prikazuje struktura poizvedbe na sliki 4.16.

```
const findChatGql = gql`query chat($friend_id: Int, $offset: Int, $limit: Int) {
  chat(friend_id: $friend_id) {
    id,
    user1 {
      id,
      first_name,
      last_name
    },
    user2 {
      id,
      first_name,
      last_name
    },
    messages(offset: $offset, limit: $limit) {
      body,
      createdAt,
      owner {
        first_name,
        last_name
      }
    }
  }
}`
```

Slika 4.16: Poizvedba v odjemalcu.

Pozvedbe izvršimo z Apollo odjemalcem, ki nam omogoča izvajanje dodatnih operacij ob določenih dogodkih. Tako lahko posodobimo spletno stran ob spremembi podatkov ali ob napaki izpišemo primerno sporočilo. Odjemalec omogoča pomnenje poizvedb. Ob izvedbi poizvedbe je potrebno določiti tudi zahtevane vhodne podatke. Na sliki 4.17 je prikazan primer izvajanja poizvedb za pogovor med prijatelji, ki nam omogoča avtomatsko posodabljanje spletne strani ob novem sporočilu.

```
findChat(limit) {
  this.feedQuery = this.apollo.watchQuery<any>({
    query: findChatGql,
    variables: {
      friend_id: this.id,
      offset: 0,
      limit: limit
    },
    fetchPolicy: 'network-only'
  });

  this.feed = this.feedQuery
    .valueChanges
    .subscribe(({data}) => {
      this.feed = data.chat.messages;
      this.user1 = data.chat.user1;
      this.user2 = data.chat.user2;
      this.chatId = data.chat.id;
    },(error) => {
    });
}
```

Slika 4.17: Poizvedba v odjemalcu.

Pridobljeni podatki so shranjeni v objektne spremenljivke, ki so nam na voljo v HTML datoteki. Do njih dostopamo preko oznak (ang. tags), ki so določene v AngularJS ogrodju. Ob spremembi objektnih spremenljivk se posodobijo tudi oznake.

```
{{ user1?.first_name }} {{ user1?.last_name }}
```

S tem smo povezali odjemalca z GraphQL strežnikom in podatkom dodali dodano vrednost, saj jih prikazujemo na uporabniku prijazen način. Z dodajanjem virov in zmožnostmi GraphQL strežnika zgradimo spletno stran, ki predstavlja socialno omrežje.

# Poglavje 5

# Sklepne ugotovitve

## 5.1 Analiza uporabnosti

Poizvedbeni jezik GraphQL nam že v osnovi ponuja pregledno dokumentacijo in preverjanje tipov uporabljenih elementov. S tem si olajšamo izgradnjo in uporabo aplikacij. Za izgradnjo GraphQL sheme in implementacijo v Symfony ogrodju je bilo porabljenega več časa od pričakovanega. A po postavitvi je nadgradnja in uporaba strežnika postala preprosta in razumljiva. Prednosti poizvedbenega jezika GraphQL so:

- dokumentacija
- tipizirani sistem
- enoten jezik
- lahek razvoj
- podprt v mnogih programskih jezikih
- učinkovito pridobivanje podatkov

S strogim tipiziranjem zagotavlja preverjanje tipov elementov, a za resno poslovno aplikacijo to ne zadostuje. Potrebno bi bilo razviti integracijske

teste za aplikaciji. S testi bi zagotavljali verodostojnost prenešenih podatkov in še izboljšali kvaliteto izdelane spletne strani.

Pri grafičnem vmesniku GraphiQL se je pokazala slabost pri uporabi z obstoječo avtentikacijo. Določene verzije vmesnika namreč ne omogočajo dodajanja dodatnih spremenljivk in žetona pri izvajanju operacij. To bi lahko rešili s postavitvijo ustrezne verzije na lastni strežnik.

GraphQL nam ponuja preglednost nad aplikacijami in za to ne zahteva dodatnih knjižnic ali produktov. S tem se zmanjša možnost napak ob nadgradnjah in uporabi, ter ohranja konsistentnost izgradnje.

## 5.2 Zaključek

V diplomskem delu je bil predstavljen poizvedbeni jezik GraphQL in njegova implementacija. Cilj je bil izdelati delujočo spletno stran z uporabo opisane tehnologije ter jo predstaviti. Prednosti pri izdelavi so se pokazale zelo zgodaj, saj se vedno sklicujemo na GraphQL shemo, na kateri gradimo naše aplikacije.

Izdelava aplikacijskega programskega vmesnika s poizvedbenim jezikom je preprosta. Pri tem je zelo pomemben korak izbira knjižnic, s katerimi ga bomo implementirali. Podpora slabše zastavljenim knjižnicam namreč lahko s časom preneha, kar nas prisili v uporabo drugih knjižnic in posledično v rekonstrukcijo naše aplikacije.

Možnosti, ki jih GraphQL ponuja, so zelo napredne in strnjene v poizvedbenem jeziku. Z drugimi orodji je možno doseči podobne rezultate, a ponavadi zahtevajo uporabo več različnih knjižnic ali produktov. Zaradi tega je ob taki uporabi lahko še več težav kot prednosti. GraphQL se temu izogne, zato je posvečen izključno izdelavi aplikacije s poizvedbenim jezikom.

Izdelava projekta za diplomsko nalogo je bila uspešna. Uporaba in izdelava zaledne aplikacije je zelo preprosta. S preverjanjem poslanih in prejetih podatkov se zagotavlja kvaliteta delovanja aplikacije. Pri tem se izognemo nepričakovanim napakam in poskrbimo za primeren prikaz že obstoječih. S

tem se tudi lažje zagotavlja stik z uporabnikom in sledi njegovim akcijam.

Spletna stran socialnega omrežja zaokrožuje razvoj GraphQL strežnika, njegovo uporabo in izvajanje v za to pripravljenem okolju. Razvit je bil po principu hitrega razvoja pri tem pa so bile uporabljena orodja in knjižnice, ki to omogočajo. Za poslovno aplikacijo bi bilo potrebno razmisliti o uporabi drugih programskih jezikov in ogrodij.

Začelna aplikacija je zgrajena z AngularJS ogrodjam za katerega pa že obstaja novejša verzija. Le-tega bi bilo dobro uporabiti, saj bi nam olajšal razvoj. Integracija PHP knjižnice za poizvedbeni jezik je bila preprosta, a bi se lahko uporabilo tudi druge knjižnice. Ker je smiselnost GraphQL aplikacij na visokem nivoju pa bi bilo najbolje uporabiti funkcijsko programiranje za izdelavo zaledne aplikacije z uporabo pravil, ki jih ponuja poizvedbeni jezik GraphQL.

# Literatura


[1] Apache Cordova. Dosegljivo: https://cordova.apache.org/. [Dostopano 14. 1. 2019].

[2] Aplikacijski programski vmesnik. Dosegljivo: https://searchmicroservices.techtarget.com/definition/application-program-interface-API. [Dostopano 14. 1. 2019].

[3] Apollo odjemalec. Dosegljivo: https://www.apollographql.com/docs/angular/. [Dostopano 14. 1. 2019].

[4] Arhitekturni slog. Dosegljivo: https://en.wikipedia.org/wiki/Software_architecture. [Dostopano 14. 1. 2019].

[5] Grafični vmesnik GraphiQL. Dosegljivo: https://github.com/graphql/graphiql. [Dostopano 14. 1. 2019].

[6] GraphQL direktive. Dosegljivo: https://facebook.github.io/graphql/draft/#sec-Language.Directives. [Dostopano 14. 1. 2019].

[7] GraphQL dokumentacija. Dosegljivo: https://facebook.github.io/graphql/draft/#sec-Descriptions. [Dostopano 14. 1. 2019].

[8] GraphQL sistem za strogo tipiziranje. Dosegljivo: https://facebook.github.io/graphql/draft/#sec-Type-System. [Dostopano 14. 1. 2019].

[9] GraphQL čez HTTP. Dosegljivo: https://graphql.org/learn/serving-over-http/. [Dostopano 14. 1. 2019].

[10] ID žeton. Dosegljivo: https://auth0.com/docs/tokens/id-token. [Dostopano 14. 1. 2019].

[11] Industrijski standard RFC 7519. Dosegljivo: https://tools.ietf.org/html/rfc7519. [Dostopano 14. 1. 2019].

[12] JavaScript ogrodje AngularJS. Dosegljivo: https://angularjs.org/. [Dostopano 14. 1. 2019].

[13] Markdown. Dosegljivo: https://en.wikipedia.org/wiki/Markdown. [Dostopano 14. 1. 2019].

[14] Monolog. Dosegljivo: https://github.com/Seldaek/monolog. [Dostopano 22. 10. 2018].

[15] O ogrodju AngularJS. Dosegljivo: https://en.wikipedia.org/wiki/AngularJS. [Dostopano 14. 1. 2019].

[16] Objektno-relacijsko mapiranje Doctrine. Dosegljivo: https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html#what-is-doctrine. [Dostopano 14. 1. 2019].

[17] Opis vsebovalnika Docker. Dosegljivo: https://www.docker.com/resources/what-container. [Dostopano 14. 1. 2019].

[18] PHP knjižnica za GraphQL uporabnika Folklore. Dosegljivo: https://github.com/Folkloreatelier/laravel-graphql. [Dostopano 14. 1. 2019].

[19] PHP knjižnica za GraphQL uporabnika Webonyx. Dosegljivo: https://github.com/webonyx/graphql-php. [Dostopano 14. 1. 2019].

[20] PHP knjižnica za GraphQL uporabnika Youshido. Dosegljivo: https://github.com/youshido-php/GraphQL. [Dostopano 14. 1. 2019].

[21] PHP ogrodje Symfony. Dosegljivo: https://symfony.com/. [Dostopano 14. 1. 2019].

[22] Podatkovna baza MySQL. Dosegljivo: https://sl.wikipedia.org/wiki/MySQL. [Dostopano 14. 1. 2019].

[23] Programski jezik JavaScript. Dosegljivo: https://www.javascript.com/. [Dostopano 14. 1. 2019].

[24] Skriptni jezik PHP. Dosegljivo: http://php.net/manual/en/preface.php. [Dostopano 14. 1. 2019].

[25] Spletni strežnik NGINX. Dosegljivo: https://www.nginx.com/resources/glossary/nginx/. [Dostopano 14. 1. 2019].

[26] Symfony Voter. Dosegljivo: https://symfony.com/doc/current/security/voters.html. [Dostopano 14. 1. 2019].

[27] Upravljalnik odvisnosti Composer. Dosegljivo: https://getcomposer.org/doc/00-intro.md. [Dostopano 14. 1. 2019].

[28] Vsebovalnik Docker. Dosegljivo: https://www.docker.com/. [Dostopano 14. 1. 2019].

[29] XML-RPC specifikacija. Dosegljivo: http://xmlrpc.scripting.com/spec.html, 1999. [Dostopano 14. 1. 2019].

[30] Arhitektura za izmenjavo podatkov med spletnimi storitvami REST. Dosegljivo: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. [Dostopano 14. 1. 2019].

[31] XML Information Set specifikacija. Dosegljivo: https://www.w3.org/TR/2004/REC-xml-infoset-20040204/, 2004. [Dostopano 14. 1. 2019].

[32] SOAP specifikacija. Dosegljivo: https://www.w3.org/TR/soap12/, 2007. [Dostopano 14. 1. 2019].

[33] JSON-RPC specifikacija. Dosegljivo: https://www.jsonrpc.org/specification, 2010. [Dostopano 14. 1. 2019].

[34] GraphQL specifikacija. Dosegljivo: https://facebook.github.io/graphql/draft/, 2012. [Dostopano 14. 1. 2019].

[35] Specifikacija OAuth 2.0. Dosegljivo: https://oauth.net/2/, 2012. [Dostopano 14. 1. 2019].

[36] Metoda HTTP GET. Dosegljivo: https://tools.ietf.org/html/rfc7231#section-4.3.1, 2014. [Dostopano 14. 1. 2019].

[37] Metoda HTTP POST. Dosegljivo: https://tools.ietf.org/html/rfc7231#section-4.3.3, 2014. [Dostopano 14. 1. 2019].

[38] Protokol HTTP. Dosegljivo: https://tools.ietf.org/html/rfc7231, 2014. [Dostopano 14. 1. 2019].

[39] Specifikacija OpenID Connect. Dosegljivo: https://openid.net/connect/, 2014. [Dostopano 14. 1. 2019].