INSTITUT ZA MATEMATIKU I INFORMATIKU
PRIRODNO-MATEMATIČKI FAKULTET
UNIVERZITET U KRAGUJEVCU

SEMINARSKI RAD
PREDMET: BAZE PODATAKA 2

GRAPHQL

Mentor
dr Ana Kaplarević-Mališić

Student
Bojan Piskulić, 93/2015

Jun 2019.

Sadržaj

Predgovor

Kada je Facebook 2011. godine odlučio da uvede News Feed, inženjeri su uvideli da imaju veliki problem sa dobijanjem podataka, naročito na mobilnim platformama, tako je nastao GraphQL.

1. Uvod

1.1. Ukratko o GraphQL-u

GraphQL predstavlja nov API standard koji predstavlja potencijalno moćniju alternativu REST-u. Razvijen je od strane Facebook-a, a danas je vlasništvo zajednice koja je svakog dana sve brojnija. Zapravo GraphQL predstavlja samo specifikaciju kako treba da izgleda API i nije ništa više osim povećeg dokumenta u kojem je opisano šta i kako treba da radi. Postoji mnogo implementacija za gotovo sve platforme i tehnologije. Ono što GraphQL čini posebnim jeste da isti ne zavisi od tehnologije, baze ili nekih drugih parametara.

API je postao sveprisutan u razvoju softverske infrastrukture. Ukratko, definiše kako klijent može učitati podatke sa servera.

U svojoj srži, GraphQL omogućava deklarativno dobijanje podataka, gde klijent može naznačiti šta mu je tačno od podataka potrebno. Umesto postojanja više endpoint-a koji vraćaju fiksne strukture podataka, GraphQL server otkriva samo jedan endpoint i odgovara sa podacima koje je klijent tražio.

1.1.1. Upitan jezik za API

Većina današnjih aplikacija ima potrebu da preuzima podatke sa servera gde se oni čuvaju u bazi podataka. API je zadužen da pruži interfejs za dobijanje potrebnih podataka. GraphQL se često meša sa tehnologijama vezanim za baze podataka, što nije tačno. GraphQL jeste upitan jezik, ali ne radi sa bazama podataka, nego sa API-jima.

1.1.2. Efikasnija alternative REST-u

REST je popularan način za dobavljanje podataka sa servera. Kada se koncept REST-a razvijao, klijentske aplikacije bile su relativno jednostavne. Danas postoje tri ključne stvari koje ispituju efikasnost REST-a:

- Povećana upotreba mobilnih podataka zahteva efikasnije učitavanje sadržaja Ovo je jedan od glavnih razloga zašto je Facebook razvio GraphQL, kako bi što više smanjila količina podataka koja se prenosi mrežom.
- Raznovrsnost frontend framework-a i platformi usled prevelike raznolikosti među frontend frejmvorcima i platformama koje pokreću klijentske aplikacije postaje teško održavati jedan API koji može zadovoljiti svačije zahteve. Pomoću GraphQL-a, **svaki klijent može sam navoditi koje podatke želi da dobije**.
- Ubrzan development i očekivanje brzog razvoja funkcionalnosti, konstantno nadograđivanje softvera postalo je standard u mnogim kompanijama, a usled čestih izmena javlja se potreba i za izmenama na REST API-ju i načinu kako i koji se podaci dostavljaju klijentu.

2. Da li je GraphQL bolji od REST-a?

U prethodnoj deceniji, REST je postao standard za dizajn web API-ja. Međutim, njegova glavna mana jeste ta što nije fleksibilan. To ćemo najlakše objasniti praktičnim primerom. Recimo, želimo da za aplikaciju koja se bavi pisanjem blogova dobijemo sve naslove objava određenog korisnika. Takođe, na istoj strani želimo da prikažemo i poslednja tri pratioca navedenog korisnika. Pogledajmo implementaciju rešenja pomoću REST-a i GraphQL-a.

2.1. Dobijanje podataka REST vs GraphQL

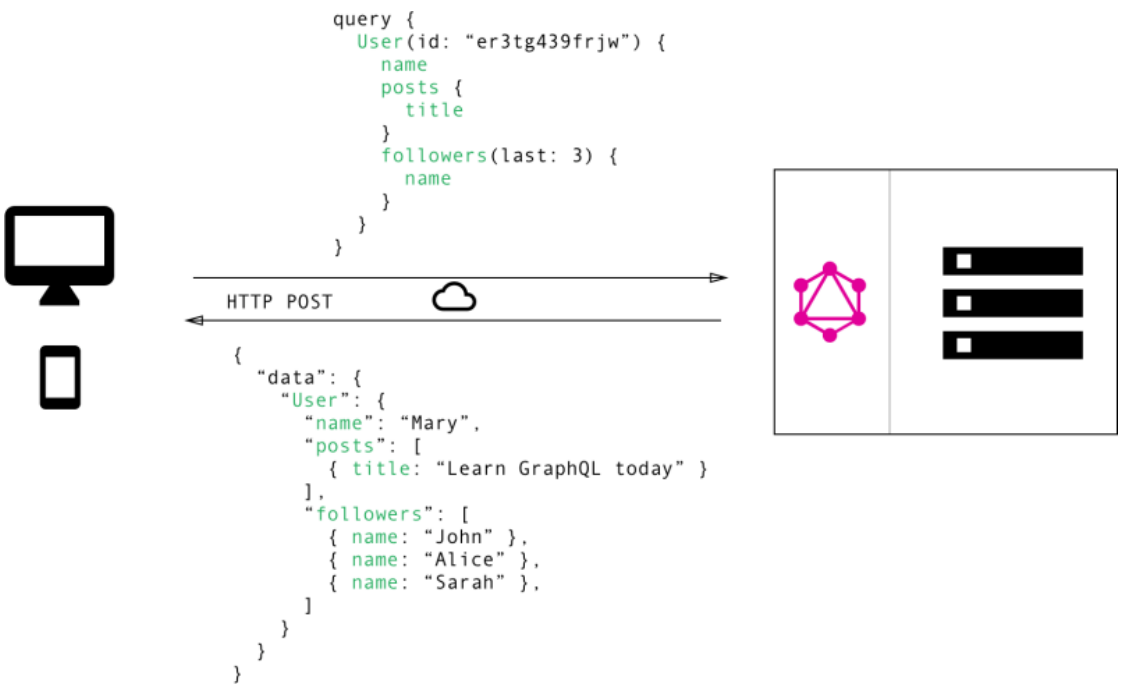
Sa REST API-jem, podatke ćemo dobiti tako što ćemo pristupiti određenim endpointima. Na primer:

- **/users/{:id}** -> ovim endpoinom ćemo dobiti podatke o korisniku.
- **/users/{:id}/posts** -> endpoint za dobijanje svih objava izabranog korisnika.
- **/users/{:id}/followers** -> endpoint koji će vratiti listu pratilaca za izabranog korisnika.

Slika 1. Rešenje problema pomoću REST API-ja.

REST: Pomoću tri zahteva dobili smo potrebne podatke. Takođe, ovde imamo i slučaj overfetching-a (slučaj kada nam endpoint vraća i podatke koji nam nisu potrebni).

GraphQL nam daje mogućnost da pošaljemo jedan upit GraphQL serveru gde konkretno navodimo koji podaci su nam potrebni. Server će vratiti JSON objekat koji zatovoljava pomenute zahteve.



Slika 2. Rešenje problema pomoću GraphQL-a.

GraphQL: Pomoću jednog zahteva dobijamo sve potrebne podatke. Obratiti pažnju na to da su podaci struktuirani tačno onako kako smo i naveli u upitu.

3. Koje su koristi i koje probleme rešava GraphQL?

3.1. Nema više Over/Underfetching-a

Ovo su najčešći problemi kod REST-a, pošto endpointi koji se gađaju vraćaju podatke koji imaju fiksnu strukturu.

"Think in graphs, not endpoints." – Lessons From 4 Years of GraphQL by Lee Byron, GraphQL Co-Inventor.

Overfetching: Preuzimanje suvišnih podataka. Ovo zapravo znači da klijent preuzima više informacija nego što je potrebno u aplikaciji. Zamislite, na primer, stranu koja treba da prikaže listu korisnika samo sa njihovim imenima. U REST API–ju, ova aplikacija bi pogodila endpoint **/users** i kao odgovor dobila JSON niz sa korisničkim podacima. Međutim, ovaj odgovor može sadržati više informacija o korisnicima, npr. njihovi rođendani, adrese… Sve su to informacije koje nam zapravo nisu potrebne pošto želimo prikazati samo njihova imena.

Underfetching i n+1 problem – Underfetching znači da određeni endpoint ne vraća sve potrebne podatke (suprotno overfetching-u), pa će klijent morati da gađa i druge endpointe kako bi dobio sve podatke koji su mu potrebni. Ovo može veoma lako da se iskomplikuje.

Na primer, aplikacija treba da prikaže poslednja ti pratioca po korisniku. API obezbeđuje endpoint **/users/{:user-id}/followers**. Da bi bila u mogućnosti da prikaže potrebne informacije, aplikacija će morati da podnese jedan zahtev do endpointa **/users**, zatim će za svakog korisnika morati da gađa endpoint **/users/{:user-id}/followers**.

3.2. Rasterećenje serverske strane

Postoji rizik da izmenom na UI delu (korisničkom interfejsu) dođe do potrebe za izmenom i na serverskoj strani aplikacije (vraćanje više/manje podataka itd.) što umanjuje produktivnost. Pomoću GraphQL-a ovaj problem prebacuje se na klijentsku stranu gde se izmenom API upita dobijaju željeni podaci.

3.3. Analitika na serveru

GraphQL poseduje opciju **praćenja performansi zahteva** koje obrađjue server kako bi se vršile dodatne optimizacije. GraphQL koristi koncept resolver funkcija kako bi prikupio podatke koje je zahtevao klijent, merenjem performansi istih, moguće je imati uvid u performanse sistema.

3.4. Koristi od šeme i tipiziranog sistema

GraphQL koristi **strogo tipizirani sistem**. Svi tipovi koji se koriste u API-ju zapisani su u schemai pomoću **GraphQL Schema Definition Language-a (SDL)**. Ova šema služi kao ugovor između klijenta i servera kako bi se definisalo kako će klijent pristupiti podacima.

Kada je šema definisana, frontend i backend timovi mogu raditi nezavisno pošto su i jedni i drugi svesni strukture podataka sa kojima će se raditi.

Frontend timovi lako mogu testirati aplikacije kreiranjem lažnih podataka, a kada serverska strana bude gotova, veoma lako je izvršiti prelaz na rad sa pravim podacima.

4. Osnovni koncepti

4.1. Schema Definition Language (SDL)

GraphQL ima svoje tipove koje koristi kako bi definisao šemu API-ja. Sintaksa za kreiranje šeme naziva se Schema Definition Language (SDL).

Primer kako pomoću SDL-a možemo definisati jednostavan tip ***Person***:

```
type Person {
    id: ID!
    name: String!
    age: Int!
}
```

Znak **!** pored tipova označava da je to polje obavezno (non-nullable). Polje id se automatski popunjava pošto je tipa **ID**, ovo je praksa koja se često koristi pri radu sa GraphQL-om.

Veoma lako možemo napraviti i relacije između tipova. Tako na primer možemo nastaviti sa aplikacijom za pisanje blogova, pa jedna osoba može imati više objava. Napravićemo nov tip **Post**:

```
type Post{
    id: ID!
    title: String!
    author: Person!
}
```

Drugi kraj relacije odnosi se na osobu i ona može imati više objava, dakle, proširićemo tip **Person** za polje **posts**:

```
type Person {
    name: String!
    age: Int!
    posts: [Post!]!
}
```

Upravo smo napravili **one-to-many** (jedan na prema više) relaciju između tipova **Person** i **Post**, pošto polje posts kod tipa **Person** predstavlja niz tipa **Post**.

4.2. Podaci i upiti

Kada dobijamo podatke pomoću REST API-ja, podaci se učitavaju sa određenih endpointa. Svaki endpoint ima nedvosmisleno definisanu strukturu podataka koju će vratiti.

Pristup koji koristi GraphQL je drugačiji, umesto postojanja više endpointa koji vraćaju fiksne strukture podataka, GraphQL API obično dostavlja samo jedan endpoint iz razloga što podaci koji se vraćaju nisu fiksni i rezultat zavisi samo od upita koji klijent formira.

Svi primeri koji su obrađeni, mogu biti testirani u playground okruženju na api.graph.cool.

4.2.1. Osnovni upiti

Zamislimo da su tabele **Person** i **Posts** popunjene podacima kaosa slike:

Person

id	name	age
cjv2ksl450hci01487z8ygpor	Johnny	23
cjv2ksl5b0hco0148u8sgj9dk	Sarah	20
cjv2ksl6a0hcs0148y59qqp75	Alice	20
cjv2ksmmw0i0001644v5an8vh	Alice	36

Post

id	title
cjv2ksl450hcj01481f46gvlz	GraphQL is awesome
cjv2ksl450hck0148fjnk0on1	Relay is a powerful GraphQL Client
cjv2ksl5b0hcp0148qvbzkxnp	How to get started with React & GraphQL

Slika 3. Tabele Person i Post sa podacima.

Posmatrajmo sledeće upite:

```
// Upit
{
    allPersons {
        name
    }
}

// Rezultat upita

{
    "data": {
        "allPersons": [
        {
            "name": "Johnny"
        },
        {
            "name": "Sarah"
        },
        {
            "name": "Alice"
        },
        {
            "name": "Alice"
        }
        ]
    }
}
```

Polje **allPersons** u ovom upitu naziva se root polje ili koren, sve što prati root polje naziva se payload upita. U ovom upitu payload jeste polje name.

Obratiti pažnju da smo kao odgovor dobili u rezultatima samo polje name, ukoliko želimo da dobijemo i vrednosti za godine, potrebno je upit proširiti na sledeći način:

```
// Upit

{

    allPersons {
        name
        age
    }
}

// Rezultat upita

{
    "data": {
        "allPersons": [
            {
            "name": "Johnny",
            "age": 23
        },
        {
            "name": "Sarah",
            "age": 20
        },
        {
            "name": "Alice",
            "age": 20
        },
        {
            "name": "Alice",
            "age": 36
        }
    ]
}
```

4.2.2. Upiti sa više informacija

Jedna od glavnih prednosti GraphQL-a ogleda se u tome što **omogućava lako pretraživanje ugneždenih informacija**. Na primer, ukoliko želimo da dobijemo sve članke koje je napisala, potrebno je samo da ispratimo strukturu tipova u kojima se nalaze potrebne informacije.

```
// Upit




{

    allPersons {
        name
        age
        posts {
            title
        }
    }
}

// Rezultat upita

{
    "data": {
        "allPersons": [
        {
            "name": "Johnny",
            "age": 23,
            "posts": [
            {
                "title": "GraphQL is awesome"
            },
            {
                "title": "Relay is a powerful GraphQL Client"
            }
            ]
        },
        {
            "name": "Sarah",
            "age": 20,
            "posts": [
            {
                "title": "How to get started with React & GraphQL"
            }
            ]
        }
        ]
    }
}
```

4.2.3. Upiti s argumentima

U GraphQL-u svako polje može imati nula ili više argumenata ukoliko je to navedeno u šemi. Tako na primer **allPersons** polje može imati last parametar kako bi vratio određen broj poslednje upisanih osoba.

```
// Upit
{
     allPersons(last: 2) {
          name
     }
}

// Rezultat upita

{
     "data": {
          "allPersons": [
          {
               "name": "Alice"
          },
          {
               "name": "Bob"
          }
          ]
     }
}
```

4.3. Unos podataka i mutacije

Pored prikaza podataka, za očekivati je da je podržana i modifikacija istih, ovo je omogućeno pomoću **mutacija**.

Postoji tri tipa mutacija:

- Kreiranje novih podataka
- Izmena postojećih podataka
- Brisanje postojećih podataka

Mutacije prate istu strukturu kao i upiti, jedina razlika jeste u tome što se na početku nalazi ključna reč mutation.

Primer mutacije:

```
// Mutacija
mutation {
     createPerson(name: "Bob", age: 36) {
          name
          age
     }
}
// Rezultat mutacije

{
     "data": {
    "createPerson": {
      "name": "Bob",
      "age": 36
    }
}
```

Slično kao i kod upita, mutacije takođe imaju root polje, u ovom slučaju to je **createPerson**. Polje createPerson ima dva argumenta koja određuju ime i godine.

Kao i kod upita moguće je specificirati payload za mutacije gde možemo zatražiti određena polja. U ovom slučaju, tražimo polja **name** i **age**, što nam i nije korisno, ali kod komplikovanijih mutacija itekako može biti korisno dobiti određene podatke kroz jedan zahtev. Primer jeste da kroz payload dobijemo **id** novog zapisa.

4.4. Izmene u realnom vremenu s pretplatom

Bitna opcija za mnoge savremene aplikacije jeste mogućnost da u realnom vremenu budu obaveštene o bitnim događajima, zato je u GraphQL-u implementiran **koncept pretplate na događaje tj. subscriptions**.

Kada se klijent pretplati na događaj, iniciraće i održavati konekciju sa serverom. Kada se desi određeni događaj, server će poslati odgovarajuće podatke klijentu. Za razliku od upita i mutacija koje prate tipičan „request-response-cycle“, pretplate predstavljaju tok podataka (stream of data) koji se šalje klijentu.

Pretplate koriste istu sintaksu kao i upiti i mutacije.

primer:

```
// Pretplata

subscription {
     newPerson {
          name
          age
     }
}

// Rezultat događaja

{
     "newPerson": {
          "name": "Jane",
          "age": 23
     }
}
```

Dakle, nakon što se otvori konekcija klijent-server, kada se kreira nova mutacija koja kreira novu osobu, server šalje informacije o osobi klijentu.

4.5. Definisanje šeme

Pošto smo obradili upite, mutacije i pretplate, možemo sve staviti u jednu celinu i videti kako se piše šema pomoću koje ćemo omogućiti izvršavanje napisanih primera. Šema je najbitniji koncept pri radu sa GraphQL API-jem. Ona definiše mogućnosti API-ja i definiše kako klijent može dobiti podatke. Obično označava ugovor između klijentske i serverske strane.

Uopšteno, šema predstavlja kolekciju GraphQL tipova. Međutim, kada se piše šema za API, postoje određeni specijalni root tipovi:

```
type Query { ... }
type Mutation { ... }
type Subscription { ... }
```

Query, **Mutation** i **Subscription** tipovi su samo ulazne tačke za zahteve poslate od strane klijenta. Kako bi se omogućio upit allpersons koji smo ranije napisali, Query tip mora biti napisan na sledeći način:

```
type Query {
  allPersons: [Person!]!
}
```

allPersons se naziva root poljem API-ja. Posmatrajući primer koji smo iznad napisali, dodali smo ragument last allPersons polju, tako da moramo proširiti tip na sledeći način:

```
type Query {
  allPersons(last: Int): [Person!]!
}
```

Slično važi i za createPerson mutaciju, potrebno je dodati root polje mutacionom tipu:

```
type Mutation {
  createPerson(name: String!, age: Int!): Person!
}
```

Na kraju, potrebno je napisati deo za pretplate gde moramo dodati newPerson root polje:

```
type Subscription {
  newPerson: Person!
}
```

Kada sve spojimo, dobijamo celu šemu za sve upite i mutacije koje smo naveli u primerima iznad.

```
type Query {
  allPersons(last: Int): [Person!]!
}

type Mutation {
  createPerson(name: String!, age: Int!): Person!
}

type Subscription {
  newPerson: Person!
}

type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}

type Post {
  title: String!
  author: Person!
}
```

5. Šira slika (Arhitektura)

GraphQL je predstavljen samo kao specifikacija. Ovo u suštini znači da GraphQL ne predstavlja ništa više nego dug dokument koji opisuje ponašanje GraphQL servera.

Obradićemo tri različite arhitekture koje uljučuju GraphQL server:

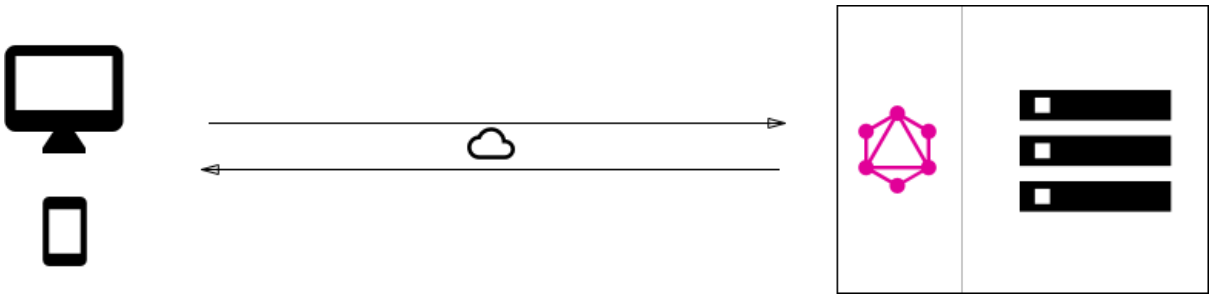
- GraphQL server povezan sa bazom podataka
- GraphQL server koji predstavlja međusloj između klijentske/klijentskih aplikacija i legacy sistema, mikroservisa i drugih API-ja kojima se pristupa preko jednog GraphQL API-ja.
- Hibridni pristup sa bazom podataka i legacy i drugim sistemima kojima se može pristupiti preko istog GraphQL API-ja

5.1. GraphQL server sa bazom podataka

Ovaj pristup se koristi kod tzv *greenfield* projekata, tj. kod projekata koji ne moraju da se restruktuiraju. Okruženje sadrži jedan (web) server koji implementira GraphQL specifikaciju. Kada upit stigne na GraphQL server, isti čita payload i dovlači odgovarajuće podatke iz baze, taj deo naziva se resolving upita.

Treba napomenuti da GraphQL zapravo predstavlja transportni sloj, što znači da se potencijalno može koristiti za bilo koji mrežni protokol. Tako da GraphQL server možemo implementirati na TCP-u, WebSoketima itd.

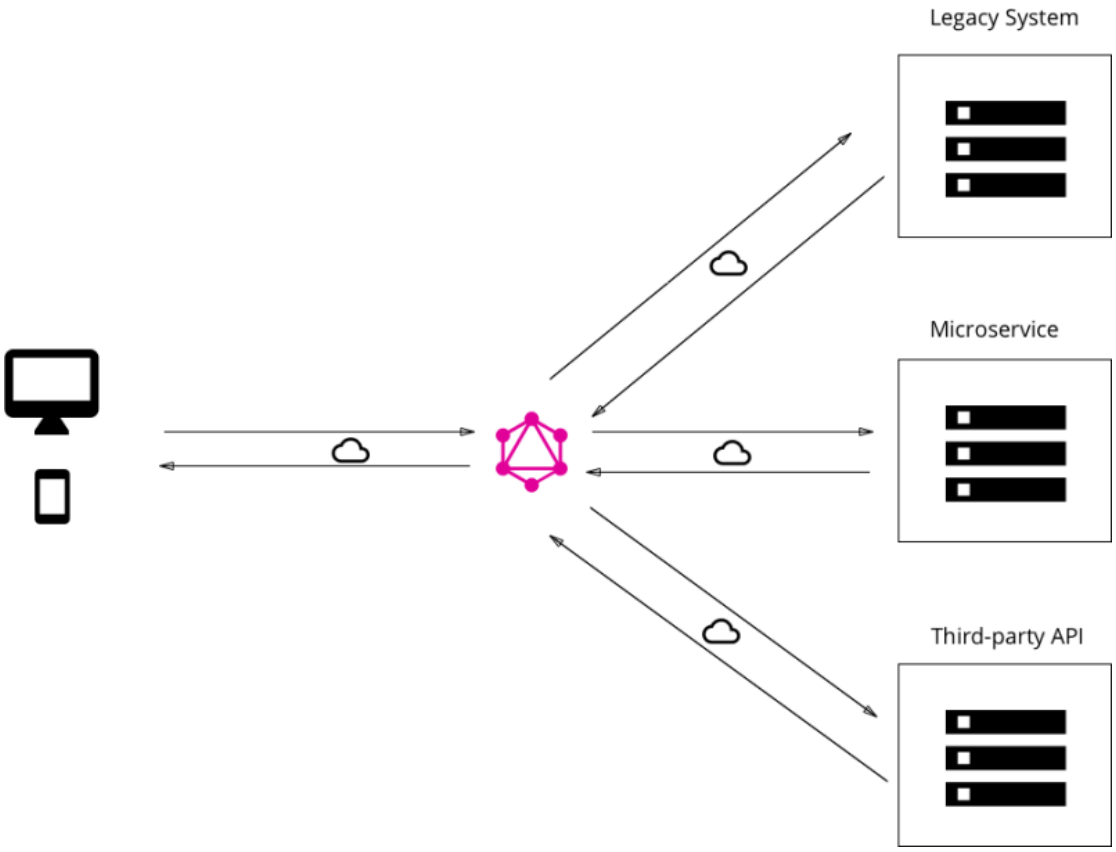
GraphQL ne zanima kakva je baza ili u kom su formatu zapisani podaci. Moguće je koristiti SQL ili NoSQL baze, apsolutno je nebitno.



Slika 4. Najčešće implementirana arhitektura GraphQL servera koji se povezuje sa jednom bazom podataka.

5.2. GraphQL kao sloj koji integriše postojeći sistem

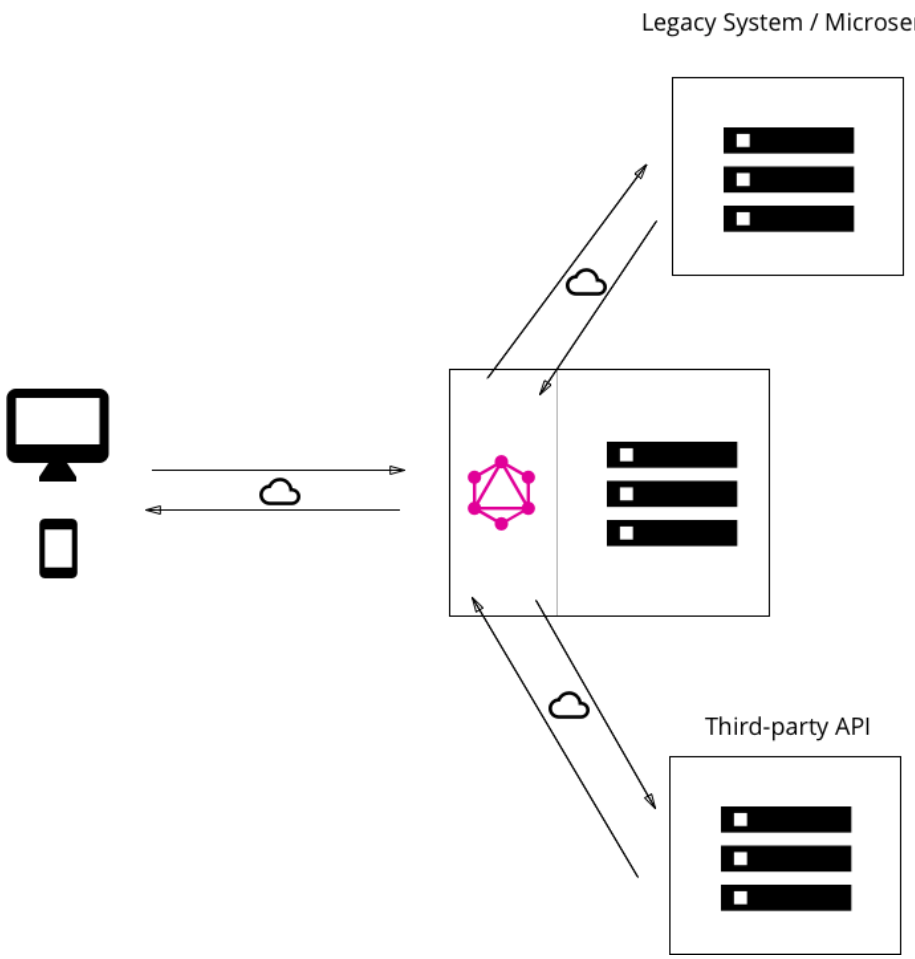
Takoće jedna od čestih upotreba GraphQL-a jeste integracija više postojećih sistema iza jednog, koherentnog, GraphQL API-ja. Ovo zapravo znači da je moguće iskoristiti u potpunosti postojeću infrastrukturu i više različitih API-ja koji su godinama razvijani i spojiti ih u jednu celinu. Na ovaj način moguće je ujediniti postojeće sisteme i sakriti kompleksnost iza GraphQL API-ja. Na ovaj način mogu se razviti nove klijentske aplikacije koje će na jednostavan način komunicirati sa GraphQL serverom kako bi dobile željene podatke.



Slika 5. GraphQL omogućava sakrivanje kompleksnosti postojećih sistema, poput mikroservisa, legacy infrastrukture i drugih API-ja.

5.3. GraphQL i hibridni pristup

Kao što i samo ime kaže, ovo zapravo predstavlja kombinovani pristup već pomenutih pristupa, gde GraphQL server može biti povezan sa bazom podataka ali takođe i da komunicira sa drugim sistemima. Kada je upit dobijen, pomoću resolver funkcije određuje se kako će isti biti obrađen.



Slika 6. Kombinovani pristup gde GraphQL dovlači podatke iz baze podataka kao i iz drugih sistema.

5.4. Resolver funkcije

Sve što je već navedeno zvuči lepo, velika fleksibilnost, nije bitno kog je tipa baza podataka, takođe nije bitno da li se radi sa legacy sistemima, mikroservisima itd. sve je rešeno resolver funkcijama. Međutim, postavlja se pitanje šta su zapravo one, i zašto su toliko moćne?

Kao što je već obrađeno, payload GraphQL upita (ili mutacija) sastoji se od niza polja. U serverskoj implementaciji GraphQL-a, svako od pomenutih polja zapravo odgovara tačno jednoj funkciji koja se poziva kao resolver. Svrha resolver funkcija jeste da dovlači podatke za odgovarajuća polja.

Kada server dobije upit, on će pozvati sve funkcije za polja koja su navedena u payload-u upita. Na taj način resolver može za upit dobiti odgovarajuće podatke za svako polje, server će upakovati podatke u formatu koji je opisan upitom i poslati ga nazad klijentu.

```
Query: {
  human(obj, args, context, info) {
    return context.db.loadHumanByID(args.id).then(
      userData => new Human(userData)
    )
  }
}
```

Primer resolvera napisanog u javaScript-u, međutim, GraphQL serveri mogu biti napisani u mnogim jezicima, tako da je ovo navedeno samo kao primer.

Slika 7. Svako polje u upitu odgovara jednoj resolver funkciji. GraphQL.

6. Zaključak i šta dalje

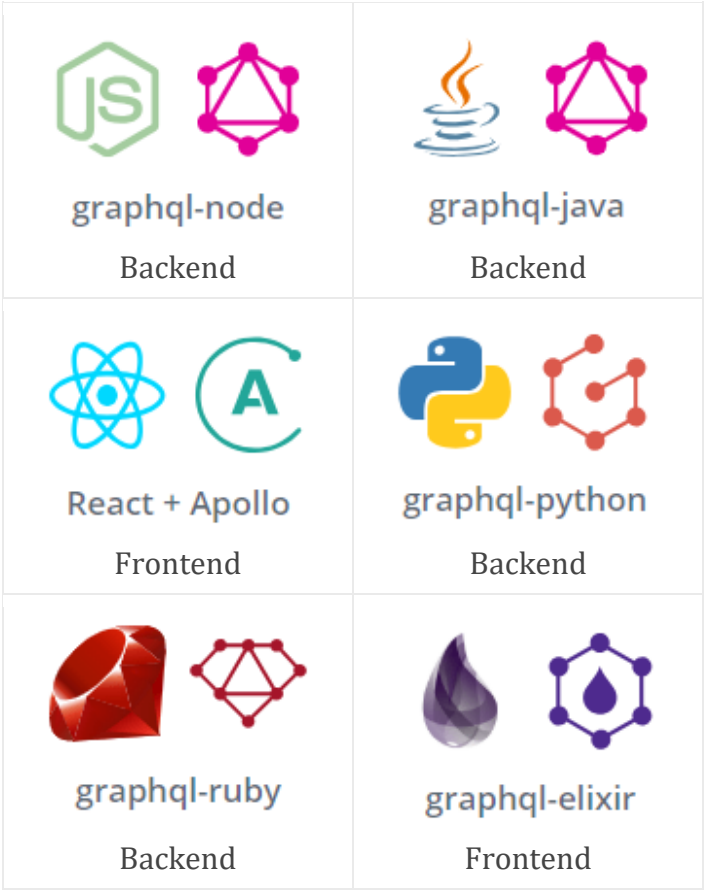
GraphQL predstavlja odličan alat za frontend developere pošto u potpunosti eliminiše određene probleme koji postoje sa REST API-jima, poput over i underfetching-a. Kompleksnost je prebačena na serversku stranu koja se nalazi na daleko moćnijim mašinama koje preuzimaju na sebe teži, proračunski deo. Klijent ne mora da zna odakle i na koji način su podaci dobijeni.

Zahvaljujući pristupu deklarativnog dobijanja podataka, klijent ima samo obavezu da uradi sledeća dva koraka:

- Opis podataka koji su potrebni
- Prikaz dobijenih podataka

GraphQL je još uvek relativno nov pristup, što znači da i dalje neće sve raditi savršeno i mogući su određeni problemi, takođe, određene struje smatraju da GraphQL zapravo i nema poentu, da predstavlja nepotrebnu tehnologiju, međutim kako vreme prolazi, takvih komentara je sve manje i manje.

Na linkovima ispod moguće je naći odlične tutorijale vezane za kreiranje i rad sa GraphQL serverom i upotrebu GraphQL-a za frontend development.



Literatura

[1] GraphQL

[2] Startit o GraphQL-u

[3] How To GraphQL

[4] Pet najčešćih problema u GraphQL aplikacijama (i kako ih rešiti)

[5] Exploring GraphQL (video)

[6] Zero to GraphQL in 30 Minutes – Steven Luscher (video)

[7] GraphQL: Designing a Data Language – Lee Byron (video)