Uvod u GraphQL

s primjerima korištenja u različitim tehnologijama

Nenad Crnko

Zagreb, 7.9.2018.

Sadržaj

Uvod (što je GraphQL i kako je nastao)

Usporedba s drugim sličnim tehnologijama

Primjeri korištenja u programskom jeziku JavaScript

Primjeri korištenja na relacijskoj bazi podataka MySQL pomoću programskog jezika PHP

Primjeri korištenja na dokumentno orijentiranoj bazi podataka MongoDB

Uvod
(Što je GraphQL
i kako je nastao)

Što je GraphQL

GraphQL predstavlja posebnu vrstu jezika upita pomoću koje aplikacija na strani klijenta na fleksibilan, ali istovremeno i optimiziran način, postavlja upite serveru, a server vraća rezultate obrade klijentu. Upiti se ne postavljaju izravno na bazu podataka nego na API podsustav.

U svojim mobilnim aplikacijama Facebook koristi ovu tehnologiju od 2012. godine.

GraphQL postaje javno dostupan 2015. godine

Ključne karakteristike

GraphQL upit vraća uvijek točno tražene i predvidljive podatke te ništa više ili manje od toga. Pri tome aplikacija na strani klijenta upravlja time kakve će podatke dobiti, a ne server, zbog čega je brza i stabilna u radu.

Jednim GraphQL upitom može se dobiti velika količina podataka zajedno s vezama između podataka (prednost u odnosu na višestruke REST API pozive). Zato korištenje ove tehnologije može biti vrlo djelotvorno i na sporim (mobilnim) vezama.

GraphQL upiti organizirani su prema objektim i poljima kojima se pristupa preko jedinstvene adrese.

Ključne karakteristike

Razvoj novih GraphQL upita je jednostavniji, jer ne utječe na djelovanje starih upita, koji se i dalje mogu normalno izvoditi. Također, prekid mogućnosti korištenja starijih verzija upita je jednostavnija nego kod drugih tehnologija.

Tehnologija je dobro provjerena u praksi, a nakon što je postala javno dostupna moguće je napraviti njezinu implementaciju u različitim okruženjima i programskim jezicima: Python, PHP, Java, JavaScript, C# / .NET, Go, Ruby, …

Osnovni primjer „Hello world”

Upit:

```
query {
   hello
}
```

Odgovor:

```
{
   "data": {
   "hello": "Hello world!"
   }
}
```

Malo složeniji primjer upita

Upit na polje i objekt s navođenjem naziva operacije (obavezno samo kod većeg broja operacija, ali korisno kod traženja grešaka):

```
query HeroNameAndFriends {
  hero {
   # upit na polje
    name
    # upit na objekt
    friends {
      name
    }
  }
}
```

Rezultati malo složenijeg primjera upita

Odgovor (iz „Star Wars” baze podataka):

```
{
   "data": {
      "hero": {
         "name": "R2-D2",
          "friends": [
            {
             "name": "Luke Skywalker"
            },
            {
             "name": "Han Solo"
            },
```

Rezultati malo složenijeg primjera upita


```
            {
             "name": "Leia Organa"
            }
        ]
    }
  }
}
```


Korištenje argumenata u upitima

Zadavanje upita navođenjem argumenata:

```
query {
    # korištenje dva argumenta od
    # kojih je drugi enumerator
    human(id: "1000") {
        name
        height(unit: FOOT)
   }
}
```

Rezultati upita s argumentima


```
{
  "data": {
    "human": {
      "name": "Luke Skywalker",
      "height": 5.6430448
    }
  }
}
```


Korištenje aliasa u upitima

Rješavanje problema ponavljanja polja u upitima:

```
query {
  empireHero: hero(episode: EMPIRE) {
    name
  }
  jediHero: hero(episode: JEDI) {
    name
  }
}
```

Rezultati upita s aliasima


```
{
  "data": {
    "empireHero": {
      "name": "Luke Skywalker"
    },
    "jediHero": {
      "name": "R2-D2"
    }
  }
}
```


Korištenje fragmenata za skupove polja

Isti fragment koristi se na više mjesta u upitu:

```
query {
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}
fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

Rezultati upita s korištenjem fragmenata

```
{
  "data": {
    "leftComparison": {
      "name": "Luke Skywalker",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Han Solo"
        }, ...
     ]
    },
```

Rezultati upita s korištenjem fragmenata

```
"rightComparison": {
      "name": "R2-D2",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Luke Skywalker"
        },...
     ]
    }
  }
}
```

Korištenje varijabli u upitima

Umjesto pisanja novog upita mijenja se samo varijabla:

```
query HeroNameAndFriends($episode:
 Episode) {
  hero(episode: $episode) {
    name
    friends {
      name
    }
  }
}
{
  "episode": "JEDI"
}
```

Rezultati upita s varijablama


```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },...
```


Podrazumijevane vrijednosti varijabli

Varijablama koje se koriste kod upita mogu se zadati i podrazumijevane vrijednosti (ako su sve podrazumijevane, može samo naziv upita):

```
query HeroNameAndFriends($episode:
Episode = JEDI) {
  hero(episode: $episode) {
    name
    friends {
      name
    }
  }
}
```

Korištenje direktiva u upitima

Omogućavaju dinamičku izmjenu strukture upita (moguće vrijednosti `@include` i `@skip`):

```
query Hero($episode: Episode, $withFriends:
 Boolean!) {
  hero(episode: $episode) {
    name
    friends @include(if: $withFriends)    {
      name
    }
  }
}
{
  "episode": "JEDI",
  "withFriends": false
}
```

Mutacije za ažuriranje podataka

Za ažuriranje se može koristiti bilo koja vrsta upita, ali se preporučuje korištenje mutacija (sličnost REST/GET):

```
mutation CreateReviewForEpisode($ep: Episode!,
 $review: ReviewInput!) {
createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

Rezultati izvođenja mutacije

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great
movie!"
    }
  }
}
```

Važna razlika mutacija u odnosu na druge upite je da se one obavezno izvode slijedno, jedna iza druge.

Sheme i tipovi

Iako GraphQL uvijek vraća očekivane rezultate na strani klijenta, moguće je precizno opisati skup mogućih podataka što dodatno olakšava pisanje upita. Također, omogućava jednostavniju provjeru ispravnosti samih upita.

To se postiže definiranjem dostupnih tipova za upit (polja i objekti), odnosno definiranjem sheme upita.

Po svojoj strukturi sheme GraphQL upita slične su samim upitima, ali su istovremeno nezavisne od programskog jezika zaduženog za implementaciju upita u pozadini.

Primjer jednostavne definicije

Kod definiranja sheme mogu se koristiti različite vrste skalarnih tipova podataka kao u slijedećem primjeru:

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

Character	- tip GraphQL objekta
Name i appearsIn	- polja u tipu Character
String	- osnovni skalarnih tip podatka
String! *not nullable*	- uvijek vraća vrijednost ili pogrešku
[Episode]! *not nullable*	- polje objekata Episode

Podrazumijevana vrijednost

I ovdje se može definirati podrazumijevana vrijednost.

```
type Starship {
  id: ID!
  name: String!
  length(unit: LengthUnit = METER): Float
}
```

Dostupne skalarne vrijednosti

Int

32-bitna cjelobrojna vrijednost s predznakom.

Float

Broj s pokretnim zarezom i predznakom dvostruke preciznosti.

String

Niz znakova prema UTF-8 standardnu kodiranja.

Boolean

Logička vrijednost true ili false.

ID

Jedinstveni identifikator objekta (u stvari isto niz znakova)

Ostale mogućnosti

Moguće je definirati i vlastiti skalarni tip podatka:

```
scalar Date
```

Postoji i mogućnost definiranja vlastitih Enum tipova podataka s ograničenim skupom vrijednosti:

```
enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}
```

Ostale mogućnosti

Pomoću znakova [ ] mogu se definirati vlastite liste:

```
myField: [String!]
myField: null                  // OK
myField: []                    // OK
myField: ['a', 'b']            // OK
myField: ['a', null, 'b']      // error

myField: [String]!
myField: null                  // error
myField: []                    // OK
myField: ['a', 'b']            // OK
myField: ['a', null, 'b']      // OK
```

Definiranje apstraktnih tipova

Moguće je navođenjem oznake **interfaces**:

```
interface Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
}
```

Svi izvedeni tipovi moraju sadržati nabrojene osnovne dijelove.

Korištenje apstraktnih tipova

```
type Human implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  starships: [Starship]
  totalCredits: Int
}

type Droid implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction: String
}
```

Korištenje apstraktnih tipova

```
query HeroForEpisode($ep: Episode!) {
  hero(episode: $ep) {
    name
    primaryFunction
  }
}
{
  "errors": [
    {
      "message": "Cannot query field
\"primaryFunction\" on type \"Character\". Did you
mean to use an inline fragment on \"Droid\"?",
      "locations": [
        {
          "line": 4,
          "column": 5
        }
      ] …
```

Korištenje unija

Koristi se za vraćanje različitih vrsta objekata iz upita. Na primjer, sliedeća definicija:

```
union SearchResult = Human | Droid |
Starship
```

Omogućava vraćanje rezultata za bilo koji od dozvoljenih tipova.

Korištenje unija

```
{
  search(text: "an") {
    ... on Human {
      name
      height
    }
    ... on Droid {
      name
      primaryFunction
    }
    ... on Starship {
      name
      length
    }
  }
}
```

Korištenje unija

```
{
  "data": {
    "search": [
      {
        "name": "Han Solo",
        "height": 1.8
      },
      {
        "name": "Leia Organa",
        "height": 1.5
      },
      {
        "name": "TIE Advanced x1",
        "length": 9.2
      } …
```

Objekti kao ulazni tipovi podataka

Osim jednostavnih skalarnih vrijednosti kao ulazni tipovi podataka mogu se koristiti i objekti.

U tom slučaju u samoj definiciji treba **type** zamijeniti s **input**. Na primjer:

```
input ReviewInput {
  stars: Int!
  commentary: String
}
```

Na temelju toga može se definirati slijedeći oblik mutacije:

Objekti kao ulazni tipovi podataka

```
mutation CreateReviewForEpisode($ep:
Episode!, $review: ReviewInput!) {
  createReview(episode: $ep, review:
$review) {
    stars
    commentary
  }
}
```

I koristiti kao:

```
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

Objekti kao ulazni tipovi podataka

```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great
movie!"
    }
  }
}
```

Usporedbe s drugim sličnim tehnologijama (REST API)

Usporedba s REST API

Umjesto većeg broja krajnjih točaka s fiksnom strukturom podataka, GraphQL omogućava korištenje samo jedne krajnje točke s različitim i precizno definiranim strukturama podataka potrebnim klijentu.

U jednom GraphQL upitu mogu se sa servera zatražiti vrlo složeni podaci s međusobno manje ili više složenim vezama, a pri tome je je sam oblik upita pregledan te prilično jednostavan za razumijevanje. Server vraća odgovor koji po svojoj strukturi točno odgovara postavljenom upitu.

Usporedba s REST API

Način postavljanja upita serveru i vraćanja podataka sa servera bitno smanjuje količinu prometa po mreži (vrlo bitno kod sporijih mobilnih mreža za prijenos podataka). Nema „Overfetching” i „Underfetching”.

GraphQL Schema Definition Language povećava produktivnost je omogućava nezavisan rad na strani servera i na strani klijenta.

Mogući načini korištenja na strani servera

1. Izgradnja potpuno novog sustava koji preko GraphQL upita pristupa bazi podataka.

2. Pojednostavljivanje pristupa postojećim (legacy) sustavima tako da se pomoću GraphQL upita „skriva” njihova kompleksnost za aplikacije klijente.

3. Hibridni pristup – integracija prethodnih točki 1 i 2. Ovakva fleksibilnost moguća je izradom različitih **resolver** funkcija na strani servera.

Mogući načini korištenja na strani klijenta

Uz pomoć klijent orijentiranih GraphQL biblioteka kao što su:

Relay (https://facebook.github.io/relay/)

Apollo (https://www.apollographql.com/)

Pristup podacima iz aplikacije klijenta može se svesti na:

1. Opis zahtjeva za podacima
2. Prikaz podataka u korisničkom sučelju

Umjesto REST API (priprema HTTP zahtjeva, primanje i parsing podataka, lokalno spremanje podataka, prikaz u korisničko sučelju).

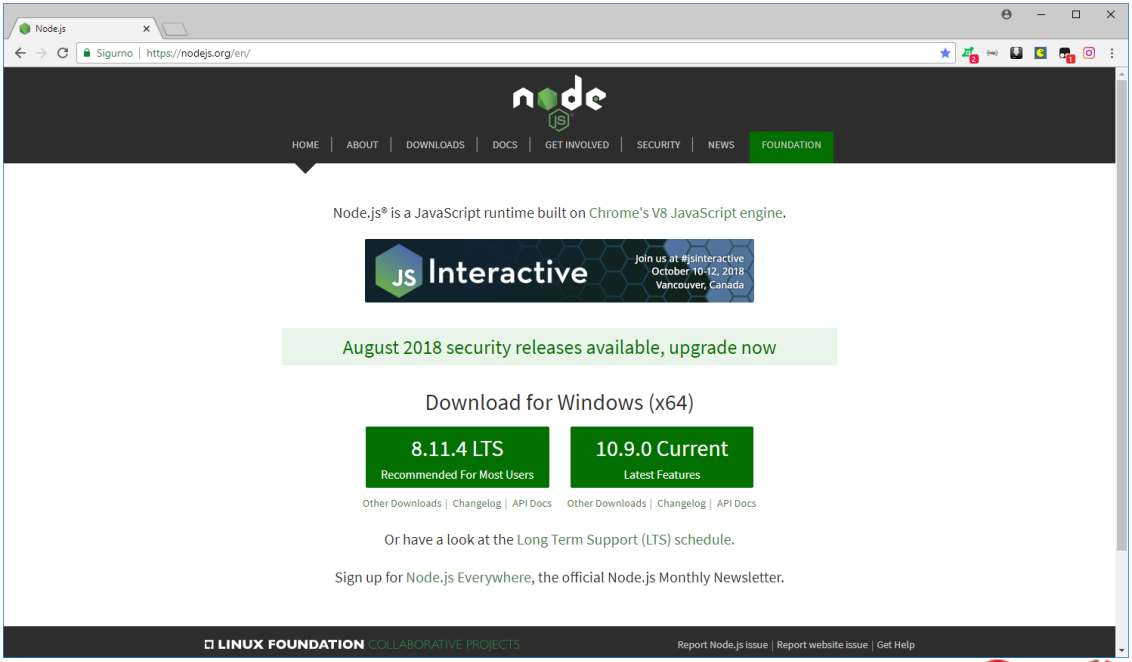
Primjer korištenja u biblioteci Relay

```
import {graphql} from 'react-relay';

graphql`
  query MyQuery {
    viewer {
      id
    }
  }
`;
```

Primjeri korištenja u programskom jeziku JavaScript

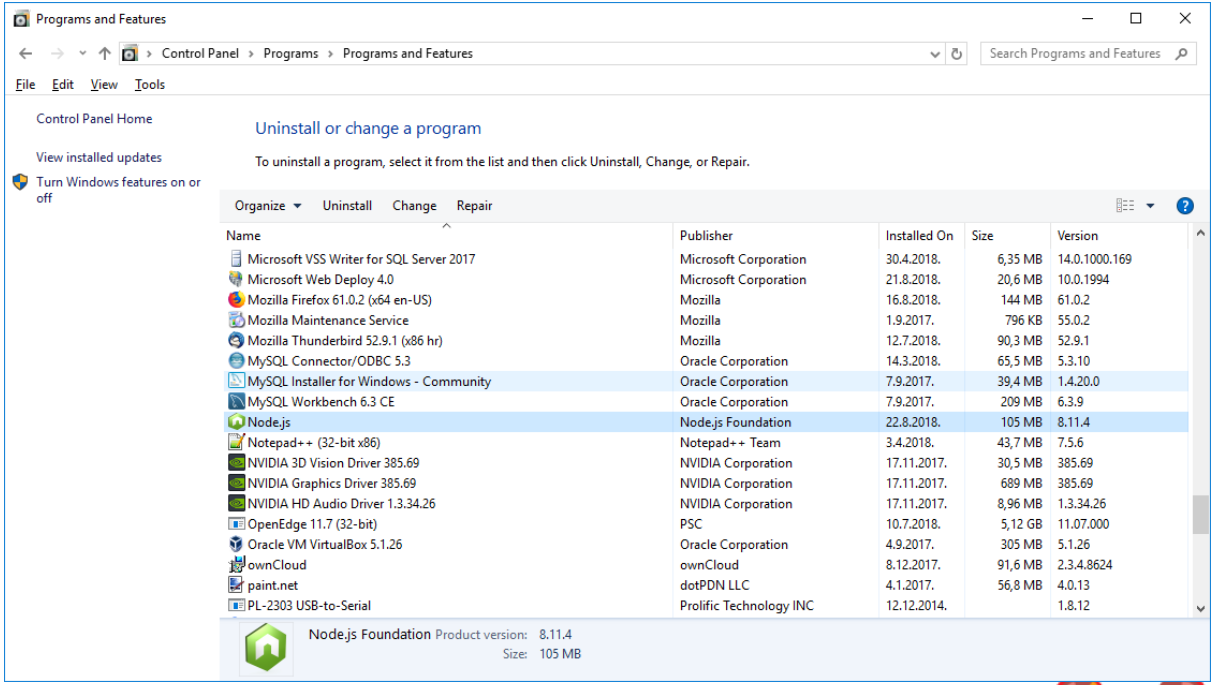
JavaScript – priprema Windows računala

Preuzimanje paketa **node.js**, ako ne postoji na računalu - https://nodejs.org/en/



JavaScript – priprema Windows računala

Instalacija paketa **node.js** izvodi se iz preuzete EXE datoteke.



JavaScript – osnovni primjer

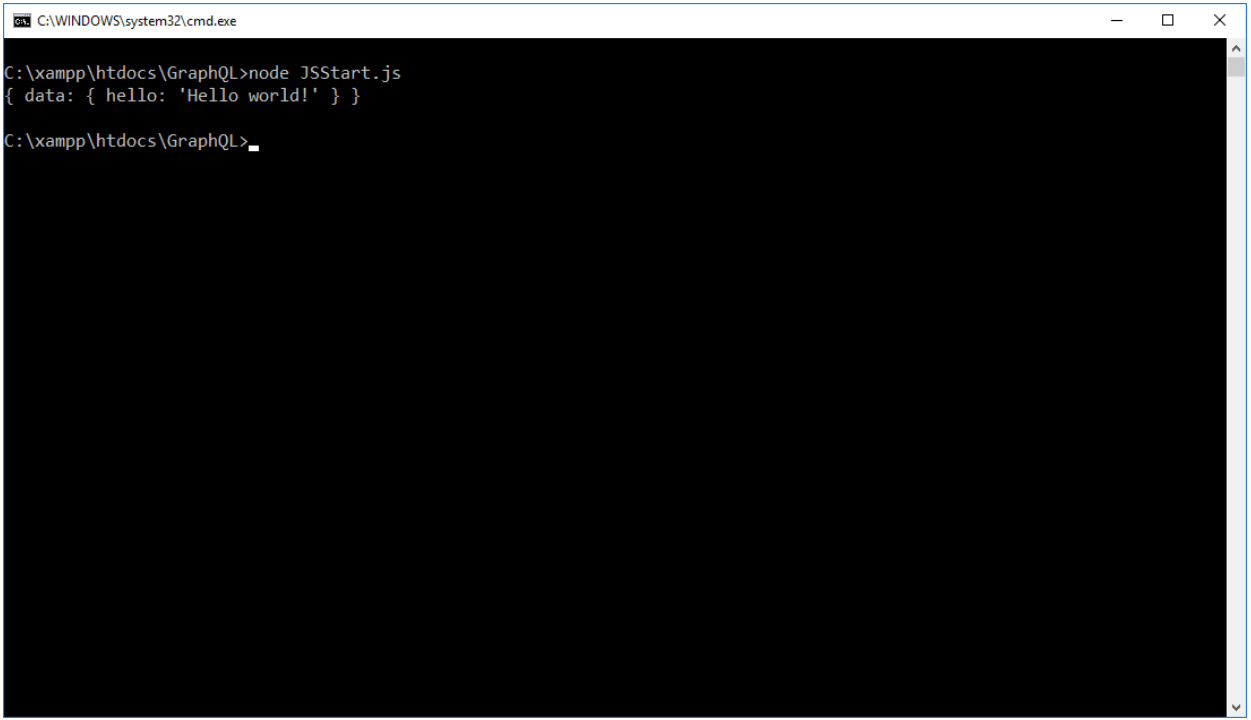
Programski kod primjera

```
var { graphql, buildSchema } =
require('graphql');

// Construct a schema, using GraphQL
schema language
var schema = buildSchema(`
  type Query {
    hello: String
  }
`);
```

JavaScript – osnovni primjer (nastavak)

```
// The root provides a resolver function
for each API endpoint
var root = {
  hello: () => {
    return 'Hello world!';
  },
};
// Run the GraphQL query '{ hello }' and
print out the response
graphql(schema, '{ hello }',
root).then((response) => {
  console.log(response);
});
```

JavaScript – osnovni primjer (izvođenje)



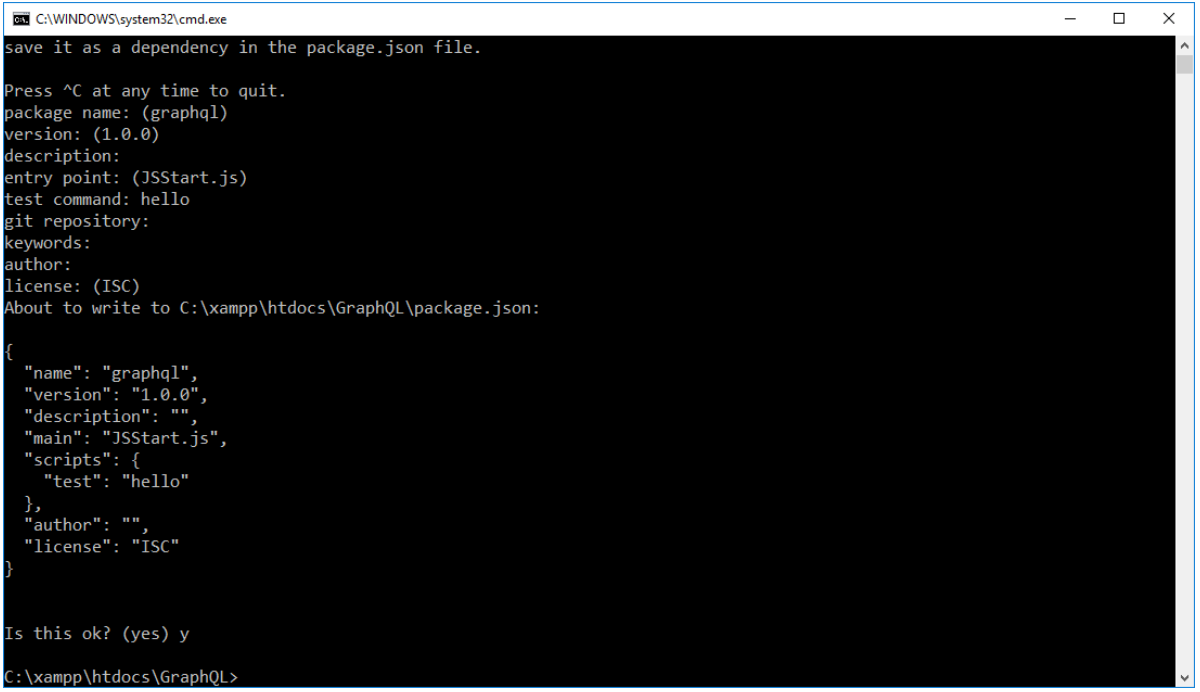
JavaScript – dodatni primjer

Koristi se dodatni web Framework za Node.js dostupan na adresi: https://expressjs.com/

Koraci:

1. Inicijalizacija projekta
2. Instalacija frameworka
3. Pisanje programskog koda
4. Izvođenje primjera

JavaScript – inicijalizacija projekta



JavaScript – inicijalizacija projekta

>npm init

```
This utility will walk you through
creating a package.json file.

It only covers the most common items, and
tries to guess sensible defaults.

See `npm help json` for definitive
documentation on these fields and exactly
what they do.

Use `npm install <pkg>` afterwards to
install a package and save it as a
dependency in the package.json file.
```

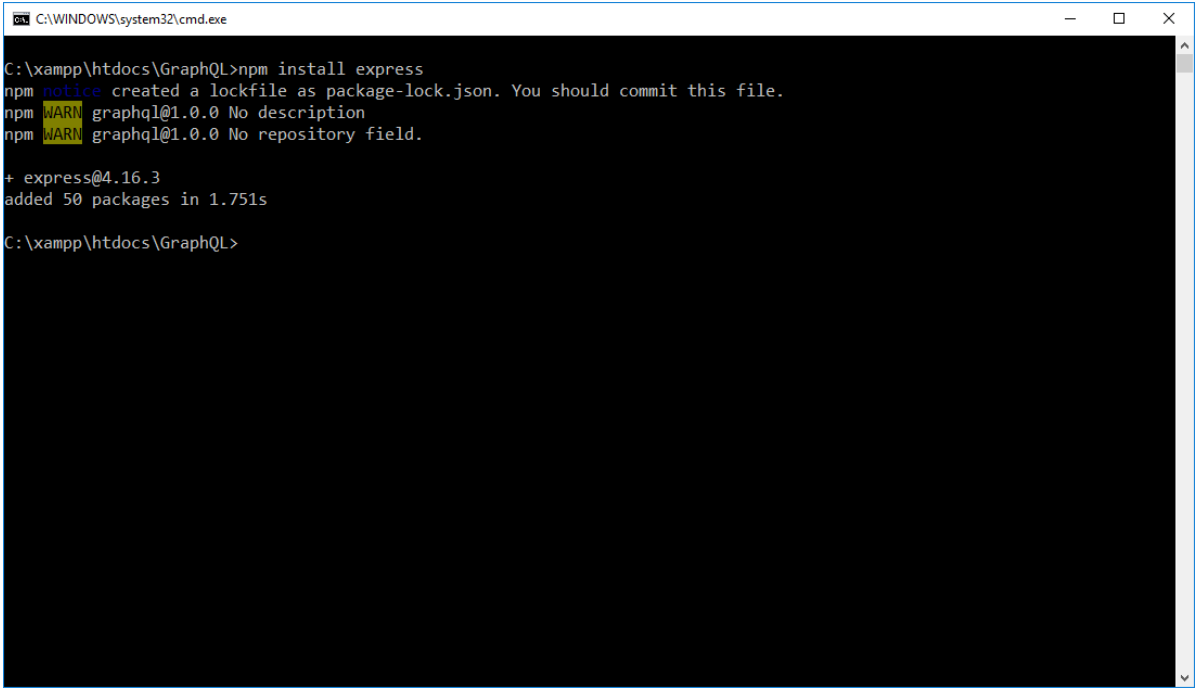
JavaScript – inicijalizacija projekta

```
Press ^C at any time to quit.
package name: (graphql)
version: (1.0.0)
description:
entry point: (JSStart.js)
test command: hello
git repository:
keywords:
author:
license: (ISC)
About to write to
C:\xampp\htdocs\GraphQL\package.json:
```

JavaScript – inicijalizacija projekta

```
{
  "name": "graphql",
  "version": "1.0.0",
  "description": "",
  "main": "JSStart.js",
  "scripts": {
    "test": "hello"
  },
  "author": "",
  "license": "ISC
}
Is this ok? (yes) y
```

JavaScript – instalacija frameworka



JavaScript – instalacija frameworka

>npm install express

```
npm notice created a lockfile as package-
lock.json. You should commit this file.
npm WARN graphql@1.0.0 No description
npm WARN graphql@1.0.0 No repository
field.

+ express@4.16.3
added 50 packages in 1.751s
```

JavaScript – programski kod

```
var express = require('express');
var graphqlHTTP = require('express-
graphql');
var { buildSchema } = require('graphql');

// Construct a schema, using GraphQL
schema language
var schema = buildSchema(`
  type Query {
    hello: String
  }
`);
```

JavaScript – programski kod

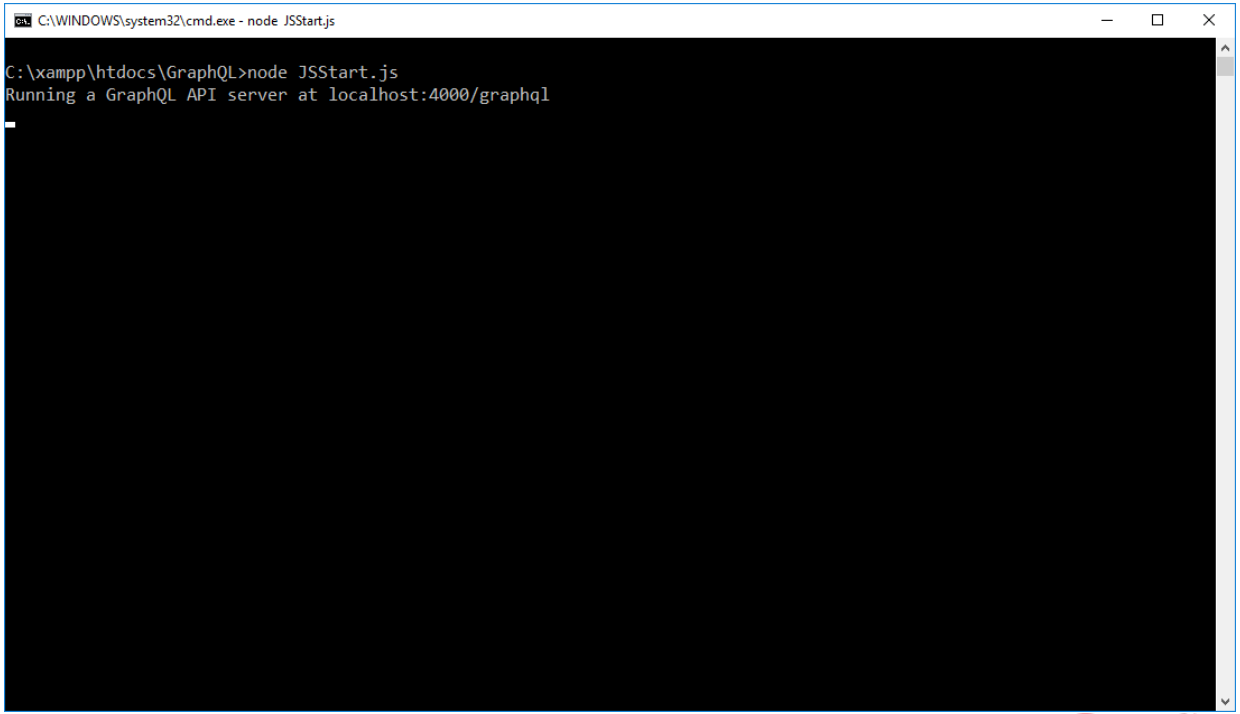
```
for each
API endpoint
var root = {
  hello: () => {
    return 'Hello world!';
  },
};
var app = express();
app.use('/graphql', graphqlHTTP({
  schema: schema,
  rootValue: root,
  graphiql: true,
}));
```

JavaScript – programski kod

```
app.listen(4000);
console.log('Running a GraphQL API server
at
localhost:4000/graphql');
```

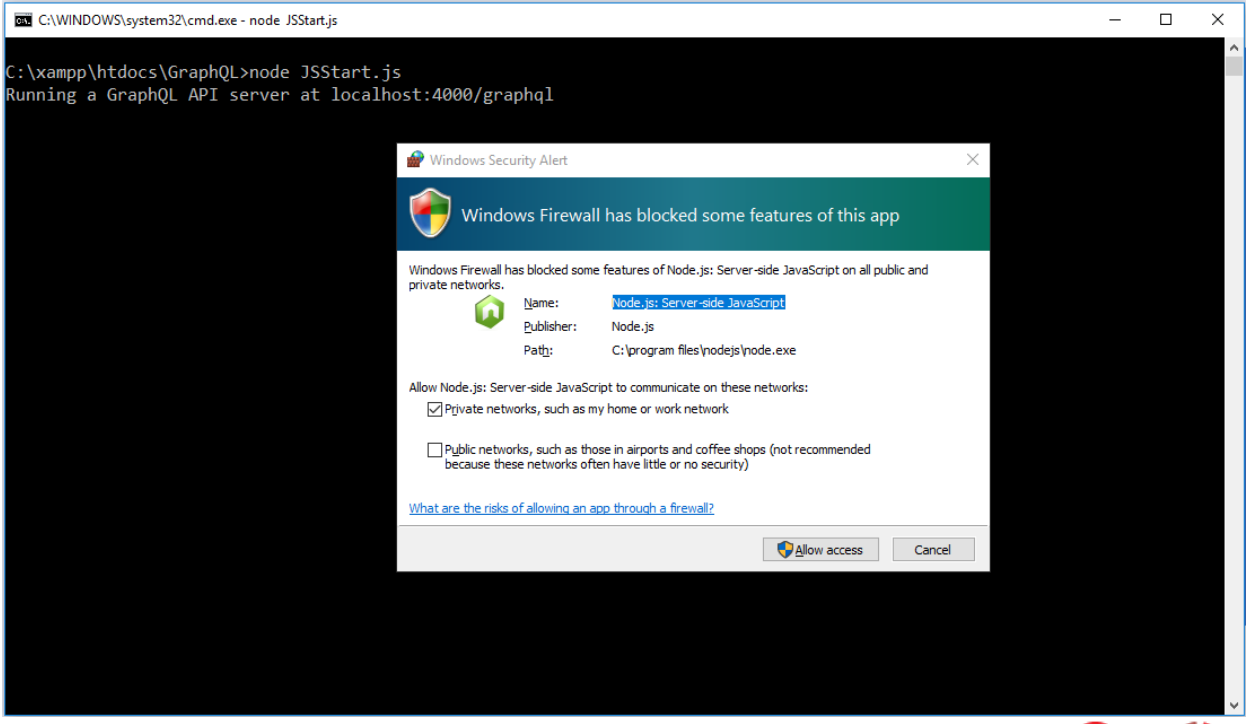
JavaScript – pokretanje primjera

> node JSStart.js



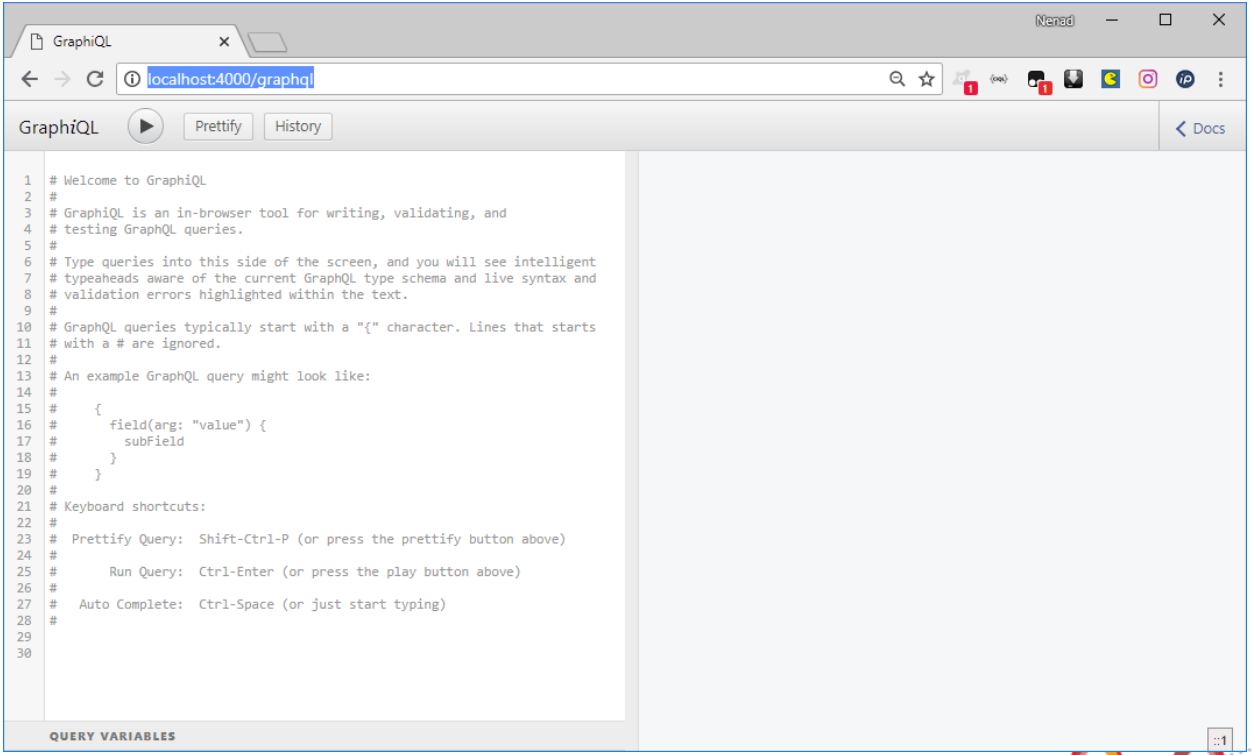
JavaScript – izvođenje (Firewall dozvola)

Pokretanje: **node JSStart.js**



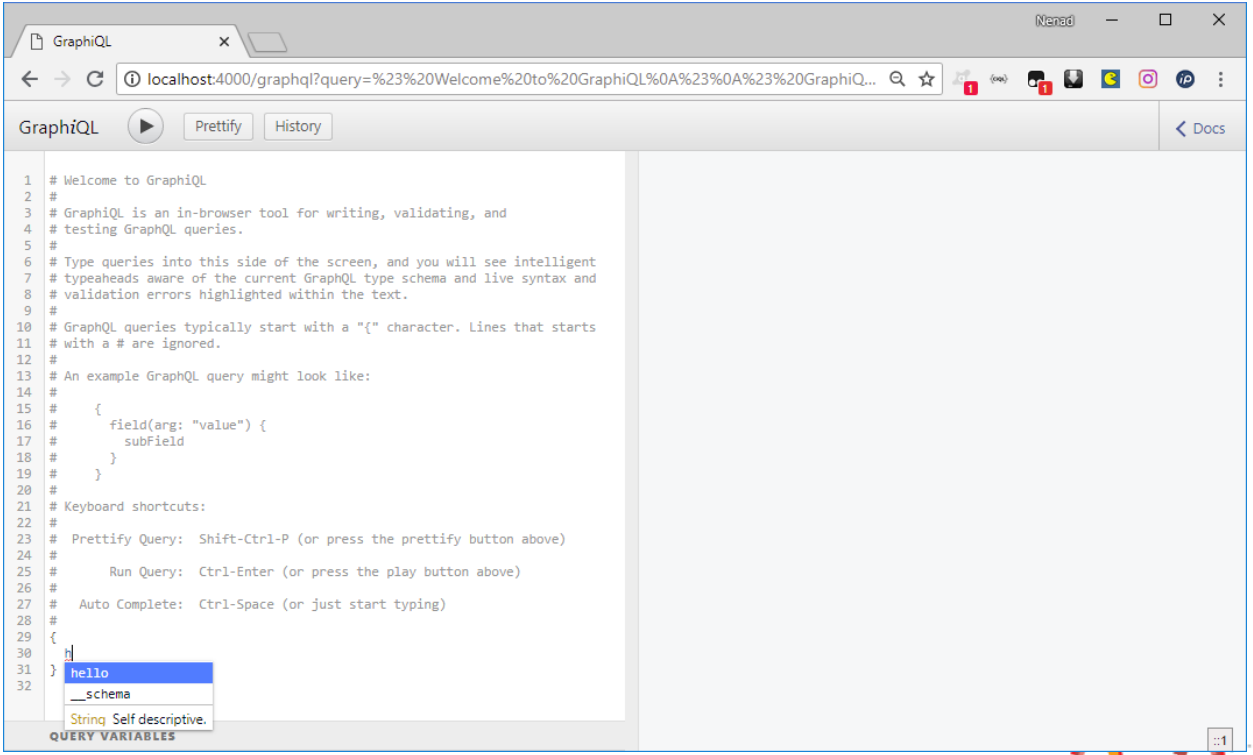
JavaScript – izvođenje u Chrome pregledniku

Upisuje se adresa: **localhost:4000/graphql**



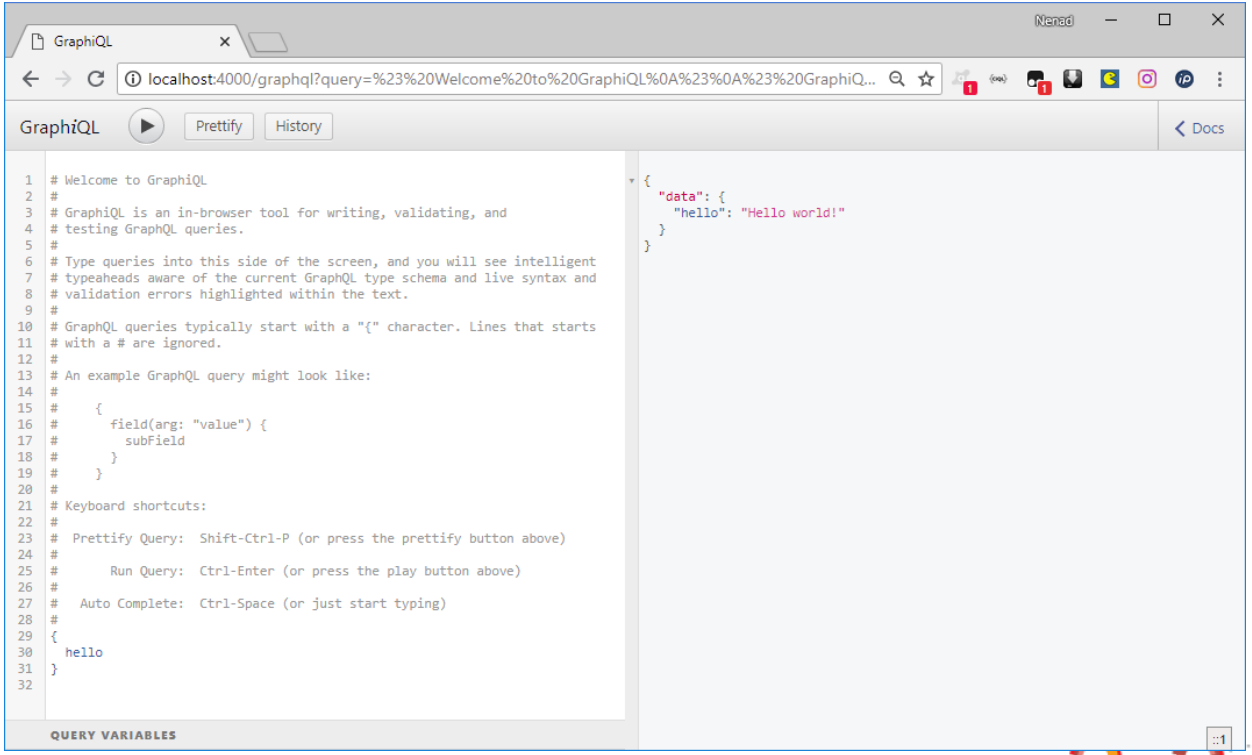
JavaScript – izvođenje u Chrome pregledniku

Upis i odabir dostupnih naredbi: **hello**



JavaScript – izvođenje u Chrome pregledniku

Izvođenje i prikaz rezultata



Primjeri korištenja na relacijskoj bazi podataka MySQL pomoću programskog jezika PHP

PHP – priprema računala za korištenje

Podrazumijeva se na je na računalu instaliran programski jezik PHP i baza podataka MySQL ili MariaDB (npr. XAMPP - www.apachefriends.org)

Dodatna PHP biblioteka za podršku GraphQL upita dostupna je na adresi:

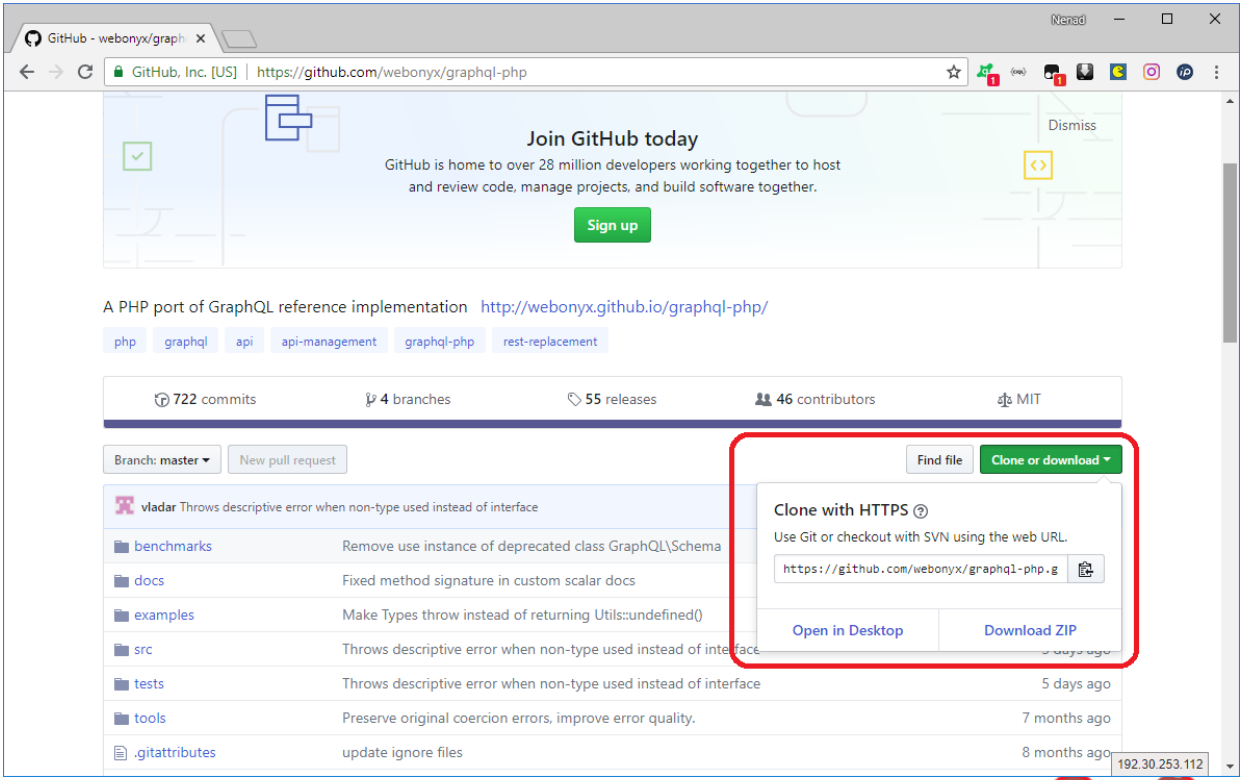
https://github.com/webonyx/graphql-php

Dva glavna oblika instalacije:

1. Korištenje gumba **Clone or Download**
2. Korištenje alata **Composer**

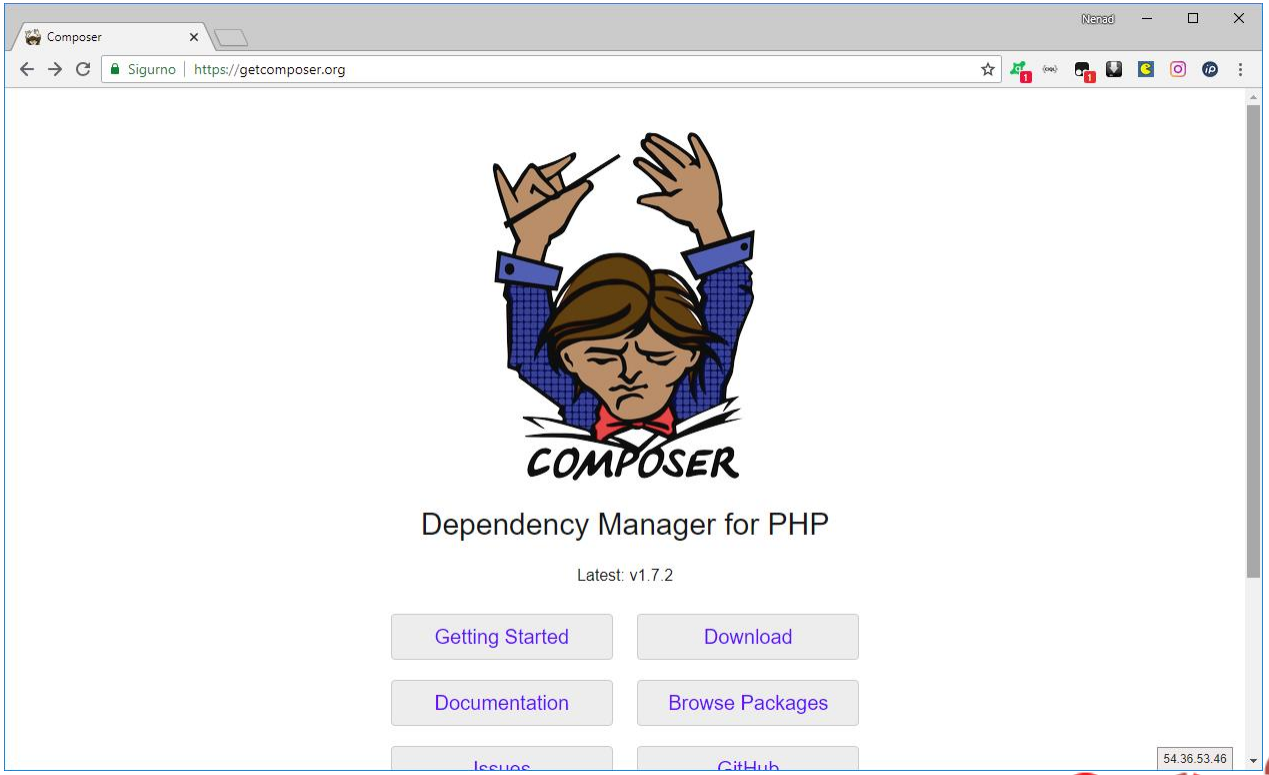
PHP – priprema računala za korištenje

Korištenje gumba **Clone or Download**



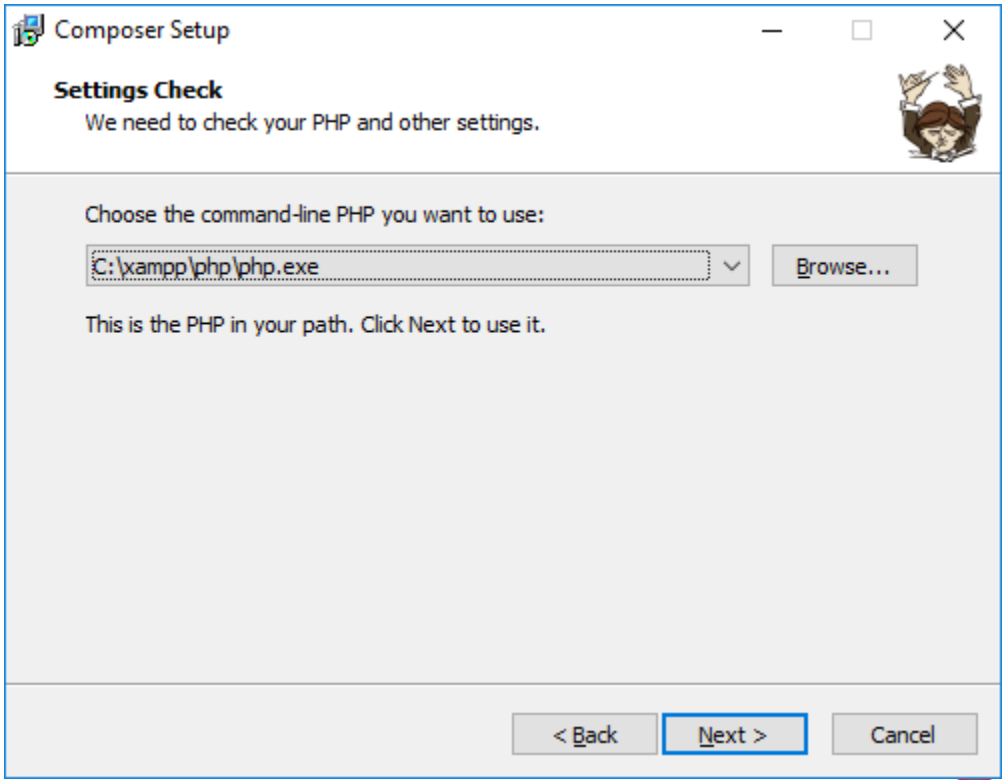
PHP – priprema računala za korištenje

Korištenje alata **Composer** (getcomposer.org)



PHP – priprema računala za korištenje

Composer – instalacija i povezivanje s PHP-om



PHP – priprema računala za korištenje

Preuzimanje biblioteke pomoću alata Composer na temelju datoteke **composer.json**

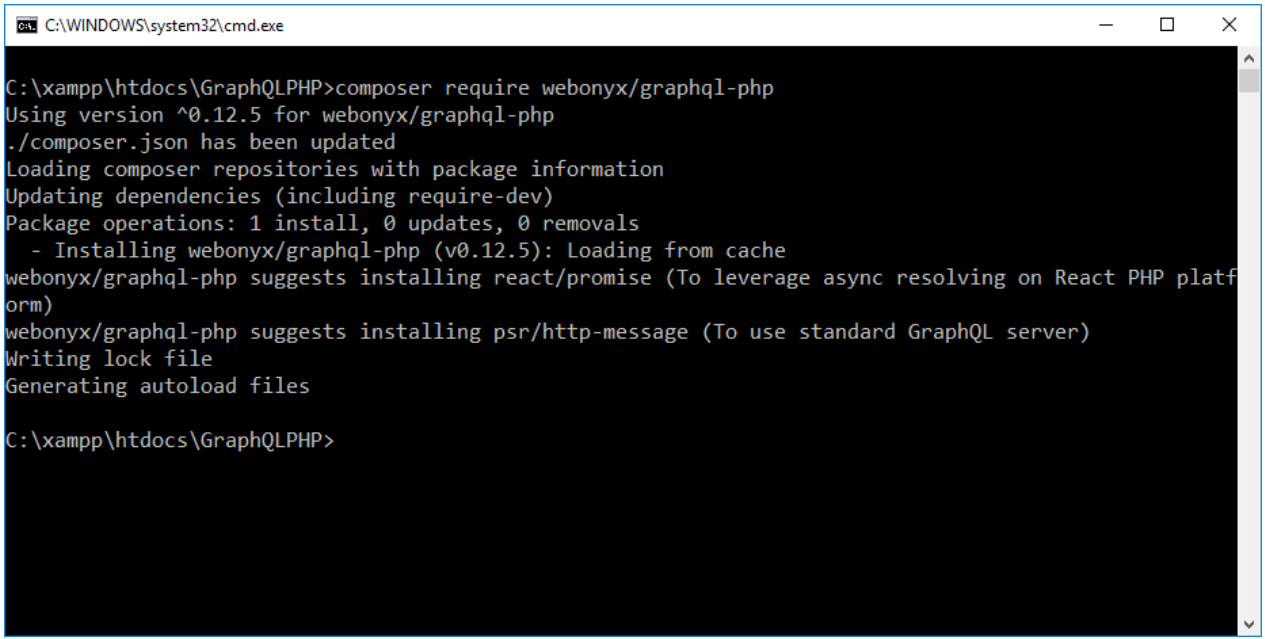
```
{
    "require": {
        "webonyx/graphql-php": "^0.12.5"
    }
}
```

> **composer require webonyx/graphql-php**

PHP – priprema računala za korištenje

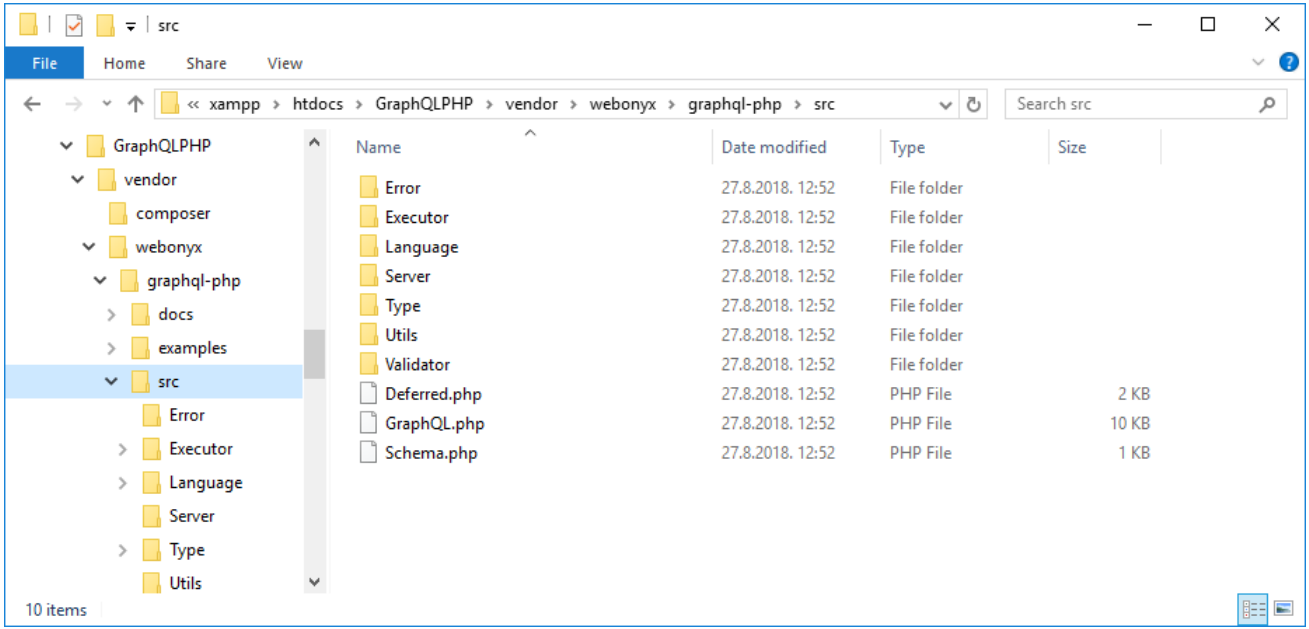
Preuzimanje biblioteke

> **composer require webonyx/graphql-php**



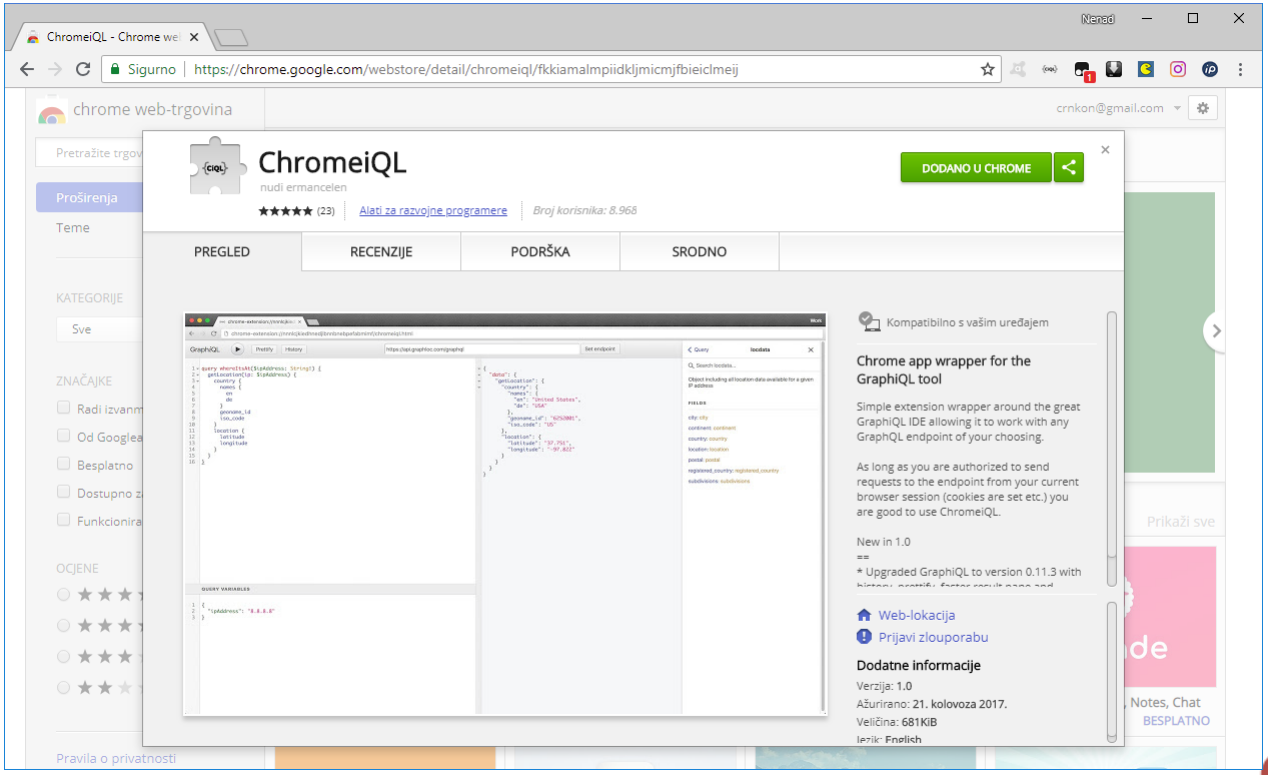
PHP – priprema računala za korištenje

Rezultati preuzimanja biblioteke



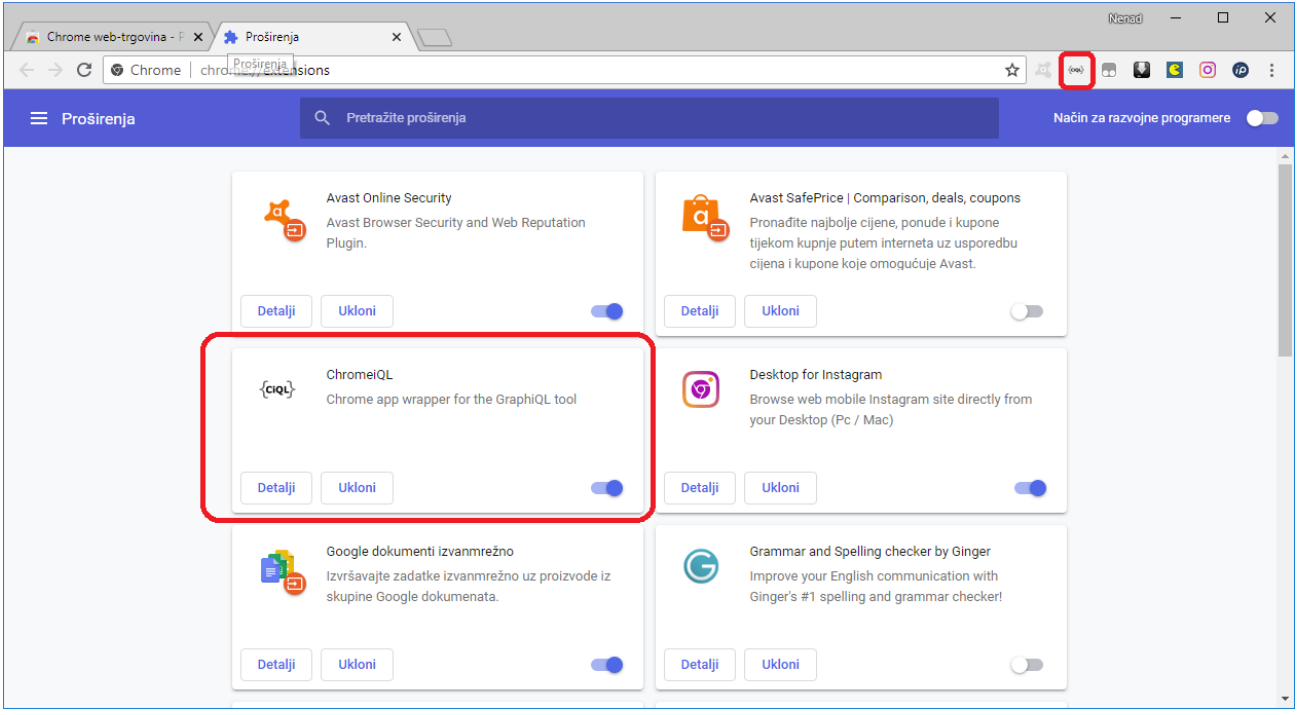
PHP – priprema računala za korištenje

Dodatni alat za Chrome: **ChromeiQL**



PHP – priprema računala za korištenje

Dodatni alat za Chrome: **ChromeiQL**



PHP – osnovni primjer

```
<?php

require_once __DIR__ . './autoload.php';

use GraphQL\Type\Definition\ObjectType;
use GraphQL\Type\Definition\Type;
use GraphQL\Type\Schema;
use GraphQL\GraphQL;

try {
    $queryType = new ObjectType([
        'name' => 'Query',
        'fields' => [
            'echo' => [
                'type' => Type::string(),
                'args' => [
```

PHP – osnovni primjer

```
'message' =>
['type' =>  Type::string()],],
 'resolve' => function ($root, $args) {
         return $root['prefix'] .
            $args['message'];
          }
       ],
    ],
]);
```

PHP – osnovni primjer

```
$mutationType = new ObjectType([
   'name' => 'Calc',
   'fields' => [
     'sum' => [
        'type' => Type::int(),
        'args' => [
            'x' => ['type' => Type::int()],
            'y' => ['type' => Type::int()],
         ],
        'resolve' => function ($root, $args) {
           return $args['x'] + $args['y'];
        },
     ],
   ],
]);
```

PHP – osnovni primjer

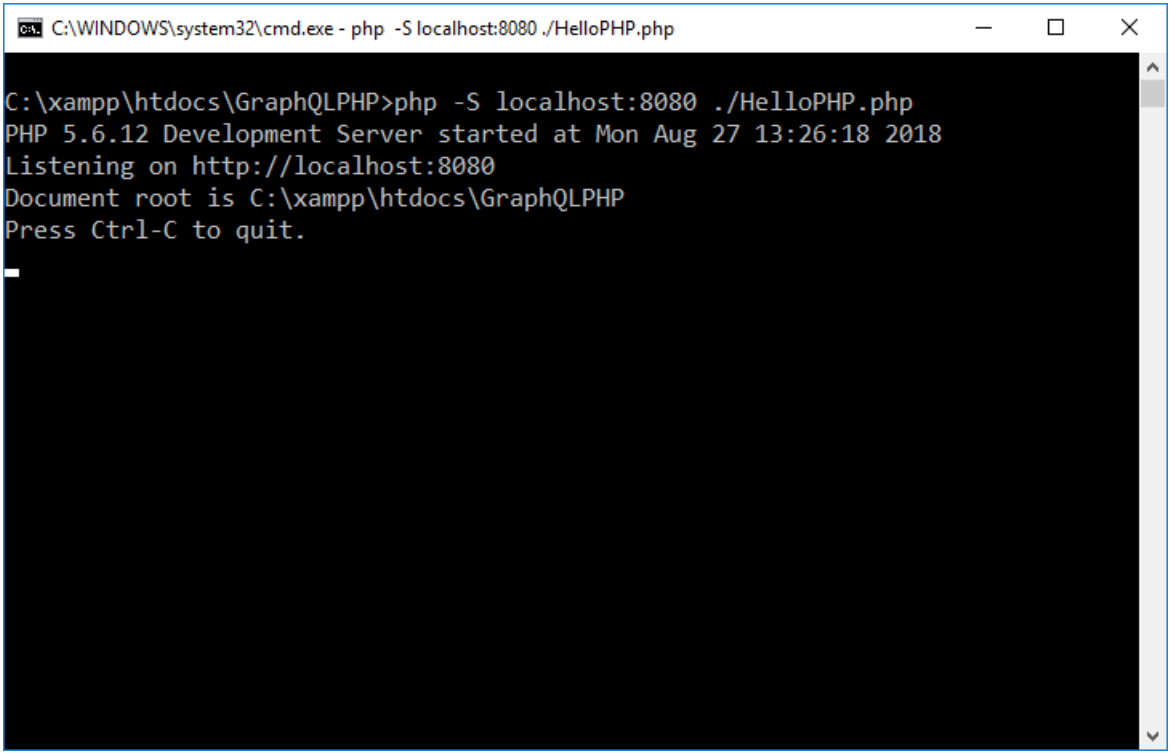
```
$schema = new Schema([
        'query' => $queryType,
        'mutation' => $mutationType,
    ]);

$rawInput
   =file_get_contents('php://input');
$input = json_decode($rawInput, true);
$query = $input['query'];
$variableValues
   =isset($input['variables']) ?
$input['variables'] : null;
$rootValue = ['prefix' => 'You said: ];
$result = GraphQL::executeQuery($schema,
   $query, $rootValue, null,
   $variableValues);
 $output = $result->toArray();
```

PHP – osnovni primjer

```
} catch (\Exception $e) {
    $output = [
        'error' => [
            'message' => $e->getMessage()
        ]
    ];
}
header('Content-Type: application/json;
charset=UTF-8');
echo json_encode($output);
```

PHP – izvođenje primjeraHelloPHP.php

> php -S localhost:8080 ./HelloPHP.php



PHP – izvođenje primjeraHelloPHP.php

Koristi se preglednik **Chrome** i dodatak **ChromeiQL**

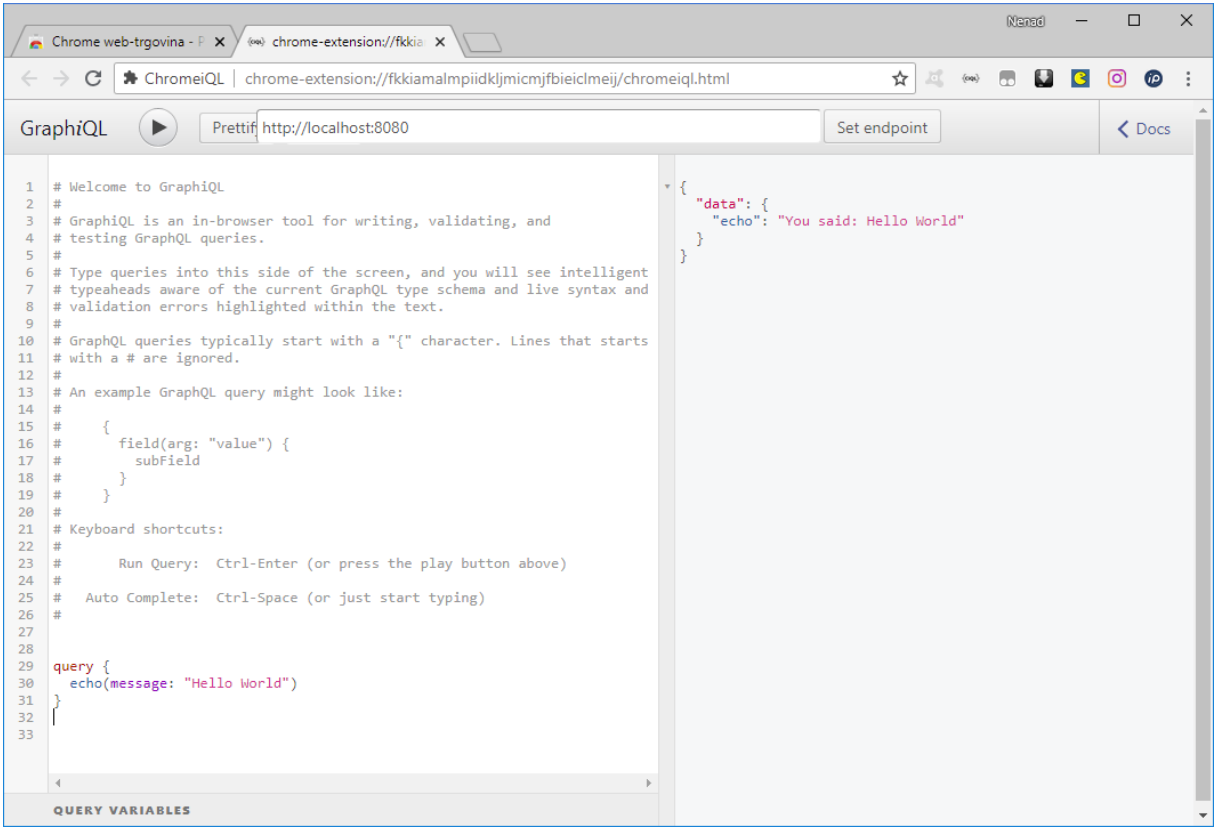
Za korištenje primjera treba upisati istu adresu kao i kod pokretanje primjera u PHP-u:

http://localhost:8080

U lijevi dio prozora upisuje se naredba:

```
query {
  echo(message: "Hello World")
}
```

PHP – izvođenje primjeraHelloPHP.php



PHP – izvođenje u ChromeiQL

Rezultat izvođenja

```
{
  "data": {
    "echo": "You said: Hello World"
  }
}
```

PHP – pogreška kod izvođenja

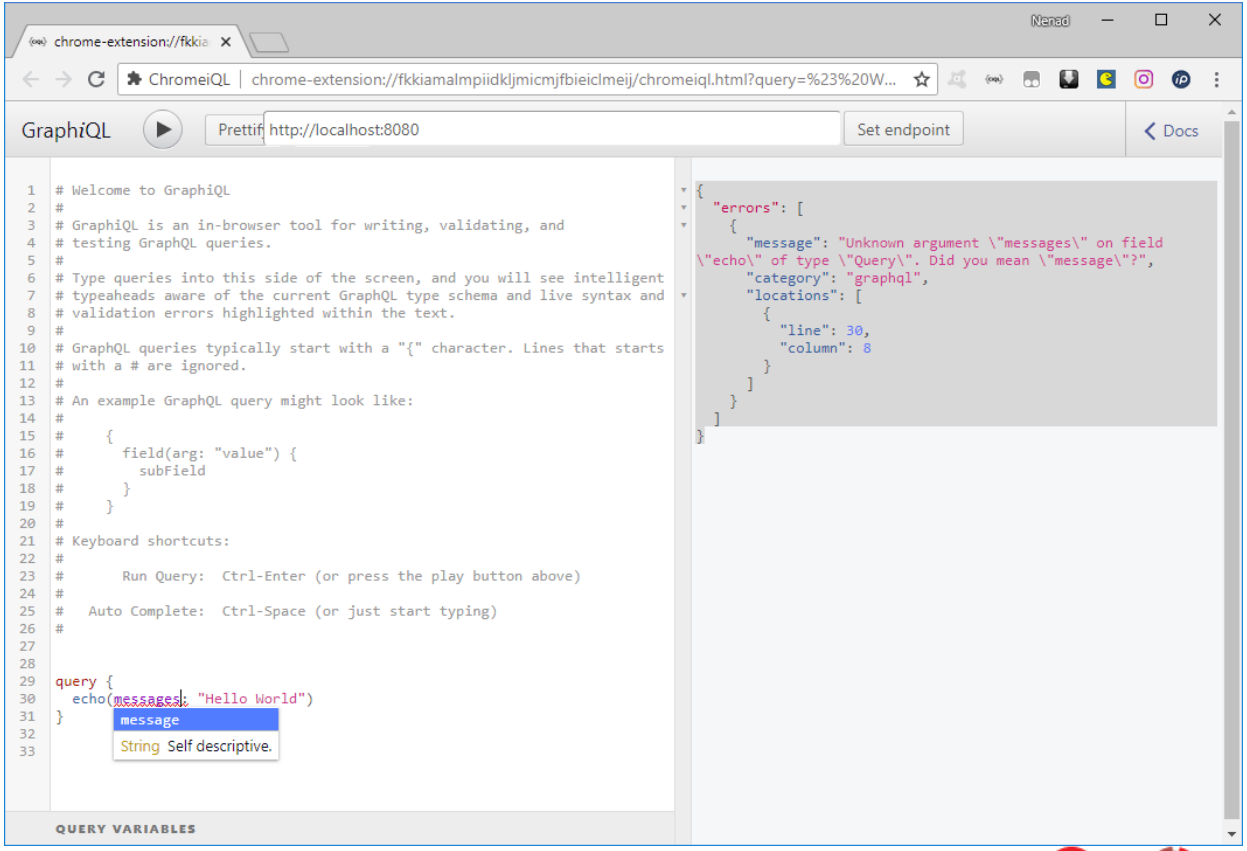
Primjer upita

```
query {
  echo(messages: "Hello World")
}
```

I obrade pogreške

```
{
  "errors": [{
      "message": "Unknown argument \"messages\" on
field \"echo\" of type \"Query\". Did you mean
\"message\"?",
      "category": "graphql",
      "locations": [{
          "line": 30,
          "column": 8
    }]
  }]
}
```

PHP – pogreška kod izvođenja



PHP – primjer korištenja mutacije

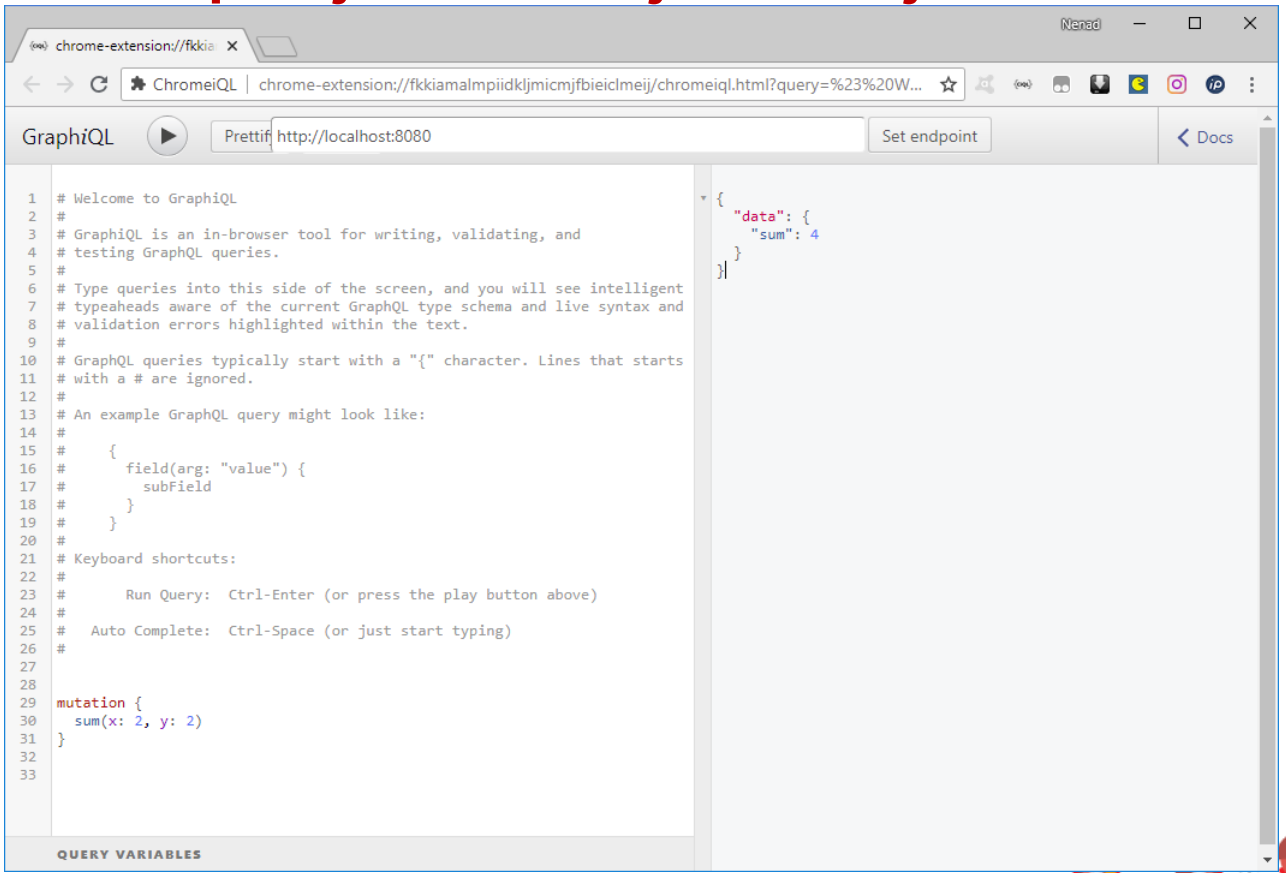
Primjer mutacije

```
mutation {
  sum(x: 2, y: 2)
}
```

Rezultat izvođenja mutacije

```
{
  "data": {
    "sum": 4
  }
}
```

PHP – primjer korištenja mutacije



Primjer povezan s MySQL bazom podataka

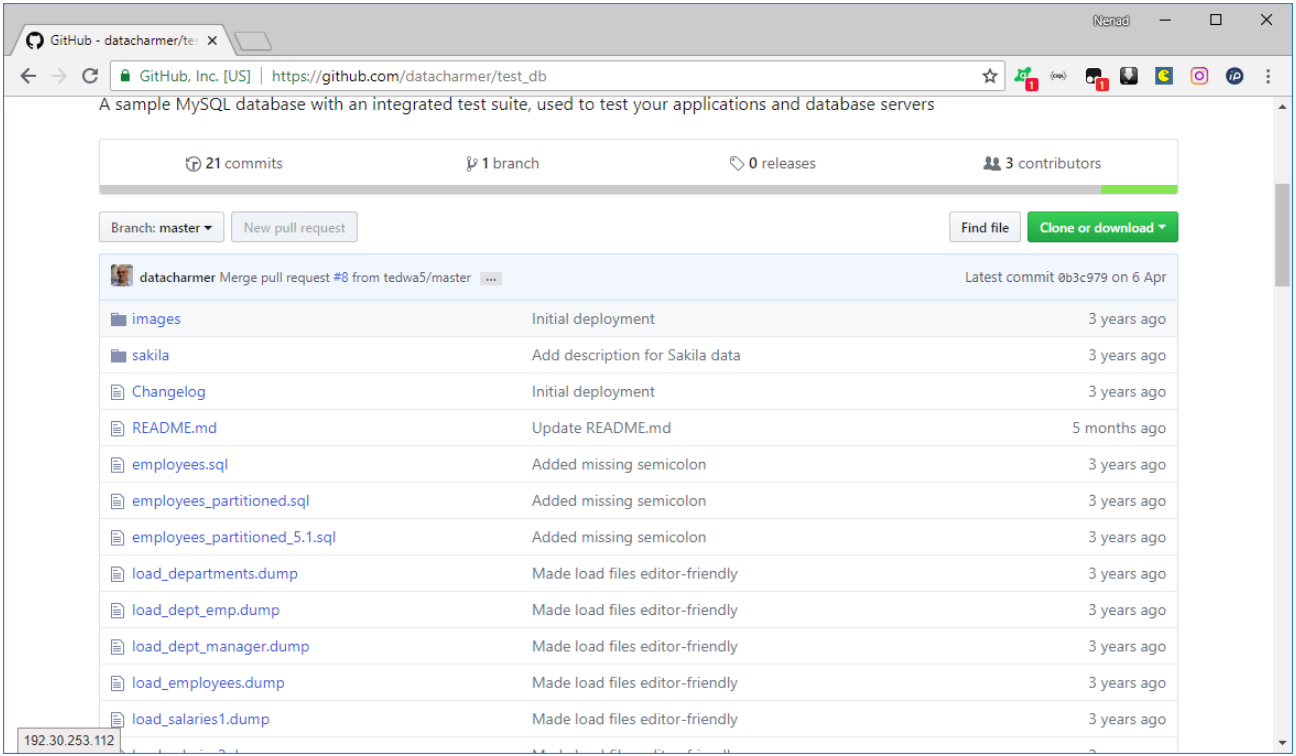
Koristi se demo baza podataka dvojice autora (Patrick Crews i Giuseppe Maxia) opisana je na adresi:
https://dev.mysql.com/doc/employee/en/employees-introduction.html

A sam sadržaj baze podataka može se preuzeti s adrese:
https://github.com/datacharmer/test_db

Po svojoj internoj strukturi baza podataka je prilično jednostavna (sastoji se od svega šest tablica i dva pogleda), ali zato sadrži poprilican broj slogova u tablicama – oko 3 000 000.

Zbog takve veličine može se koristiti za druge vrste primjera poput optimizacije upita na bazu podataka i slično.

Demo baza - preuzimanje

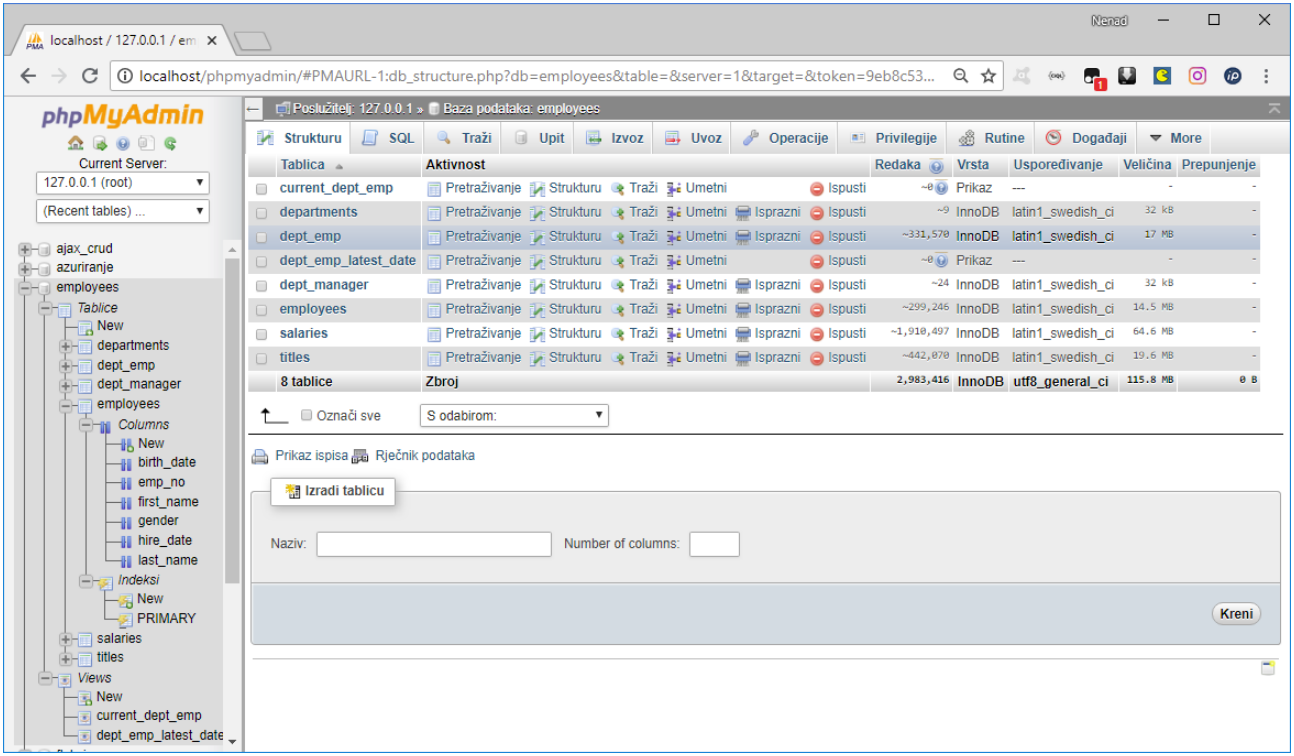


Demo baza – pregled količine podataka

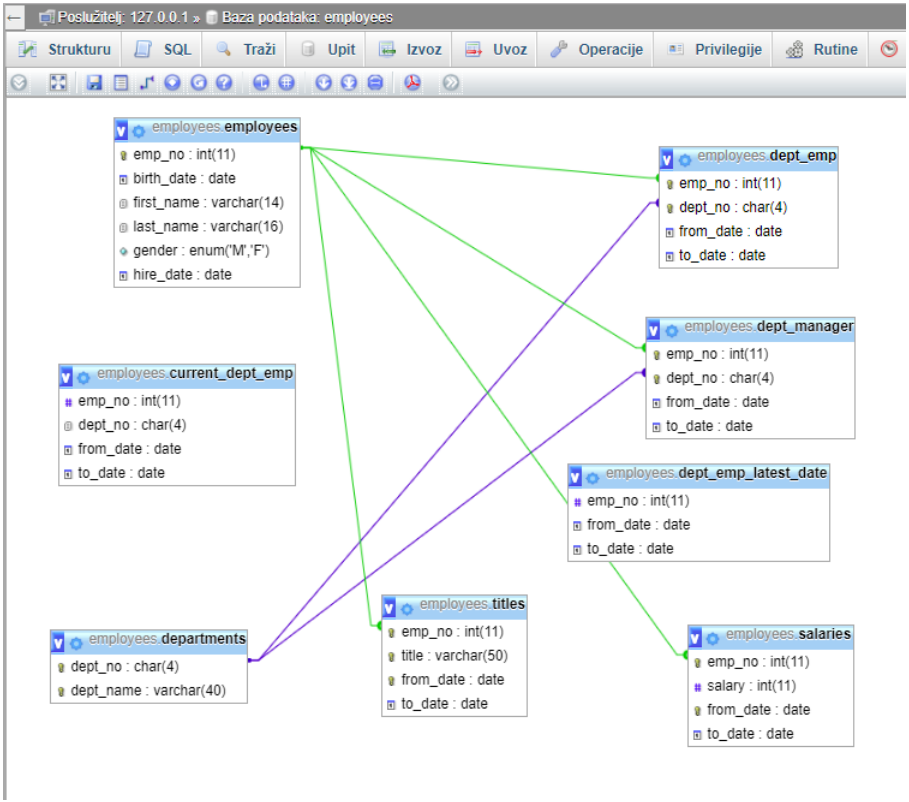
Info | Tables | Columns | Indexes | Triggers | Views | Stored Procedures | Functions | Grants | Events

Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index Length
departments	InnoDB	10	Compact	9	1820	16.0 KiB	0.0 bytes	16.0 KiB
dept_emp	InnoDB	10	Compact	331570	36	11.5 MiB	0.0 bytes	5.5 MiB
dept_manager	InnoDB	10	Compact	24	682	16.0 KiB	0.0 bytes	16.0 KiB
employees	InnoDB	10	Compact	299246	50	14.5 MiB	0.0 bytes	0.0 bytes
salaries	InnoDB	10	Compact	1910497	35	64.6 MiB	0.0 bytes	0.0 bytes
titles	InnoDB	10	Compact	442070	46	19.6 MiB	0.0 bytes	0.0 bytes

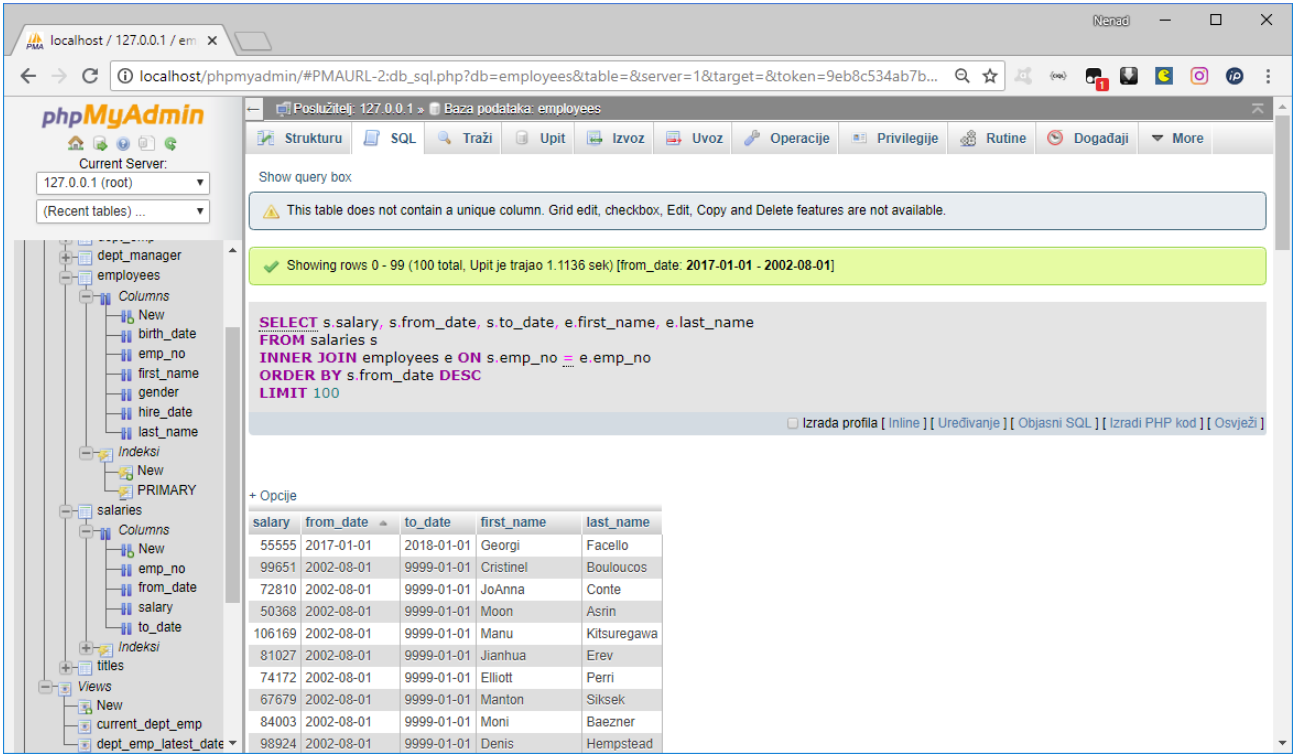
Demo baza – pregled strukture



Demo baza – pregled strukture



Demo baza – pregled podataka za primjer



UseMySQL.php – korištenje baze u PHP-u

```
<?php

require_once __DIR__ . './autoload.php';
require_once __DIR__ . "./_config.php";

use GraphQL\Type\Definition\ObjectType;
use GraphQL\Type\Definition\Type;
use GraphQL\Type\Schema;
use GraphQL\GraphQL;

try {
```

UseMySQL.php – korištenje baze u PHP-u

```
$queryType = new ObjectType([
        'name' => 'Query',
        'fields' => [
            'echo' => [
                'type' => Type::string(),
                'args' => [
                    'emp_no' => ['type' => Type::string()],
                ],
                'resolve' => function ($root, $args) {
             $query = "select s.salary, s.from_date,
                      s.to_date, e.first_name,
                      e.last_name
                      from salaries s
                      inner join employees e on s.emp_no
                      = e.emp_no
                    where s.emp_no = '" . $args['emp_no'] . "'
 order by s.from_date desc limit 1";
```

UseMySQL.php – korištenje baze u PHP-u

```
$result = mysql_query($query);
    while($rec = mysql_fetch_array($result)) {
        $salary = $rec['salary'];
        $period = $rec['from_date'] . '
                - ' . $rec['to_date'];
        $employee = $rec['first_name'] . ' ' .
                $rec['last_name'];
    }
        return $root['prefix'] . $salary . '
        (' . $period . ') ' . $employee;
                    }
                ],
            ],
     ]);
```

UseMySQL.php – korištenje baze u PHP-u

```
$mutationType = new ObjectType([
    'name' => 'Calc',
    'fields' => [
       'add' => [
          'type' => Type::string(),
          'args' => [
           'emp' => ['type' =>
              Type::string()],
     'from' => ['type' => Type::string()],
              'to' => ['type' =>
           Type::string()],
                   'salary' => ['type' =>
              Type::int()],
              ],
```

UseMySQL.php – korištenje baze u PHP-u

```
 'resolve' => function ($root, $args) {
   $query = "insert into salaries (emp_no,
salary,
              from_date, to_date)
              values ('" . $args['emp'] . "'," .
              $args['salary'] . ",'" .
              $args['from'] . "','" .
              $args['to'] . "')";
   $result = mysql_query($query);
   if ($result == 1)
      return "Added salary for " . $args['emp'];
   else
      return "Error adding salary";
                },
            ],
        ],
    ]);
```

UseMySQL.php – korištenje baze u PHP-u

```
$schema = new Schema([
        'query' => $queryType,
        'mutation' => $mutationType,
    ]);

$rawInput = file_get_contents('php://input');
$input = json_decode($rawInput, true);
$query = $input['query'];
$variableValues = isset($input['variables']) ?
$input['variables'] : null;
```

UseMySQL.php – korištenje baze u PHP-u

```
$rootValue = ['prefix' => 'Last salary: '];
$result = GraphQL::executeQuery($schema, $query,
$rootValue, null, $variableValues);
$output = $result->toArray();
} catch (\Exception $e) {
    $output = [
        'error' => [
            'message' => $e->getMessage()
        ]
    ];
}
header('Content-Type: application/json;
charset=UTF-8');
echo json_encode($output);

?>
```

config.php – korištenje baze u PHP-u

```
<?php
    $con = mysql_connect('localhost','root','')
        or die(mysql_error());
    if (!$con) {
        echo "Unable to connect to DB: " .
mysql_error();
        exit;
    }
    if (!mysql_select_db("employees")) {
        echo "Unable to select mydbname: " .
mysql_error();
        exit;
    }
    mysql_query("SET NAMES 'utf8'");
    mysql_query("COLLATE 'utf8_general_ci'");
```

PHP – primjer upita na bazu

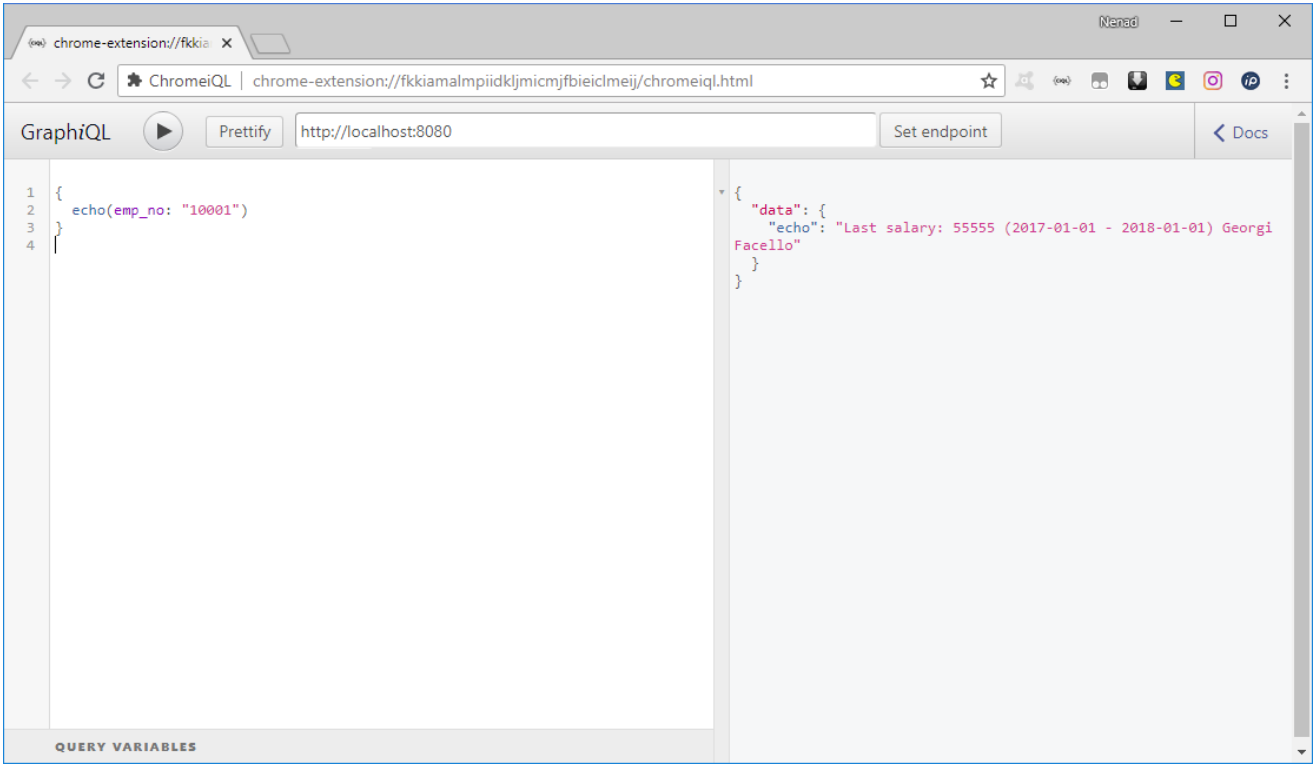
Primjer upita

```
{
  echo(emp_no: "10001")
}
```

Rezultat izvođenja upita

```
{
  "data": {
    "echo": "Last salary: 55555 (2017-01-01 -
2018-01-01) Georgi Facello"
  }
}
```

PHP – primjer upita na bazu



PHP – primjer mutacije na bazu

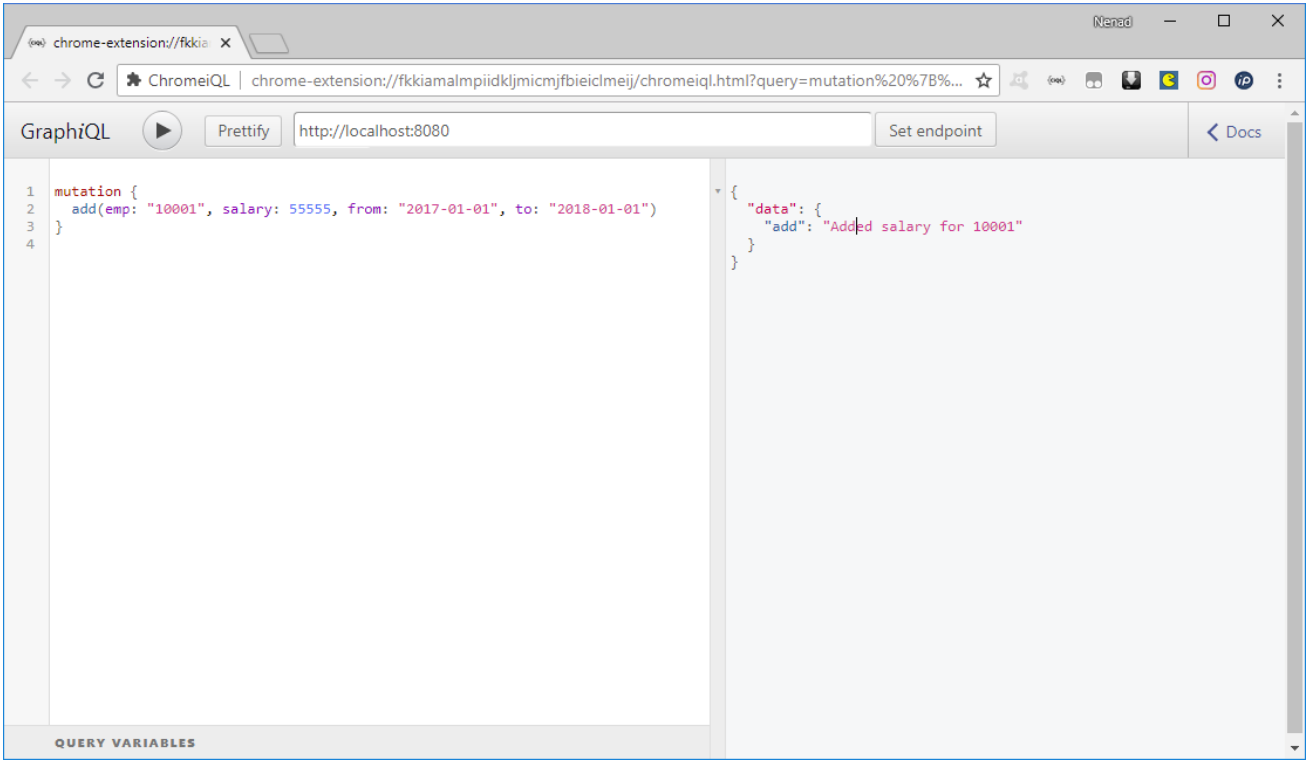
Primjer mutacije

```
mutation {
add(emp: "10001", salary: 55555, from: "2017-
01-01", to: "2018-01-01")
}
```

Rezultat izvođenja mutacije

```
{
  "data": {
    "add": "Added salary for 10001"
  }
}
```

PHP – primjer mutacije na bazu



PHP – Ponavljanje iste mutacije

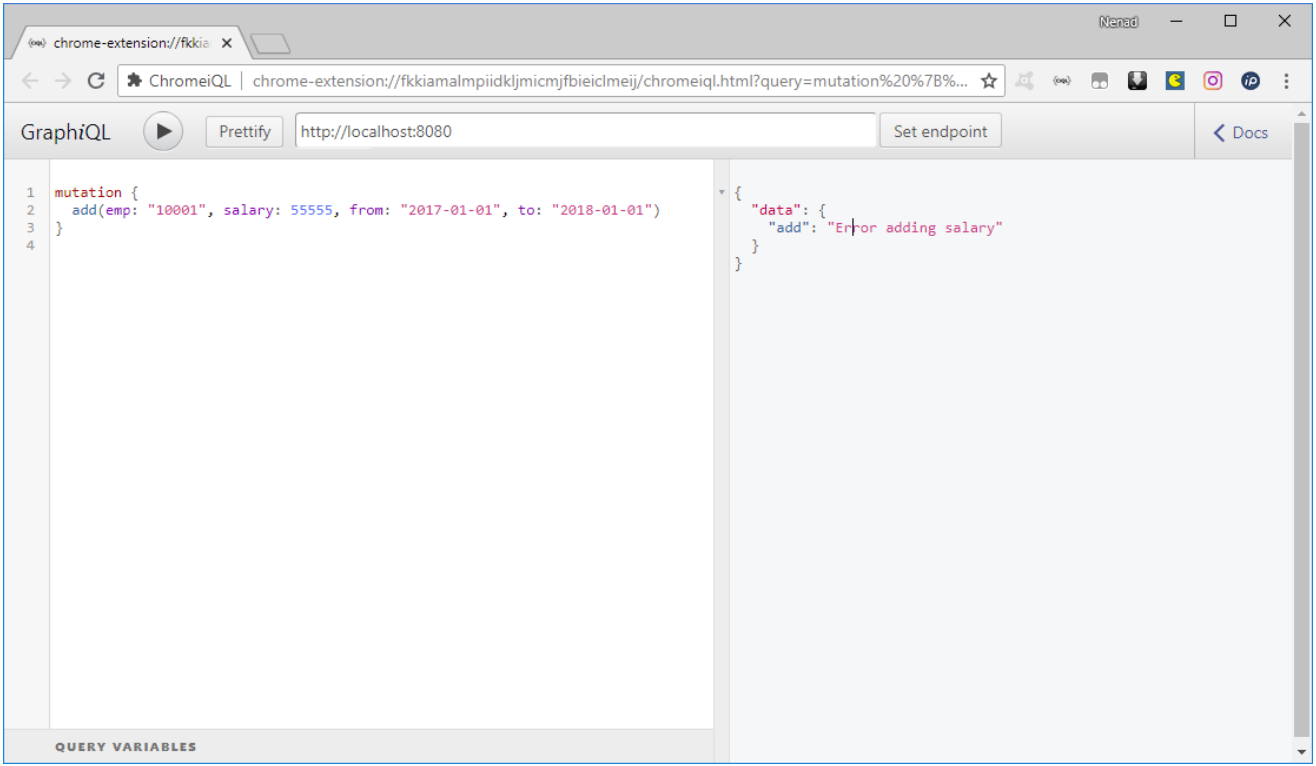
Primjer mutacije

```
mutation {
add(emp: "10001", salary: 55555, from: "2017-
01-01", to: "2018-01-01")
}
```

Rezultat izvođenja mutacije

```
{
  "data": {
    "add": "Error adding salary"
  }
}
```

PHP – primjer mutacije na bazu



UseMySQL.php – izmijenjeni primjer

```
'name' => 'Query',
   'fields' => [
      'LastSalary' => [
         'type' => Type::string(),
         'args' => [
         'emp_no' => ['type'
            => Type::string()],
         ],
```

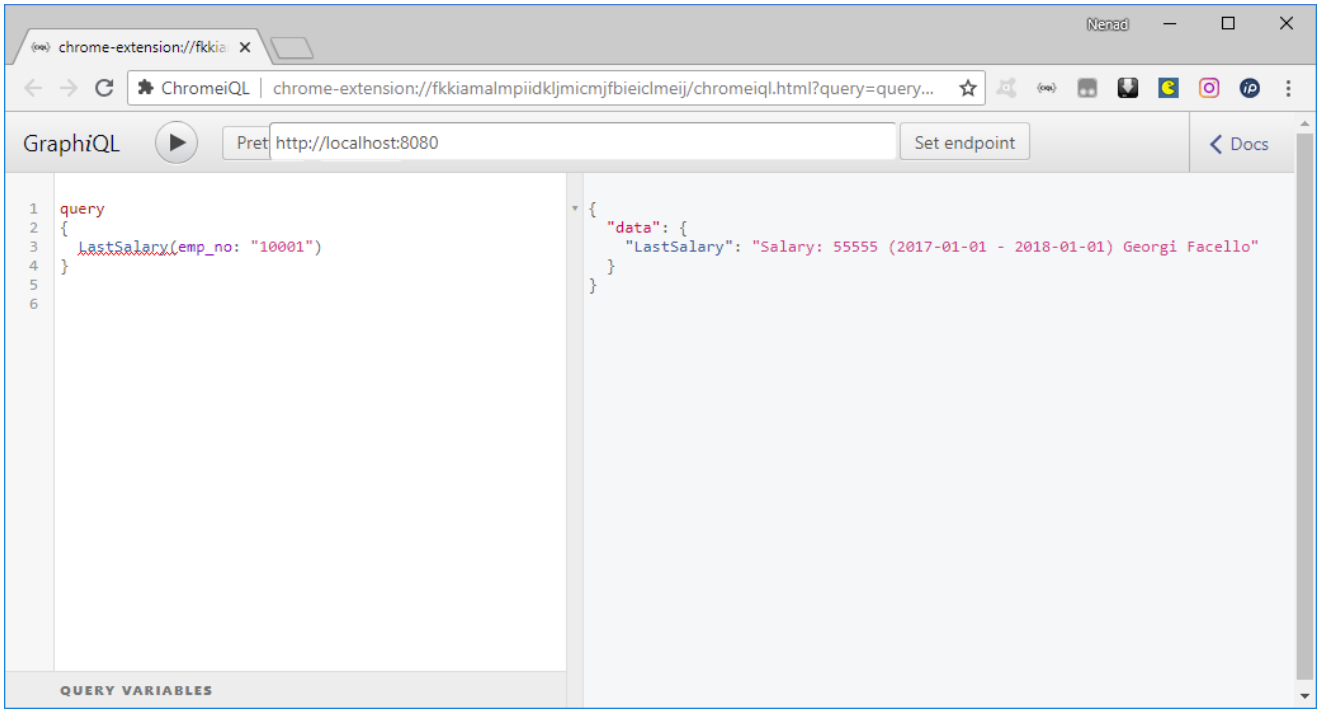
UseMySQL.php – izmijenjeni primjer

```
'name' => 'Query',
   'fields' => [
      'FirstSalary' => [
         'type' => Type::string(),
         'args' => [
         'emp_no' => ['type'
            => Type::string()],
         ],
```

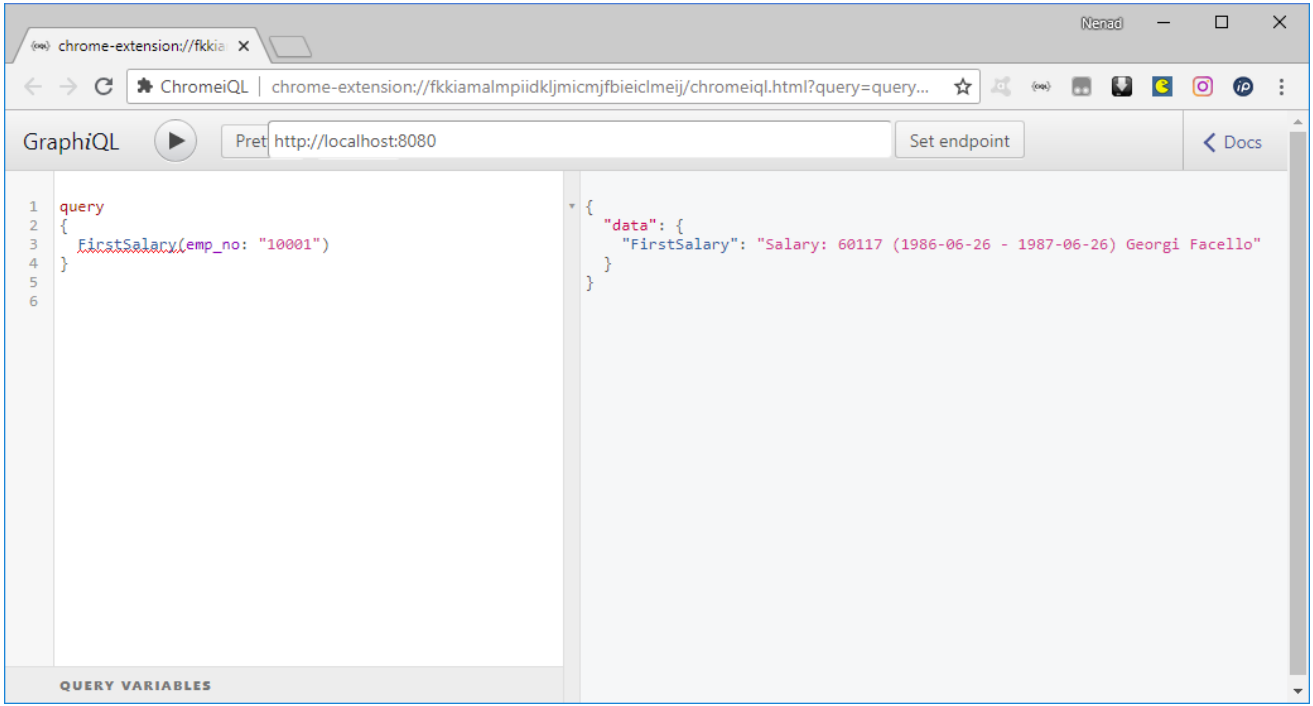
UseMySQL.php – izmijenjeni primjer

```
$query = "select s.salary, s.from_date,
s.to_date, e.first_name, e.last_name
from salaries s
inner join employees e on s.emp_no = e.emp_no
where s.emp_no = '" . $args['emp_no'] . "'
order by s.from_date asc limit 1";
```

PHP – izvođenje izmijenjenog primjera



PHP – izvođenje izmijenjenog primjera



Primjeri korištenja na dokumentno orijentiranoj bazi podataka MongoDB

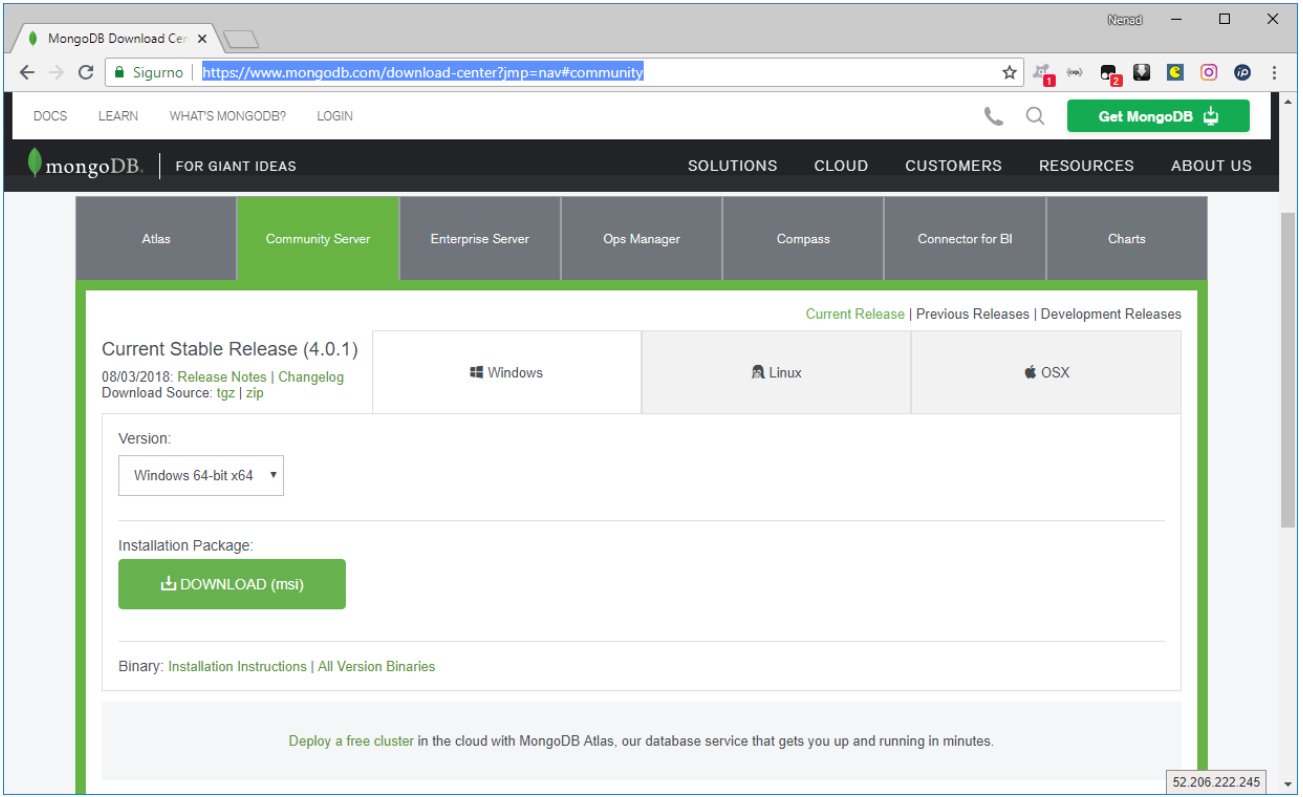
MongoDB priprema računala

MongoDB je dokumentno (JSON-like) orijentirana baza podataka slobodna za korištenje.

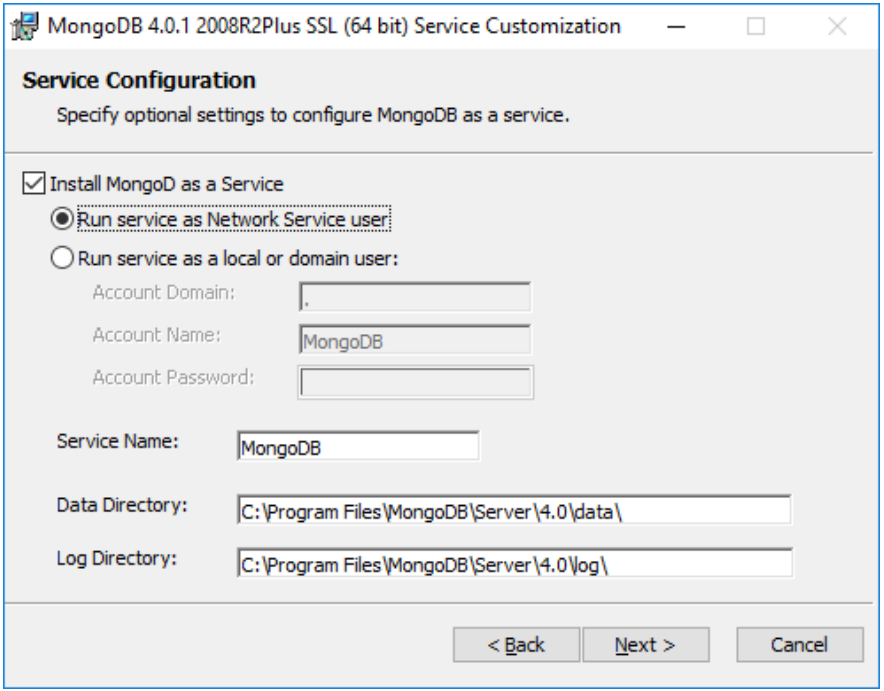
Omogućava korištenje u različitim programskim jezicima.

Instalacija za Windowse preuzima se s adrese
https://www.mongodb.com/download-center?jmp=nav#community

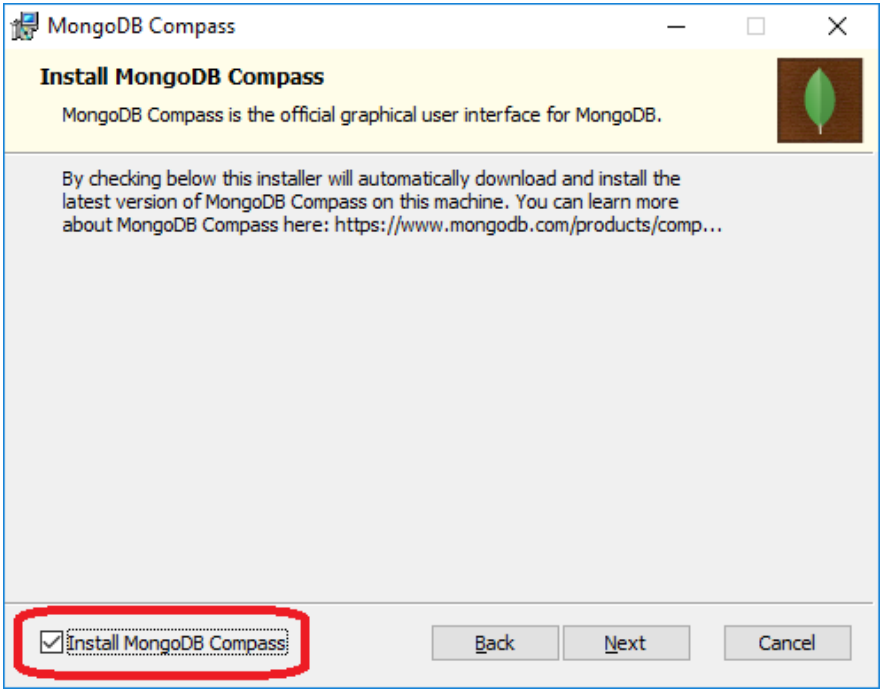
MongoDB – preuzimanje i instalacija



MongoDB – preuzimanje i instalacija



MongoDB – preuzimanje i instalacija



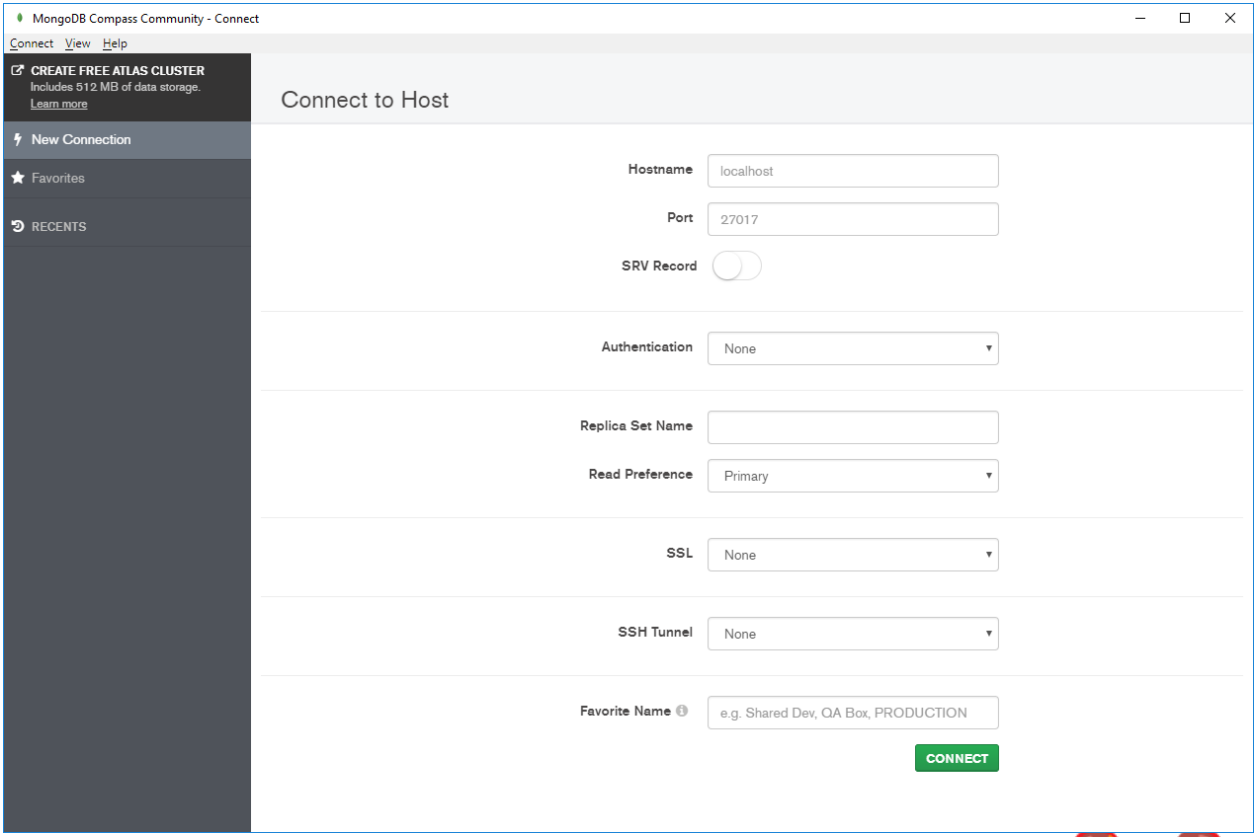
MongoDB – uvod u korištenje

Nakon instalacije automatski su pripremljene tri lokalne baze podataka: **admin, config** i **local**, te jedna kolekcija dokumenata (**startup_log** u bazi **local**).

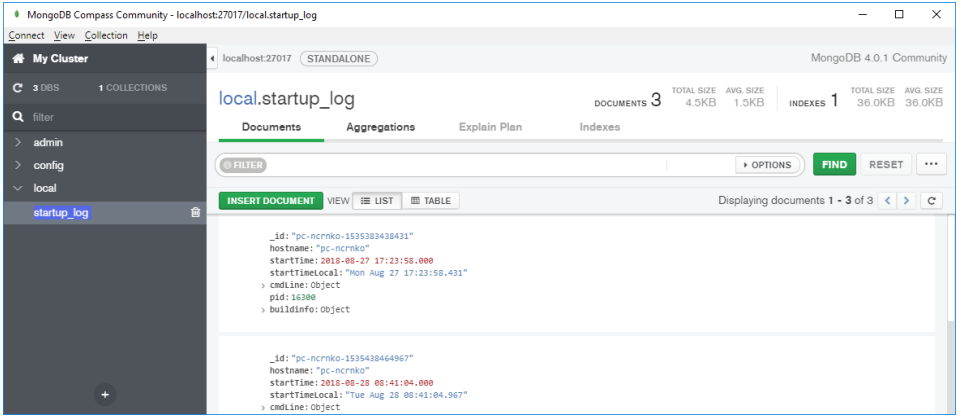
Baza se može koristiti preko grafičkog korisničkog sučelja **MongoDB Compass Community.**

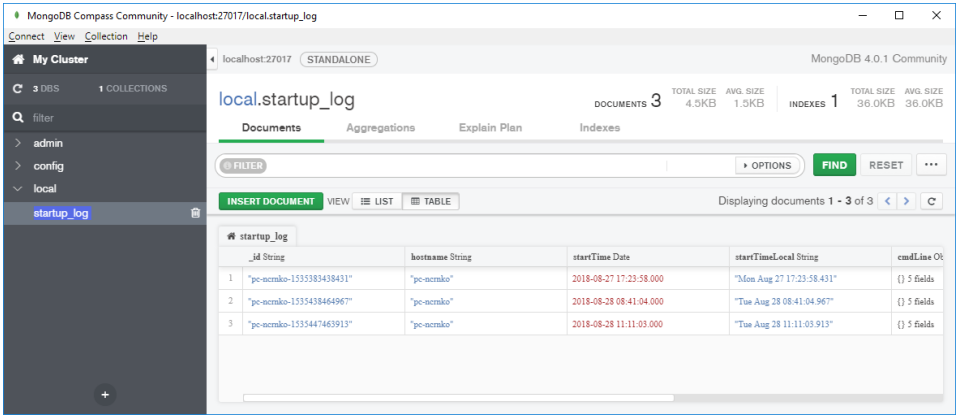
Za to se koristi podrazumijevana veza prema bazi podataka također automatski pripremljena kod instalacije baze podataka.

MongoDB – uvod u korištenje

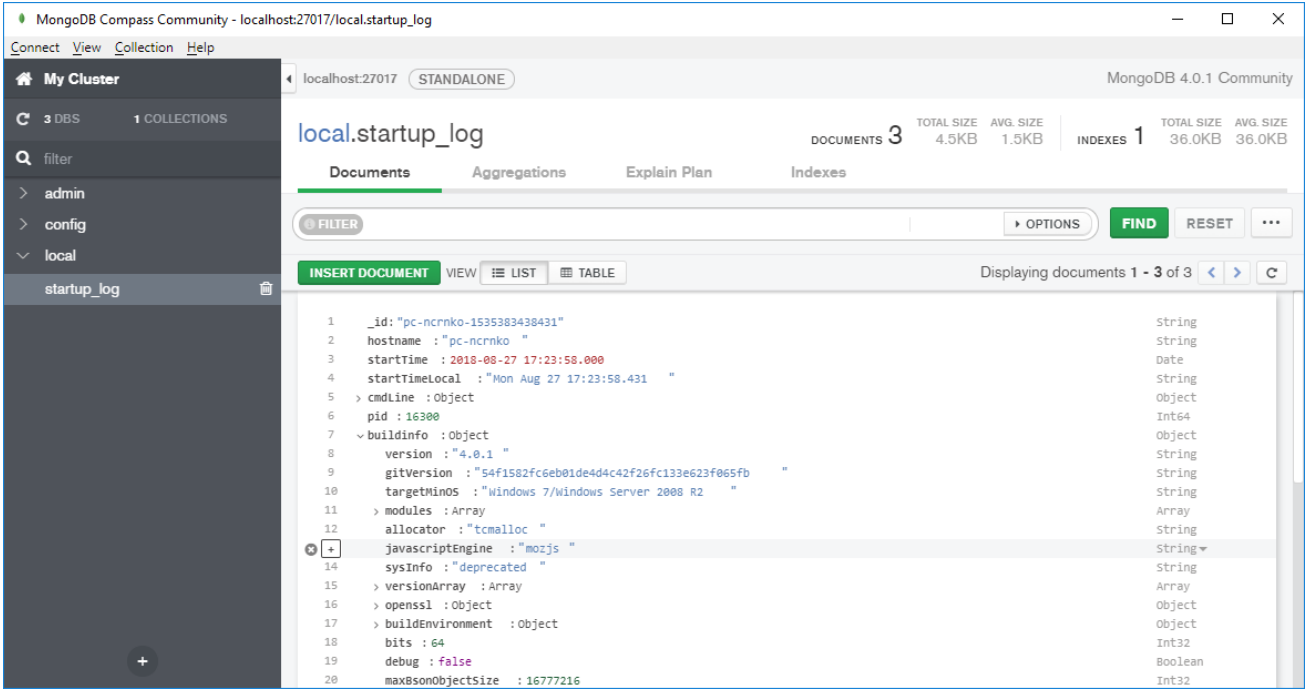


MongoDB – pregled podataka





MongoDB – unos/ažuriranje podataka



MongoDB – uvod u korištenje

Dokumenti u bazi podataka mogu se pregledavati u obliku popisa dokumenata ili tablice.

Različiti dijelovi dokumenta mogu se unositi / ažurirati preko korisničkog sučelja ili programskog koda.

Moguće je uvesti podatke iz drugih baza podataka. Na primjer, MySQL baza podataka employees. Primjer upita za izvoz podataka:

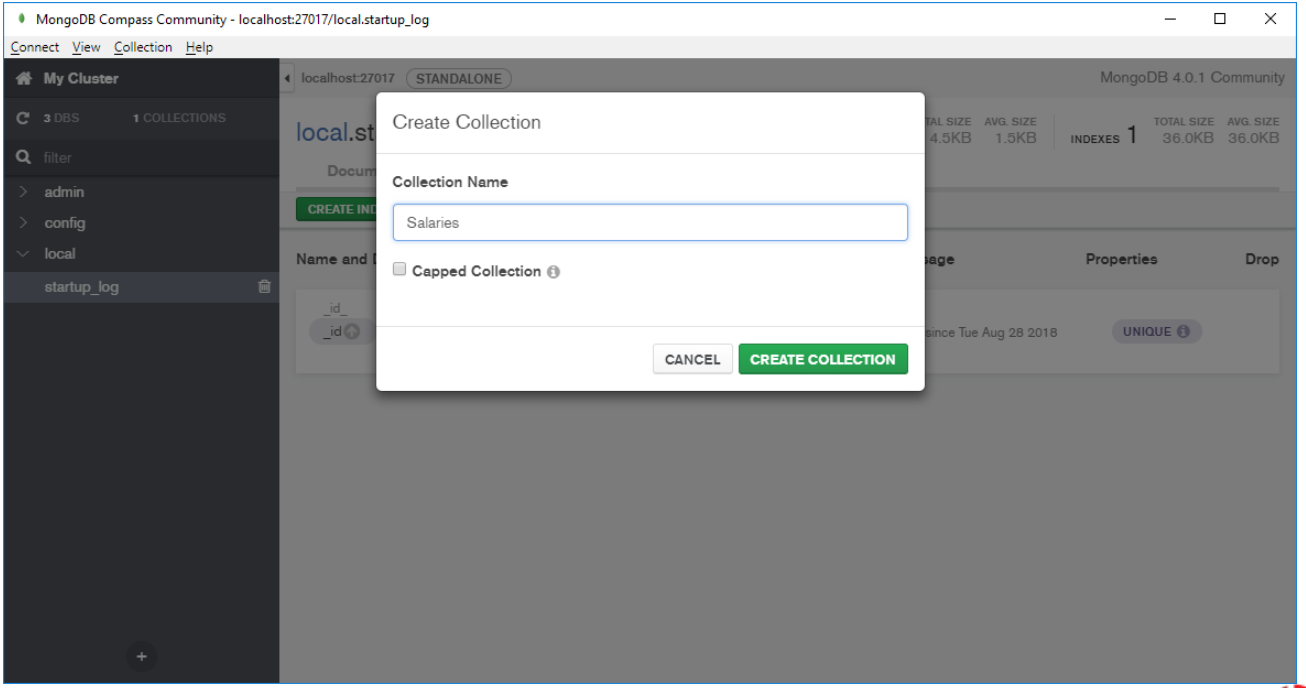
```
select e.emp_no, e.first_name, e.last_name,
s.salary, s.from_date, s.to_date
from salaries s
inner join employees e on s.emp_no = e.emp_n
oorder by 1 asc
limit 100000
```

MongoDB – uvoz podataka u MongoDB

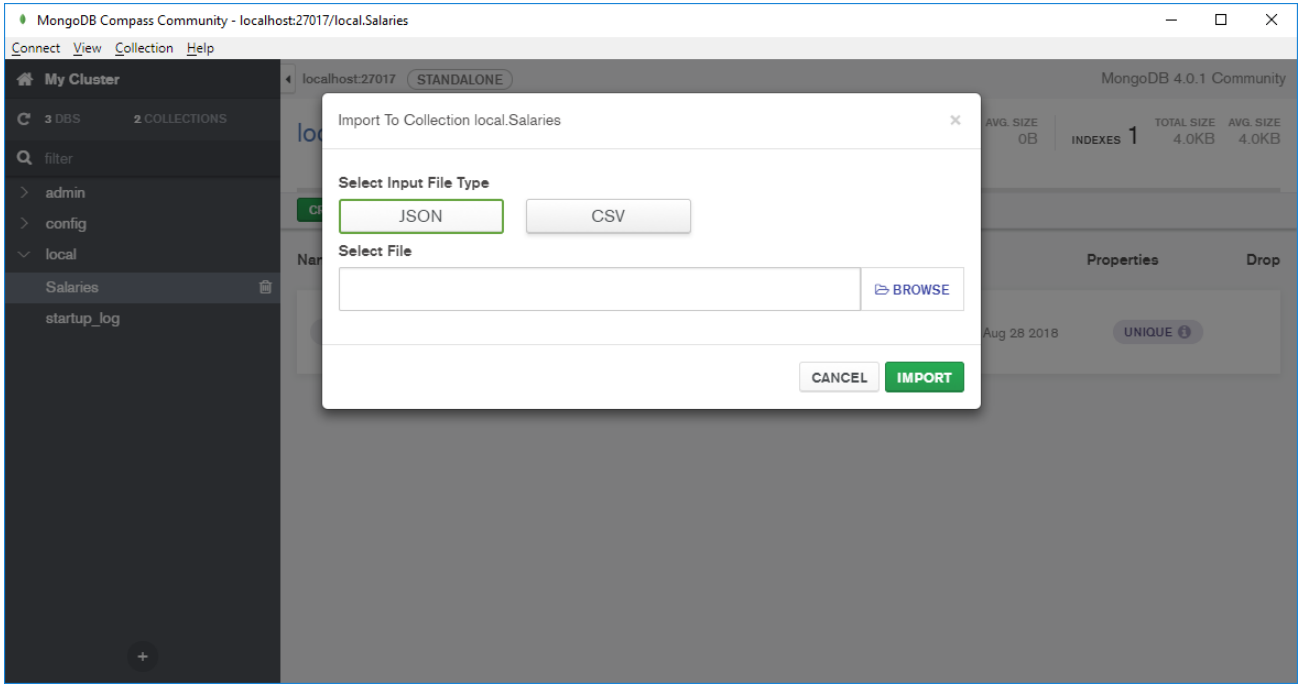
Izvezeni podaci iz relacijske baze podataka pogodni za uvoz u MongoDB:

```
emp_no,first_name,last_name,salary,from_date,to_date
10001,Georgi,Facello,60117,1986-06-26,1987-06-26
10001,Georgi,Facello,62102,1987-06-26,1988-06-25
10001,Georgi,Facello,66074,1988-06-25,1989-06-25
10001,Georgi,Facello,66596,1989-06-25
10002,Bezalel,Simmel,65828,1996-08-03,1997-08-03
10002,Bezalel,Simmel,65909,1997-08-03,1998-08-03
10002,Bezalel,Simmel,67534,1998-08-03,1999-08-03
10002,Bezalel,Simmel,69366,1999-08-03,2000-08-02
10002,Bezalel,Simmel,71963,2000-08-02,2001-08-02
10002,Bezalel,Simmel,72527,2001-08-02,9999-01-01
```

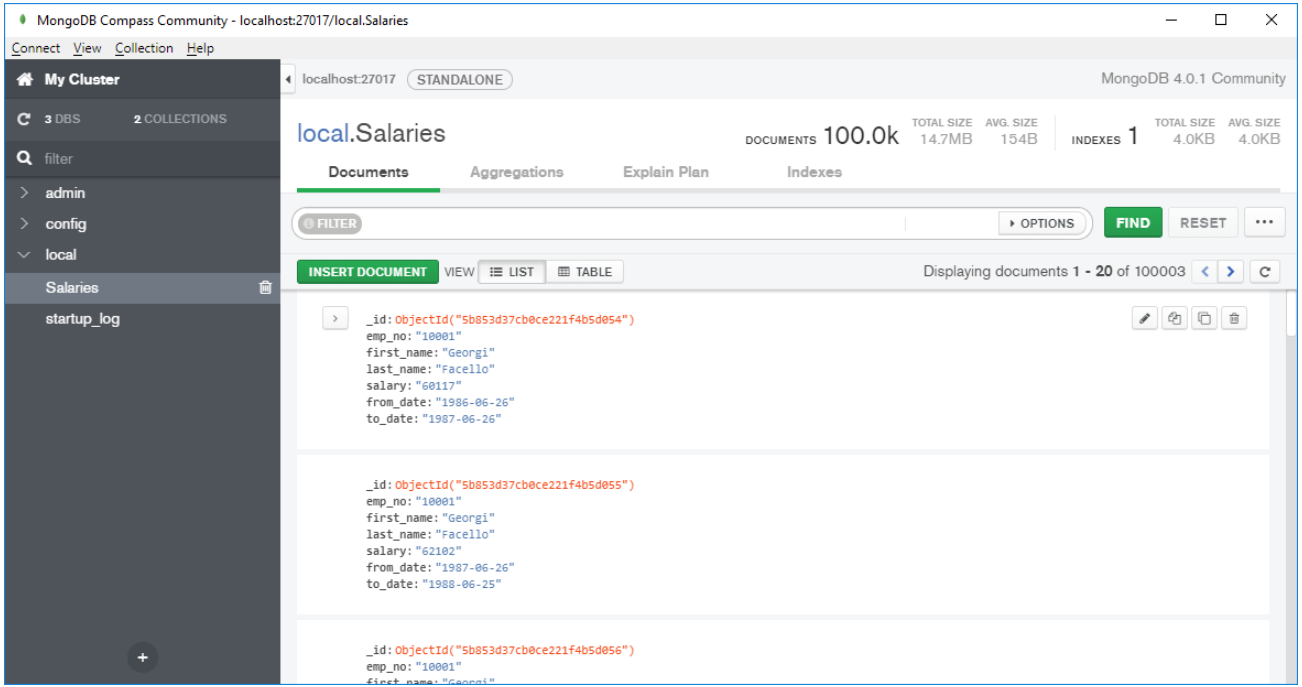
Korak 1 – priprema nove kolekcije za uvoz



Korak 2 – odabir polaznog formata datoteke



Korak 3 – provjera podataka nakon uvoza



Mongoose – MongoDB framework

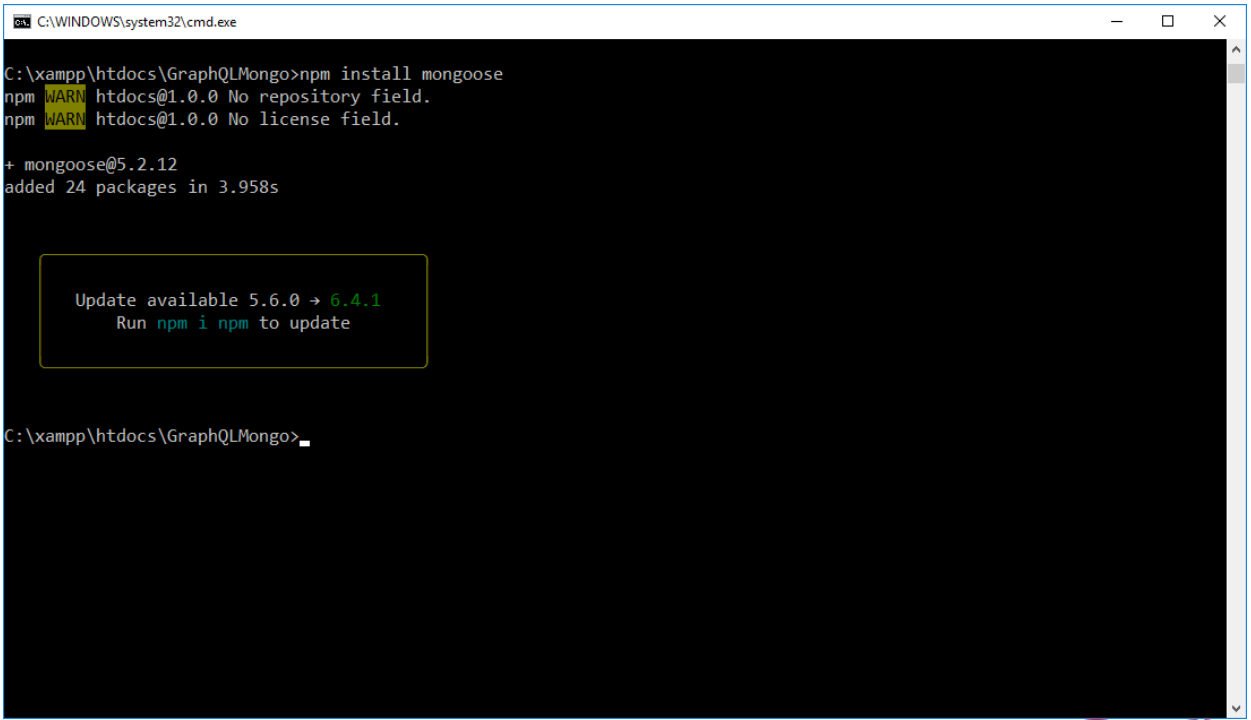
Za korištenje dokumentno orijentirane baze podataka MongoDB uz pomoć jezika upita GraphQL, potreban je još jedan dodatak – Mongoose.

Osim baze podataka MongoDB ma računalu treba biti instaliran i **Node.js**.

Mongoose je dostupan na web adresi:
https://mongoosejs.com/

Mongoose – instalacija

>npm install mongoose



MongoDB/Mongoose – GraphQL primjer

```
var express = require('express');
var graphqlHTTP = require('express-
graphql');
var { buildSchema } = require('graphql');

var schema = buildSchema(`
  type Query {
    salary: String
  }
`);
```

MongoDB/Mongoose – GraphQL primjer

```
var root = {
  salary: () => {
      var mongoose = require('mongoose'),
      Schema = mongoose.Schema;
      var db =
mongoose.connect('mongodb://localhost/local'
,  { useNewUrlParser: true });

      var Salaries = new Schema({
        emp_no: String,
        first_name: String,
        last_name: String,
        salary: String,
        from_date: String,
        to_date: String
      });
```

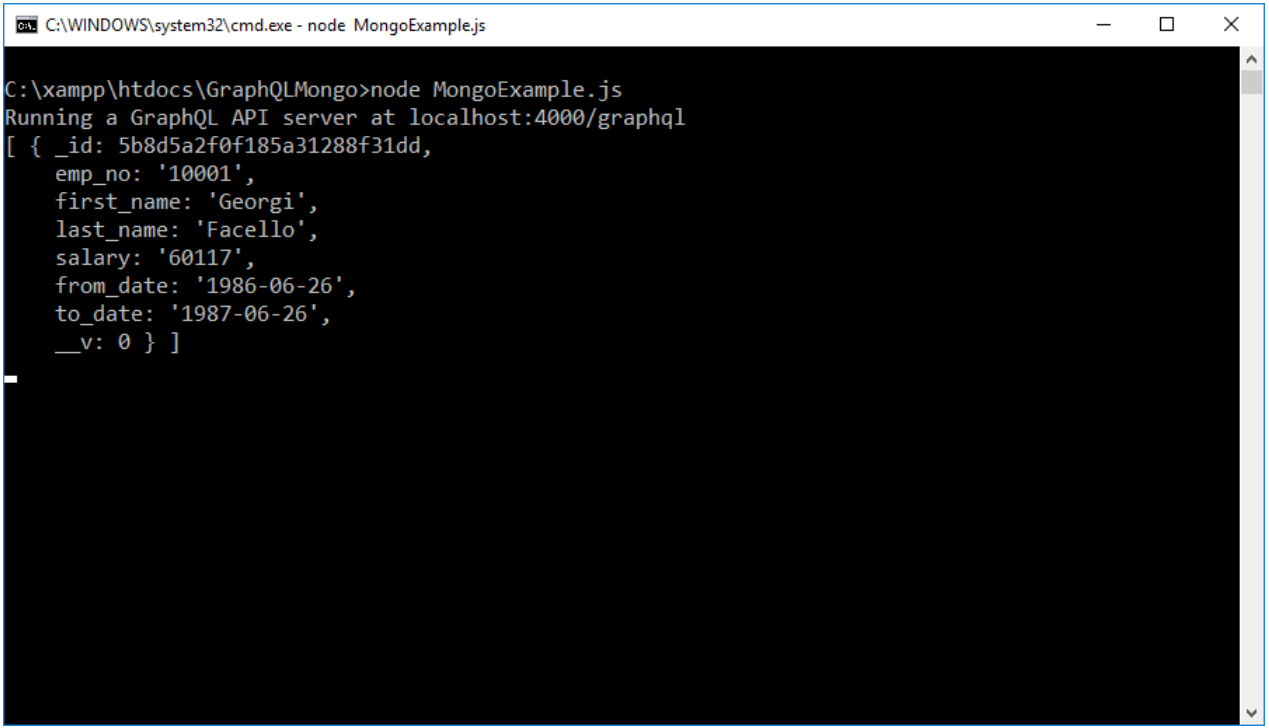
MongoDB/Mongoose – GraphQL primjer

```
        var SalaryModel =
mongoose.model('Salaries', Salaries);


        var query = SalaryModel.find({ emp_no:
'10001'}, function(err, Salaries)
{}).sort([['from_date', 'ascending']]);


    return query.then(function (retsalaries) {
        console.log(retsalaries);
        return 'First salary for ' +
    retsalaries[0].first_name + " " +
        retsalaries[0].last_name + " = " +
        retsalaries[0].salary;
    });
    }
};
```

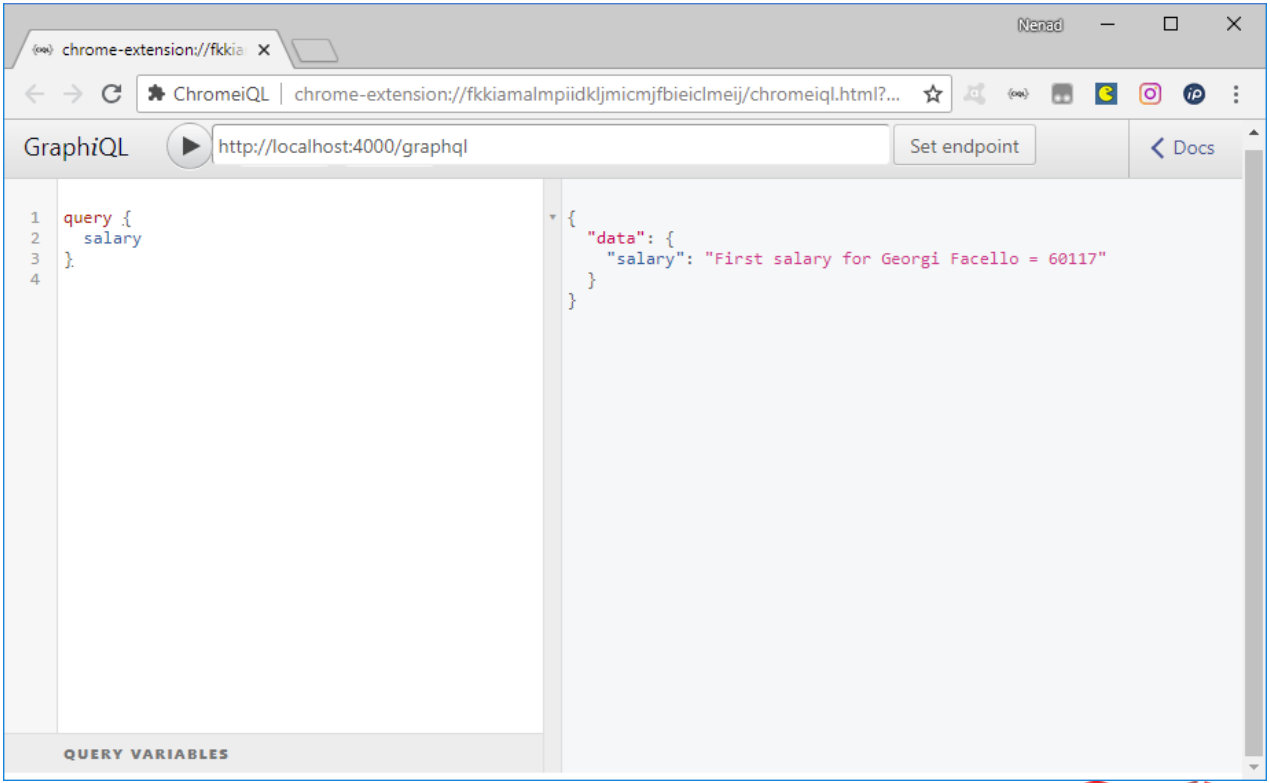
MongoDB/Mongoose – GraphQL primjer

```
var app = express();
app.use('/graphql', graphqlHTTP({
  schema: schema,
  rootValue: root,
  graphiql: true,
}));
app.listen(4000);
console.log('Running a GraphQL API server
at localhost:4000/graphql');
```

MongoDB/Mongoose – izvođenje



MongoDB/Mongoose – izvođenje



MongoDB - primjer koda za ažuriranje baze

```
var SalaryModel =
mongoose.model('Salaries', Salaries);

var record = new SalaryModel();
record.emp_no = '10001';
record.first_name = 'Georgi';
record.last_name = 'Facello';
record.salary = '60117';
record.from_date = '1986-06-26';
record.to_date = '1987-06-26';

record.save(function (err) { });
```

Sažetak predavanja

Uvod u GraphQL s osnovnim primjerima korištenja neovisno o implementaciji. Dodatni materijali - cjelokupan StarWars primjer dostupan je na adresi:

https://github.com/graphql/graphql-js/blob/master/src/__tests__

Primjeri korištenja u programskom jeziku JavaScript.

Primjeri korištenja na relacijskoj bazi podataka MySQL pomoću programskog jezika PHP.

Primjeri korištenja na dokumentno orijentiranoj bazi podataka MongoDB.

Komentari i pitanja?