

Mengenal GraphQL Menggunakan ExpressJS #1



Fuadit Muhammad · Follow

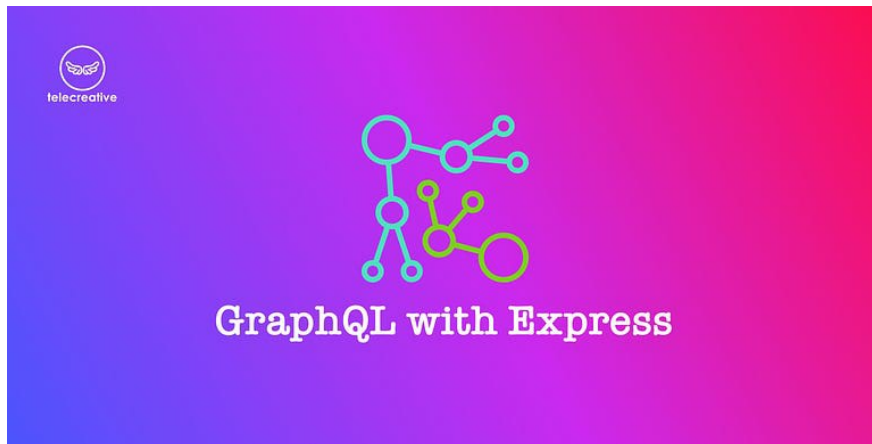
Published in Telecreative · 11 min read · Apr 15, 2018



478



1



Telecreative tutorial for GraphQL w/ Express.

Setelah saya melakukan sedikit review dari beberapa artikel berbahasa bule dan dokumentasi langsung dari website **GraphQL** nya, saya menemukan secerca cahaya yang membuat saya mengambil kesimpulan, “*GraphQL itu easy as f*ck, tidak sesulit yang kita pikir*”. Dan memang, sebelumnya banyak review dari para mentor di channel Youtube yang memberikan testimonial pula bahwasanya GraphQL itu gampang untuk dipelajari dan logika yang bercabang tidak terlalu banyak.

Kenapa logika bercabang tidak terlalu banyak? Maksudnya apa? Ya, GraphQL ketika diimplementasikan di real-project, kamu tidak perlu membuang cara(logic) yang pernah kamu pakai di project sebelumnya. Jadi ketika pernah setup GraphQL untuk satu project, kamu juga bisa memakai nya kembali di project yang lain dan tinggal mengganti beberapa nama atau parameter data yang ingin kamu ambil. Bingung? Ya, analogikan saja ini hampir sama ketika kita setup skema database di SQL untuk aplikasi marketplace dan kita gunakan skema yang sedikit sama dengan aplikasi hotel listing, ya tidak akan jauh berbeda.

Walaupun secara posisi arsitektur system GraphQL ini bisa dibilang termasuk kedalam kategori middleware(seperti redux atau flux), ketika kita pelajari saya yakin tidak akan sesulit dua setan middleware yang sudah melegenda itu(redux atau flux).

Source Code

Github: [**graphql-with-express**](https://github.com/fuadit/graphql-with-express)

1. Just start it!

Tanpa terlalu banyak mukadimah, disini saya akan langsung menjelaskan

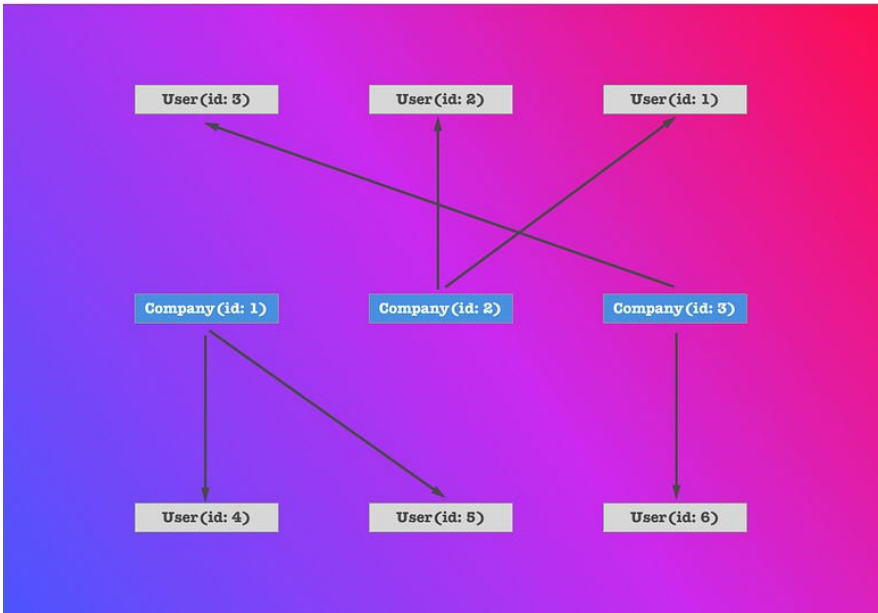
project apa yang akan kita bangun pada artikel kali ini.

1.1 GraphQL di Express

Saya memikirkan dimana project termudah tempat saya bisa mencontohkan betapa gampangnya GraphQL ini. Berhubung saya familiar betul dengan Javascript, maka saya memilih ExpressJS sebagai framework agar saya bisa mengimprovisasi dengan bebas beberapa hal yang telah saya dapatkan mengenai GraphQL ini dari beberapa sumber tadi.

1.2 Profile Listing Project

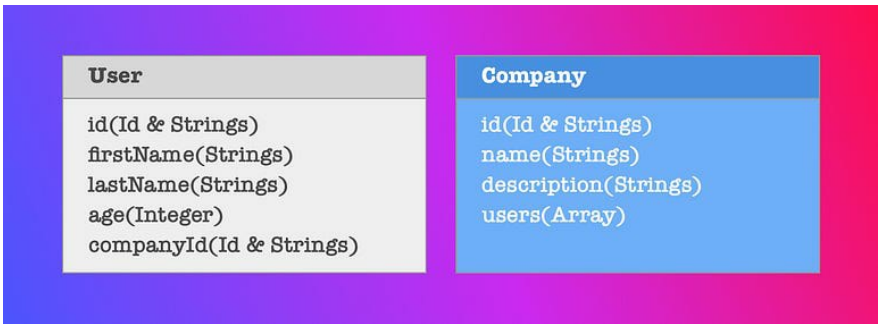
Kita akan membangun satu aplikasi untuk memuntahkan data profil orang-orang dari ‘fake json’ beserta data perusahaan tempat mereka bekerja. Bisa anda lihat dari gambar skema dibawah agar lebih paham bagaimana profile listing ini akan bekerja.



Skema Data Aplikasi

Kita mulai dari skema yang sangat sederhana yaitu dengan dua jenis data; **user profile** dan **company profile**. Pelan-pelan, agar kalian semua bisa mencernanya dengan baik. Jadi, dimana data dari perusahaan tempat si user bekerja akan kita ambil dari relasi `id` user dan `id` perusahaan.

Jadi bisa kita lihat juga dibawah bentuk susunan datanya akan seperti ini;



Struktur Data

Ya, dua data, cuma dua jenis data. Dan jangan salah, dari dua struktur data ini akan menghasilkan style data yang beragam. Kembali kepada tujuan GraphQL sendiri, **mengkombinasikan beragam struktur data untuk menghasilkan satu style data yang akan memudahkan kita menampilkannya**

di frontend.

Maksudnya satu style data itu bagaimana? Bayangkan jika kamu ngefetch data dari REST API untuk ditampilkan di frontend. Dimana dalam satu halaman profile saja memiliki element data-data yang tidak berasal dari satu sumber. Misalnya; nama, umur, lokasi, tanggal lahir diambil URL A. Sedangkan avatar, foto, teman dan email diambil dari URL B. Dan tentunya ini akan merepotkan sisi frontend mengkombinasikannya menjadi satu object(style) data. Dengan GraphQL, memungkinkan kita untuk membangun satu object data yang sangat fleksibel dan dapat diubah style nya sesuai dengan kebutuhan kita.

1.3 Faedah GraphQL

Seberapa bermanfaat sih konsep GraphQL ini kita pelajari? Sekarang saya akan menjabarkannya dengan singkat agar kita semua bisa mengerti kenapa ini sangat berfaedah;

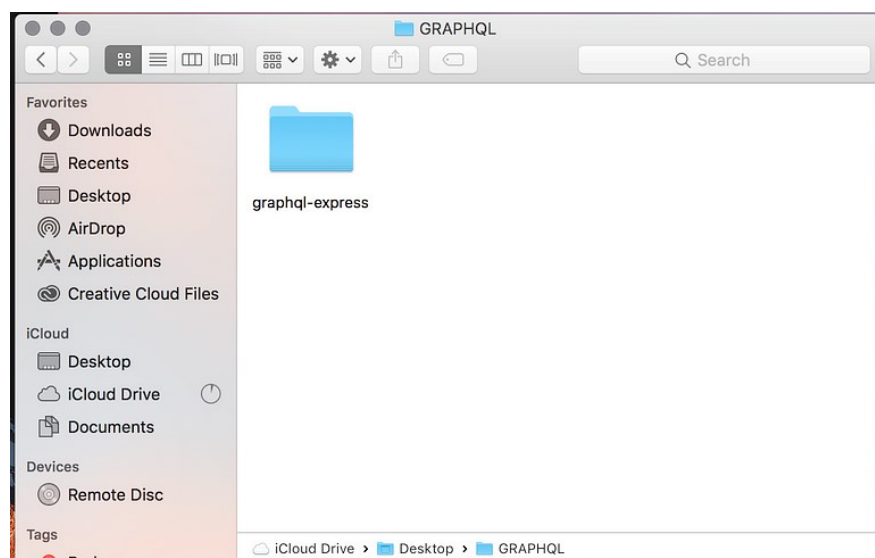
1. Definisi data yang ingin kita pakai bisa ditentukan semudah membuat object di Javascript. Bahkan lebih simple.
2. Berkomunikasi dengan server hanya melewati satu endpoint untuk mendapatk suatu data. Jadi gak perlu pusing lagi mau taro database di server A, server B, ngambilnya cuma dari satu sumber dan nyusunnya gampang banget.
3. **Non Overfetching**, dimana user tidak akan mendapatkan data berlebih yang tidak dibutuhkan ketika merender suatu komponen, sehingga membantu performa aplikasi lebih cepat.

2. Setup Express

Seperti judul, kita akan membuat aplikasi ini menggunakan framework Express. Bagi kalian yang belum pernah belajar ExpressJS jangan khawatir, karena disini saya akan menyusun semua langkah-langkahnya satu persatu diikuti dengan penjelasan yang simple dan gampang dimengerti.

Note: Artikel ini tidak dianjurkan kepada kamu yang belum mengenal Javascript dan CLI(Command Line Interface). Setidaknya mengambil tutorial singkat dari Youtube atau source/website penyedia tutorial Javascript dan CLI.

2.1 Bangun Direktori



Buat satu directory/folder dengan nama dan lokasi sesuai keinginan kamu. Disini saya memberikan nama folder project ini `graphql-express`. Folder ini akan menjadi satu environment khusus untuk pengembangan aplikasi GraphQL kita kedepannya.

2.2 Inisialisasi NPM Environment

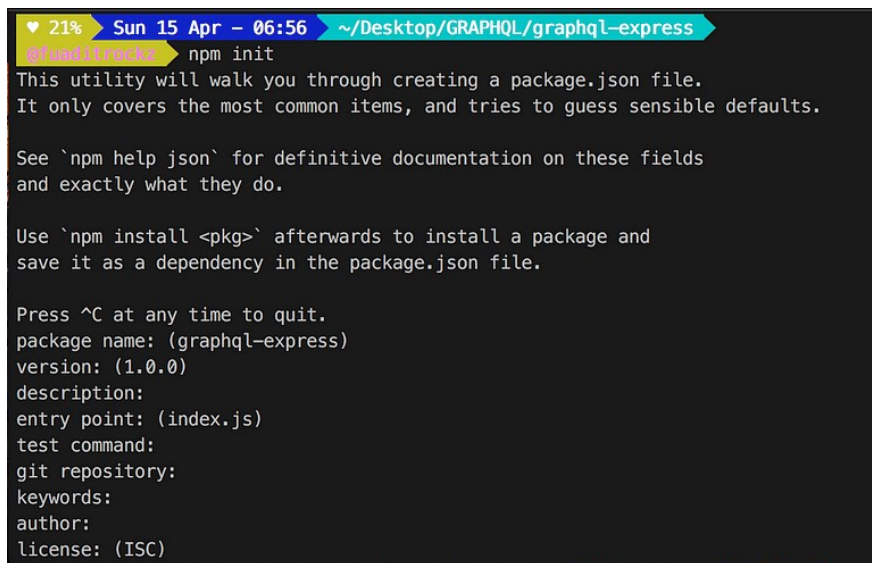
Buka terminal atau iterm2 dan 'change directory' terminal kamu di lokasi tempat folder project kita berada.

```
$ cd graphql-express
```

Lalu, kita install NPM(Node Package Manager) di folder ini agar kita bisa modul-modul yang kita butuhkan untuk membangun aplikasi menggunakan Express dan GraphQL.

```
$ npm init
```

Ikuti langkah-langkah yang diberikan oleh NPM di terminal, seperti nama package(project), entry point, author name dan license, kamu bisa isi input tersebut di terminal atau men-skip dengan cara ketik enter berulang kali sampai input tsb hilang.



```
♥ 21% Sun 15 Apr - 06:56 ~/Desktop/GRAPHQL/graphql-express
@fuzdizrocka 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.

Press ^C at any time to quit.
package name: (graphql-express)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```

2.3 Install Semua Kebutuhan Modul Javascript

Disini kita membutuhkan 6 paket modul, yaitu; **express**, **express-graphql**, **graphql**, **json-server**, **nodemon** dan **axios**. Ketik command dibawah ini di terminal.

```
$ npm install --save express express-graphql graphql json-server axios
```

express berguna sebagai layer awal, framework atau pondasi dari aplikasi yang akan kita bangun.

express-graphql berfungsi sebagai konektor antara dua modul berbeda(Express & GraphQL) agar kita bisa menggunakan GraphQL di dalam framework Express.

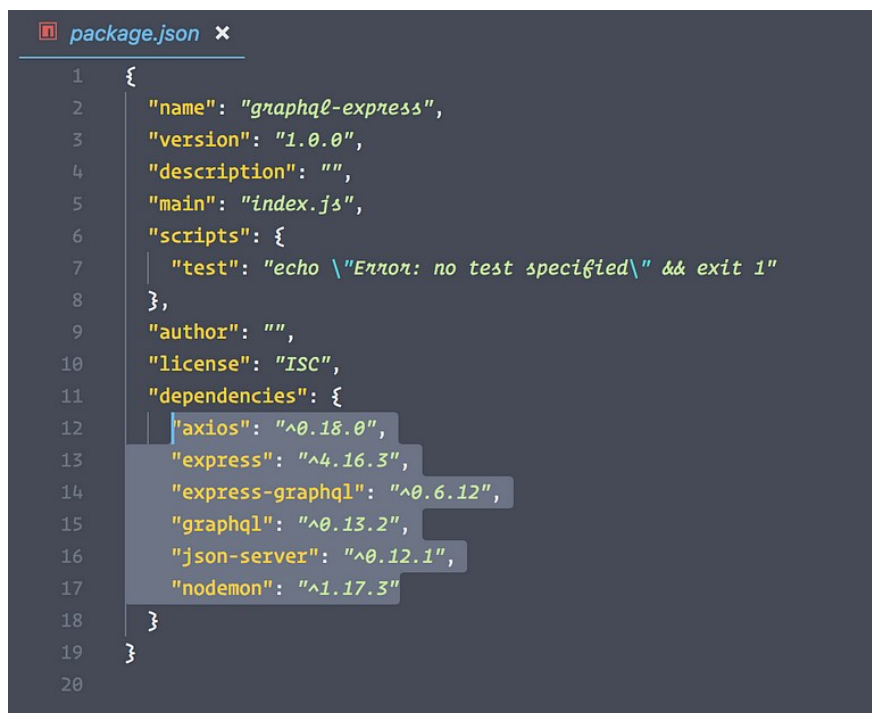
graphql adalah modul yang menyimpan semua fungsi-fungsi dari GraphQL itu sendiri.

json-server adalah modul yang berguna untuk membuat 'fake REST API' agar kita gak perlu cape bikin REST API ampe berjam-jam. Pake json-server gak lebih dari 30 detik(katanya).

nodemon adalah modul yang bisa kita instal globally ataupun perdependencies, akan lebih baik kita instal secara globally agar bisa dipakai berulang-ulang di semua project kita yang akan datang. Modul ini sangat berguna agar kita tidak perlu merestart running aplikasi setelah melakukan perubahan pada kode, karena kalau kita hanya mengandalkan `node` sebagai komando untuk running aplikasi, kita akan melakukannya berulang kali.

axios adalah modul yang membantu kita request data dari 'fake REST API' menggunakan metode *promise*.

Setelah semuanya di instal silahkan check di file `package.json` untuk memastikan semua modul sudah tersedia di dalam folder project kita.



```
1 {
2   "name": "graphql-express",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "axios": "^0.18.0",
13    "express": "^4.16.3",
14    "express-graphql": "^0.6.12",
15    "graphql": "^0.13.2",
16    "json-server": "^0.12.1",
17    "nodemon": "^1.17.3"
18  }
19 }
```

File package.json

2.4 Setup Running Commands and package.json

Sekarang kita akan ubah sedikit settingan default dari file `package.json`.

Ubah main atau entry point aplikasi kita dari `index.js` menjadi `server.js`.

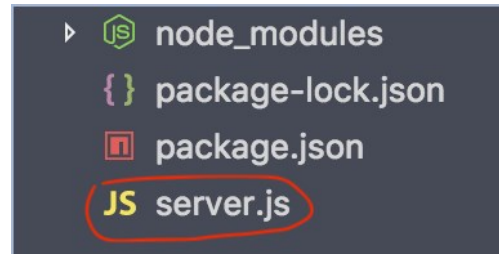
Kalau kita lihat file `package.json`, main berada di **baris ke 5**, yang isinya `"main": "index.js"`. Ini gak ada pengaruh apa-apa sih kalau kita tetap memakai `index.js`, cuma untuk *make sure* aja kalau pura-pura nya kita lagi bikin server untuk aplikasi kita.

Ubah komando running aplikasi menggunakan nodemon, ini berada di **baris ke 6 sampai ke 8** di file `package.json`. Ubah menjadi seperti berikut;

```
## tadinya ini...
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
## Ubah menjadi seperti ini...
"scripts": {
  "start": "nodemon server"
},
```

2.5 File server.js

Nah, pada sesi ini kita akan mulai kodingan dari framework ExpressJS ini. Buat file yang bernama `server.js` didalam folder project kita.



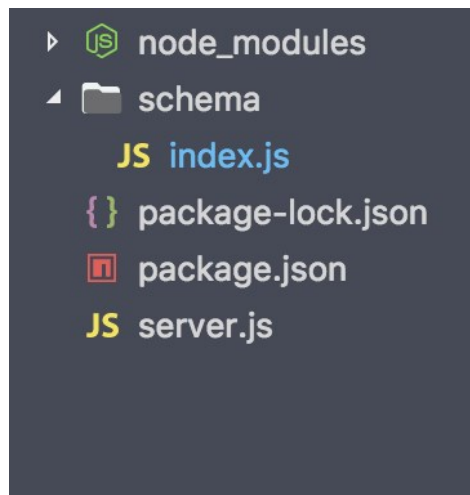
Lokasi file server.js

Ingat, file ini setara level direktori nya dengan `node_modules`, `package-lock.json`, dan `package.json` oke?

Buka file `server.js` di text editor dan ikuti kode saya dibawah;

2.6 Folder schema & File index.js

Ditempat ini skema data akan kita bangun dan disini jugalah kode dari konsep GraphQL kita jahit. Disini saya membuatnya menjadi satu folder dengan alasan kecenderungan pengembangan aplikasi dimasa yang akan datang. Sehingga kode kita lebih termanajemen dengan adanya folder `schema` ini.



Bisa kita lihat diatas, didalam folder `schema` ada satu file yang bernama `index.js`. File ini adalah entry point dari semua skema yang akan kita bangun nantinya(jika banyak).

Selain itu manfaat dari kita membuatnya dengan nama `index` adalah, kita dapat mengimport `index.js` ini hanya dengan mengetik map directory seperti ini;

```
const schema = require('./schema')
```

daripada seperti ini;

```
const schema = require('./schema/index.js')
```

Secara default ES6 sudah mendukung fitur untuk mengenal entry point dari suatu folder dengan nama `index`. Jadi gausah cape-cape ketik nama filenya deh.

Note: Ini tidak berlaku kalau namanya selain 'index'.

3. The GraphQL Starter

Oke, tanpa banyak *tete benge'* langsung buka file `schema/index.js` yang telah kalian buat barusan. Silahkan baca dengan seksama dan ikuti kode saya dibawah ini. Kita akan fokus pada satu file ini saja.

3.1 Import Semua Kebutuhan

Disini kita akan mengimport semua modul dari **graphql** dan **axios** agar kita bisa menggunakannya.

```
const graphql = require('graphql')
const axios = require('axios')
```

3.3 Fungsi-Fungsi GraphQL

Ambil semua fungsi-fungsi yang kita butuhkan untuk membangun definisi data kita dari core GraphQL.

```
const {
  GraphQLObjectType,
  GraphQLInt,
  GraphQLString,
  GraphQLNonNull,
  GraphQLSchema,
  GraphQLList
} = graphql
```

Dari yang kita lihat dari kode diatas ada beberapa kata dari penamaan fungsi GraphQL yang tidak asing bagi kita, seperti Int, String, NonNull, Schema, List. Bagi kamu yang sudah pernah belajar SQL (dibaca sequel), saya yakin memahami maksud dari GraphQL ini akan lebih cepat dari orang yang belum pernah belajar SQL. Karena secara konsep nyaris sama dengan menentukan tipe data di SQL, kita bahkan juga bisa bilang GraphQL ini adalah konsep bahasa query versi frontend.

3.4 GraphQLObjectType()

Suatu fungsi yang akan sangat sering kita gunakan dan paling mudah kita kenali. Kenapa? Karena ini adalah fungsi starter untuk kamu menyusun semua definisi data yang akan kamu lakukan secara suka-suka. Misalnya kamu mau membuat data product, maka data product mu akan dibangun didalam fungsi ini.

Sekarang kita coba membuat satu objek data dengan nama 'UserType', yang dimana objek ini akan menyusun semua informasi dari data user di aplikasi kita.

```
const UserType = new GraphQLObjectType({
  name: 'User',
  fields: {
    id: { type: GraphQLString },
    firstName: { type: GraphQLString },
    lastName: { type: GraphQLString },
  }
})
```

Dari kode diatas, kita membuat satu `const` yang bernama `UserType` yang berisikan fungsi baru `GraphQLObjectType()`.

Setelah itu didalam fungsi `GraphQLObjectType()` terdapat satu argument yang berupa objek dengan isi data nya; nama data -> `name: 'User'` dan field dari data-data User ini -> `fields: 'ALL DATA'`.

Data-data didalam `fields` tersebut ditentukan didalam `fields` ini langsung. Misalnya ada `firstname` yang kita berikan tipe data tersebut adalah String, dan juga data-data lainnya.

3.5 RootQuery

Ini adalah satu root yang akan kita buat dan export nantinya agar bisa di import di file lain. `RootQuery` adalah **suatu object yang dibangun dengan fungsi `GraphQLObjectType()` juga, tetapi ditujukan untuk menjadi root utama dari semua skema query yang kita bangun di file `schema/index.js` saat ini.**

Sekarang mari kita buat `RootQuery` dengan mengikuti kode dibawah ini;

Pertama kita buat pondasi untuk **ObjectType** dari `RootQuery` ini. Kita beri

nama `RootQueryType` agar **UI GraphQL bisa menampilkan namanya nanti ketika kita buka di browser.**

```
const RootQuery = new GraphQLObjectType({
  name: 'RootQueryType',
})
```

Kedua, setelah `name`, kita isi `fields` dari `RootQuery` ini. `fields` tersebut tempat kita mendeklarasikan `type`, `args` dan `resolve` dari data `UserType`, tapi disini kita menamakannya dengan `user` saja.

```
const RootQuery = new GraphQLObjectType({
  name: 'RootQueryType',
  fields: {
    user: {
      type: UserType,
      args: { id: { type: GraphQLString } },
      resolve(parentValue, args) {
        return
          axios.get(`http://localhost:3000/users/${args.id}`)
            .then(response => response.data)
      }
    }
  }
})
```

`type` adalah tempat kita mendeklarasikan tipe dari field, tipe nya kita dapatkan saja dari object `UserType`, jadi tidak perlu membuat ulang tipe baru.

`args` adalah argumentasi atau parameter yang kita gunakan untuk mendapatkan isi data dari suatu field. Disini kita hanya menggunakan `id` saja sudah cukup.

`resolve` adalah satu fungsi didalam `field` yang bertugas untuk mengkoneksikan skema dengan data yang ada. Ya, tentunya disini `resolve` akan melakukan request untuk mendapatkan data JSON dari 'fake REST API' yang akan kita buat menggunakan modul `axios`.

```
return axios.get(`http://localhost:3000/users/${args.id}`)
  .then(response => response.data)
```

`axios` melakukan request terhadap API menggunakan metode promise. URL yang dipakai `axios` untuk mendapatkan data disini adalah alamat URL fake API yang nanti akan kita buat setelah ini dengan modul `json-server`.

3.6 Modul json-server

Sebelum kita melanjutkan kode di file `schema/index.js`, kita akan melakukan setup sedikit terhadap 'fake API' menggunakan `json-server`.

Buat file yang bernama `db.json` yang dimana file tersebut lokasinya setara dengan `server.js`. Didalam file ini kita buat data User dengan JSON yang struktur data nya persis seperti yang kita bangun di skema.

Data DB example.

Disini kita membuat 2 jenis data yaitu; **Users** dan **Companies**. Companies kita persiapkan datanya dari sekarang, agar kita tidak bolak-balik ke file `db.json` lagi.

Setup `json-server` **di file** `package.json` agar kita bisa melakukan running terhadap fake API yang ingin kita konsumsi.

```
## Yang tadinya hanya seperti ini.
"scripts": {
  "start": "nodemon server"
},
## Tambah satu komando lagi dibawah start.
"scripts": {
  "start": "nodemon server",
  "json:server": "json-server --watch db.json"
}
```

Running json-server dengan mengetik ini di terminal.

```
$ npm run json:server
```

Maka terminal akan mengeluarkan log seperti ini, taraaaaa!!

```
> json-server --watch db.json

\{^_^}/ hi!

Loading db.json
Done

Resources
http://localhost:3000/users
http://localhost:3000/companies

Home
```

```
http://localhost:3000
```

```
Type s + enter at any time to create a snapshot of the database  
Watching...
```

Buka home dari `json-server` di browser seperti yang diperintahkan oleh console diatas(yang otomatis akan mengambil port 3000). Selain itu kamu juga bisa melihat hasil data yang tadi telah kita buat dengan mengikuti informasi diatas, apakah mau melihat **users** ataupun **companies**.

3.7 Export & Import Schema

Sekarang kembali ke file `schema/index.js` dan saatnya kita mengexport semua skema yang telah kita buat barusan.

```
module.exports = new GraphQLSchema({  
  query: RootQuery  
})
```

Sebelum kita running aplikasi **GraphiQL** (dibaca grafikel) yang akan menjadi user interface untuk hasil dari konsep GraphQL, kamu wajib mengimport file `schema/index.js` kedalam file root utama aplikasi ini di `server.js`. Cara kita mengimport file ini telah saya jelaskan pada artikel di sesi **2.6 Folder schema & File index.js**.

```
const schema = require('./schema')
```

Kita tidak perlu menggunakan `index.js` seperti yang saya jelaskan sebelumnya. Karena dengan mengimport dari nama folder saja, **ES6** sudah otomatis mencari file yang bernama `index.js`.

3.8 Menggunakan Schema

Setelah skema kita import, cara menggunakannya adalah, meletakkan modul skema tersebut didalam fungsi yang bernama `expressGraphQL` dan sudah kita buat sebelumnya.

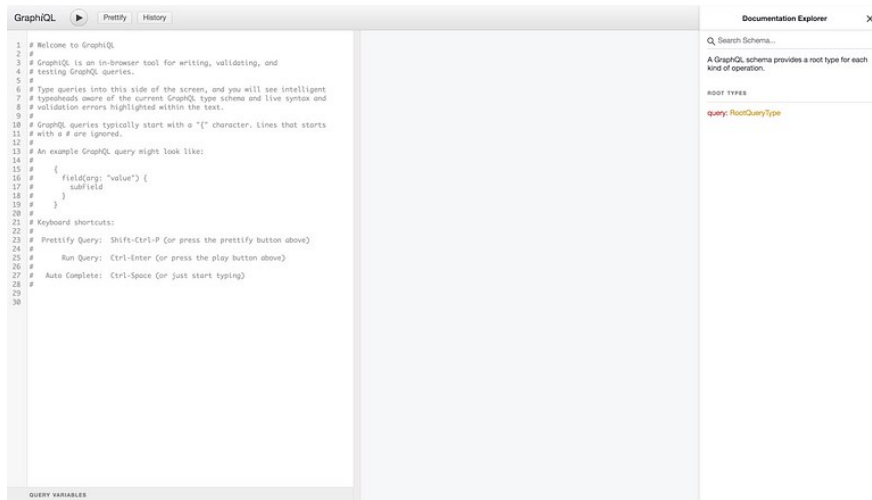
```
app.use('/graphql', expressGraphQL({  
  schema,  
  graphiql: true  
}))
```

3.9 Running Aplikasi

Tadi kan kita telah melakukan running `json-server` nya, nah sekarang buka satu tab lagi di terminal untuk **running aplikasi dari GraphQL** yang telah kita buat dengan mengetik;

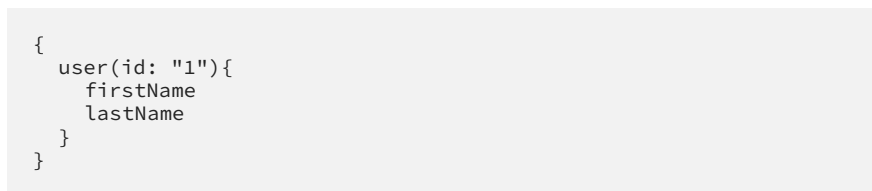
```
$ npm start
```

Masih ingat kan? Tadi saya membuat port nya di 4000 pada file `server.js`, nah sekarang **buka user interface workspace untuk GraphQL** di port 4000 di browser. Buka seperti ini;



GraphQL di Browser.

Disini adalah tempat kita mencoba data yang tadi telah kita buat. Sekarang kita akan mencoba data user dari skema `UserType`. Ketik di workspace GraphQL di browser seperti ini;



Hasilnya akan seperti ini;



GraphQL user interface.

Kamu bisa mengeluarkan berapapun jumlah data dari `UserType` sesuai dengan kebutuhan kamu, selama id nya sesuai dengan data yang ada di `db.json`.

. . .

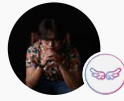
Summary

Kita telah berhasil membuat satu skema untuk data user dan ditampilkan di workspace GraphQL di browser. Selain itu pada sesi ini kita juga sudah berhasil melakukan setup sederhana mengimplementasikan GraphQL di framework Express. Selanjutnya atau di sesi berikutnya (minggu depan) saya akan menjelaskan bagaimana cara setup data companies dan melakukan relasi dengan data users.

To be Continued ...

👏 478 💬 1

🔖 📄 ⋮



Written by Fuadit Muhammad

Follow

257 Followers · Editor for Telecreative

Programmer,, UI UX Specialist, and ex blues guitarist.

More from Fuadit Muhammad and Telecreative



Fuadit Muhammad in Telecreative

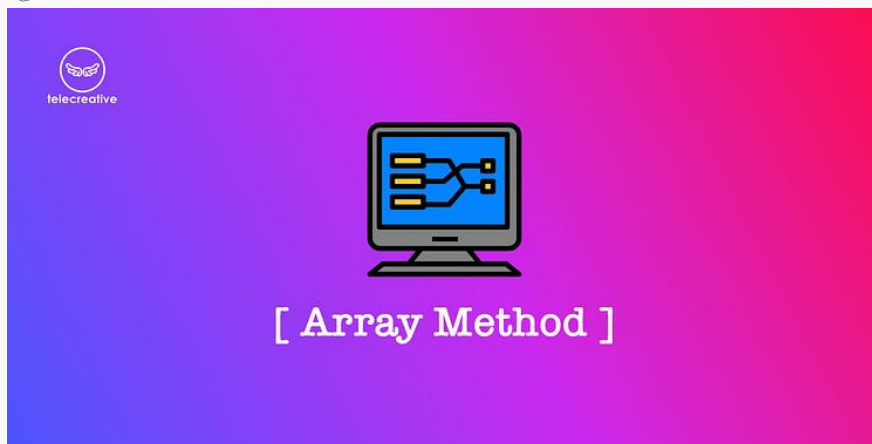
Welcome to Telecreative DevDiary!

Selamat datang teman-teman, para pencinta Javascript, dan orang-orang yang sedang kebingungan dengan dinamisnya pemrograman, tools, library...

4 min read · Mar 27, 2018

👏 153 💬

🔖 ⋮



Fuadit Muhammad in Telecreative

Method Untuk Mengolah Array di Javascript #1

Langsung saja, tanpa basa-basi, kali ini saya akan membahas beberapa method/fungsi untuk mengolah data array di Javascript. Kenapa...

4 min read · Apr 10, 2018

👏 337 💬

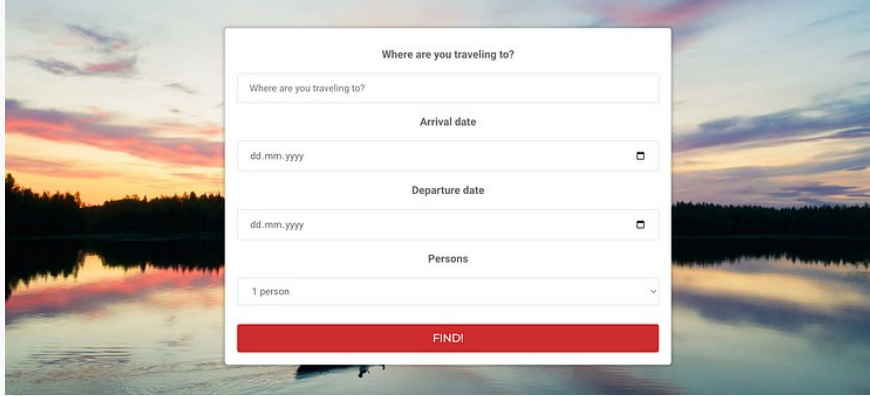
🔖 ⋮

See all from Fuadit Muhammad

See all from Telecreative

Recommended from Medium

parhaatmokit.fi



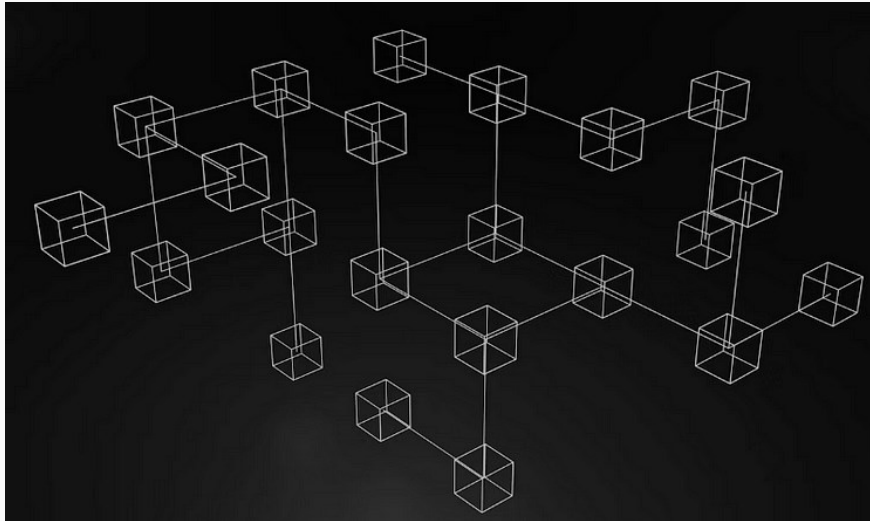
Artturi Jalli

I Built an App in 6 Hours that Makes \$1,500/Mo

Copy my strategy!

🔥 · 3 min read · Jan 23, 2024

👏 18.7K 💬 201



Gayathri Chandrasekarin99P Labs

Behind the Scenes: Hyperledger Fabric Network and Gateway-UI Integration Demystified

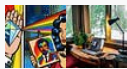
In this article, we will delve into the more technical aspects of our project—Designing a Decentralized Data Marketplace on Blockchain...

8 min read · Dec 12, 2023

👏 3 💬



Lists



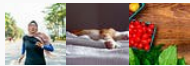
Staff Picks

651 stories · 1006 saves



Stories to Help You Level-Up at Work

19 stories · 623 saves



Self-Improvement 101

20 stories · 1916 saves



Productivity 101

20 stories · 1746 saves



Abhishek KariainSimform Engineering

Node.js and Database-Per-Service: A Practical Implementation Guide

Join database tables efficiently using an aggregator pattern with database-per-service.

7 min read · May 13, 2024

👏 155 💬 1



Booking.com

Talha Şahin

High-Level System Architecture of Booking.com

Take an in-depth look at the possible high-level architecture of Booking.com.

8 min read · Jan 10, 2024

4.95K 40



Minko GechevinAngular Blog

Angular v18 is now available!

Today we are excited to share the next milestone in the evolution of Angular! Over the past three releases we've introduced a lot of new...

13 min read · May 23, 2024

3.8K 26



Furqan Ahmad

Building a Web Scraper with Puppeteer and Cheerio: A Step-by-Step Guide

Introduction

5 min read · Feb 10, 2024

See more recommendations

