WLUG
COMMUNITY | KNOWLEDGE | SHARE

MAPPING THE GRAPH

THE FOSS FILES SEASON 4 | EPISODE 3

API ALCHEMY

Yash Patil

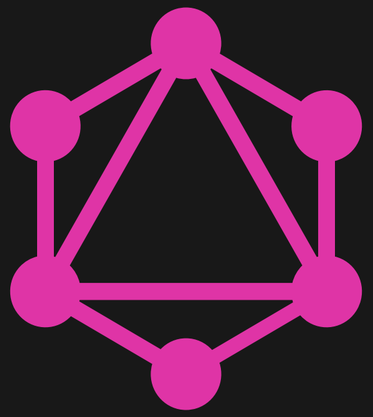
Tanmay Shingde

Aditya Aparadh

30 June 2024

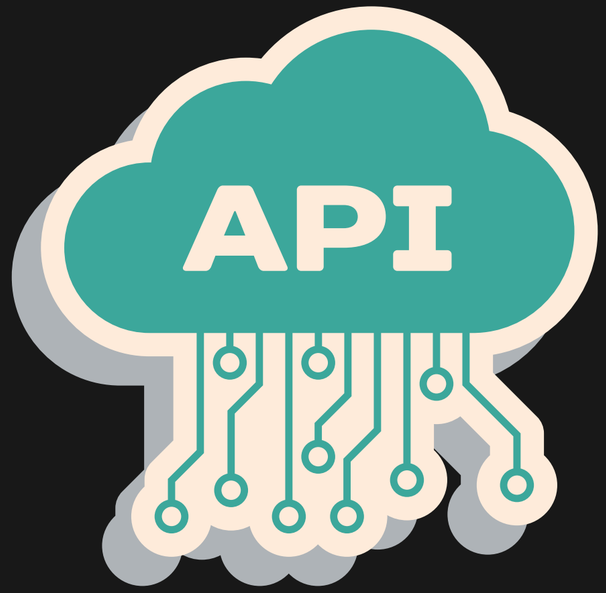
WLUG
COMMUNITY | KNOWLEDGE | SHARE

Table of Contents



Let's revise REST

- Full form of REST?
- Which HTTP method to retrieve data ?
- Status code for NOT FOUND status



WLUG
COMMUNITY | KNOWLEDGE | SHARE

Introduction to GraphQL

- What is GraphQL?
- How was it invented?
- Flexible and strongly typed

WLUG
COMMUNITY | KNOWLEDGE | SHARE

GraphQL vs REST

- Data Fetching
- Overfetching and Underfetching
- Endpoint Structure

Example

pokemon.com/api

Names	Type	Master
pikachu	electric	Ash
bulbasaur	grass	Rock
charmender	fire	Misty
squirtle	water	Red

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Data Fetching

REST

pokemon.com/api/NameId

pokemon.com/api/TypeId

pokemon.com/api/MasterId

GraphQL

pokemon.com/api

getInfo

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Over Fetching
Pokemon name

REST

pokemon.com/api/ID

```
{
    "Id": 25,
    "Name": "charizard",
    "Type": "fire",
    "Master": "ash",
    "Height": 1.7,
    "Weight": 900
}
```

GraphQL

pokemon.com/api

getName

```
query getName {
    name
}
```

```
{
  "data": {
      "name": "charizard"
  }
}
```

Under Fetching

Pokemon name, type and other Pokemon names from it’s master

REST

pokemon.com/api/ID

```
{
    "Id": 25,
    "Name": "charizard",
    "Type": "fire",
    "Master": "ash",
    "Height": 1.7,
    "Weight": 900
}
```

GraphQL

pokemon.com/api
getDetail

```
query getDetail {
  pokemon {
    name
    type {
      typeName
    }
    master {
      pokemons {
        name
      }
    }
  }
}
```

```
{
  "data": {
    "pokemon": {
      "name": "Pikachu",
      "type": {
        "typeName": "Electric"
      },
      "master": {
        "pokemons": [
          {
            "name": "Pikachu"
          },
          {
            "name": "Bulbasaur"
          },
          {
            "name": "Charizard"
          }
        ]
      }
    }
  }
}
```

Schema

- Blueprint of GraphQL API
- Defines structure of data, their fields, types and relationships
- Defines available operations - Queries and Mutations

Example

```
type Master{
    id: ID!
    name: String!
    age: Int!
    pokemons: [Pokemon]
}

type Pokemon {
    id: ID!
    name: String!
    type: String!
    master: Master
}

type Query {
    masters: [Master]
    pokemons: [Pokemon]
}

type Mutation {
    addMaster(name: String!, age: Int!): Master
    addPokemon(name: String!, type: String!): Pokemon
    addPokemonToMaster(masterId: ID!, pokemonId: ID!): Master
}
```

Types

- Scalar Types - Int, Float, String, Boolean, ID

```
type Pokemon {
    id: ID!
    name: String!
    level: Int!
    captured: Boolean!
}
```

Types

- Object Types - Represent entities in your API

```
type Master {
    id: ID!
    name: String!
    pokemons: [Pokemon]
}

type Pokemon {
    id: ID!
    name: String!
    master: Master
}
```

Types

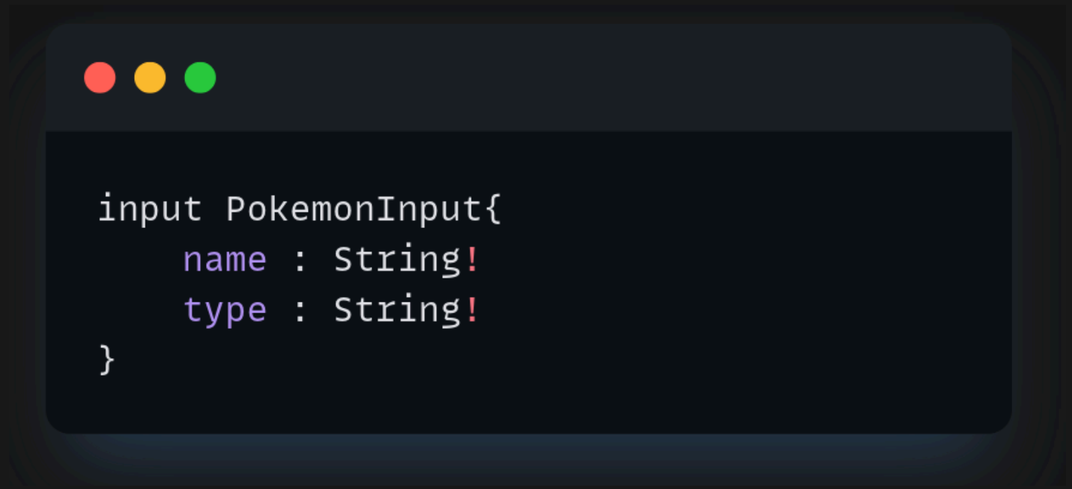
- Enum Types - Restrict a field to a fixed set of values

```
enum PokemonType {
    NORMAL
    FIRE
    WATER
    DARK
}

type Pokemon {
    id: ID!
    name: String!
    type: PokemonType!
}
```

Types

- Input Types - Define the structure of the data clients can send to the server



```
input PokemonInput{
    name : String!
    type : String!
}
```

Queries

- Used to fetch data from the server

```
type Query {
    getPokemon(id: ID!): Pokemon
    getPokemons: [Pokemon]
}
```

```
Query{
    getPokemon(id: 1){
        id
        name
        type
    }
}
```

```
{
    "data":{
        "getPokemon":{
            "id":1,
            "name":"Bulbasaur",
            "type":"Grass",
        }
    }
}
```

Queries

- Nested Queries

```
Query{
    getPokemon(id: 2){
        name
        type
        master{
            id
            name
        }
    }
}
```

```
{
    "data":{
        "getPokemon":{
            "name":"Picachu",
            "type":"Electric",
            "master":{
                "id":1,
                "name":"Ash",
            }
        }
    }
}
```

Mutations

- Used to modify data on the server

```
input PokemonInput {
  name: String!
  type: String!
}

type Mutation {
  createPokemon(input: PokemonInput!): Pokemon!
}
```

```
mutation {
    createPokemon(input: {
        name: "Pikachu",
        type: "Electric"
    }) {
        id
        name
        type
    }
}
```

```
{
    "data": {
        "createPokemon": {
            "id": "1",
            "name": "Pikachu",
            "type": "Electric"
        }
    }
}
```

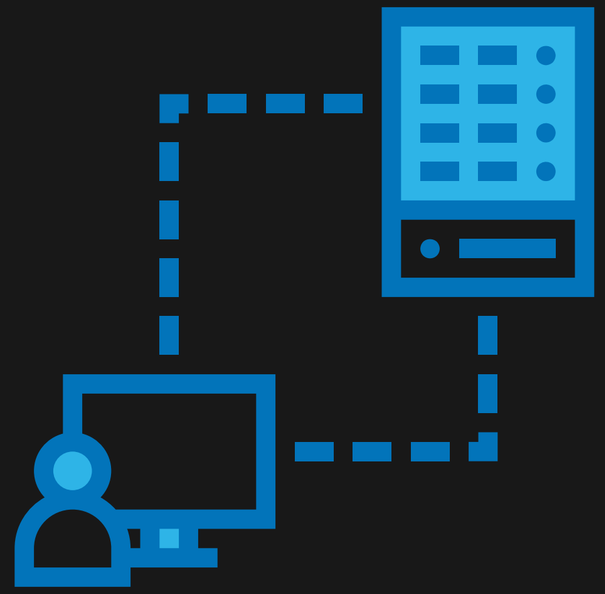
Resolvers

- Functions that fetch or modify data for the fields defined in your schema

```
{
    resolvers:{
        Query:{
            getPokemon:async(id){
                return await db.pokemon.find({id:id});
            }
        }
        Mutation:{
            createPokemon:async(name, type){
                return await db.pokemon.create({name:name, type:type});
            }
        }
    }
}
```

Subscriptions

- Allow clients to receive real-time updates from the server
- Subscriptions use a publish-subscribe pattern



WLUG
COMMUNITY | KNOWLEDGE | SHARE

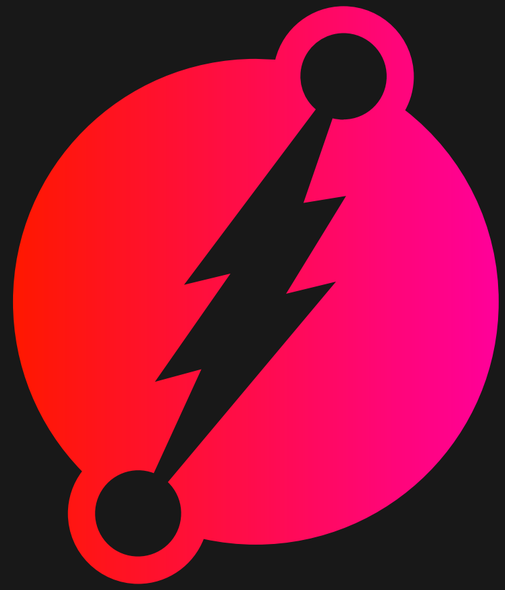
Apollo

- GraphQL client and server
- Development tool
- Apollo engine

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Dgraph

- Open source
- Graph database
- Natively uses GraphQL



WLUG
COMMUNITY | KNOWLEDGE | SHARE

Get Ready!

WLUG
COMMUNITY | KNOWLEDGE | SHARE

THANK YOU

Community | Knowledge | Share

API ALCHEMY

Yash Patil

Tanmay Shingde

Aditya Aparadh

30 June 2024