香港中文大學
The Chinese University of Hong Kong

AN INTRODUCTION TO GRAPHQL

ESTR2106 2022-23 Term 1

Building Web Applications

Dr. Chuck-jee Chau
chuckjee@cse.cuhk.edu.hk

OUTLINE

- GraphQL API
- Using GraphQL in Node.js/Express
- Sending queries
- GraphQL with MongoDB/Mongoose

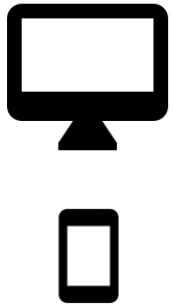
GRAPHQL

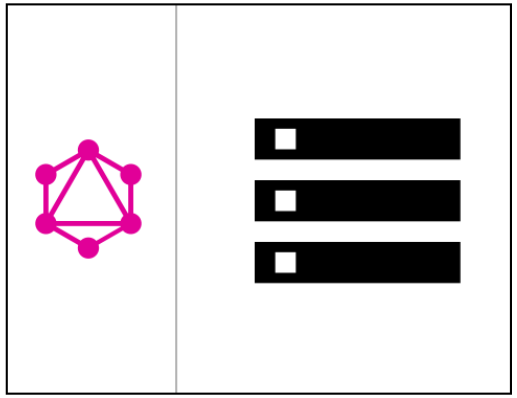
- "Query language for APIs"
  - Types and fields
- Developed by Facebook in 2012
- Moved to non-profit GraphQL Foundation in 2018

- "Ask for what you need, get exactly that"
  - Specifying only the wanted fields from API
- Multiple resources in one ***single request***
- Easier maintenance with evolving version

GRAPHQL API

```
query {
  User(id: “er3tg439frjw”) {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```





```
{
  “data”: {
    “User”: {
      “name”: “Mary”,
      “posts”: [
        { title: “Learn GraphQL today” }
      ],
      “followers”: [
        { name: “John” },
        { name: “Alice” },
        { name: “Sarah” },
      ]
    }
  }
}
```

See: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

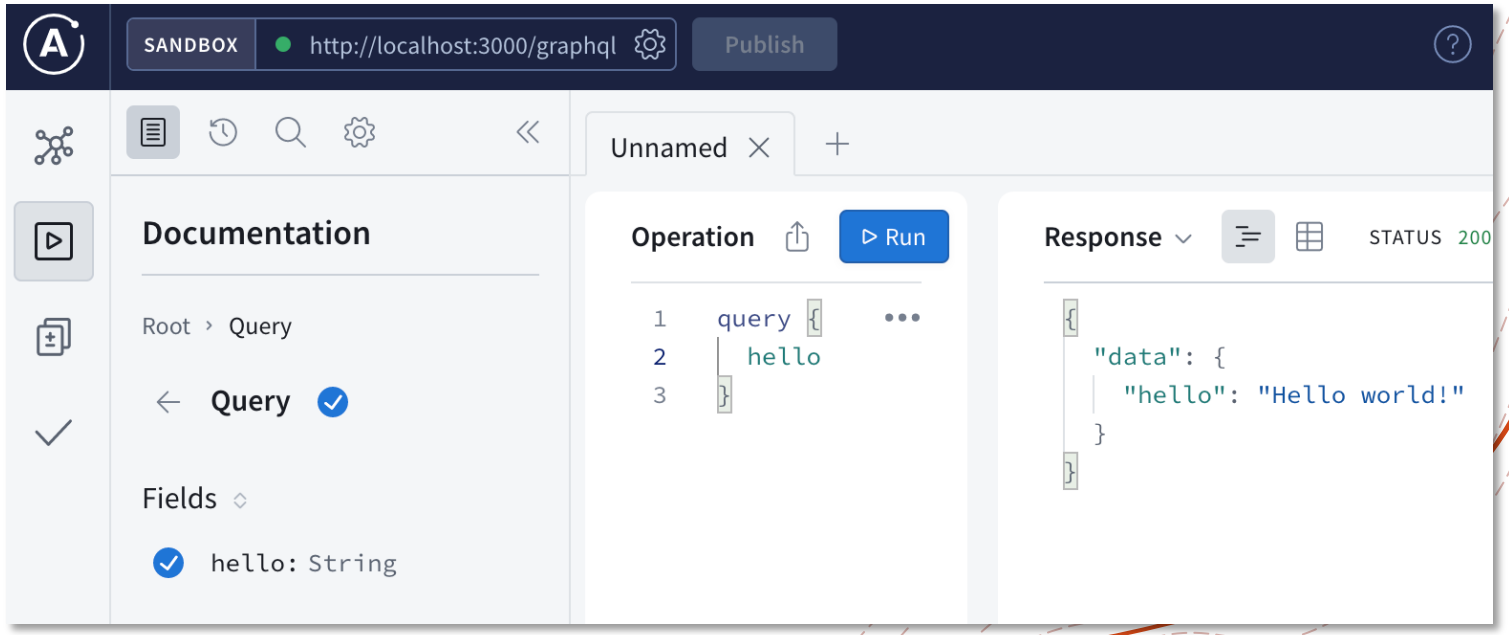
HELLO WORLD

- Apollo Server is a popular GraphQL server
- GraphQL schema
  - Defining types of data to manipulate
- GraphQL resolvers
  - Defining the query results

```
const express = require('express');
const app = express();
const { ApolloServer, gql } = require('apollo-server-
express');

const typeDefs = gql`
  type Query {
    hello: String
  }
`;                          // queries used by resolvers

const resolvers = {
  Query: {
    hello: () => 'Hello world!',
  },
};                          // what to execute

const s = new ApolloServer({ typeDefs, resolvers });
s.start().then(res => {
 s.applyMiddleware({ app });
});

const server = app.listen(3000);
```

HELLO WORLD

- By running Apollo server, a `/graphql` endpoint is automatically provided on the Node/Express server
  - New version of Apollo points it to Apollo Studio, and accepts queries at your Node/Express server (Note: for Google Chrome only)
  - Some other versions and GraphQL provides *GraphiQL*, another query interface



SOME MORE DATA AND QUERIES

- For example, here is a JS array of data

```
const students = [
        { id: 1, name: 'Alice', age: 18.1, courses: ["csci2720"], friends: [2] },
        { id: 2, name: 'Bob', age: 19.0, courses: ["csci2720","csci3100"], friends: [] },
        { id: 3, name: 'Charlie', age: 17.5, friends: [1,2] }
];
```

```
type Student {
  id: Int!
  name: String!
  age: Float
  courses: [String]
}
type Query {
  hello: String
  students: [Student]!
}
```

typeDefs

- We can write a new type **Student**, and a resolver to return all students

```
Query: {
  hello: () => 'Hello world!',
  students: () => students,
  // the whole students array is returned
},
```

Query

SOME MORE DATA AND QUERIES

- Then, you can query for students in `/graphql`
- Yet, you must specify your desired subfields (e.g., `id`, `name`, …)

Operation

ExampleQuery

```
query ExampleQuery {
  students {
    name
    courses
  }
}
```

Variables Headers

JSON

```
1
```

R...

STATUS 200 | 9.00ms | 147B

```
{
  "data": {
    "students": [
      {
        "name": "Alice",
        "courses": [
          "csci2720"
        ]
      },
      {
        "name": "Bob",
        "courses": [
          "csci2720",
          "csci3100"
        ]
      },
      {
        "name": "Charlie",
        "courses": null
      }
    ]
  }
}
```

GRAPHQL SDL

- The schema is defined with the GraphQL schema definition language (SDL)

```
type Student {
  id: Int!
  name: String!
  age: Float
  courses: [String]
  friends: [Student]
}
```

- ***Student*** is the GraphQL object type, with fields
  - ***id***, ***name***, ***age***, ***courses*** are fields
- Default scalar types in GraphQL includes **`Int`** (32-bit), **`Float`** (single precision), **`String`**, **`Boolean`**, and **`ID`**
  - You can use GraphQL types too

- There are also arrays denoted by **`[ ]`**
- The exclamation mark **`!`** denotes non-nullable fields
- *See: https://graphql.org/learn/schema/*

QUERY ARGUMENTS

```
query {
  student(id: 2) {
    name
  }
}
```

The query, not JS code

- Arguments can be added to a query, specified for an object or a field, and handled by the resolver

```
type Query {
  hello: String
  students: [Student]!
  student(id: Int!): Student
}
```

```
Query: {
  hello: () => 'Hello world!',
  students: () => students,
  student: (parent, {id}) => students.find(
                stu => stu.id === id)
},
```

- The resolver has this function signature

  fieldName: (***parent, args, context, info***) => *data*;

  - When not used, the latter arguments are skipped from the code

NESTED OBJECT

- An object can also contain pointers towards another object
- For example, the id of students in friends

```
{ id: 3, name: 'Charlie', age: 17.5, friends: [1,2] }
```

- The schema needs to contain what to output

```
type Student {
  id: Int!
  name: String!
  age: Float
  courses: [String]
  friends: [Student]
}
```

- A new resolver for friends in student query

```
Student: {
  friends: (parent)=> {
    const {friends} = parent;
    return students.filter(
      stu => friends.includes(stu.id));
  }
},
```

NESTED OBJECT

Operation ▷ Run

```
1  query {
2    students {
3      name
4      friends { name }
5    }
6  }
```

Response STATUS 200 | 10.0ms | 159B

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

UPDATING DATA WITH GRAPHQL

- GraphQL is usually used for data fetching
- Yet, data updating is also possible with Mutation endpoints instead of Query endpoints
  - Appropriate **`Mutation`** and **`Input`** types are needed
- *See: https://graphql.org/graphql-js/mutations-and-input-types/*

SENDING QUERIES TO GRAPHQL

- GraphQL accepts queries via GET or POST
  - e.g., JS fetch calls on client side

```
fetch('http://localhost:3000/graphql?query=query{students{name}}')
.then(res=>res.json())
.then(data=>console.log(data))
)
```

```
▼ {data: {…}}
  ▼ data:
    ▼ students: Array(3)
      ▶ 0: {name: 'Alice'}
      ▶ 1: {name: 'Bob'}
      ▶ 2: {name: 'Charlie'}
        length: 3
```

```
fetch('http://localhost:3000/graphql', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({query: `
query {
  students {
    name
  }
}`
    }),
})
.then(res=>res.json())
.then(data=>console.log(data))
```

GRAPHQL WITH MONGODB/MONGOOSE

```
/* require, init mongoose and apollo ... */

const Schema = mongoose.Schema;
const UserSchema = Schema({
  name: { type: String, required: true },
  email: { type: String, required: true },
  password: { type: String, required: true }
});
const User = mongoose.model('User',
UserSchema);

const typeDefs = gql`
  type User {
    name: String!
    email: String!
    password: String!
  }
  type Query {
    users: [User]
  }
`;

const resolvers = {
  Query: {
    users: async () => {
      const result = await User.find().exec();
      return result;
    } // async needed for DB access
  },
};

/* ... other server options ... */
```

READ FURTHER…

Introduction to GraphQL

https://graphql.org/learn/

Apollo: Full-stack Tutorial

https://www.apollographql.com/docs/tutorial/introduction