REST API vs GraphQL

mash up 10기 노드팀 10분 세미나

mash up 10기 노드팀
이소형

REST

- Representational State Transfer

- 서버의 자원을 정의하고 자원에 대한 주소를 지정하는 방법

- 리소스의 유형과 해당 리소스를 가져오는 방식(엔드 포인트)이 결합되어 있다.
- 리소스에 대한 구조화된 액세스

  클라이언트가 해당 엔드 포인트에 간단히 액세스하여 필요한 모든 정보를 얻을 수 있으므로 편리하다.

REST의 구성 요소

- 자원(Resource): URI
  - 무엇을
  - 명사
  - URL로 식별
- 행위(Verb): Method
  - 어떻게 한다
  - 동사
  - HTTP Method: POST(Create) / GET(Read) / PUT(update) / DELETE(Delete)
- 표현(Representation): Body(Message Payload)
  - 리소스의 내용은 json, xml, yaml 등의 다양한 표현 언어로 정의

- 메소드와 URI를 조합해서, 예측 가능하고 일정한 정보와 작업을 요청
- 주소 하나가 요청 메서드를 여러 개 가질 수 있다.

REST의 특징

- Self - descriptiveness
- Uniform interface
- Stateless
- Cacheable
- Layered System
- Client - Server 구조

REST API 예

- 서버가 이해하기 쉬운 주소로 잘 정의된 API

- GET /posts 게시글 리스트 가져오기
- GET /posts/1 게시글 1을 가져오기
- POST /posts 게시글 생성하기
- PUT /posts/1 게시글 1을 수정하기
- DELETE /posts/1 게시글 1을 삭제하기
- GET /posts/1/comments
- GET /posts/1/comments/1

REST API 단점

- Over Fetching

→ 전송되는 데이터의 양 측면에서 소모가 크다

- Under Fetching

→ n + 1 문제

- 액세스하는 클라이언트의 빠르게 변화하는 요구 사항을 따라 잡기에는 너무 유연하지 않다.

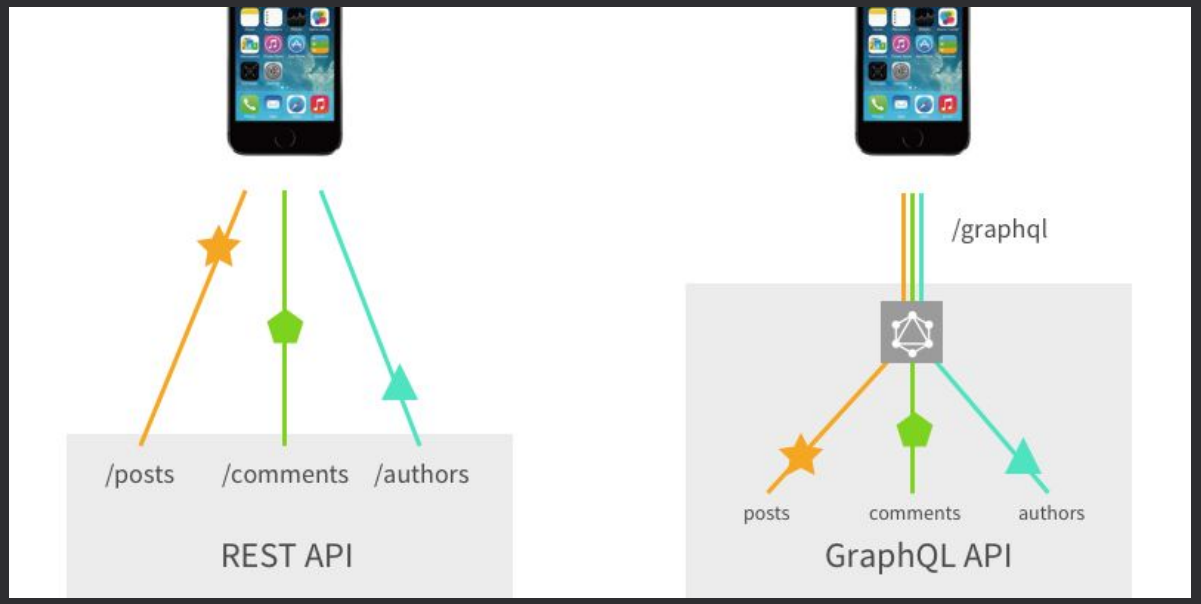
⇒ UI가 변경될 때마다 새로운 데이터 요구사항을 고려하여 백엔드도 조정해야 한다.

→ 이로 인해 생산성이 저하되고 사용자 피드백을 제품에 통합하는 기능이 현저히 느려진다.

GraphQL

- API를 위한 서버 측 런타임 및 쿼리 언어
- 데이터의 type과 데이터를 가져오는 방법이 완전히 분리되어 있다.
- 서버에서 정의한 엔드 포인트 대신 쿼리를 전송하여 하나의 요청(단일 쿼리)으로 원하는 데이터를 정확히 얻을 수 있다.

→ 대역폭을 절약하고 요청을 줄일 수 있다.



API에서 데이터를 가져올 때 주요 차이점

여러 관련 함수를 호출하기 위해

REST API는 여러 엔드 포인트에 액세스

GraphQL API는 하나의 요청

GraphQL 탄생 배경

GraphQL: A data query language

2015. 9. 14. by Lee Byron

When we built Facebook's mobile applications, we needed a data-fetching API powerful enough to describe all of Facebook, yet simple and easy to learn so product developers can focus on building things quickly. We developed GraphQL three years ago to fill this need. Today it powers hundreds of billions of API calls a day. This year we've begun the process of open-sourcing GraphQL by drafting a specification, releasing a reference implementation, and forming a community around it here at graphql.org.

GraphQL 데이터의 진입점

- 구체적인 데이터 요구 사항을 포함하는 단일 쿼리를 GraphQL 서버에 보내면 서버는 이러한 요구 사항이 충족되는 JSON 객체로 응답

- POST 요청

- Query type: 읽기
- Mutation type: 쓰기
- Subscription type: 특정 이벤트가 발생할 때 서버가 클라이언트에게 데이터를 보낼 수 있도록 하는 기능 (WebSocket)

예

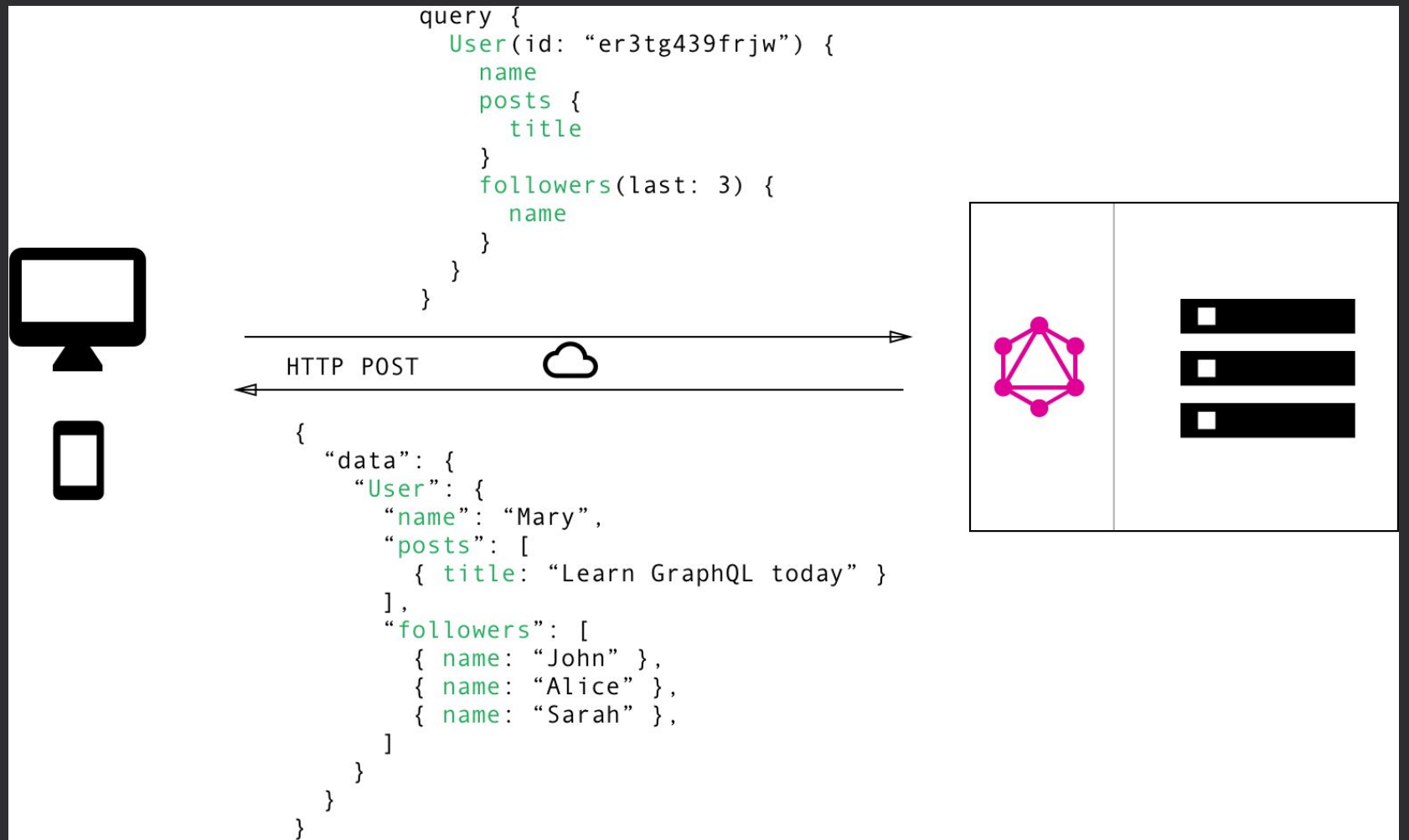
- 특정 사용자의 게시물 제목을 표시
- 해당 사용자의 마지막 팔로워 3명의 이름도 표시

예

- REST API를 사용하면 필요한 데이터를 가져 오기 위해 서로 다른 엔드 포인트에 세 가지 요청을 해야 한다.
  - /users/<id>: 초기 사용자 데이터를 가져오는 엔드 포인트
  - /users/<id>/posts: 사용자의 모든 게시물을 반환하는 엔드 포인트
  - /users/<id>/followers: 사용자 당 팔로워 목록을 반환하는 엔드 포인트
- 엔드 포인트가 필요하지 않은 추가 정보를 반환하기 때문에 오버 페치가 발생한다.

예

- GraphQL을 사용하여 클라이언트는 쿼리에 필요한 데이터를 정확하게 지정할 수 있다.



GraphQL 장점

- 진입점이 하나다
- 딱 원하는 column만 요청해서 받을 수 있다.

- 쿼리 작성에 자유도가 높다

⇒ 클라이언트는 정확한 데이터 요구 사항을 지정할 수 있으므로 프런트 엔드에서 설계 및 데이터가 변경될 때 서버에 대한 추가 작업 없이 클라이언트 측에서 변경할 수 있다.

⇒ REST의 가장 일반적인 문제 중 하나인 오버 페치 및 언더 페치 방지

- 오버 페치: 불필요한 데이터 다운로드
- 언더 페치: 특정 엔드 포인트가 필요한 정보를 충분히 제공하지 않음

→ 유연성과 효율성을 높인다

- API 문서화 작업을 하지 않아도 된다.

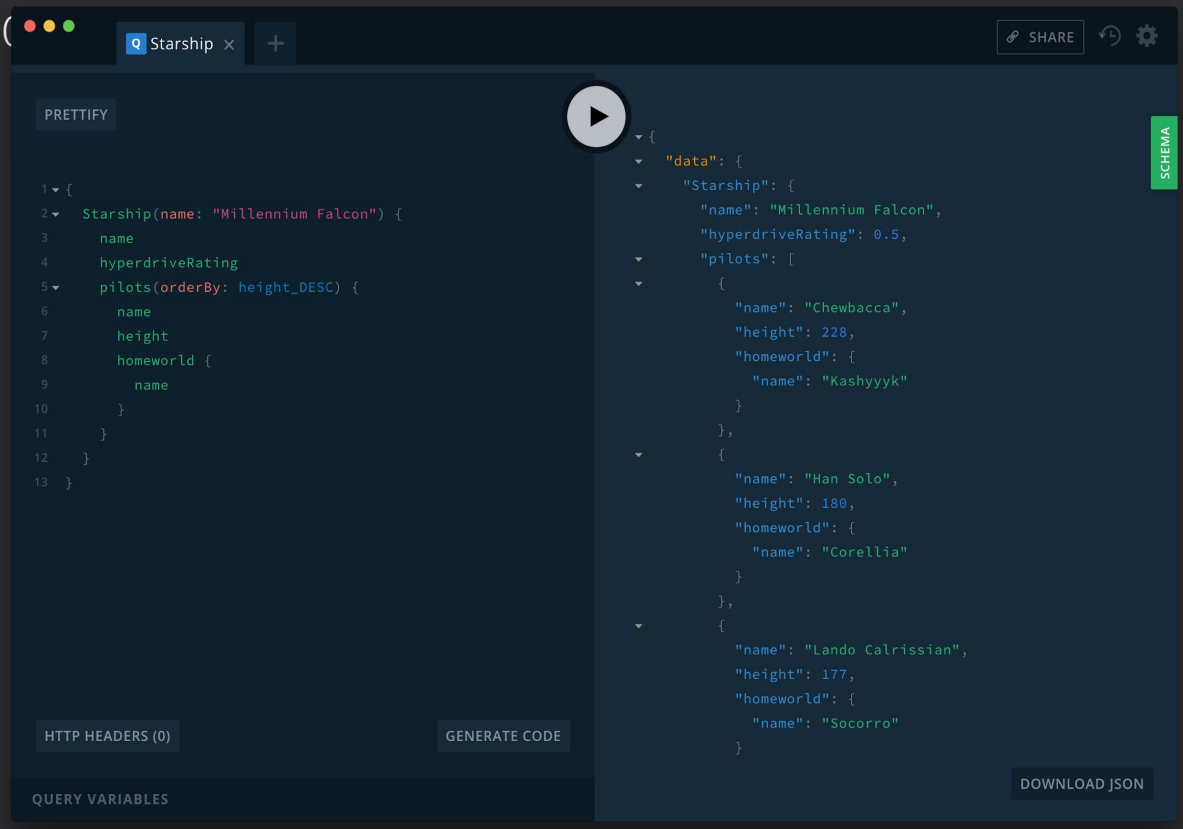
API 문서

- 호출할 수 있는 항목과 결과로 수신할 항목에 대한 설명

- RESTful API: 문서화 도구 사용

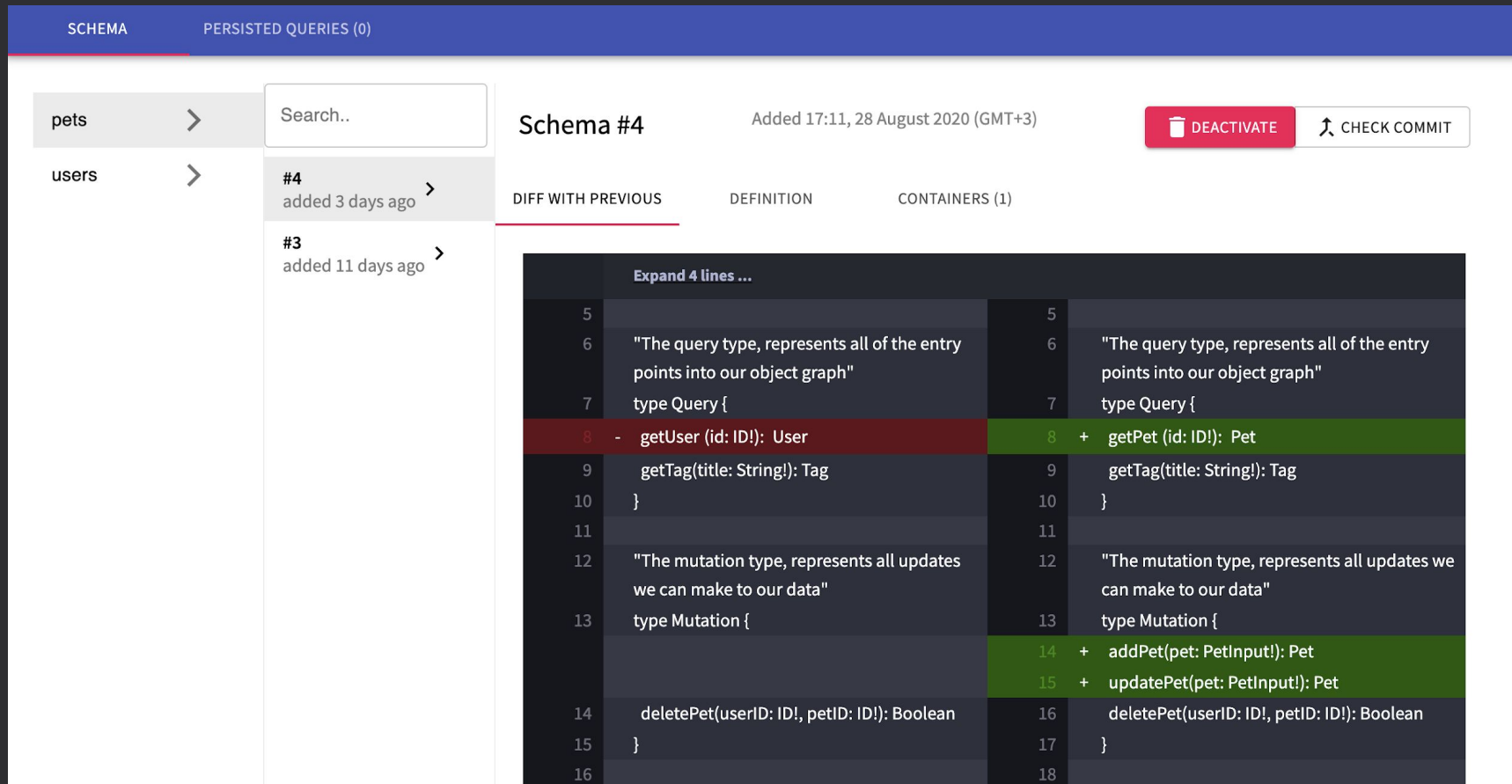
필드 타입이 정해져 있지 않아서 따로 문서화를 하지 않으면 협업(

- GraphQL: 자체 문서화 스키마
  - Apollo Studio의 Explorer
  - GraphQL Playground
  - GraphiQL



버전 관리

- REST API: 버전 번호
- GraphQL: 스키마 레지스트리



GraphQL 구현

- node.js에 Express로 구현

- graphql-yoga 모듈을 사용

GraphQL 서버 구축에 필요한 기능을 제공하는 graphql.js, express 및 apollo-server와 같은 다양한 다른 패키지 위에서 시작

→ 빠르게 시작할 수 있다

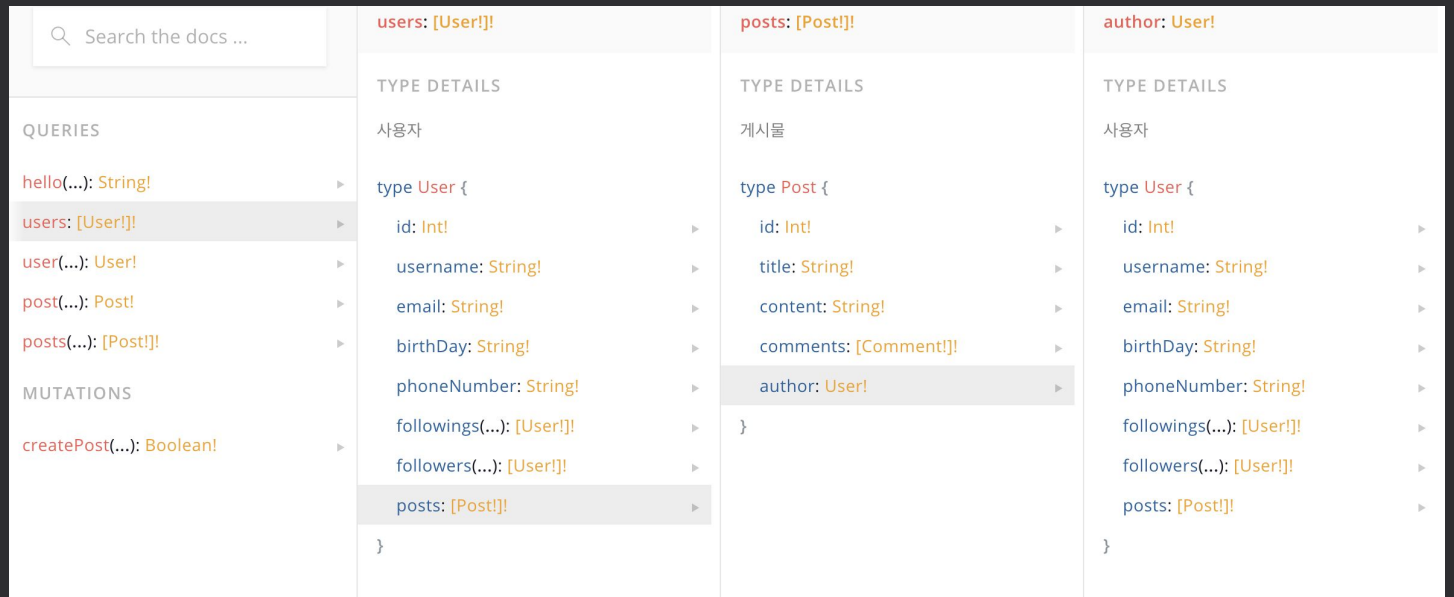
```
import { GraphQLServer } from 'graphql-yoga'
// ... or using `require()`
// const { GraphQLServer } = require('graphql-yoga')

const typeDefs = `
  type Query {
    hello(name: String): String!
  }
`

const resolvers = {
  Query: {
    hello: (_, { name }) => `Hello ${name || 'World'}`,
  },
}

const server = new GraphQLServer({ typeDefs, resolvers })
server.start(() => console.log('Server is running on localhost:4000'))
```

API 내부 구현

- REST API: 라우트 핸들러

REST의 엔드 포인트가 서버에서 호출하는 함수

- GraphQL
  - 스키마: 데이터의 구조나 표현법, 관계를 나타내는 자료형
  - 리졸버: GraphQL의 필드가 서버에서 호출하는 함수
    - parent, args, context, info



GraphQL 중첩 리졸버

- 하나의 요청은 여러 필드의 결과를 모두 얻을 수 있다.

```
# Write your query or mutation here
query {
  users {
    id
    username
  }
  posts {
    id
    title
  }
}
```

GraphQL 중첩 리졸버

- 스키마에 정의된 관계에 따라 추가 데이터를 가져오는 쿼리를 보낼 수 있다.

```
query {
  user(id: 1) {
    username
    email
  }
}
```

```
query {
  user(id: 1) {
    username
    email
    posts {
      title
    }
  }
}
```

```
export default {
  Query: {
    user: async (_, { id }) => {
      const user = data.users.find(user => user.id === id)
      return user;
    },
  },
  User: {
    posts: async (user) => {
      const posts = data.posts.filter(post => post.author === user.id);
      return posts;
    },
  },
};
```

GraphQL 단점

- 파일 전송 등 json 형식이 아닌 데이터에 대한 처리가 복잡하다

- 받아야 하는 항목들이 많고 딱 정해져 있는 경우에도 그것들을 하나하나 GraphQL 요청에 적어 보내야 한다.

→ 요청의 크기가 커진다

REST API & GraphQL

index.js > ...

```
1   import { GraphQLServer } from 'graphql-yoga';
2
3   const typeDefs = "./graphql/schema.graphql";
4
5   import resolvers from './graphql/resolvers.js';
6
7   const options = {
8     port: 4000,
9     endpoint: '/graphql',
10    subscriptions: '/subscriptions',
11    playground: '/playground',
12  }
13
14  const server = new GraphQLServer({ typeDefs, resolvers })
15
16  server.express.get('/', (req, res) => {
17    res.send('Hello World!')
18  })
19
20  server.start(options, () => console.log(`Server is running ⚡ on localhost:${options.port}`))
21  .catch(err => console.error('connection Error', err))
22
```

감사합니다

mash up 10기 노드팀 10분 세미나