

Choi, Jansen Kai Xuan

## Building Microservices with Database Migrations and GraphQL CRUD Endpoints

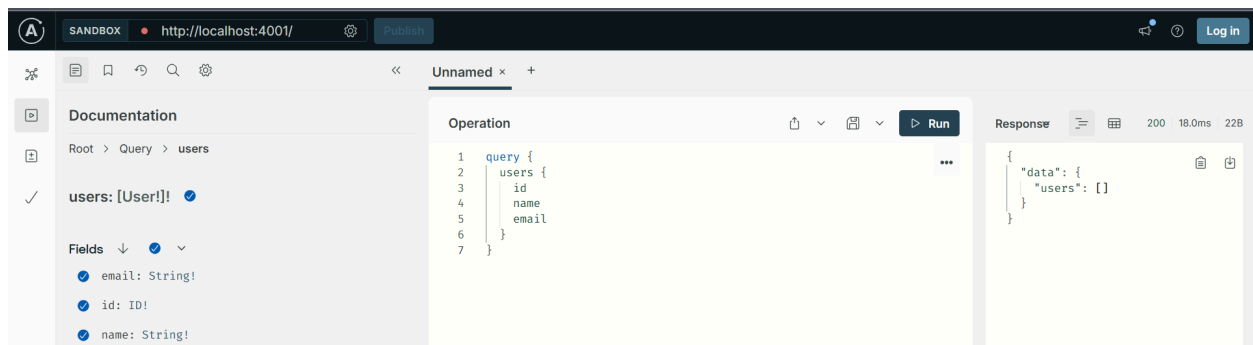
### What do database migrations do and why are they useful?

- The purpose of database migrations is to allow for version control changes to a database schema that was already defined, ensuring consistency across environments. They help transition database schemas from a current state to a new desired state whether it involves adding tables, removing elements, updates, etc. They help manage schema evolution, making it easier to apply, track, and revert changes. This is useful in cases where a microservice architecture is used, where multiple services interact with their own databases.

### How does GraphQL differ from REST for CRUD operations?

- GraphQL differs from REST for CRUD operations as it provides a more flexible query system where users can directly specify the data they need. This reduces over-fetching or under-fetching because they can alter and filter queries based on their requirements. In addition to that, while REST uses multiple endpoints for different CRUD operations, GraphQL uses a single endpoint for all queries, whether reading, inserting, updating, or deleting. This results in more efficient data retrieval and making users less constrained.

### USERS DATABASE



The screenshot shows the GraphQL Playground interface. The URL bar indicates the endpoint is `http://localhost:4001/`. The left sidebar shows the documentation for the `users` query, which returns a list of `User!` objects. The fields `email`, `id`, and `name` are selected. The main editor shows the following query:

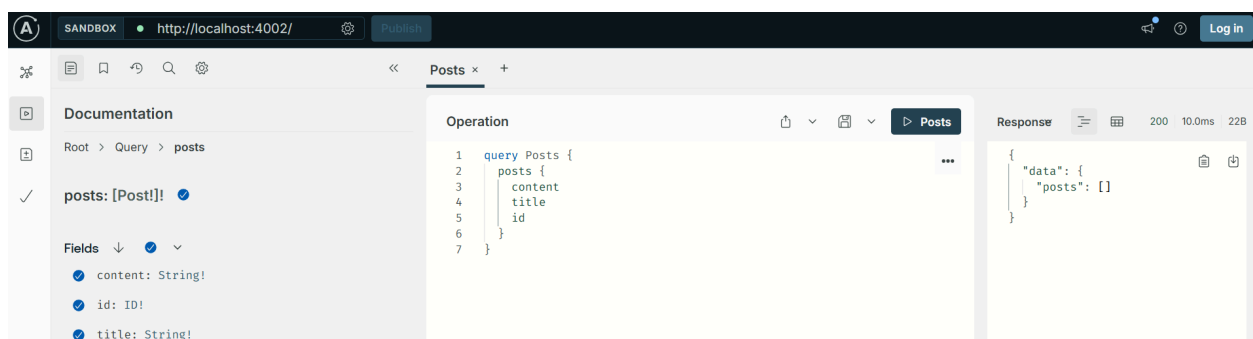
```
1 query {
2   users {
3     id
4     name
5     email
6   }
7 }
```

The right sidebar shows the response, which is a JSON object:

```
{
  "data": {
    "users": []
  }
}
```

The status bar at the bottom indicates a 200 status code, 18.0ms execution time, and 22B response size.

### POSTS DATABASE



The screenshot shows the GraphQL Playground interface. The URL bar indicates the endpoint is `http://localhost:4002/`. The left sidebar shows the documentation for the `posts` query, which returns a list of `Post!` objects. The fields `content`, `id`, and `title` are selected. The main editor shows the following query:

```
1 query Posts {
2   posts {
3     content
4     title
5     id
6   }
7 }
```

The right sidebar shows the response, which is a JSON object:

```
{
  "data": {
    "posts": []
  }
}
```

The status bar at the bottom indicates a 200 status code, 10.0ms execution time, and 22B response size.

Unset

## USERS

**//SELECT**

```
query {  
  users {  
    id  
    name  
    email  
  }  
}
```

**//INSERT**

```
mutation {  
  createUser(name: "Jansen", email: "jansen@gmail.com") {  
    id  
    name  
    email  
  }  
}
```

**//UPDATE**

```
mutation {  
  updateUser(id: "1", name: "Jans", email: "jans@gmail.com") {  
    id  
    name  
    email  
  }  
}
```

**//DELETE**

```
mutation {  
  deleteUser(id: "1") {  
    id  
    name  
    email  
  }  
}
```

Unset

## POST

**//SELECT**

```
query {  
  posts {  
    id
```

```
        title
        content
    }
}

//INSERT
mutation {
  createPost(title: "SIA", content: "SIA Notes") {
    id
    title
    content
  }
}

//UPDATE
mutation {
  updatePost(id: "1", title: "Lesson 1", content: "Microservice with Database
Migrations") {
    id
    title
    content
  }
}

//DELETE
mutation {
  deletePost(id: "1") {
    id
    title
    content
  }
}
```