

Building Microservices with Database Migrations and GraphQL CRUD Endpoints

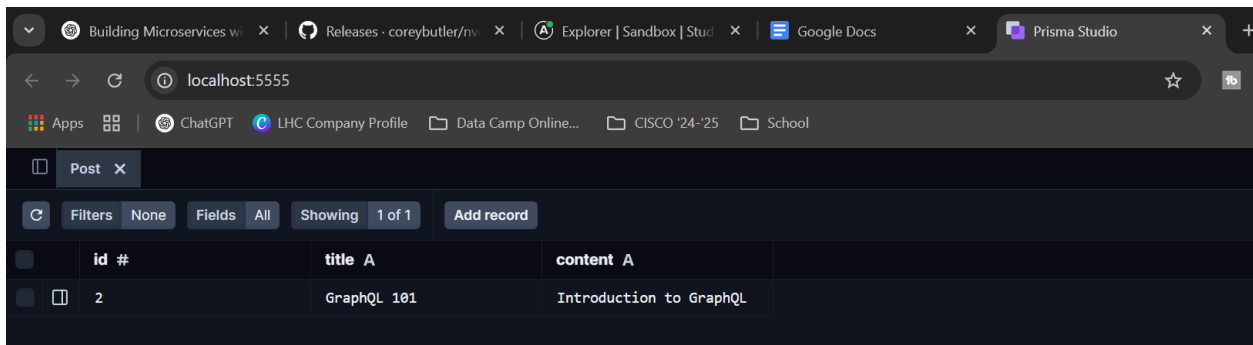
What do database migrations do and why are they useful?

- Database migrations help manage changes to the database schema over time by keeping track of modifications in a structured way. They ensure that all team members and environments have the same database structure without needing to update tables manually. This is especially useful in microservices, where different services have their databases that need to stay in sync. Migrations also allow developers to roll back changes if something goes wrong, making database management safer. Overall, they make updating and maintaining databases more efficient and reliable.

How does GraphQL differ from REST for CRUD operations?

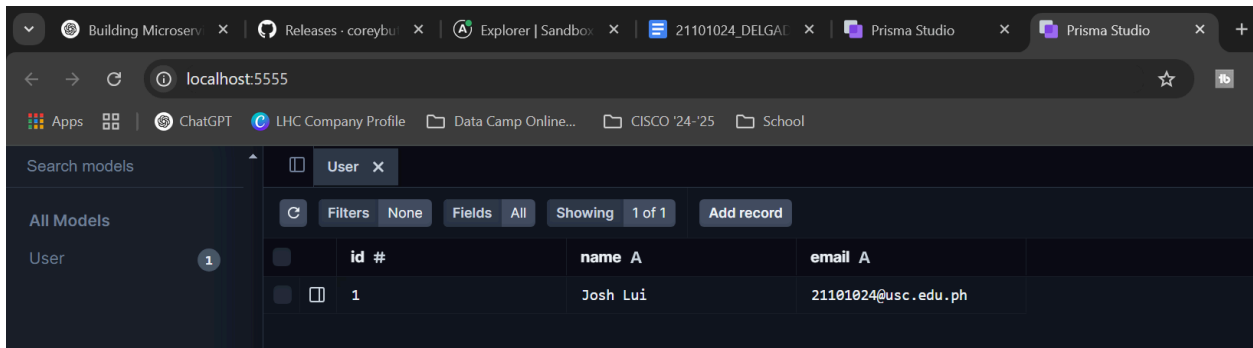
- GraphQL differs from REST in how it handles data retrieval and manipulation, making it more flexible and efficient. Instead of multiple endpoints like in REST, GraphQL has a single endpoint where clients request exactly the data they need. This helps prevent over-fetching data and under-fetching data, improving performance. GraphQL also allows clients to request multiple related resources in a single query, reducing the number of network requests. This makes it a great choice for applications that need fast and dynamic data fetching.

Post Database



	id #	title A	content A
	2	GraphQL 101	Introduction to GraphQL

User Database



	id #	name A	email A
	1	Josh Lui	21101024@usc.edu.ph

Post Database Endpoint Testing

```
Unset
# Create
mutation {
  createPost(title: "GraphQL 101", content: "Introduction to GraphQL") {
    id
    title
    content
  }
}

# Read
query {
  posts {
    id
    title
    content
  }
}

# Update
mutation {
  updatePost(id: 1, title: "Advanced GraphQL") {
    id
    title
    content
  }
}

# Delete
mutation {
  deletePost(id: 1){
    id
    title
    content
  }
}
```

User Database Endpoint Testing

```
Unset
# Create
mutation {
```

```
    createUser(name: "Jio", email: "21101024@usc.edu.ph") {
      id
      name
      email
    }
  }
}

# Read
query {
  users {
    id
    name
    email
  }
}

# Update
mutation {
  updateUser(id: 1, name: "Josh Lui") {
    id
    name
    email
  }
}

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