

## Post - Services

### Creation Example:

The screenshot displays the GraphQL Studio interface for a POST mutation. The 'Operation' tab on the left contains the following query:

```
1 mutation($title: String!, $content: String!){  
2   createPost(title: $title, content: $content) {  
3     id  
4     title  
5     content  
6   }  
7 }  
8
```

The 'Response' tab on the right shows the JSON output:

```
{  
  "data": {  
    "createPost": {  
      "id": 1,  
      "title": "I love JL!",  
      "content": "Best Friendo!"  
    }  
  }  
}
```

At the bottom, the 'Variables' tab shows the input variables:

```
1 {  
2   "title": "I love JL!",  
3   "content": "Best Friendo!"  
}
```

The status bar at the top right indicates a 200 status code, 116ms execution time, and 80B response size.

### Read Example:

The screenshot displays the GraphQL Studio interface for a GET query. The 'Operation' tab on the left contains the following query:

```
1 query{  
2   posts {  
3     id  
4     title  
5     content  
6   }  
7 }  
8  
9
```

The 'Response' tab on the right shows the JSON output:

```
{  
  "data": {  
    "posts": [  
      {  
        "id": 1,  
        "title": "I love JL!",  
        "content": "Best  
        Friendo!"  
      }  
    ]  
  }  
}
```

At the bottom, the 'Variables' tab shows the input variables:

```
1 {  
2   "title": "I love JL!",  
3   "content": "Best Friendo!"  
}
```

The status bar at the top right indicates a 200 status code, 27.0ms execution time, and 77B response size.

## Update Example:

Operation

📄

⌵

💾

⌵

▶ Run

```
1 mutation($updatePostId: Int!, $title: String!, $content: String!){ ...
2   updatePost(id: $updatePostId, title: $title, content: $content) {
3     id
4     title
5     content
6   }
7 }
8
9
```

Response

≡

📄

200

26.0ms

92B

📄

🔗

```
{
  "data": {
    "updatePost": {
      "id": 1,
      "title": "I Still Love
      JL!",
      "content": "Still Best
      Friendo!"
    }
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

⌵

```
1 {
2   "title": "I Still Love JL!",
3   "content": "Still Best Friendo!",
4   "updatePostId": 1
5 }
```

JSON

## Delete Example:

Operation

📄

⌵

💾

⌵

▶ Run

```
1 mutation($deletePostId: Int!){
2   deletePost(id: $deletePostId) {
3     id
4   }
5 }
6
7
```

Response

≡

📄

200

24.0ms

33B

📄

🔗

```
{
  "data": {
    "deletePost": {
      "id": 1
    }
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

⌵

```
1 {
2   "deletePostId": 1
3 }
```

JSON

## Users - Services

### Create Users

Operation

📄

▼

📁

▼

▶ Run

...

```
1 mutation($name: String!, $email: String!){
2   createUser(name: $name, email: $email) {
3     id
4     name
5     email
6   }
7 }
8
```

Response

≡

📄

200

62.0ms

92B

📄

🔗

```
{
  "data": {
    "createUser": {
      "id": 1,
      "name": "John Lorenz",
      "email": "Lanceisbestfriend@gmail.com"
    }
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

⌵

```
1 { "name": "John Lorenz",
2   "email": "Lanceisbestfriend@gmail.com"
3 }
```

JSON

### Read Users

Operation

📄

▼

📁

▼

▶ Run

...

```
1 query{
2   users {
3     id
4     name
5     email
6   }
7 }
8
```

Response

≡

📄

200

30.0ms

89B

📄

🔗

```
{
  "data": {
    "users": [
      {
        "id": 1,
        "name": "John Lorenz",
        "email": "Lanceisbestfriend@gmail.com"
      }
    ]
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

⌵

```
1 { "name": "John Lorenz",
2   "email": "Lanceisbestfriend@gmail.com"
3 }
```

JSON

## Update Users

Operation

1

mutation(\$updateUserId: Int!, \$name: String!, \$email: String){

2

updateUser(id: \$updateUserId,name:\$name,email: \$email) {

3

id

4

name

5

email

6

}

7

}

8

Run

Response

200

25.0ms

95B

```
{
  "data": {
    "updateUser": {
      "id": 1,
      "name": "John Lorenz",
      "email": "LanceisNOTbestfriend@gmail.com"
    }
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

1

{

2

"name": "John Lorenz",

3

"email": "LanceisNOTbestfriend@gmail.com",

4

"updateUserId": 1

}

JSON

## Delete Users

Operation

1

mutation(\$deleteUserId: Int!){

2

deleteUser(id: \$deleteUserId) {

3

id

4

}

5

}

6

Run

Response

200

25.0ms

33B

```
{
  "data": {
    "deleteUser": {
      "id": 1
    }
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

1

{

2

"deleteUserId": 1

3

}

JSON

## Reflection

This activity gave me some hands-on experience in building microservices with Node.js, Prisma, and Apollo Server. One of the main takeaways was really grasping the concept of database migrations and why they matter. Migrations are essential for tracking changes to the database schema, ensuring everything stays consistent across different environments, and making it easier to tweak the structure without risking data loss. Prisma's migration system made this process a breeze, allowing for smooth schema evolution.

Another key point was diving into GraphQL for CRUD operations. Unlike REST, which has set endpoints for each action, GraphQL offers a more flexible way to fetch data by letting clients request only the fields they need. This approach cuts down on both over-fetching and under-fetching, boosting efficiency. Setting up separate microservices for users and posts really highlighted the perks of a modular and scalable system architecture, where each service can work independently but still communicate when necessary.

All in all, this exercise really deepened my understanding of microservice architecture, database management, and the benefits of using GraphQL in today's application development landscape.