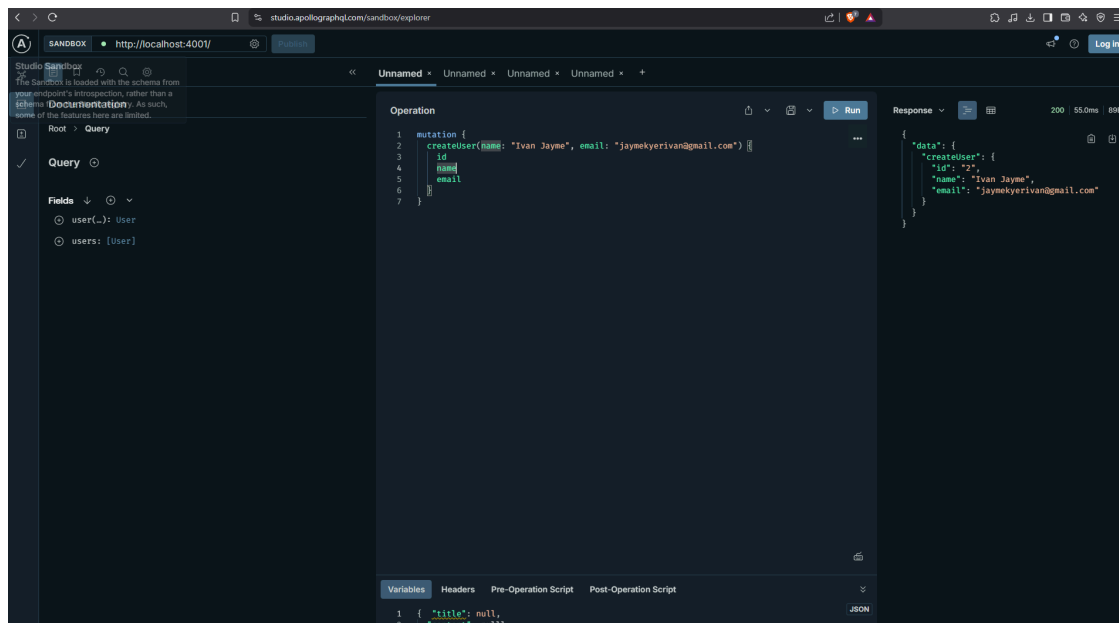


Building Microservices with Database Migrations and GraphQL CRUD Endpoints

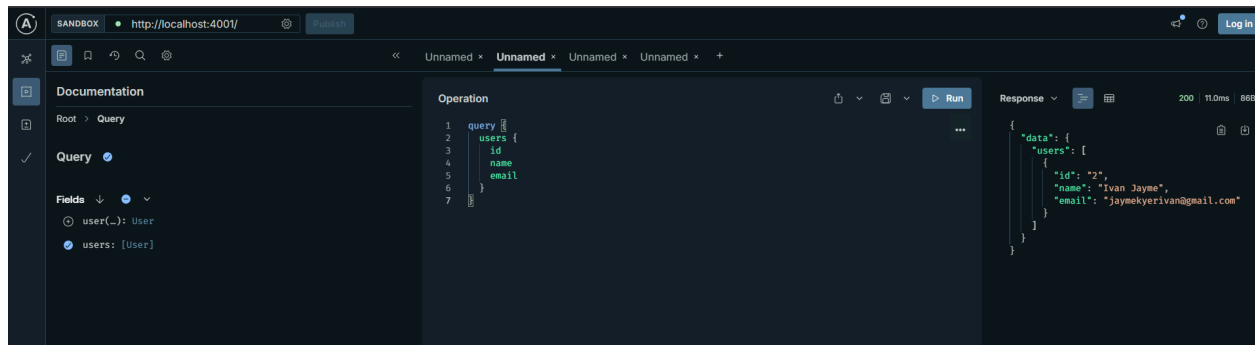
CREATE NEW USER

```
mutation {  
  createUser(name: "Ivan Jayme", email: "jaymekyerivan@gmail.com") {  
    id  
    name  
    email  
  }  
}
```



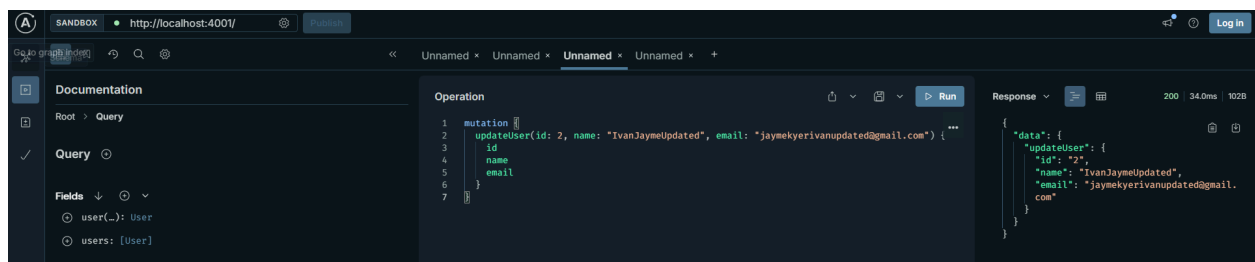
GET ALL USERS

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

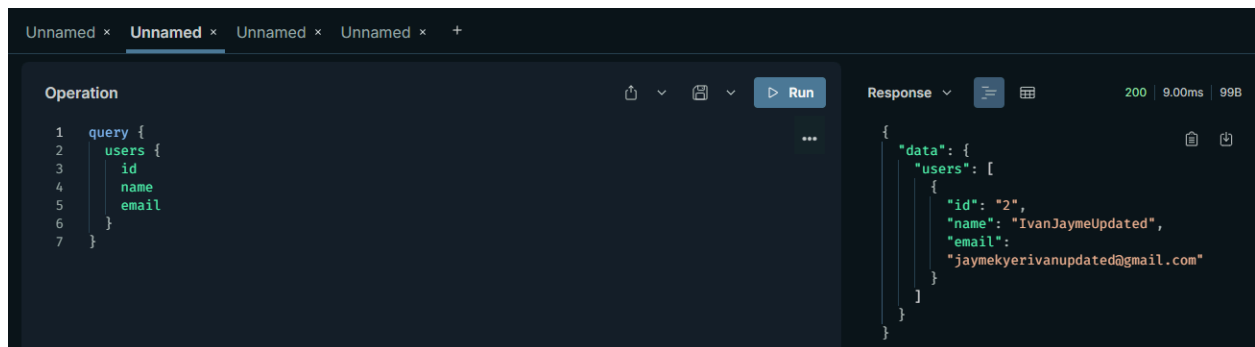


UPDATE USER

```
mutation {
  updateUser(id: 2, name: "IvanJaymeUpdated", email:
    "jaymekyerivanupdated@gmail.com") {
    id
    name
    email
  }
}
```



GETTING ALL USERS (UDPATED)



Deletion of user

```
mutation {  
  updatePost(id: 1, title: "Jaymer's journey to heaven", content: "Did he  
make it?") {  
    id  
    title  
    content  
  }  
}
```

The screenshot shows a GraphQL IDE interface with a dark theme. The 'Operation' pane on the left contains a mutation to delete a user by ID:

```
1 mutation{  
2   deleteUser(id: 2) {  
3     id  
4     name  
5     email  
6   }  
7 }
```

. The 'Response' pane on the right shows the JSON output:

```
{  
  "data": {  
    "deleteUser": {  
      "id": "2",  
      "name": "IvanJaymeUpdated",  
      "email": "jaymekyerivanupdated@gmail.com"  
    }  
  }  
}
```

. The status bar at the top right indicates a 200 status code, 38.0ms execution time, and 102B response size.

GETTING ALL USERS (DELETED)

The screenshot shows the same GraphQL IDE interface. The 'Operation' pane on the left contains a query to fetch all users:

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

. The 'Response' pane on the right shows the JSON output:

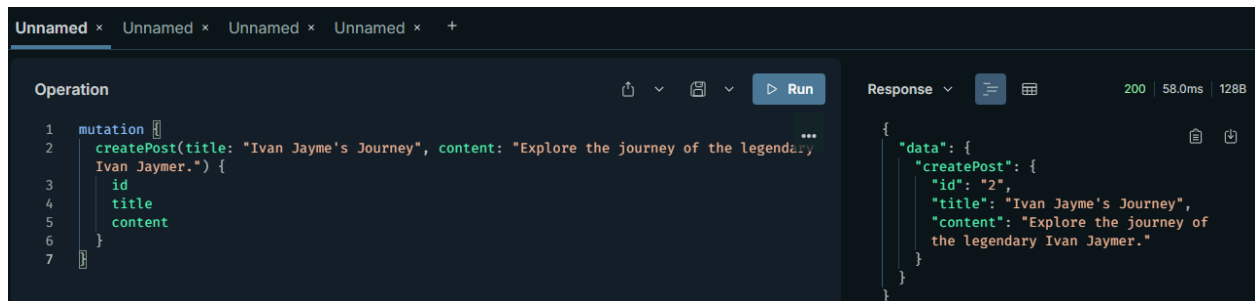
```
{  
  "data": {  
    "users": []  
  }  
}
```

. The status bar at the top right indicates a 200 status code, 14.0ms execution time, and 22B response size.

POST

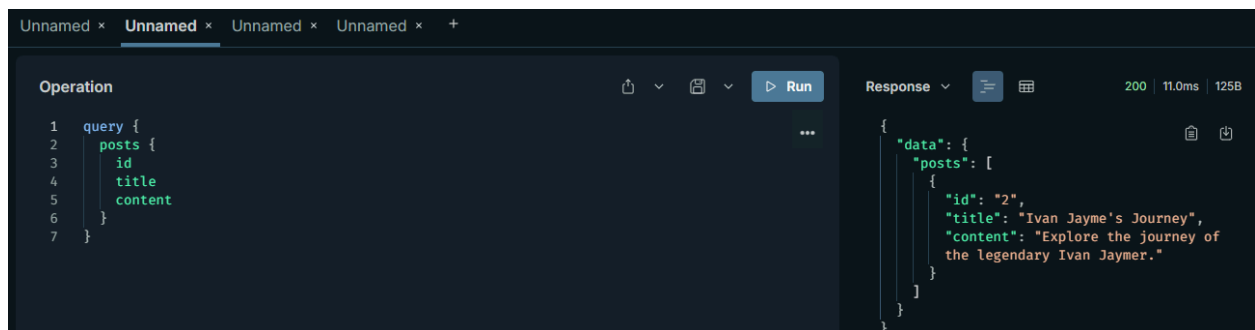
Creating new post

```
mutation {  
  createPost(title: "Ivan Jayme's Journey", content: "Explore the journey  
of the legendary Ivan Jaymer.") {  
    id  
    title  
    content  
  }  
}
```



Getting all posts

```
query {  
  posts {  
    id  
    title  
    content  
  }  
}
```



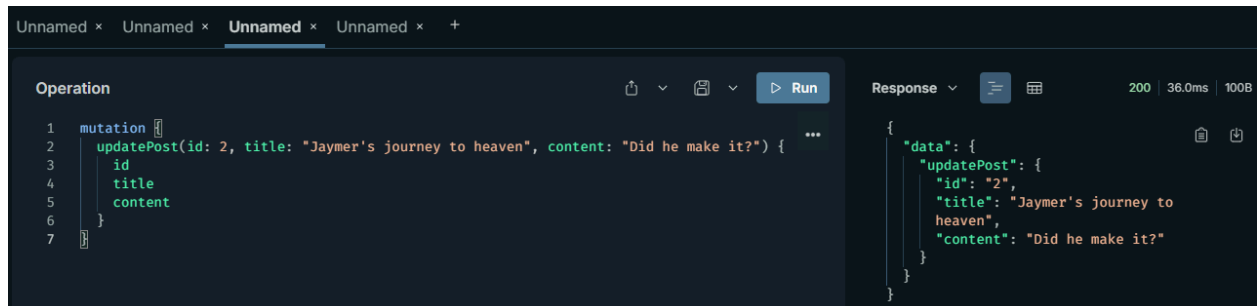
Update Post

```
mutation {
```

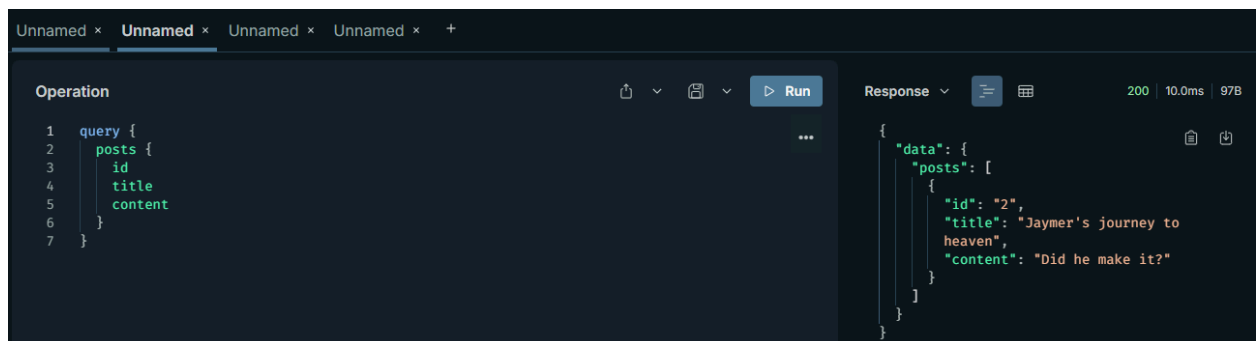
```

    updatePost(id: 2, title: "Jaymer's journey to heaven", content: "Did he
make it?") {
      id
      title
      content
    }
  }
}

```



Getting all post (updated)



Deleting Post

```

mutation{
  deletePost(id: 2) {
    id
    title
    content
  }
}

```

The screenshot shows a GraphQL IDE with a dark theme. The top bar has four tabs labeled 'Unnamed'. The 'Operation' pane on the left contains a GraphQL mutation:

```
1 mutation{
2   deletePost(id: 2) {
3     id
4     title
5     content
6   }
7 }
```

. The 'Response' pane on the right shows the JSON response:

```
{
  "data": {
    "deletePost": {
      "id": "2",
      "title": "Jaymer's journey to heaven",
      "content": "Did he make it?"
    }
  }
}
```

. The status bar at the top right indicates a 200 status code, 40.0ms execution time, and 100B response size.

UPDATED GET ALL POSTS

The screenshot shows a GraphQL IDE with a dark theme. The top bar has four tabs labeled 'Unnamed'. The 'Operation' pane on the left contains a GraphQL query:

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

. The 'Response' pane on the right shows the JSON response:

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

. The status bar at the top right indicates a 200 status code, 9.00ms execution time, and 22B response size.

What do database migrations do and why are they useful?

I believe that database migrations are a way to keep track of changes in the database as time passes. Instead of just manually editing tables, migrations can help us apply, change, or rollback changes in a much more structured and version-controlled way. They're pretty useful because they prevent data loss when updating the database, it also allows teams to collaborate and make sure that they work with the same database.

How does GraphQL differ from REST for CRUD operations?

The rest API's are the get/users, post/users, put and delete. The problem with rest is that it over-fetches or under-fetches data. Like for example if i only need the user's name, the API returns name, email, password hash and all that information that i don't even need. But with graphql it solves it by only using one endpoint instead of multiple routes we just get what we need.