

## Post-service

### Create:

The screenshot displays the GraphQL Studio interface for a 'Create' operation. 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 }
```

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

```
{  
  "data": {  
    "createPost": {  
      "id": 4,  
      "title": "testing",  
      "content": "testing"  
    }  
  }  
}
```

Below the query editor, the 'Variables' tab is active, showing the input variables:

```
1 {  
2   "title": "testing",  
3   "content": "testing"  
}
```

The status bar at the top right indicates a 200 status code, 119ms execution time, and 71B response size. A '+ Add files' button is located at the bottom left.

### Read:

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

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

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

```
{  
  "data": {  
    "posts": [  
      {  
        "id": 3,  
        "content": "Testing",  
        "title": "Testing"  
      },  
      {  
        "id": 4,  
        "content": "testing",  
        "title": "testing"  
      }  
    ]  
  }  
}
```

Below the query editor, the 'Variables' tab is active, showing the input variables:

```
1 {  
2   "title": "testing",  
3   "content": "testing"  
}
```

The status bar at the top right indicates a 200 status code, 14.0ms execution time, and 115B response size. A '+ Add files' button is located at the bottom left.

## Update:

Operation

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

Run

Response

```
{
  "data": {
    "updatePost": {
      "id": 4,
      "title": "test successful",
      "content": "test successful"
    }
  }
}
```

200 | 116ms | 87B

Variables

Headers

Pre-Operation Script

Post-Operation Script

```
1 {
2   "title": "test successful",
3   "content": "test successful",
4   "updatePostId": 4
5 }
```

JSON

+ Add files

Operation

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

Run

Response

```
{
  "data": {
    "posts": [
      {
        "content": "Testing",
        "id": 3,
        "title": "Testing"
      },
      {
        "content": "test successful",
        "id": 4,
        "title": "test successful"
      }
    ]
  }
}
```

200 | 9.00ms | 131B

Variables

Headers

Pre-Operation Script

Post-Operation Script

```
1 {
2   "title": "test successful",
3   "content": "test successful",
4   "updatePostId": 4
5 }
```

JSON

+ Add files

## Delete:

Operation

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

Run

Response

```
{
  "data": {
    "deletePost": {
      "id": 4
    }
  }
}
```

Variables

```
1 {
2   "deletePostId": 4,
3 }
```

Headers

Pre-Operation Script

Post-Operation Script

JSON

+ Add files

Operation

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

Run

Response

```
{
  "data": {
    "posts": [
      {
        "content": "Testing",
        "id": 3,
        "title": "Testing"
      }
    ]
  }
}
```

Variables

```
1 {
2   "deletePostId": 4,
3 }
```

Headers

Pre-Operation Script

Post-Operation Script

JSON

# Users-Table

## Create:

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

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

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

```
{
  "data": {
    "createUser": {
      "id": 3,
      "name": "Dusky",
      "email": "Dusky@gmail.com"
    }
  }
}
```

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

```
1 {
2   "name": "Dusky",
3   "email": "Dusky@gmail.com"
}
```

The status bar at the top right indicates a 200 status code, 141ms execution time, and 74B response size.

## Read:

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

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

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

```
{
  "data": {
    "users": [
      {
        "id": 2,
        "name": "Dusky",
        "email": "Dusky@gmail.com"
      },
      {
        "id": 3,
        "name": "Dusky",
        "email": "Dusky@gmail.com"
      }
    ]
  }
}
```

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

```
1 {
2   "name": "Dusky",
3   "email": "Dusky@gmail.com"
}
```

The status bar at the top right indicates a 200 status code, 13.0ms execution time, and 121B response size.

## Update:

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 }
```

Response

```
{
  "data": {
    "updateUser": {
      "id": 3,
      "name": "Dusky Updated",
      "email": "DuskyUpdated@gmail.com"
    }
  }
}
```

Variables Headers Pre-Operation Script Post-Operation Script

```
1 {
2   "name": "Dusky Updated",
3   "email": "DuskyUpdated@gmail.com",
4   "updateUserId": 3
5 }
```

JSON

Operation

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

Response

```
{
  "data": {
    "users": [
      {
        "id": 2,
        "name": "Dusky",
        "email": "Dusky@gmail.com"
      },
      {
        "id": 3,
        "name": "Dusky Updated",
        "email": "DuskyUpdated@gmail.com"
      }
    ]
  }
}
```

Variables Headers Pre-Operation Script Post-Operation Script

```
1 {
2   "name": "Dusky Updated",
3   "email": "DuskyUpdated@gmail.com",
4   "updateUserId": 3
5 }
```

JSON

Add files

## Delete:

Operation

```
1 mutation($deleteUserId: Int!) {
2   deleteUser(id: $deleteUserId) {
3     id
4   }
5 }
```

Run

Response

200 | 138ms | 33B

```
{
  "data": {
    "deleteUser": {
      "id": 3
    }
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

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

JSON

+ Add files

Operation

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

Run

Response

200 | 10.0ms | 71B

```
{
  "data": {
    "users": [
      {
        "id": 2,
        "email": "Dusky@gmail.com",
        "name": "Dusky"
      }
    ]
  }
}
```

Variables

Headers

Pre-Operation Script

Post-Operation Script

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

JSON

+ Add files

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

Based from my understanding, database migrations is basically described as a “snapshot” in which it captures the state of what your database is in that time. Each migration then builds on the previous whenever there should be any changes or additions to the database. Migrations are useful because it keeps track of the historical versions of each database. This also helps in case you want to go back to a previous version of the database.

- How does GraphQL differ from REST for CRUD operations?

GraphQL offers a single endpoint that allows clients to request exactly the data they need, reducing over-fetching or under-fetching. In contrast, REST typically involves multiple endpoints with fixed data responses, which can lead to inefficiencies. GraphQL’s type system also improves client-server communication by ensuring that both parties agree on the data structure.