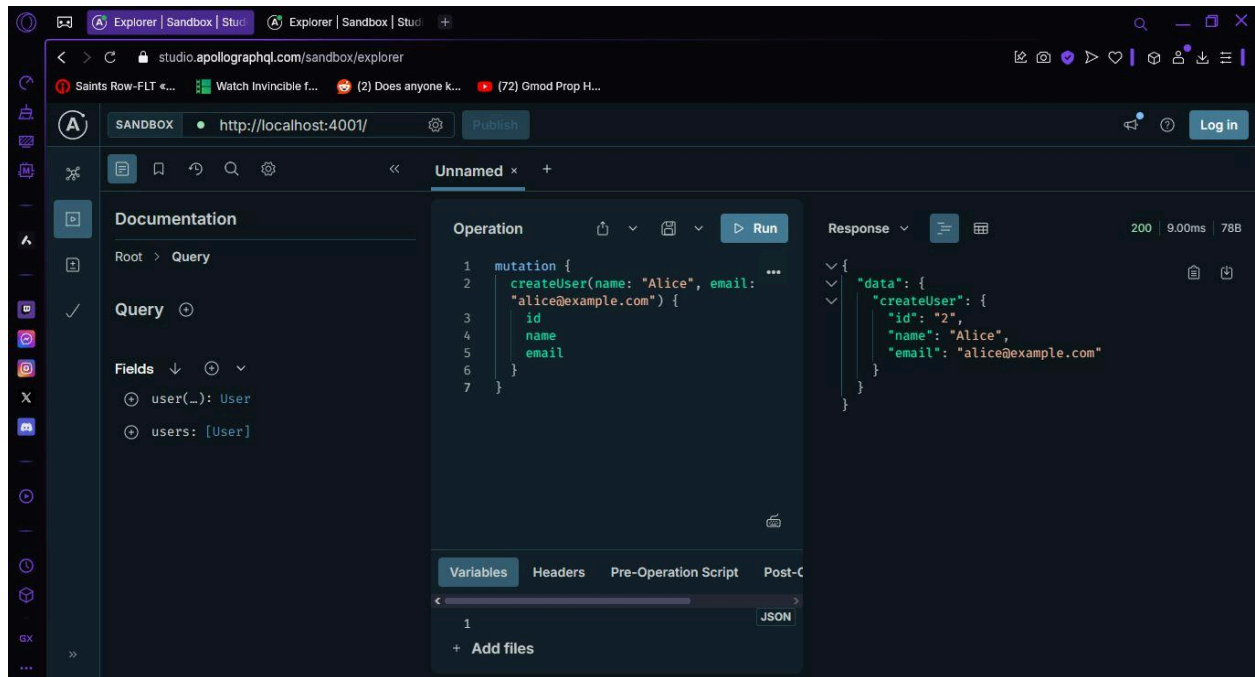
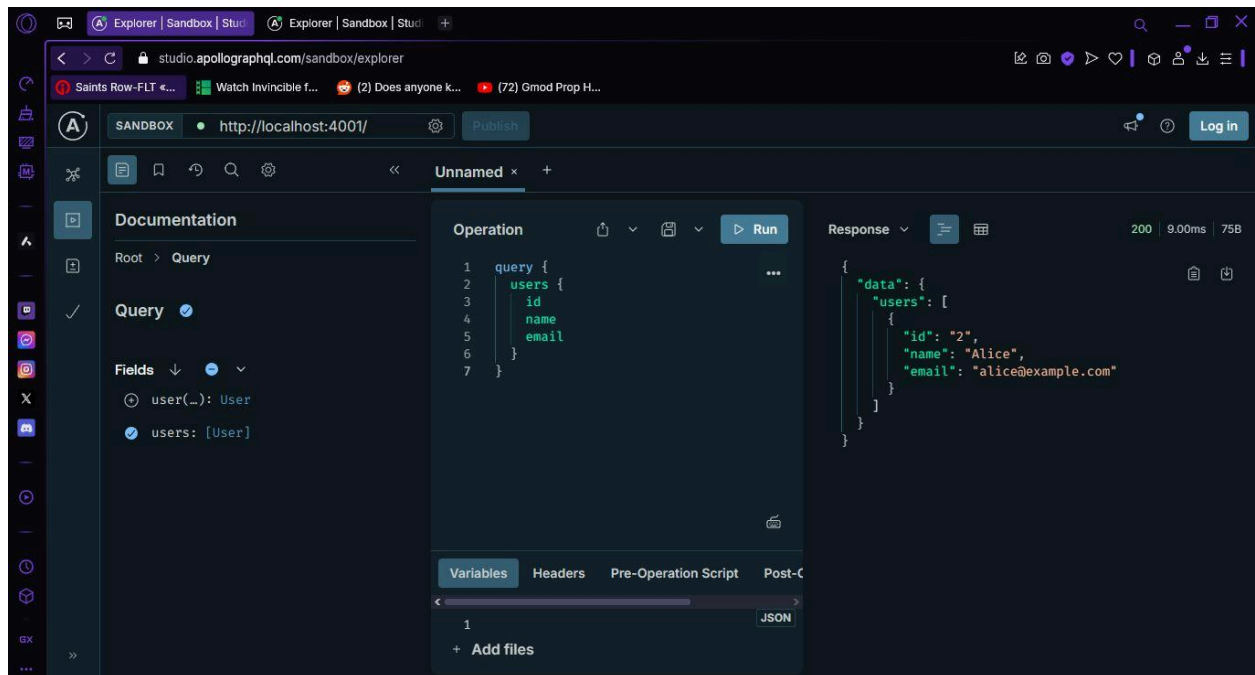


Users Service - Create, Read, Update, and Delete Records Verification:

Create Record:



Read Record:



Update Record:

The screenshot displays the Apollo Studio Explorer interface, which is used for managing GraphQL queries and mutations. The interface is divided into several panels:

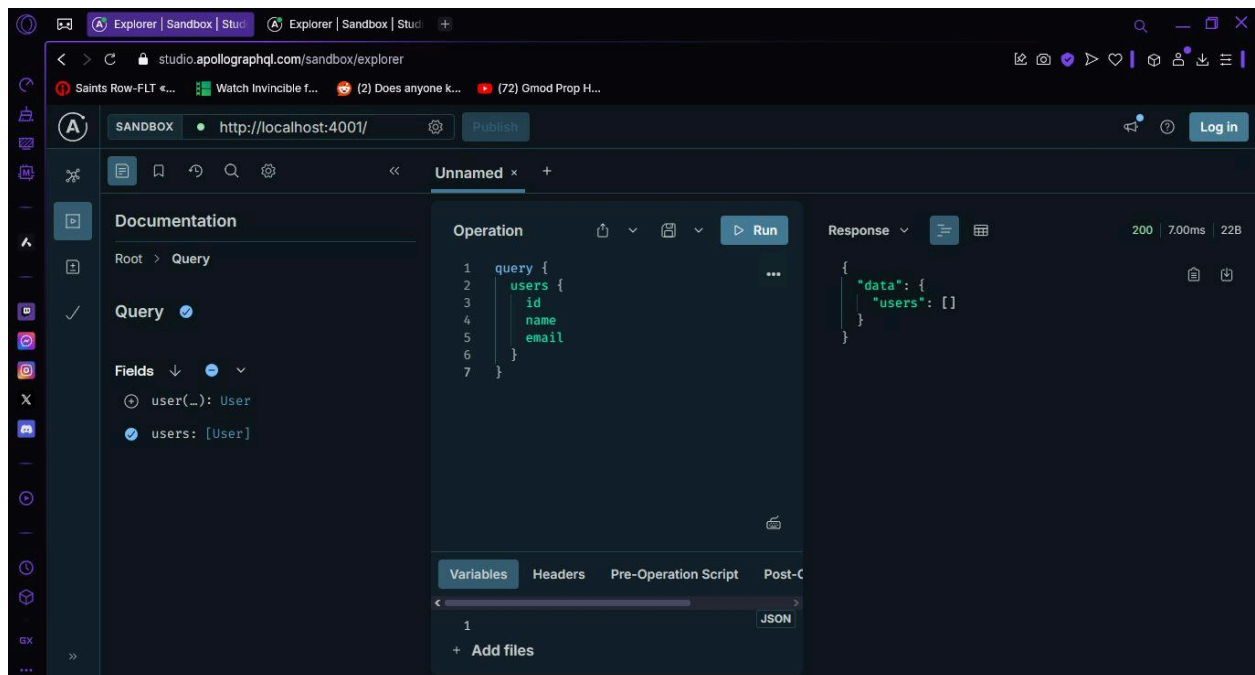
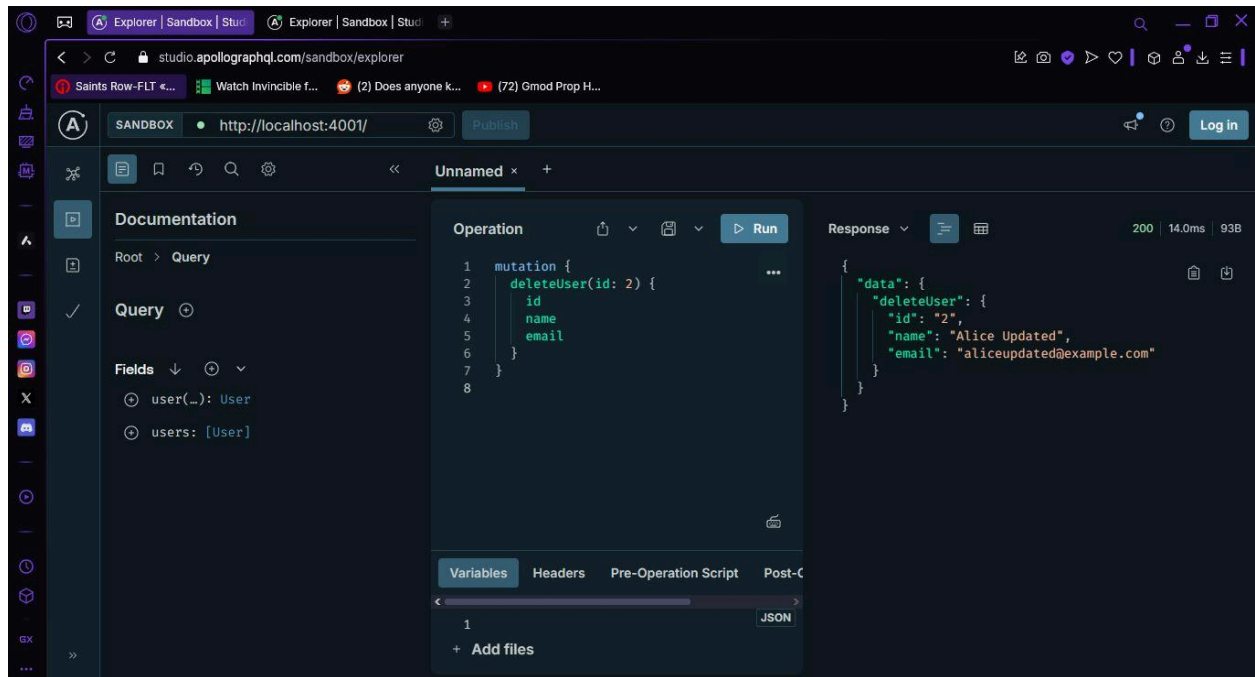
- Left Panel:** Contains the 'Documentation' section with a 'Query' dropdown and a 'Fields' section listing available fields: `user(...): User` and `users: [User]`.
- Top Panel:** Shows the 'Sandbox' environment with the URL `http://localhost:4001/` and a 'Publish' button.
- Operation Panel:** Displays the GraphQL mutation being executed. The mutation is as follows:

```
1 mutation {  
2   updateUser(id: 2, name: "Alice  
   Updated", email:  
   "aliceupdated@example.com") {  
3     id  
4     name  
5     email  
6   }  
7 }  
8
```
- Response Panel:** Shows the JSON response from the server, indicating a successful update:

```
{  
  "data": {  
    "updateUser": {  
      "id": "2",  
      "name": "Alice Updated",  
      "email": "aliceupdated@example.com"  
    }  
  }  
}
```
- Bottom Panel:** Includes tabs for 'Variables', 'Headers', 'Pre-Operation Script', and 'Post-Operation Script'. The 'Variables' tab is currently selected, showing a single variable `1` with a value of `JSON`.

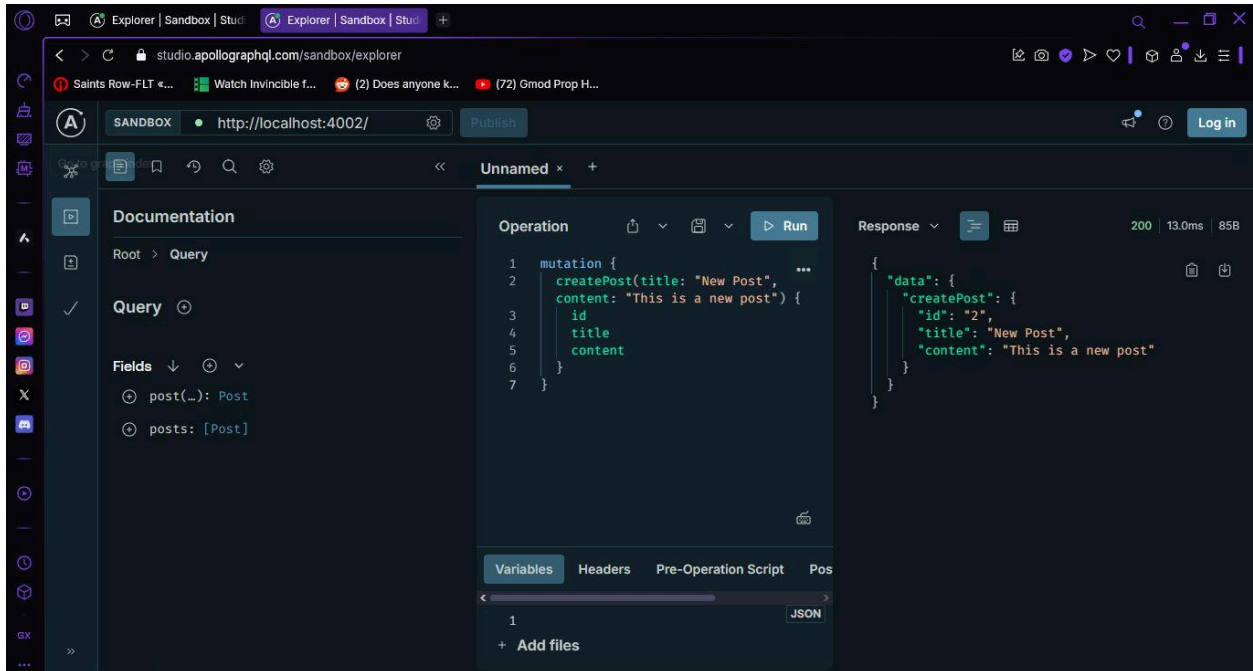
The status bar at the top right indicates a successful response with a status code of 200, a response time of 11.0ms, and a response size of 93B.

Delete Record:

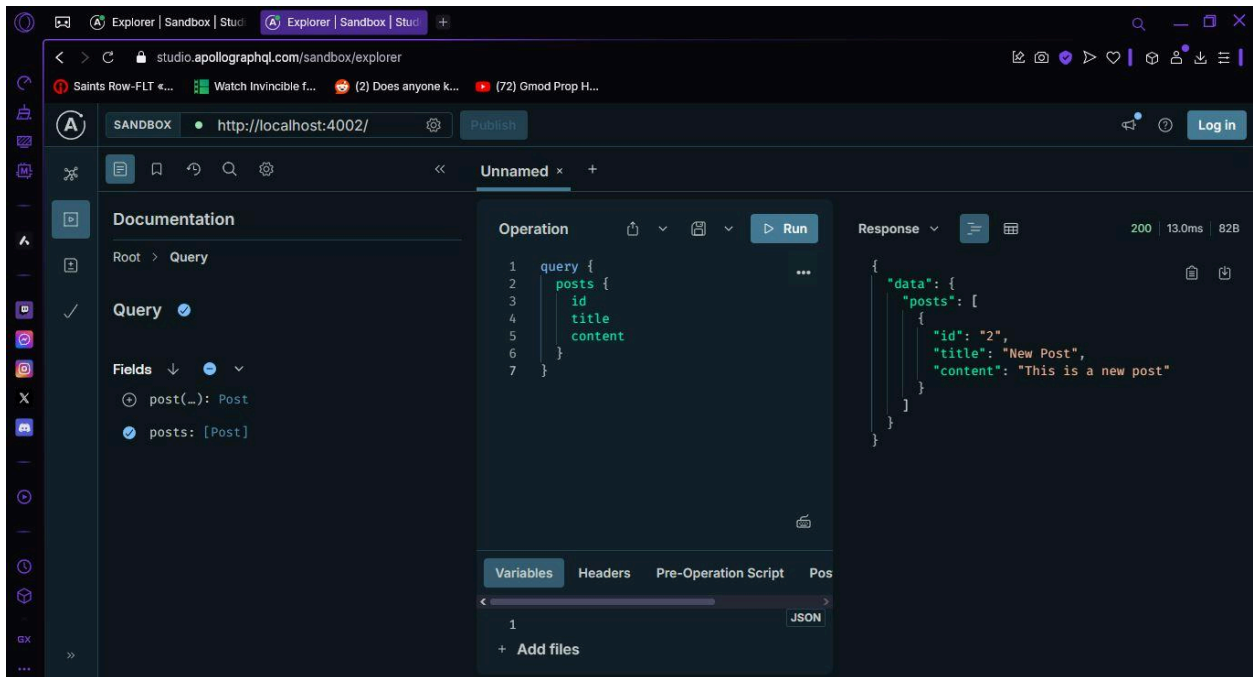


Post Service - Create, Read, Update, and Delete Records Verification:

Create Record:



Read Record:



Update Record:

The screenshot displays the Apollo Studio Explorer interface, which is used for testing and debugging GraphQL queries and mutations. The interface is divided into several panels:

- Left Panel:** Contains the "Documentation" section, showing the "Root > Query" path. Below this, the "Query" section is visible, listing fields: `post(...): Post` and `posts: [Post]`.
- Operation Panel:** Displays the GraphQL mutation being executed. The code is as follows:

```
1 mutation {
2   updatePost(id: 2, title: "Updated
   Post", content: "This is an
   updated post") {
3     id
4     title
5     content
6   }
7 }
```
- Response Panel:** Shows the JSON response from the server. The status is 200, with a response time of 10.0ms and a size of 94B. The response structure is:

```
{
  "data": {
    "updatePost": {
      "id": "2",
      "title": "Updated Post",
      "content": "This is an updated post"
    }
  }
}
```
- Bottom Panel:** Includes tabs for "Variables", "Headers", "Pre-Operation Script", and "Post-Operation Script". The "Variables" tab is active, showing a single variable `1` with a value of `JSON`. There is also an option to "Add files".

Delete Record:

This screenshot shows the Apollo Studio interface with a successful DELETE mutation. The left sidebar displays the 'Query' section with fields for 'post(...): Post' and 'posts: [Post]'. The main 'Operation' pane contains the following GraphQL code:

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

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

```
{  
  "data": {  
    "deletePost": {  
      "id": "2",  
      "title": "Updated Post",  
      "content": "This is an updated post"  
    }  
  }  
}
```

Below the operation code, the 'Variables' tab is active, showing a single variable '1' with a 'JSON' type.

This screenshot shows the Apollo Studio interface with a successful GET query. The left sidebar displays the 'Query' section with fields for 'post(...): Post' and 'posts: [Post]'. The main 'Operation' pane contains the following GraphQL code:

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

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

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

Below the operation code, the 'Variables' tab is active, showing a single variable '1' with a 'JSON' type.

Short Reflection (3-5 sentences):

- What do database migrations do and why are they useful?
Database migrations are used to make changes to a database, like adding or changing tables. They help keep the database organized and up to date, and make sure the changes work smoothly across different environments. It's really useful because it helps avoid errors and keeps everything consistent.
- How does GraphQL differ from REST for CRUD operations?
After doing a bit of research, GraphQL is different from REST because it lets you get exactly the data you need with just one request. With REST, you often need multiple endpoints for different actions, but GraphQL gives you more control and makes things more efficient. It reduces the chances of getting too much or too little data, which makes everything faster and more flexible.