

GraphQL – GERN Stack

Tip: Creating a sample nodeJS Project Using VSCode

Tip: Querying the data

Tip: Exposing the GUI for graphiql

Tip: Fetching MongoDB Data with graphql

Tip: Project MgmtApp – Brad Traversity

Tip: GUI for MongoDB

Tip: Adding a record to MongoDB using GraphQL

Tip: Getting History from graphiql

Tip: Adding bootstrap using the CDN for our client project

Tip: Using Fragment shorthand syntax

Tip: Wiring up Apollo Server to fetch graphql data

Tip: Building a re-usable component for rows in a table

Tip: Using react-icons

Tip: Creating a Spinner with built-in react spinner

Tip: Deleting and re-fetching data with graphiql

Tip: Adding data with graphql with a MODAL dialog

Tip: Creating a not Found page

Tip: Grabbing the network response data with developer tools to view graphql data

Tip: CSS syntax and FontAwsome react

Tip: Working with Modals

Tip: 400 Bad Request with graphical

Extensions to add to VSCode for GraphQL

<https://marketplace.visualstudio.com/items?itemName=GraphQL.vscode-graphql>

Youtube Link

<https://youtu.be/Dr2dDWzThK8>

REST vs GraphQL (In terms of how it handles routes/endpoints)

In traditional REST, your endpoints would look like:

/users
/travels
/books
etc...

In Graphql you only have “**one**” endpoint

/graphql

Sample Data

To get some fake data to work with, you can go to a website called

<https://www.mockaroo.com/>

The screenshot shows the Mockaroo website interface. At the top, there's a navigation bar with links: SCHEMAS, DATASETS, MOCK APIS, SCENARIOS, and PROJECTS. Below the navigation bar, there's a dark grey section with a green header containing the Mockaroo logo and the text: "Looking to generate fake data based on your production data? Mimic your databases with a trial account from".

Below this, there's a dark grey box with white text: "Need some mock data to test your app? Mockaroo lets you generate up to 1,000 rows of realistic test data in CSV, JSON, SQL, and Excel formats. Need more data? Plans start at just \$60/year. Mockaroo is also available as a docker image that you can deploy in your own private cloud."

The main part of the interface is a table with three columns: Field Name, Type, and Options. The table contains six rows of fields:

Field Name	Type	Options
id	Row Number	blank: 0 % Σ ×
first_name	First Name	blank: 0 % Σ ×
last_name	Last Name	blank: 0 % Σ ×
email	Email Address	blank: 0 % Σ ×
gender	Gender	blank: 0 % Σ ×
ip_address	IP Address v4	blank: 0 % Σ ×

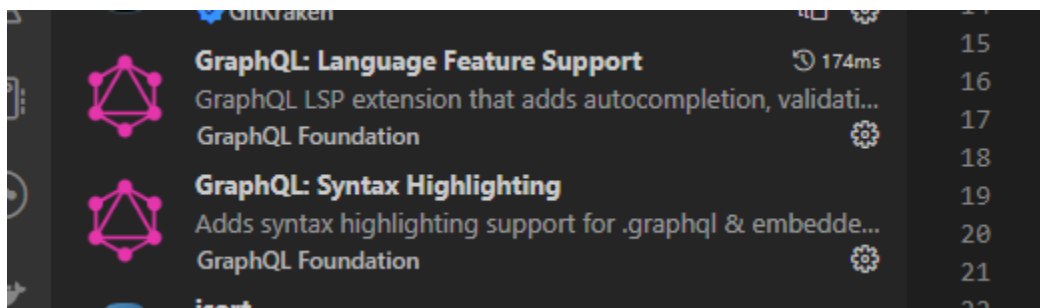
Below the table, there's a button labeled "ADD ANOTHER FIELD".

At the bottom, there's a section for configuration: "# Rows: 1000", "Format: JSON", and two checked checkboxes: "array" and "include null values". Below this, there's a hint: "Hint: Use '*' in column names to generate nested json objects, brackets to generate arrays. More information...".

At the very bottom, there's a dark green bar with a "Mock your back-end API and start" button, a "DOWNLOAD DATA" button, a "PREVIEW" button, a "SAVE THIS SCHEMA" button, and a "MORE" button with a dropdown arrow.

I changed the type to JSON

Add the VSCode Extensions to your VSCODE



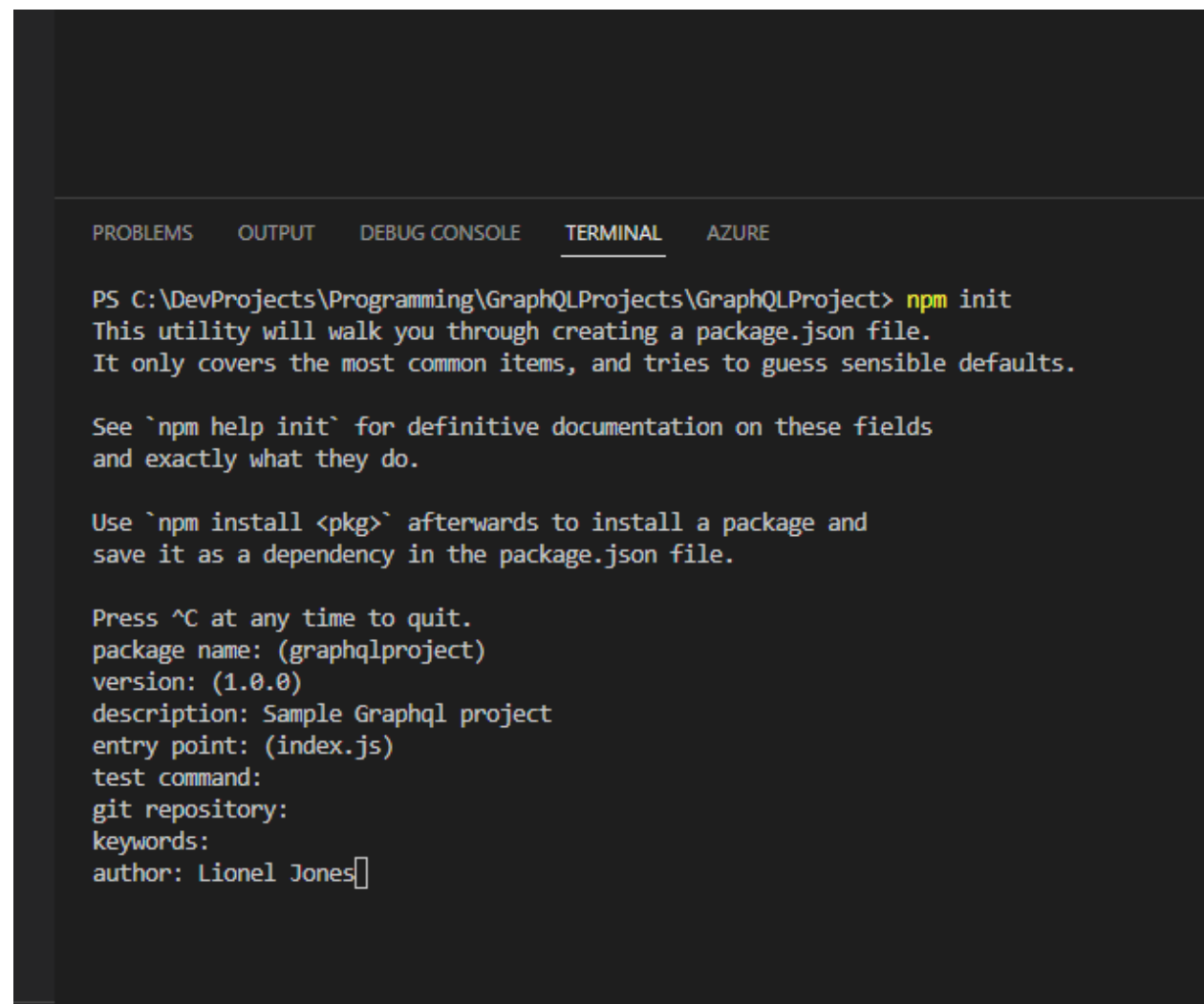
Tip: Creating a sample nodeJS Project Using VSCode

First I create a directory called GraphQLProject

I download the fake data and add it to my directory

Then to initialize the project, I open a new terminal, type
npm init

I just follow the prompts, enter in generic information



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  AZURE

PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (graphqlproject)
version: (1.0.0)
description: Sample GraphQL project
entry point: (index.js)
test command:
git repository:
keywords:
author: Lionel Jones
```

This will create my package.json file for my dependencies

```
package.json X
package.json > ...
1  {
2    "name": "graphqlproject",
3    "version": "1.0.0",
4    "description": "Sample Graphql project",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Lionel Jones",
10   "license": "ISC"
11 }
12
```

Next, type the following command to install expressjs
npm install express

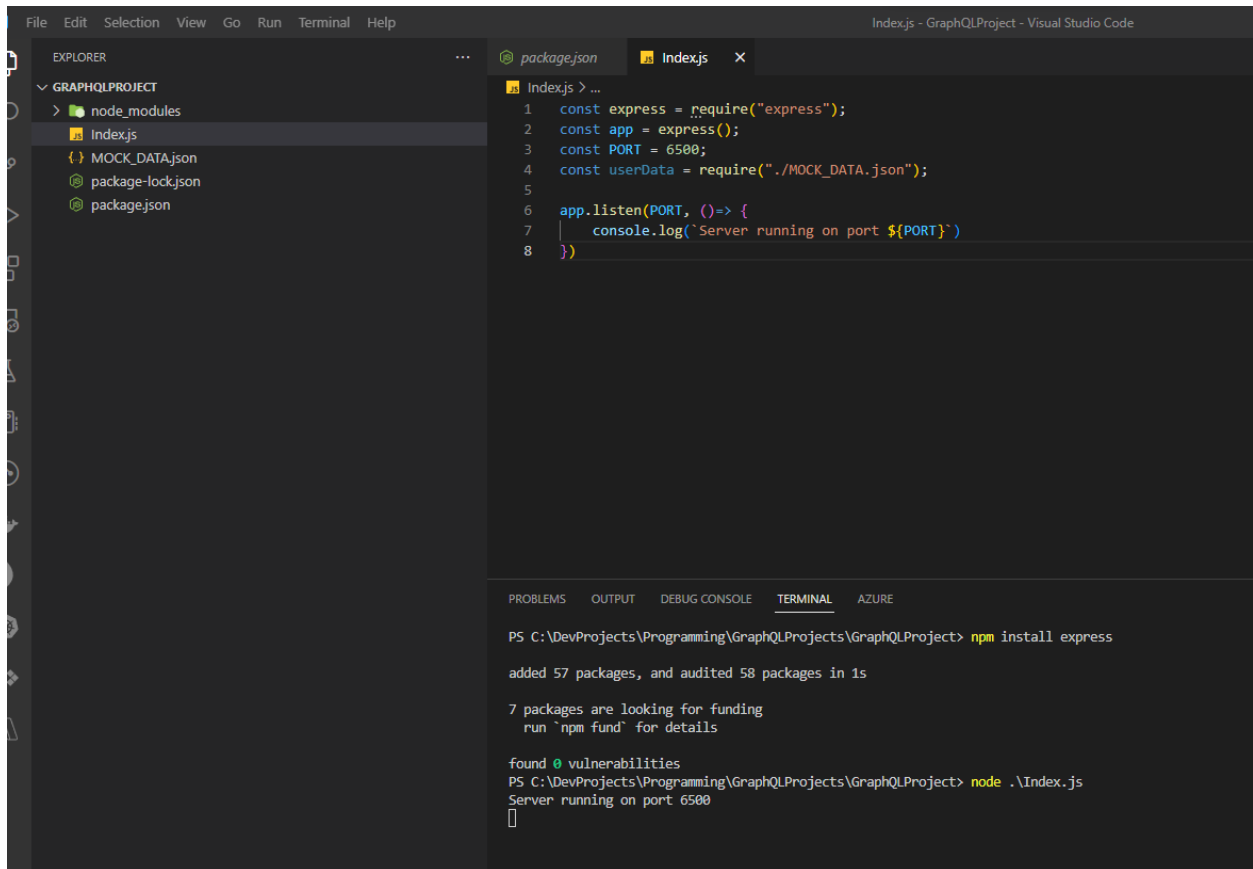
```
File Edit Selection View Go Run Terminal Help
package.json - GraphQLProject - Visual Studio Code

EXPLORER
GRAPHQLPROJECT
  node_modules
  index.js
  MOCK_DATA.json
  package-lock.json
  package.json

package.json X index.js
package.json > ...
1  {
2    "name": "graphqlproject",
3    "version": "1.0.0",
4    "description": "Sample Graphql project",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Lionel Jones",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.18.2"
13   }
14 }
15

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> npm install express
added 57 packages, and audited 58 packages in 1s
7 packages are looking for funding
run 'npm fund' for details
found 0 vulnerabilities
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject>
```

Next I create an index.js file and add the following lines of code to it



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the project structure: GRAPHQLPROJECT, node_modules, Indexjs, MOCK_DATA.json, package-lock.json, and package.json. The Indexjs file is selected. The main editor shows the following code in index.js:

```
1 const express = require("express");
2 const app = express();
3 const PORT = 6500;
4 const userData = require("./MOCK_DATA.json");
5
6 app.listen(PORT, () => {
7   console.log(`Server running on port ${PORT}`)
8 })
```

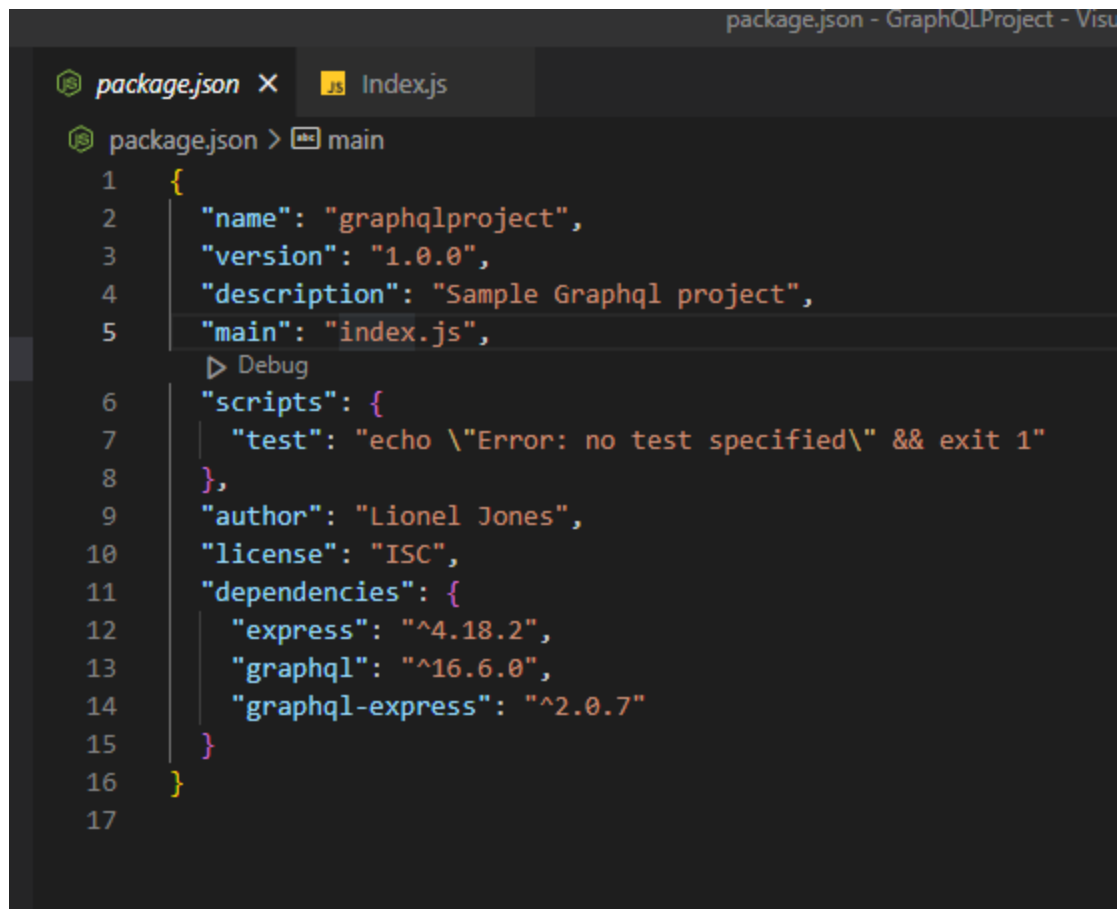
The TERMINAL panel at the bottom shows the following output:

```
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> npm install express
added 57 packages, and audited 58 packages in 1s
7 packages are looking for funding
run 'npm fund' for details
found 0 vulnerabilities
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> node .\Index.js
Server running on port 6500
```

If you go to browser
<http://localhost:6500/>
You will get a
Cannot GET /

This is because we are not creating a traditional REST api, but we still need to have some sort of service running to expose a port.

Next, install the next two packages at the terminal
npm i graphql
npm install --save express-graphql --force

A screenshot of a code editor window titled "package.json - GraphQLProject - Visual Studio Code". The editor shows the "package.json" file with the following content:

```
1  {
2    "name": "graphqlproject",
3    "version": "1.0.0",
4    "description": "Sample Graphql project",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "author": "Lionel Jones",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.18.2",
13     "graphql": "^16.6.0",
14     "graphql-express": "^2.0.7"
15   }
16 }
```

The editor interface includes a tab for "package.json" and another for "Index.js". The "package.json" tab is active, and the cursor is positioned at the end of line 16.

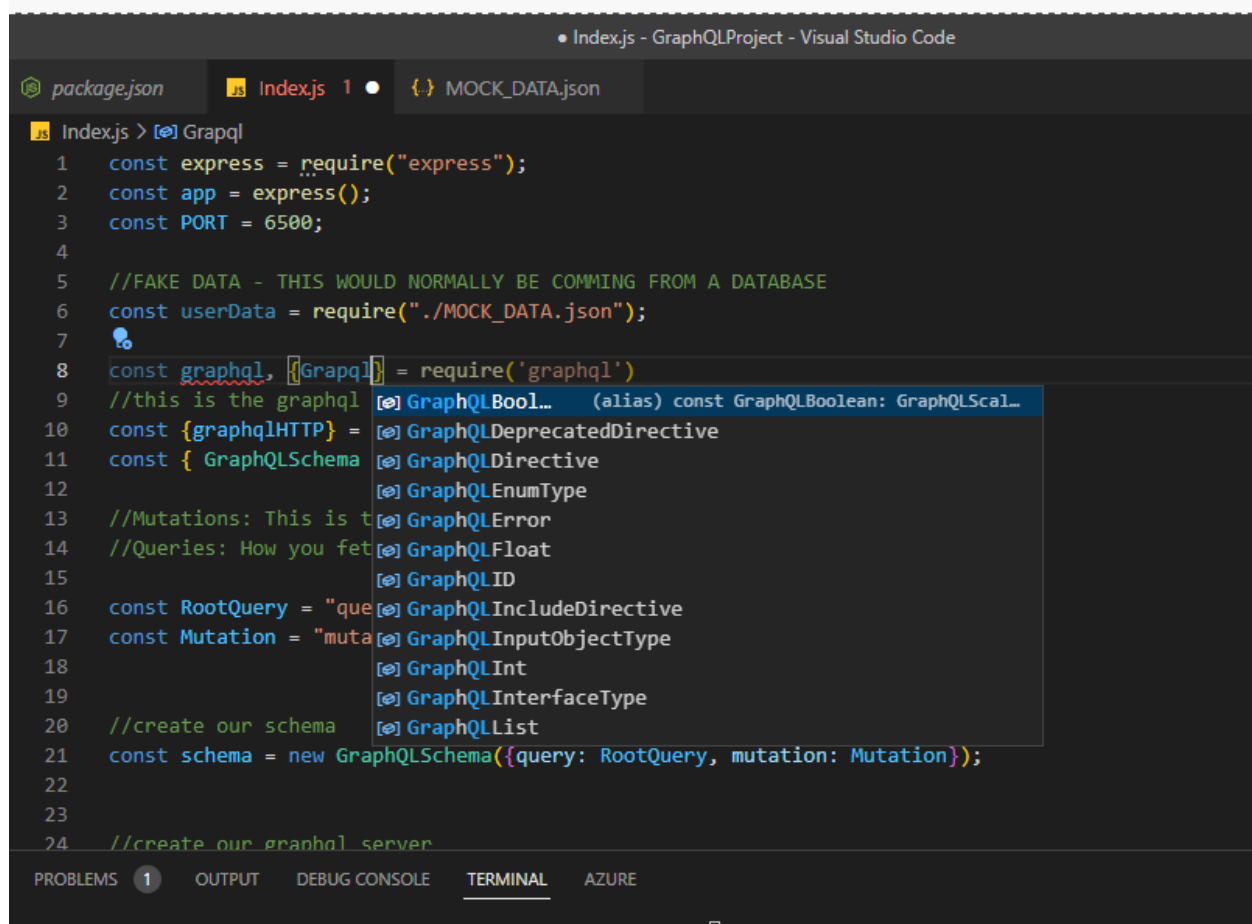
As you can see, as you install packages, it updates your package.json file

GraphQL Concepts

Mutations: This is the same as CRUD (Create, Read, Update, Delete)

Queries: How you fetch the data you need

Object types, before we create our schema, we need import the different object types

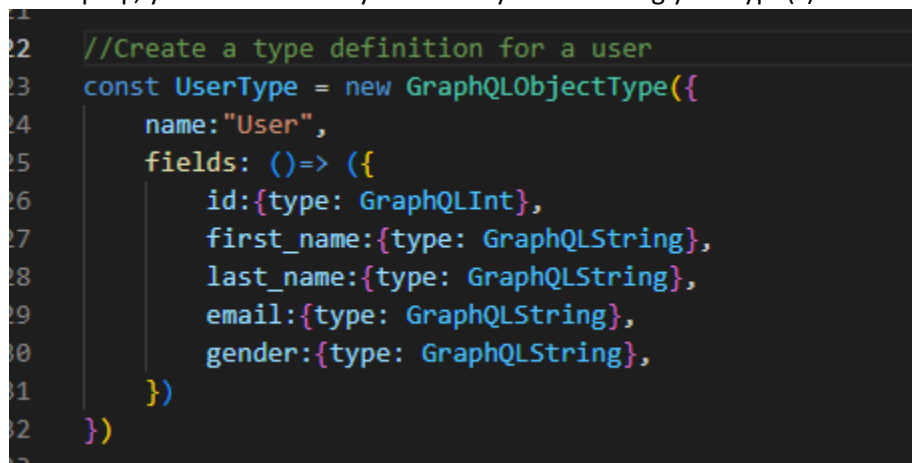


```
Index.js - GraphQLProject - Visual Studio Code
package.json  index.js 1  MOCK_DATA.json

Index.js > [Grapql]
1  const express = require("express");
2  const app = express();
3  const PORT = 6500;
4
5  //FAKE DATA - THIS WOULD NORMALLY BE COMING FROM A DATABASE
6  const userData = require("../MOCK_DATA.json");
7
8  const graphql, { GraphQL } = require('graphql')
9  //this is the graphql
10 const { graphqlHTTP } = require('graphql-http')
11 const { GraphQLSchema } = require('graphql')
12
13 //Mutations: This is t
14 //Queries: How you fet
15
16 const RootQuery = "que
17 const Mutation = "muta
18
19 //create our schema
20 const schema = new GraphQLSchema({query: RootQuery, mutation: Mutation});
21
22
23
24 //create our graphql server
```

Creating the type(s):

In GraphQL, you interact with your data by first defining your type(s)



```
1
22 //Create a type definition for a user
23 const UserType = new GraphQLObjectType({
24   name: "User",
25   fields: () => ({
26     id: {type: GraphQLInt},
27     first_name: {type: GraphQLString},
28     last_name: {type: GraphQLString},
29     email: {type: GraphQLString},
30     gender: {type: GraphQLString},
31   })
32 })
33
```

Creating the query(s)

As explained above, you create your mutations (queries as coded below)

```
//CREATE QUERY(S)
//Remember GraphQL only has "one" endpoint, below is where you create your queries
const RootQuery = new GraphQLObjectType( {
  name: "RootQueryType",
  fields: {
    //below is a query called getAllUsers
    getAllUsers: {
      type: new GraphQLList(UserType) , //list of users - the type is defined above for our user
      args: {id: {type: GraphQLInt}},

      //the resolve function is where you would make your database call, i:e SELECT * .. or with MongoDB db.findById (...)
      resolve(parent,args) {
        return userData //this is the MOCK_DATA.json data
      }
    }
    //If i wanted to create another query, it would be below here seperated by a , i:e getUserByID
  }
});
```

Creating the mutation

Also explained before, you create (CRUD OPERATIONS) via "Mutations"

```
//CREATE MUTATION
const Mutation = new GraphQLObjectType ( {
  name: "Mutation",
  fields: {
    createUser: {
      type: UserType,
      args: {
        first_name: {type: GraphQLString},
        last_name: {type: GraphQLString},
        email: {type: GraphQLString},
        gender: {type: GraphQLString},
      },
      resolve(parent,args) {
        //THIS IS WHERE YOU PUT YOUR MUTATION LOGIC, INSERT,DELETE,UPDATE ...
        userData.push({id:userData.length + 1,
          first_name: args.first_name,
          last_name: args.last_name,
          email: args.email,
          gender: args.gender
        })
        return args
      } //resolve
    }
  }
});
```

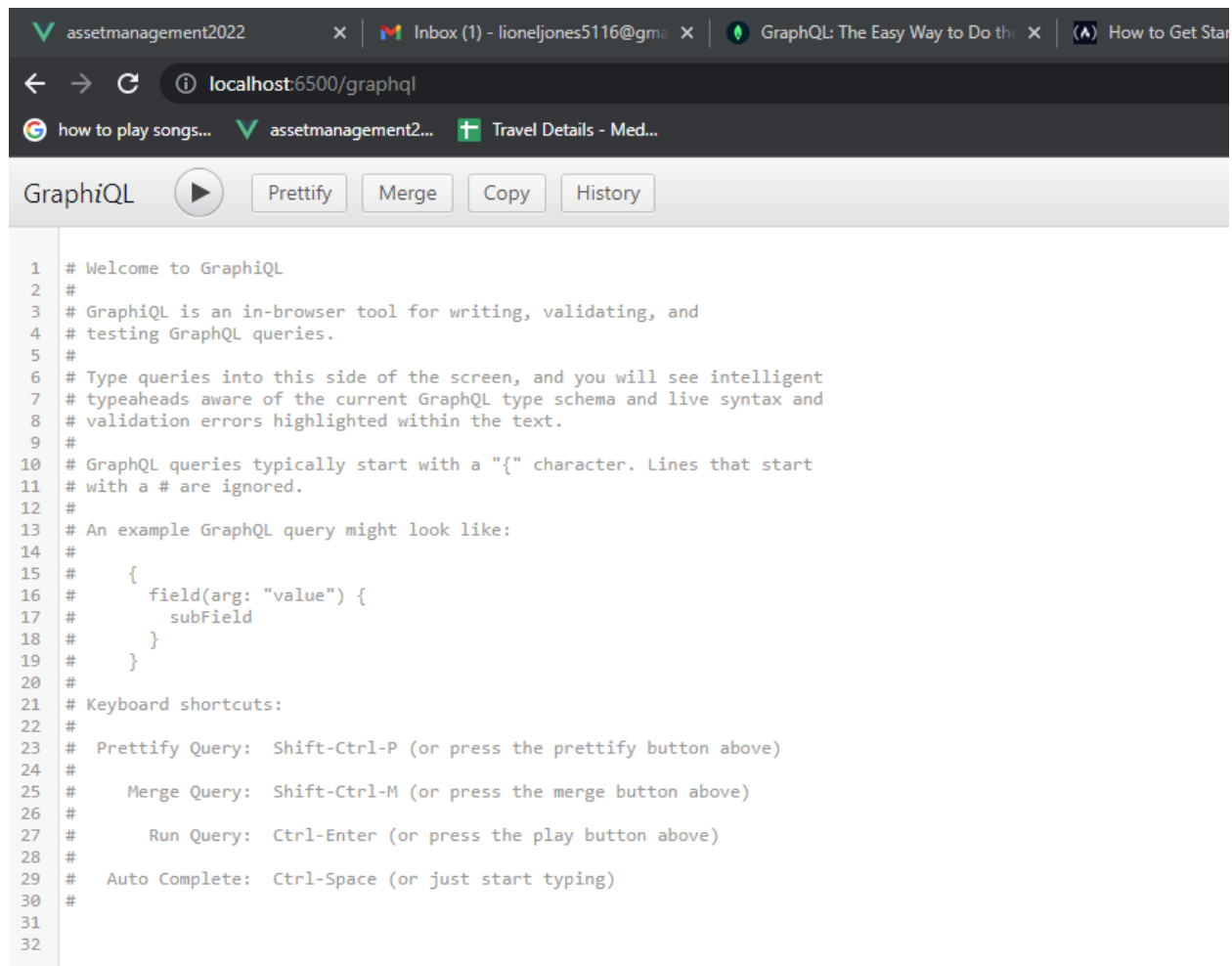

Next, let's fire up our graphql server
Enter the command at the terminal

node index.js

Then in the browser type:

Tip: Exposing the GUI for graphiql
<http://localhost:6500/graphql>

When you do this:



You get a graphical UI to view your results of your query against

This is surfaced up via this entry:

```

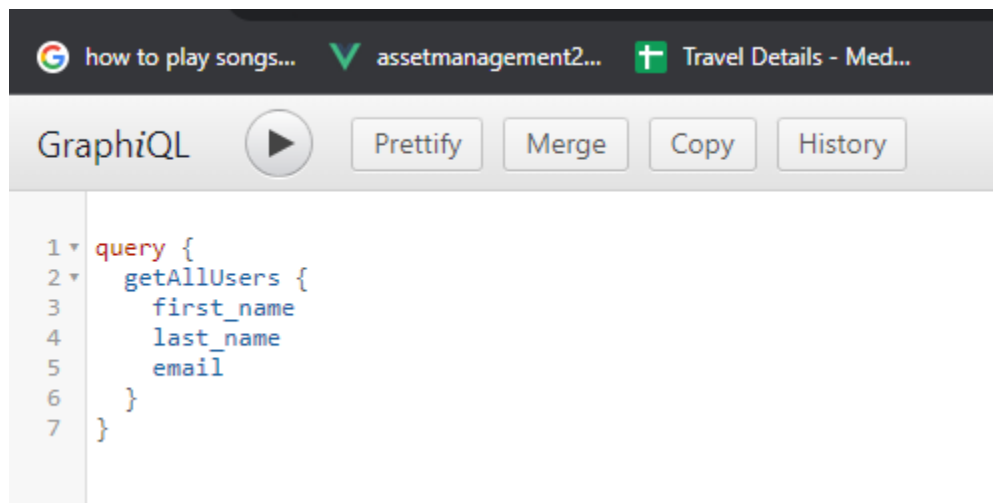
23
24 //create our graphql server
25 //Remember GraphQL only has "one" endpoint, below is where you create your queries
26 app.use('/graphql',graphqlHTTP( {
27   schema,
28   graphiql:true
29 }));
30
31

```

This is like using a API test like POSTMAN or any other API tester

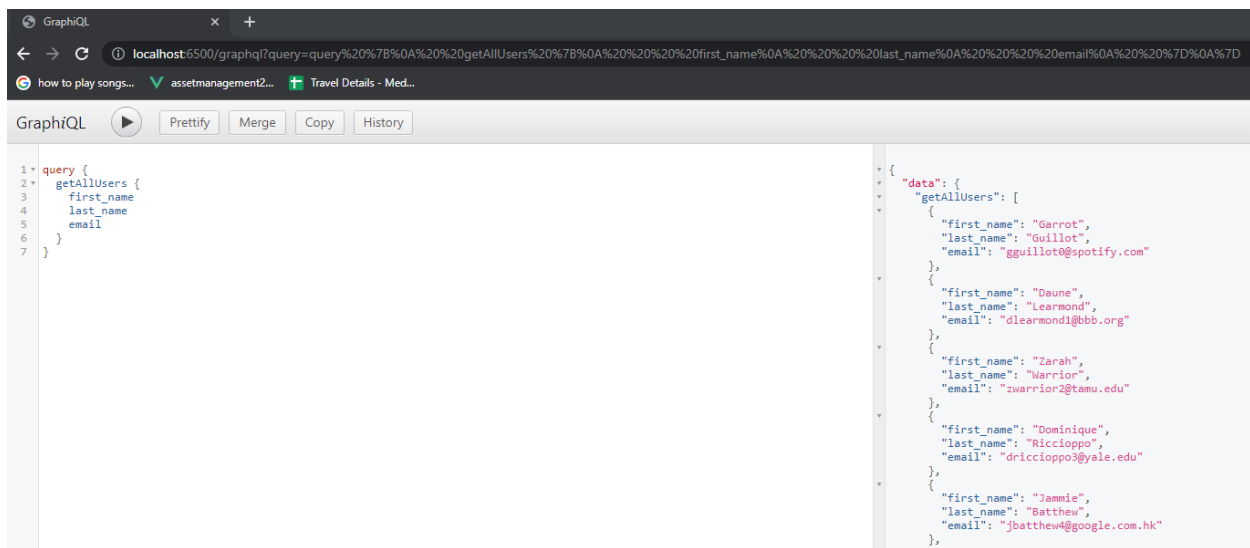
Tip: Querying the data

To fetch some data, just type below



As you type your information inside of the query, it will autocomplete the query you defined in your code.

Then just hit the run button and you will see the results in the right pane



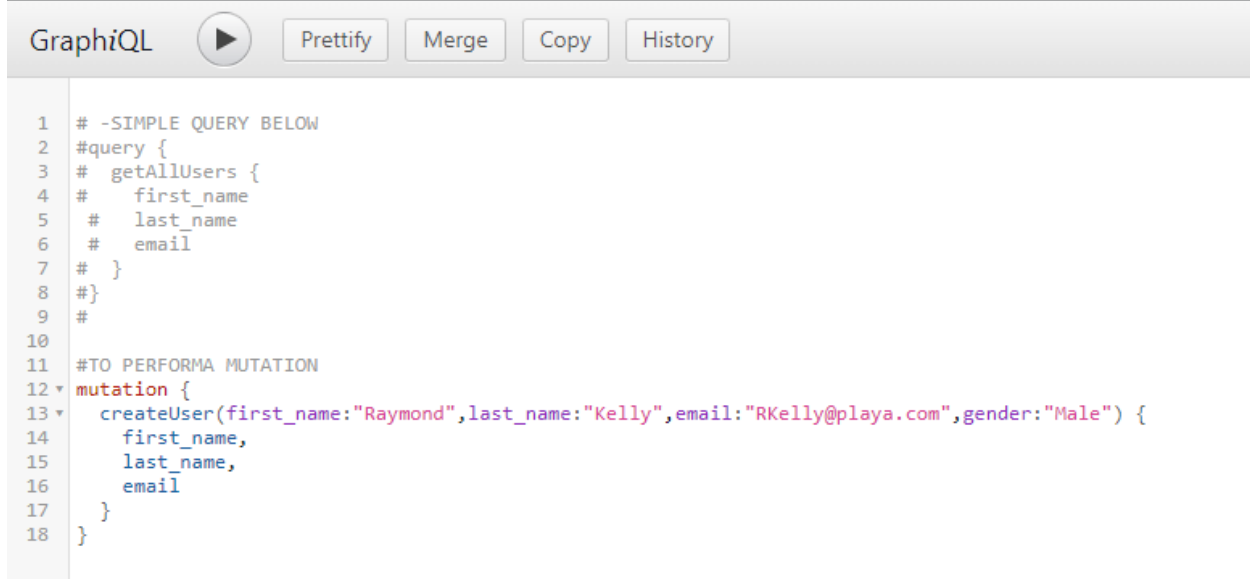
The screenshot shows the GraphQL IDE interface. The left pane contains a query:

```
1 query {
2   getAllUsers {
3     first_name
4     last_name
5     email
6   }
7 }
```

 The right pane shows the JSON response:

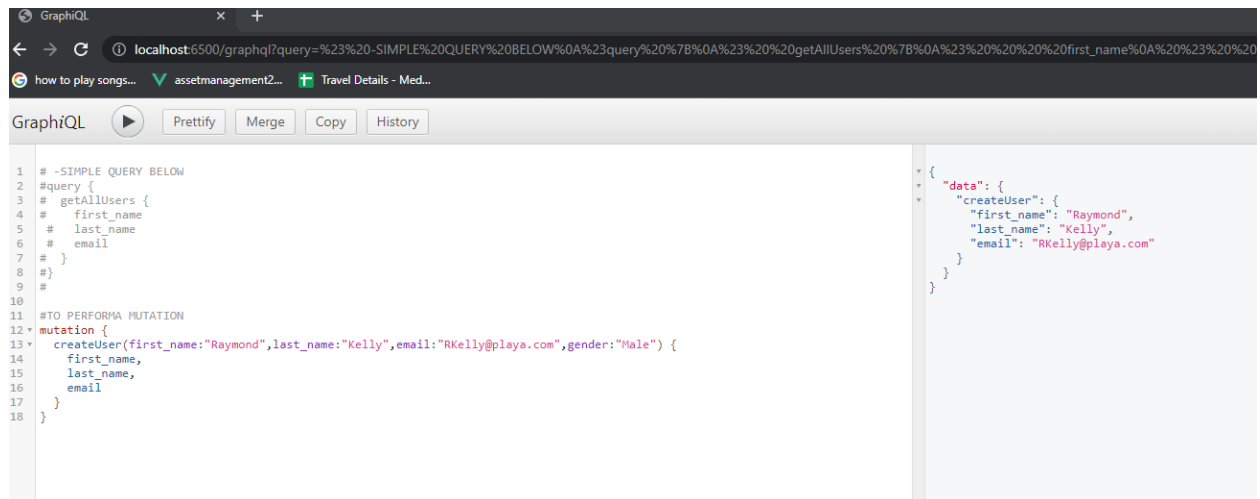
```
{
  "data": {
    "getAllUsers": [
      {
        "first_name": "Garrot",
        "last_name": "Guillot",
        "email": "gguillot@spotify.com"
      },
      {
        "first_name": "Daune",
        "last_name": "Learnmond",
        "email": "dlearnmond1@bbb.org"
      },
      {
        "first_name": "Zarah",
        "last_name": "Warrior",
        "email": "zwarrior2@tamu.edu"
      },
      {
        "first_name": "Dominique",
        "last_name": "Riccioppo",
        "email": "driccioppo3@yale.edu"
      },
      {
        "first_name": "Jammie",
        "last_name": "Batthew",
        "email": "jbatthew4@google.com.hk"
      }
    ]
  }
}
```

To perform a mutation:



The screenshot shows the GraphQL IDE interface with a mutation query in the editor:

```
1 # -SIMPLE QUERY BELOW
2 #query {
3 #   getAllUsers {
4 #     first_name
5 #     last_name
6 #     email
7 #   }
8 #}
9 #
10
11 #TO PERFORMA MUTATION
12 mutation {
13   createUser(first_name:"Raymond",last_name:"Kelly",email:"RKelly@playa.com",gender:"Male") {
14     first_name,
15     last_name,
16     email
17   }
18 }
```



The screenshot shows the GraphQL IDE interface. The left pane contains a query and a mutation. The right pane shows the JSON result of the query.

```
1 # -SIMPLE QUERY BELOW
2 #query {
3 #  getAllUsers {
4 #    first_name
5 #    last_name
6 #    email
7 #  }
8 #}
9 #
10
11 #TO PERFORMA MUTATION
12 #mutation {
13 #  createUser(first_name:"Raymond",last_name:"Kelly",email:"RKelly@playa.com",gender:"Male") {
14 #    first_name,
15 #    last_name,
16 #    email
17 #  }
18 #}
19
```

```
{
  "data": {
    "createUser": {
      "first_name": "Raymond",
      "last_name": "Kelly",
      "email": "RKelly@playa.com"
    }
  }
}
```

Run the query:

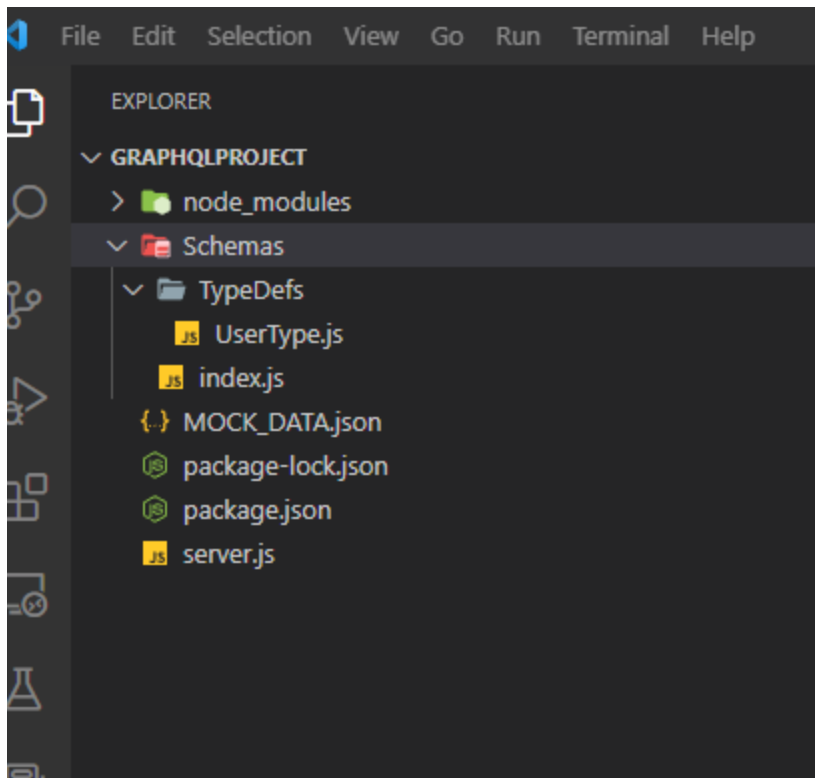
```
19
20 query {
21   getAllUsers {
22     id
23     first_name
24     last_name
25     email
26   }
27 }
28
29
```

And you will see the item added to the array:

```
{
  "id": 1001,
  "first_name": "Raymond",
  "last_name": "Kelly",
  "email": "RKelly@playa.com"
}
]
```

IMPORTANT: The user was added to an “in-memory” instance of the MOCK_DATA file, the actual .json file was not modified

To clean up the project structure:



Rename index.js on our root to "Server.js"

We place our types inside of the typedefs folder

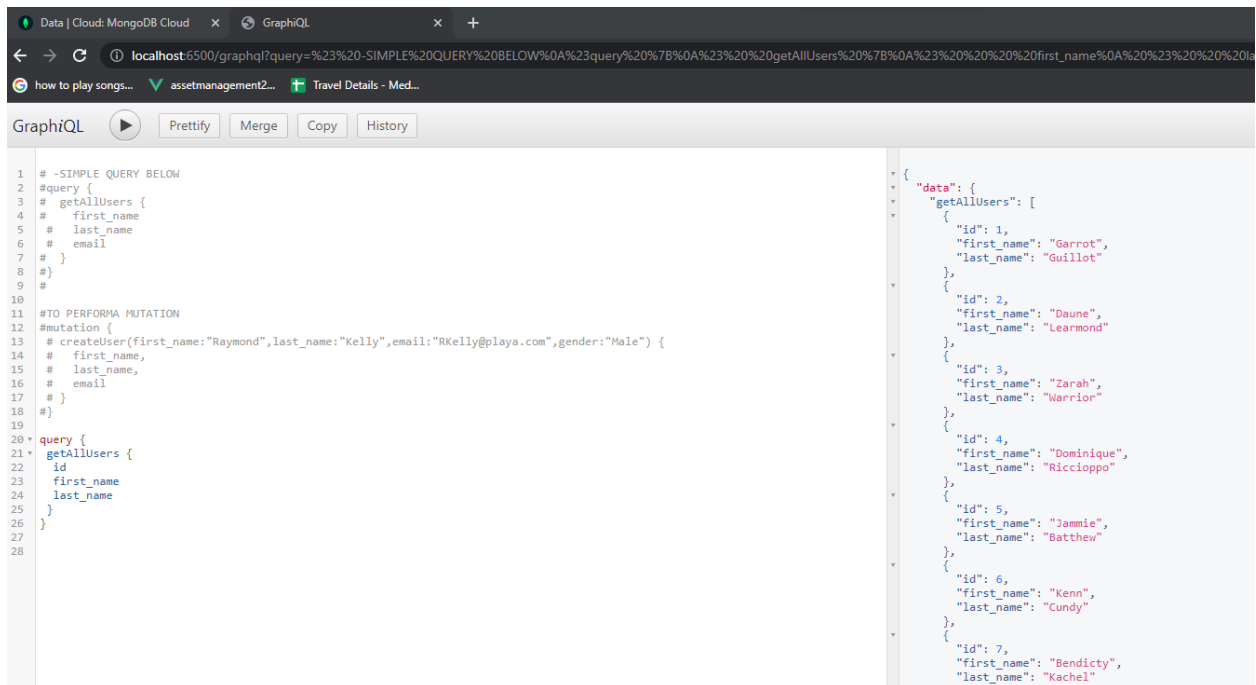
We place our mutations and queries in the index.js file under the schema folder

Then we execute

node server.js

```
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> node .\server.js
Server running on port 6500
█
```

And everything works:



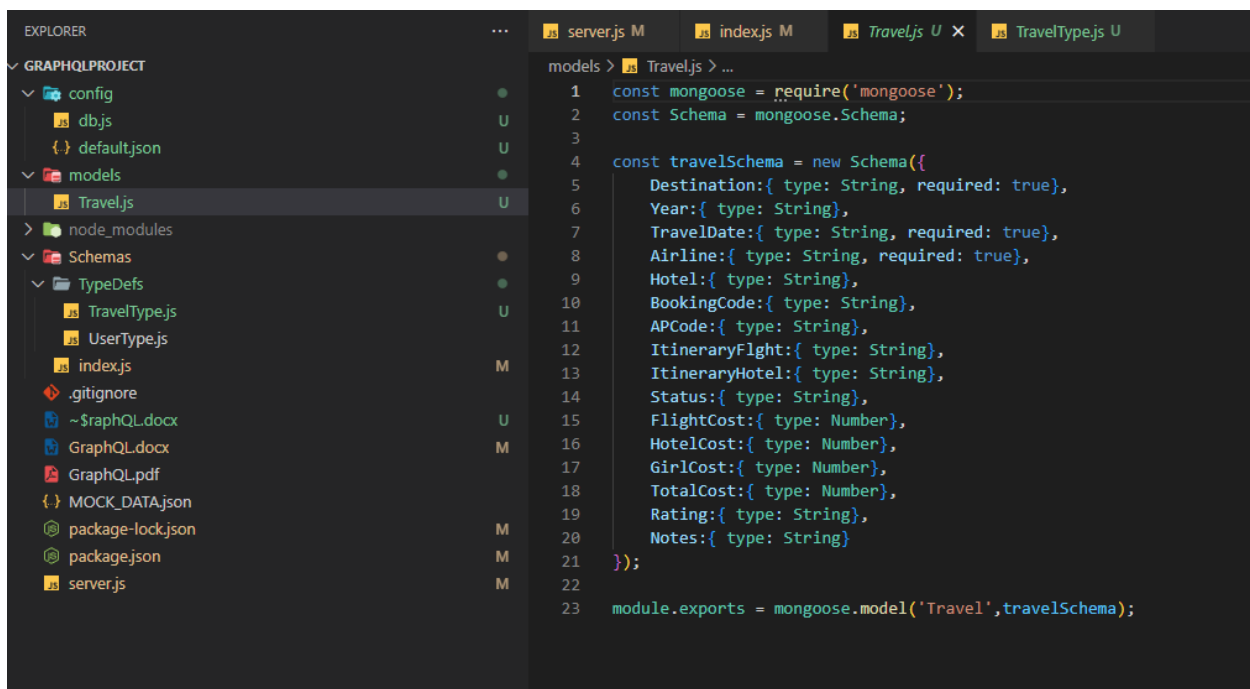
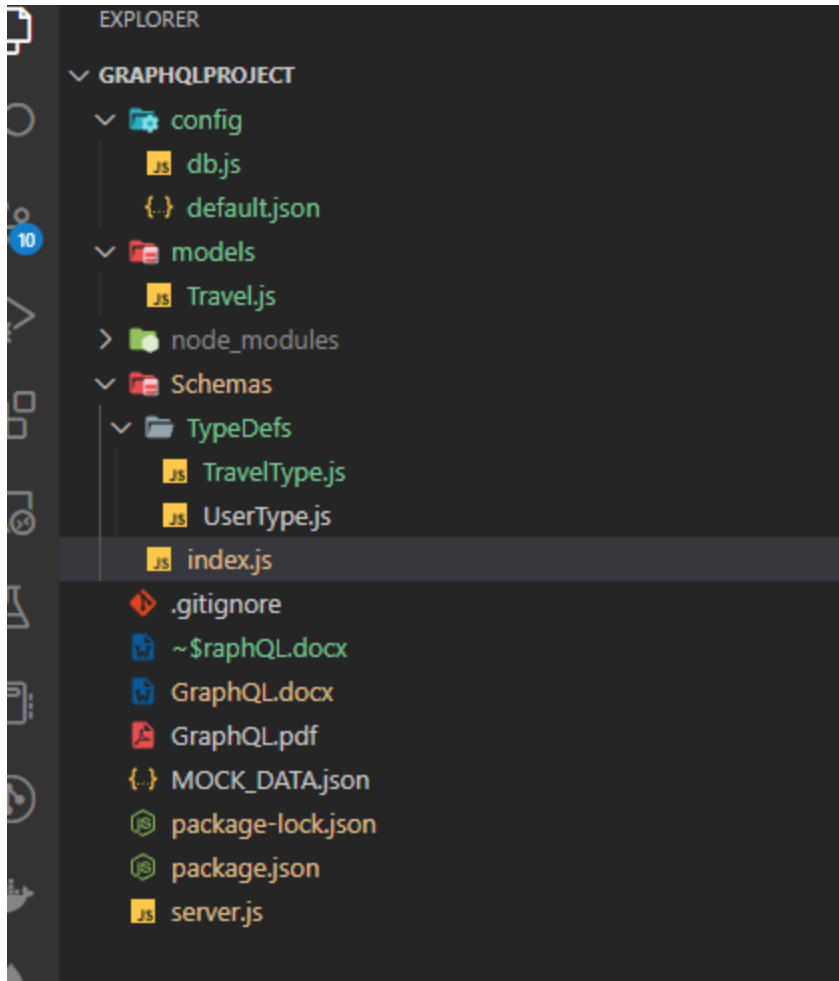
Tip: Fetching MongoDB Data with graphql

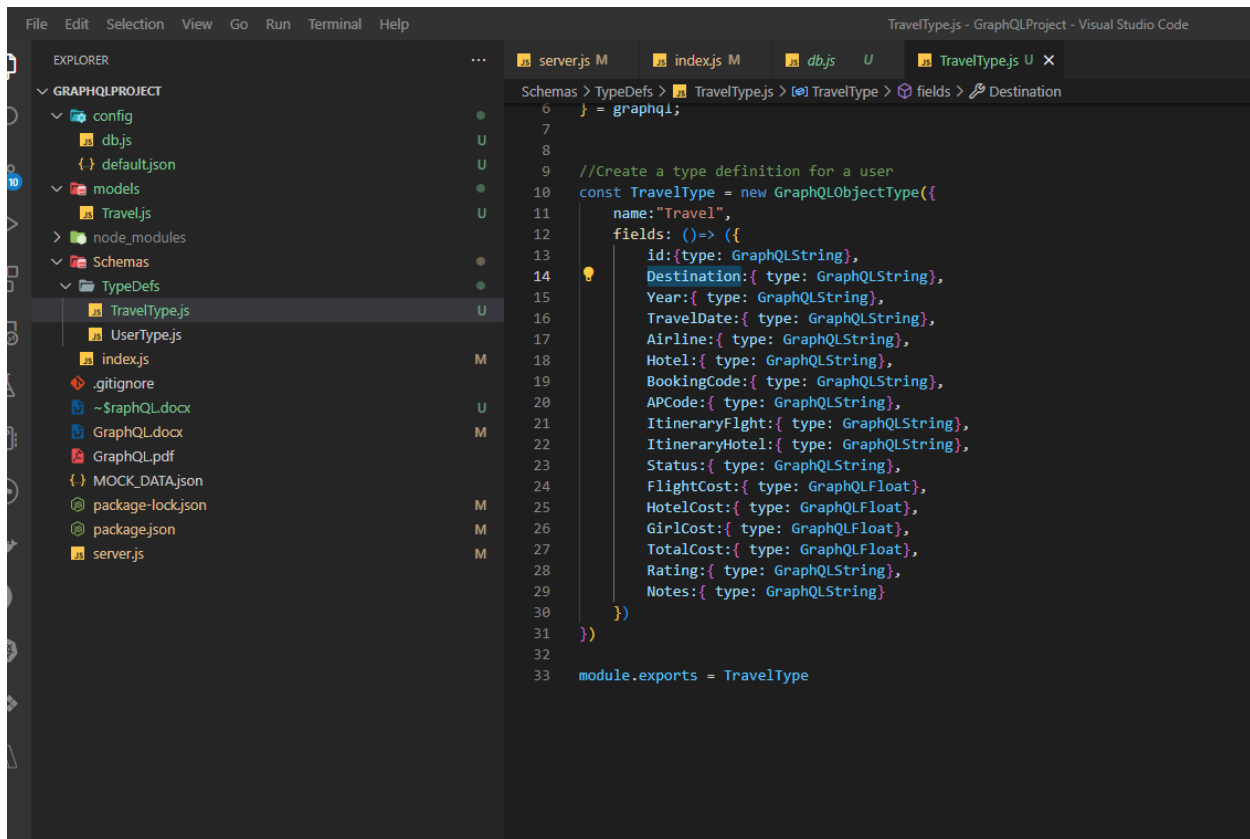
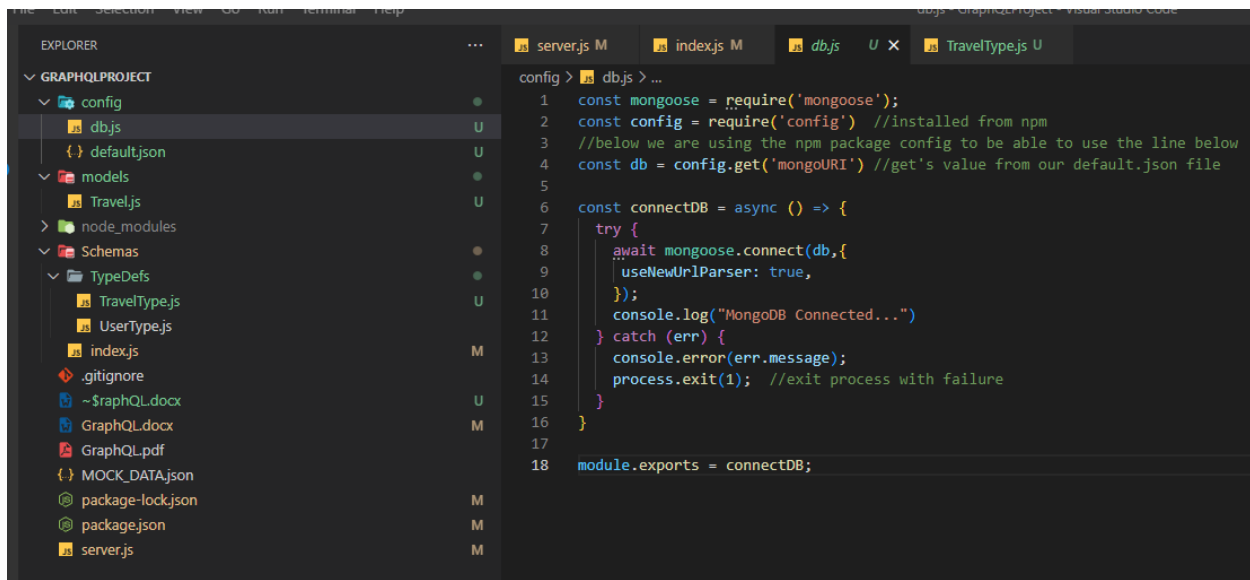
It was super easy, I just leveraged my code from my assetmgmt service using mongo express, connected to mongoDB, created my models and schema for graphql, then just did a mongoddb call to grab travel records. The resolve function in your RootQuery method where you create your queries takes the data argument as it's return.

```

    }
  },
  getAllTravelData:{
    type: new GraphQLList(TravelType) ,
    args:{id: {type: GraphQLString}},

    //the resolve function is where you would make
    resolve(parent,args) {
      return getTravelDetails() ;
    }
  },
  //If i wanted to create another query, it would
}
  
```






```

10 //FAKE DATA - THIS WOULD NORMALLY BE COMING FROM A DATABASE
11 const userData = require("../MOCK_DATA.json");
12
13 const UserType = require('../TypeDefs/UserType');
14 const TravelType = require('../TypeDefs/TravelType');
15 const Travel = require('../models/Travel');
16
17
18 async function getTravelDetails() {
19     travelRecord = await Travel.find();
20     return travelRecord;
21 }
22 //CREATE QUERIES
23 //Remember GraphQL only has "one" endpoint, below is where you create your queries
24 const RootQuery = new GraphQLObjectType({
25     name: "RootQueryType",
26     fields: {
27         //below is a query called getAllUsers
28         getAllUsers: {
29             type: new GraphQLList(UserType), //list of users - the type is defined above for our user
30             args: {id: {type: GraphQLInt}},
31
32             //the resolve function is where you would make your database call, i.e SELECT * .. or with MongoDB db.find
33             resolve(parent, args) {
34                 return userData //this is the MOCK_DATA.json data
35             }
36         },
37         getAllTravelData: {
38             type: new GraphQLList(TravelType),
39             args: {id: {type: GraphQLString}},
40
41             //the resolve function is where you would make your database call, i.e SELECT * .. or with MongoDB db.find
42             resolve(parent, args) {
43                 return getTravelDetails();
44             }
45         },
46         //If I wanted to create another query, it would be below here separated by a , i.e getUserByID
47     }
48 });
49

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS AZURE



AND IT WORKS!!!!

Tip: Project MgmtApp – Brad Traversity

<https://www.youtube.com/watch?v=BcLNfwF04Kw>

This is from the three hour video above that outlines the building of a MERN stack project using reactjs, graphql, mongodb atlas

(the following command creates the package.json file and fills in all of the defaults for you)

`npm init -y`

Install the packages below

`npm i express express-graphql graphql mongoose cors colors --force`

Nodemon below is so we don't have keep restarting and want see changes right away

dotenv is so we can use environment variables

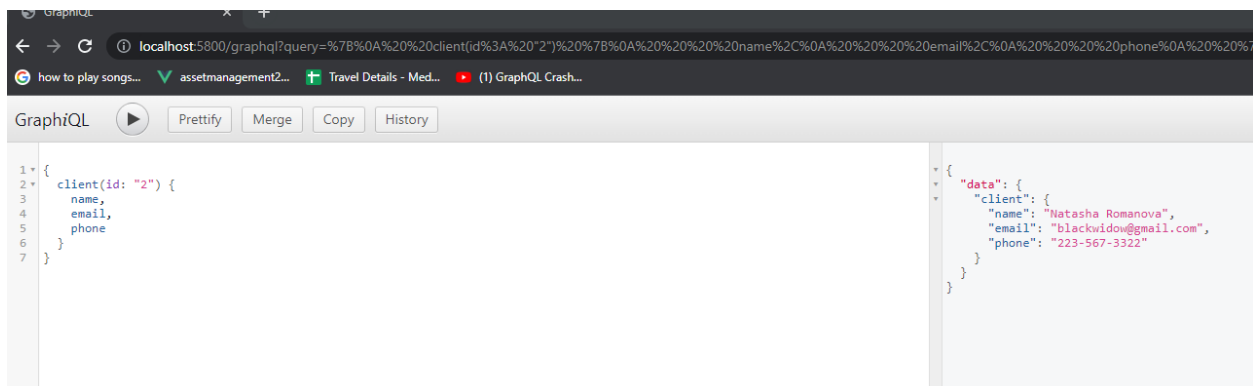
`npm i -D nodemon dotenv --force`

Git Repo

<https://github.com/lionel5116/ProjectMgmtAppGraphQL.git>

We make a call after we have wired up our code:

`http://localhost:5800/graphql`



```

// Clients
const clients = [
  {
    id: '1',
    name: 'Tony Stark',
    email: 'ironman@gmail.com',
    phone: '343-567-4333',
  },
  {
    id: '2',
    name: 'Natasha Romanova',
    email: 'blackwidow@gmail.com',
    phone: '223-567-3322',
  },
  {
    id: '3',
    name: 'Thor Odinson',
    email: 'thor@gmail.com',
    phone: '324-331-4333'
  }
]

```

Creating a relationship between two entities:

```

// Project Type
const ProjectType = new GraphQLObjectType({
  name: 'Project',
  fields: () => ({
    id: {type: GraphQLID},
    name: {type: GraphQLString},
    description: {type: GraphQLString},
    status: {type: GraphQLString},
    //establishes a relationship between two data entities
    client: {
      type: ClientType,
      resolve(parent, args) {
        return clients.find(client => client.id === parent.clientId)
      }
    }
  })
});

```

```

// Projects
const projects = [
  {
    id: '1',
    clientId: '1',
    name: 'eCommerce Website',
    description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit',
    status: 'In Progress',
  },
  {
    id: '2',
    clientId: '2',
    name: 'Dating App',
    description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit',
    status: 'In Progress',
  },
  {
    id: '3',
    clientId: '3',
    name: 'SEO Project',
    description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit',
    status: 'In Progress',
  }
]

```

```

26
27 {
28   project(id: "3") {
29     id,
30     name,
31     description,
32     status,
33     client {
34       name,
35       phone
36     }
37   }
38 }

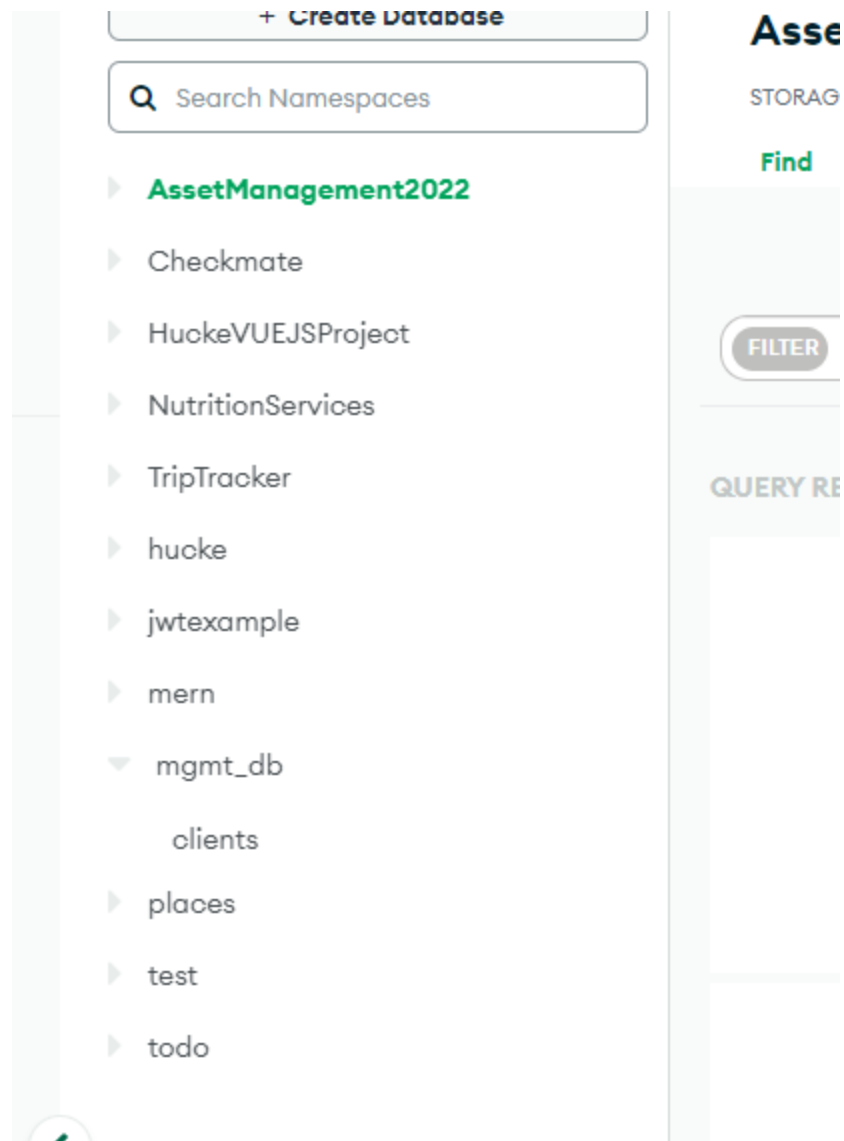
```

```

{
  "data": {
    "project": {
      "id": "3",
      "name": "SEO Project",
      "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu.",
      "status": "In Progress",
      "client": {
        "name": "Thor Odinson",
        "phone": "324-331-4333"
      }
    }
  }
}

```

Created a database and collection Atlas



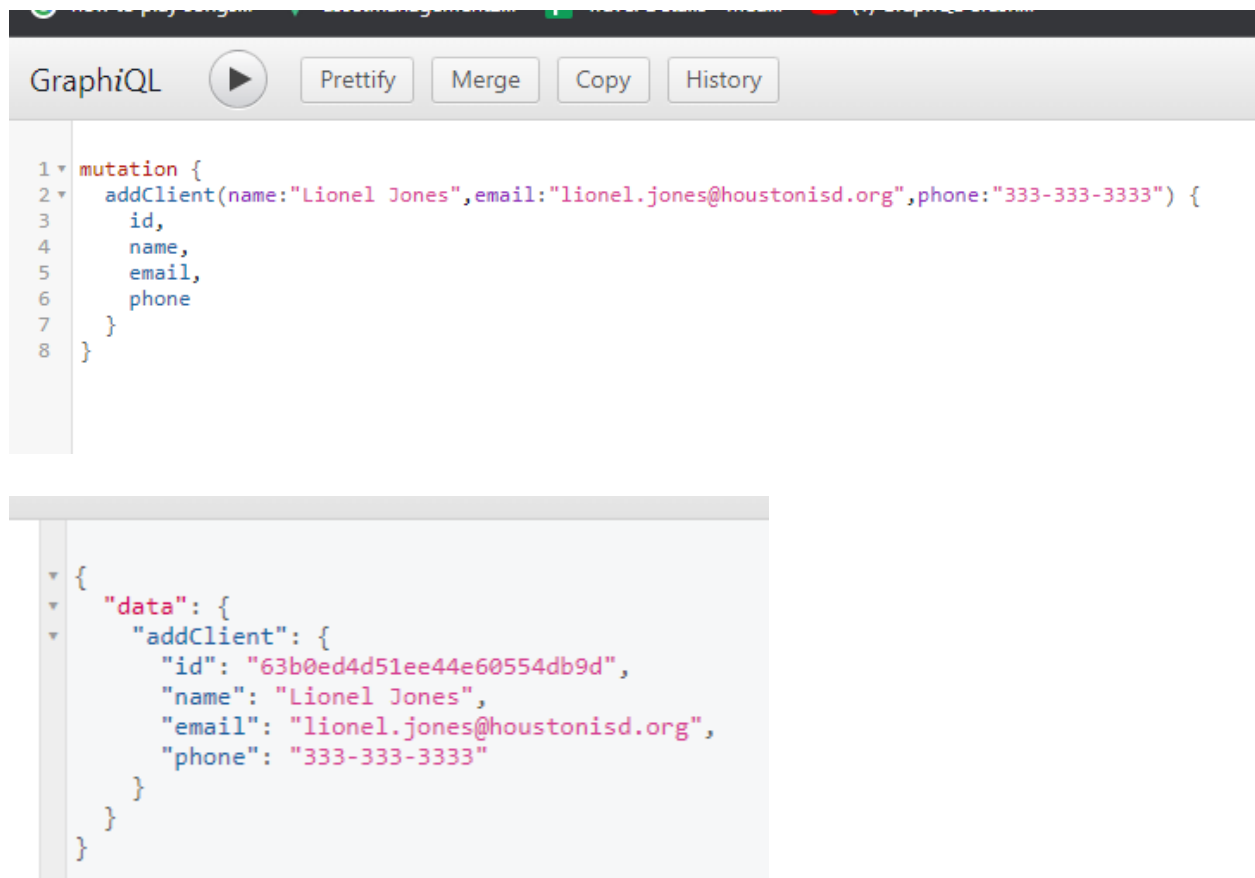
See the tip below about MongoDB Atlas GUI (Free)

Tip: Adding a record to MongoDB using GraphQL

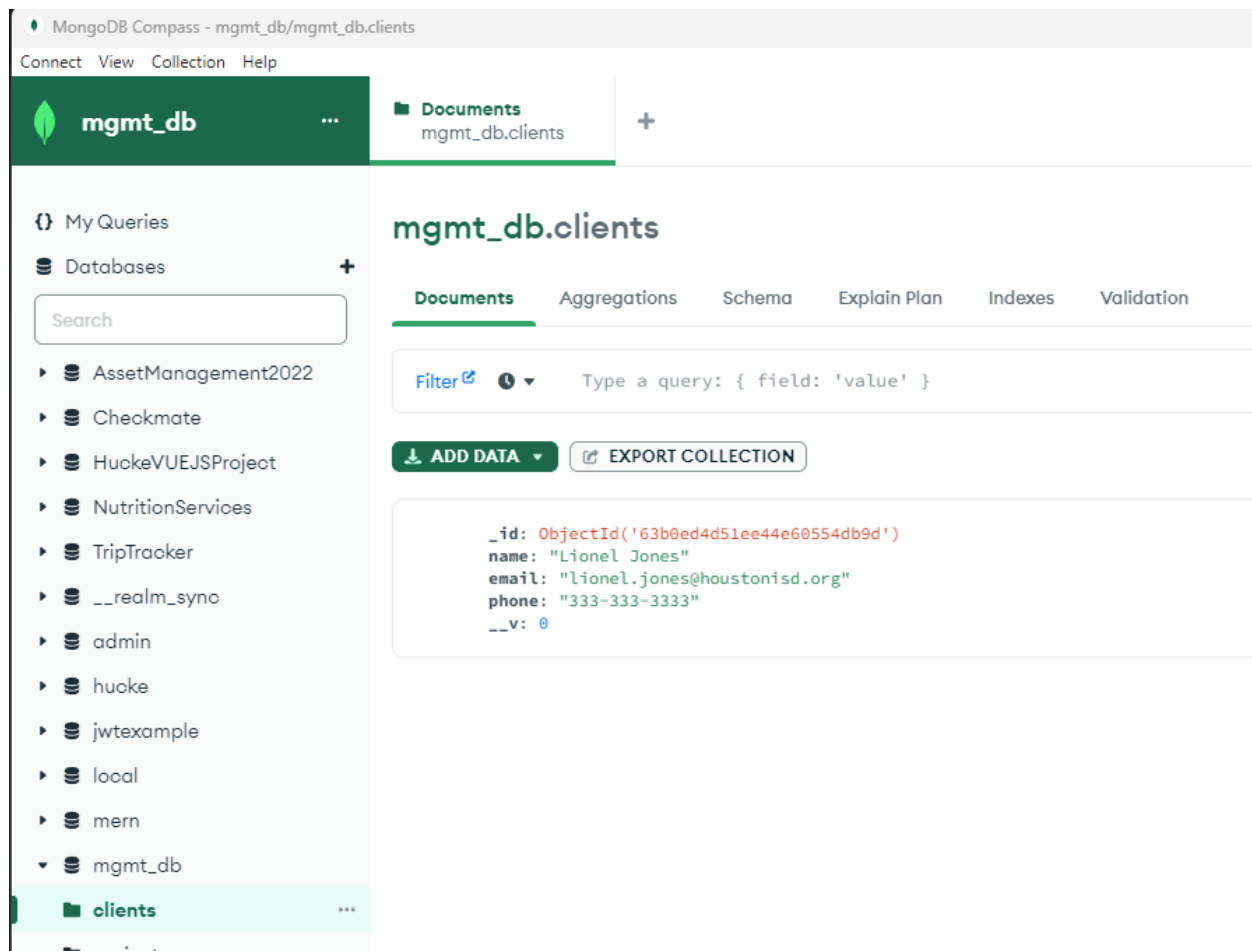
This is done via a mutation

```
1 //Mutations
2
3 const mutation = new GraphQLObjectType({
4   name: 'Mutation',
5   fields: {
6     addClient: {
7       type: ClientType,
8       args: {
9         name: { type: GraphQLNonNull(GraphQLString) },
10        email: { type: GraphQLNonNull(GraphQLString) },
11        phone: { type: GraphQLNonNull(GraphQLString) }
12      },
13      resolve(parent,args) {
14        const client = new Client( {
15          name: args.name,
16          email: args.email,
17          phone: args.phone,
18        });
19        return client.save();
20      }
21    },
22  },
23 });
```

```
mutation {
  addClient(name:"David Lee Jones",email:"david.jones@optonline.org",phone:"444-333-3333") {
    id,
    name,
    email,
    phone
  }
}
```

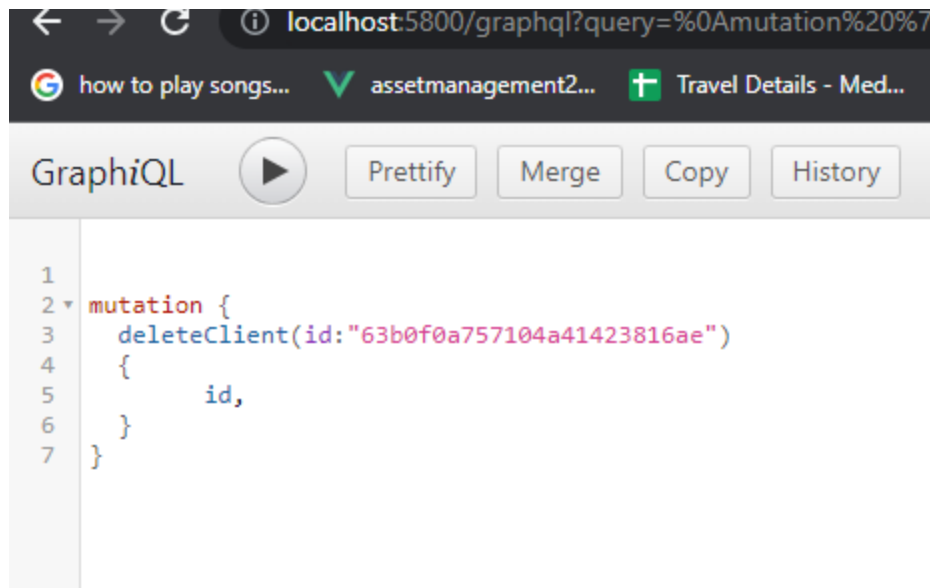


Then when the record is added, we go over to Atlas



To delete a client

```
//Delete Client
deleteClient: {
  type: ClientType,
  args: {
    id: { type: GraphQLNonNull(GraphQLID)},
  },
  resolve(parent,args) {
    return Client.findByIdAndRemove(args.id)
  }
},
```

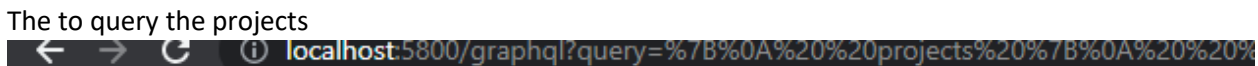
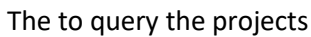


Stopped at 1:02 minutes – 12/31/2022

Mutation for Project Add

```
mutation {  
  addProject(clientId:"63b0ee2474e435b3f886f5e0",  
    name:"The Bourne Identity",  
    description:"Fast Action Paced Movie",  
    status: new) {  
    name,  
    id,  
    description,  
    status  
  }  
}
```

Notice above for the status in which is an enum, you have to use the enum key



```
{  
  projects {  
    name,  
    status  
    client {  
      name,  
      email  
    }  
  }  
}
```

Update project

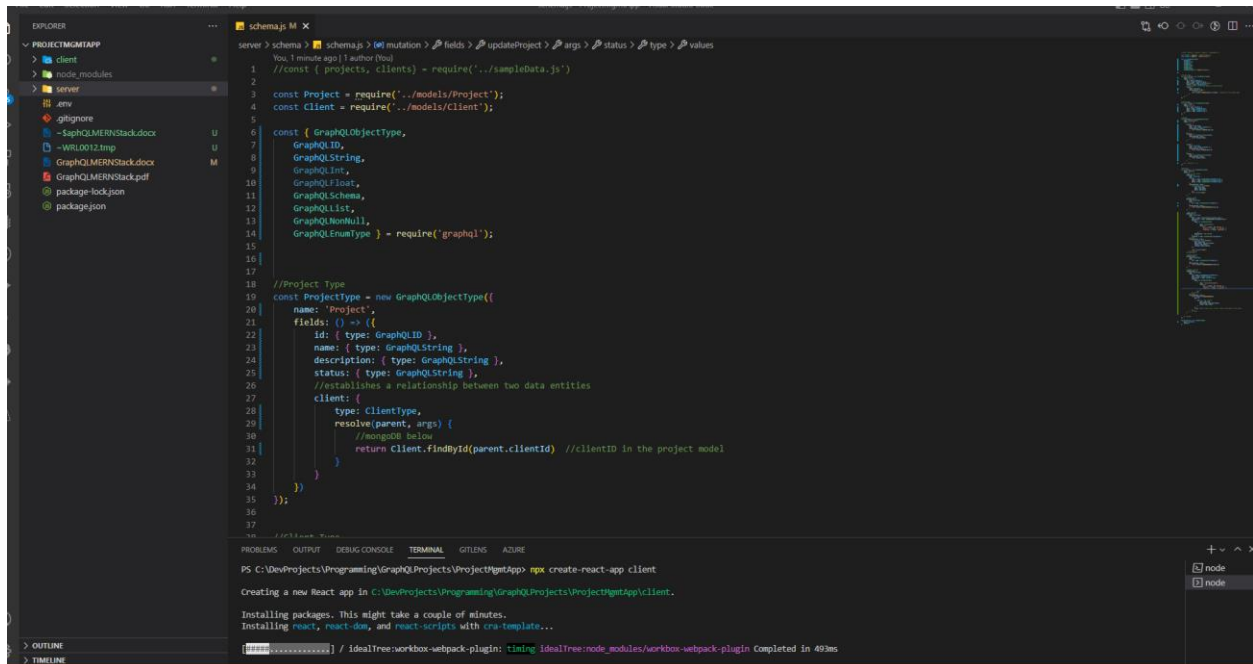
```

//update a project
updateProject: {
  type: ProjectType,
  args: {
    id: { type: GraphQLNonNull(GraphQLID)},
    name: { type: GraphQLString },
    description: { type: GraphQLString },
    status: {
      type: new GraphQLEnumType(
        {
          name: 'ProjectStatusUpdate',
          values: {
            'new': { value: 'Not Started' },
            'progress': { value: 'In Progress' },
            'completed': { value: 'Completed' },
          }
        }
      ),
    }, //status
  }, //args
  resolve(parent,args) {
    return Project.findByIdAndUpdate (
      args.id,
      {
        // You, 1 second ago • Uncommitted changes
        $set: { //set = patch
          name: args.name,
          description: args.description,
          status: args.status,
        }
      },
      {new: true} //flag: true = it will create a new project if not exists
    );
  } //resolve
},

```

```
mutation {  
  updateProject(id:"63b1dedfe008c8d0a86d4e67",  
    name:"The Bourne ReBooted 2023",  
    description:"This is a good sequel to original",  
    status:progress) {  
    name,  
    description,  
    status  
  
  }  
}
```

Creating the react app
Open up another terminal, run the
npx create-react-app client



And as shown above, it creates a folder in the root called client (this is our front end)

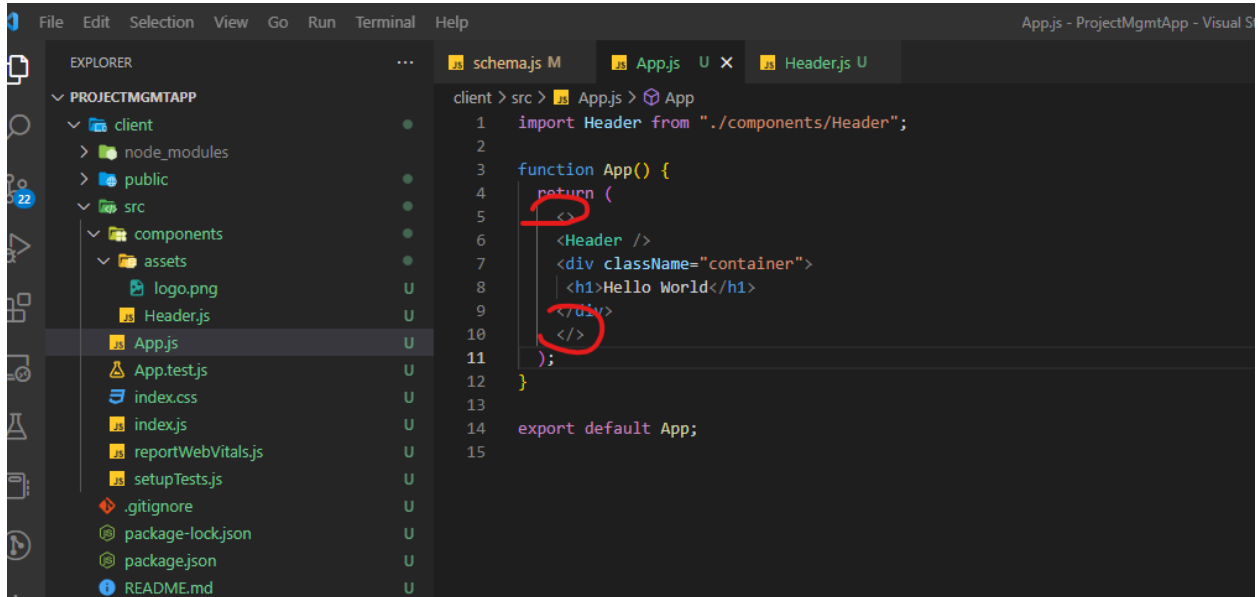
Next, cd into the client folder to install dependencies
npm i @apollo/client graphql react-router-dom react-icons

Above is adding the apollo client (to run queries against our graphql server)
Router and react icons (font-awesome icons) – from react

Tip: Using Fragment shorthand syntax

Shorthand syntax for a fragment:

`<> </>`



```
client > src > App.js > App
1  import Header from "../components/Header";
2
3  function App() {
4    return (
5      <>
6      <Header />
7      <div className="container">
8        <h1>Hello World</h1>
9      </div>
10     </>
11   );
12 }
13
14 export default App;
15
```

Tip: Adding bootstrap using the CDN

In this project he is not using bootstrap react via an npm, he is using the CDN

<https://getbootstrap.com/>



Include via CDN

When you only need to include Bootstrap's compiled CSS or JS, you can use [jsDelivr](#). See it in action with our simple [quick start](#), or [browse the examples](#) to jumpstart your next project. You can also choose to include Popper and our JS [separately](#).

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/
```

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/di
```

Add entries in the public/index.html file

```
File Edit Selection View Go Run Terminal Help
index.html - ProjectMgmtApp - Visual Studio Code

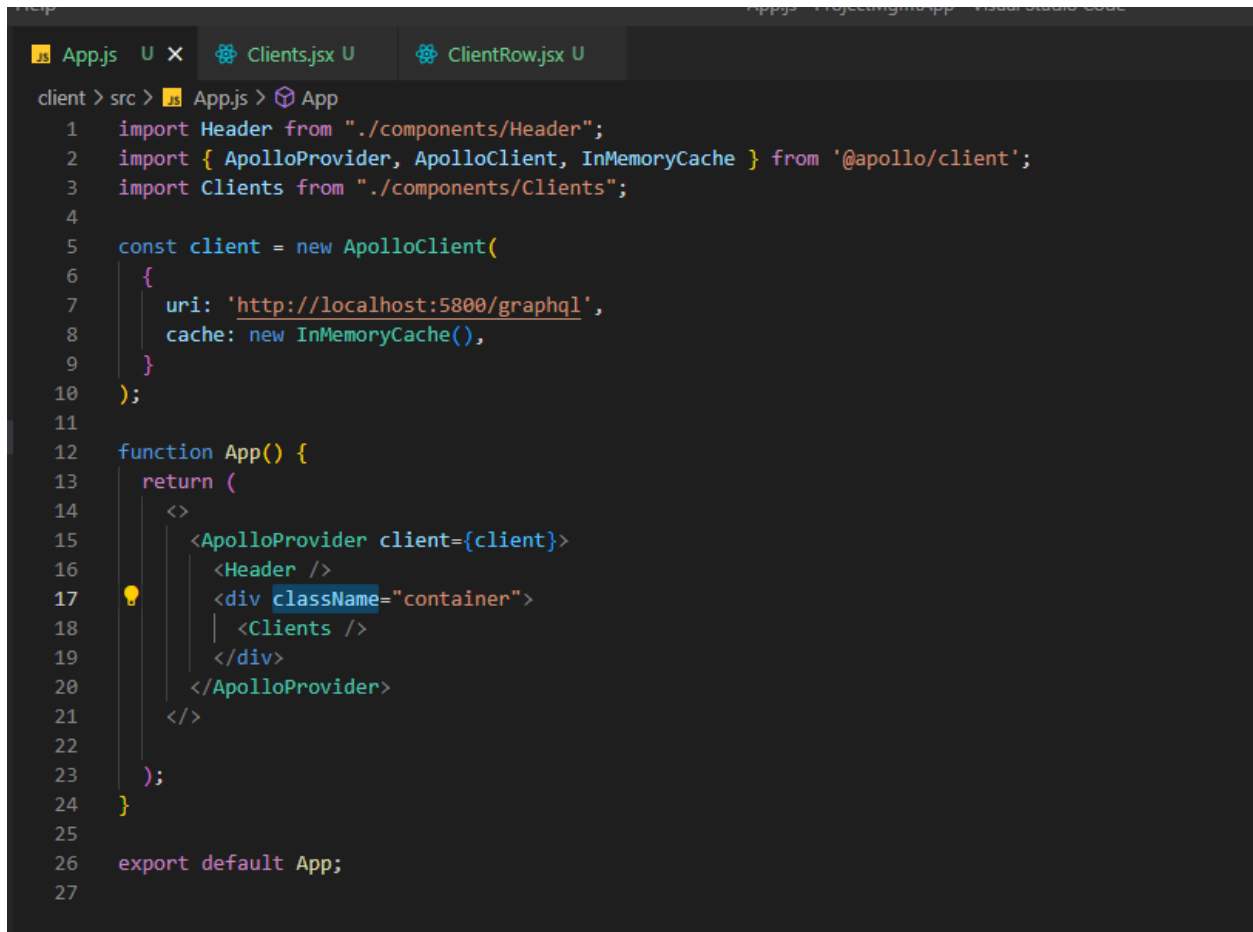
EXPLORER
PROJECTMGMTAPP
├── client
│   ├── node_modules
│   └── public
│       ├── favicon.ico
│       └── index.html
│           ├── logo192.png
│           ├── logo512.png
│           ├── manifest.json
│           └── robots.txt
├── src
│   ├── components\assets
│   │   ├── logo.png
│   │   ├── App.js
│   │   ├── App.test.js
│   │   ├── index.css
│   │   ├── index.js
│   │   ├── reportWebVitals.js
│   │   ├── setupTests.js
│   │   ├── .gitignore
│   │   ├── package-lock.json
│   │   ├── package.json
│   │   └── README.md
│   └── node_modules
├── server
└── config

client > public > index.html > html > head
9   name="description"
10   content="Web site created using create-react-app"
11 />
12 <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13 />
14   manifest.json provides metadata used when your web app is installed on a
15   user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
16   -->
17   <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18   <!--
19   Notice the use of %PUBLIC_URL% in the tags above.
20   It will be replaced with the URL of the 'public' folder during the build.
21   Only files inside the 'public' folder can be referenced from the HTML.
22
23   Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
24   work correctly both with client-side routing and a non-root public URL.
25   Learn how to configure a non-root public URL by running 'npm run build'.
26   -->
27   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
28   rel="stylesheet" integrity="sha384-GlhtlQ81RABdZL1603oVMWSktQOp6b7In12l3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
29   crossorigin="anonymous" />
30
31   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
32   integrity="sha384-w76AqPFDKMBDXo30j51Sgez6pr3x5M1Q1ZAGC+nuzB+EYdGRZgiwxhTBTkF7CXvN"
33   crossorigin="anonymous"></script>
34   <title>Project Management GraphQL</title>
35 </head>
36 <body>
37   <noscript>You need to enable JavaScript to run this app.</noscript>
```

Tip: Wiring up Apollo Server to fetch graphql data

To fetch data from our graphql server, we use apollo (as with the npm packages we installed)

In your app.js file

A screenshot of a code editor with a dark theme. The editor has three tabs at the top: 'App.js', 'Clients.jsx', and 'ClientRow.jsx'. The 'App.js' tab is active, showing a JavaScript file. The code starts with imports for 'Header' from './components/Header', and 'ApolloProvider', 'ApolloClient', and 'InMemoryCache' from '@apollo/client'. It then imports 'Clients' from './components/Clients'. A constant 'client' is created as a new 'ApolloClient' with a 'uri' of 'http://localhost:5800/graphql' and a 'cache' of 'new InMemoryCache()'. A function 'App()' is defined, which returns a JSX element. This element is a root tag containing an 'ApolloProvider' with 'client={client}', which contains a 'Header' component, a 'div' with 'className="container"' containing a 'Clients' component, and another 'Header' component. The 'App' function is then exported as the default export. Line numbers 1 through 27 are visible on the left side of the code editor.

```
client > src > App.js > App
1  import Header from "../components/Header";
2  import { ApolloProvider, ApolloClient, InMemoryCache } from '@apollo/client';
3  import Clients from "../components/Clients";
4
5  const client = new ApolloClient({
6    {
7      uri: 'http://localhost:5800/graphql',
8      cache: new InMemoryCache(),
9    }
10 });
11
12 function App() {
13   return (
14     <>
15       <ApolloProvider client={client}>
16         <Header />
17         <div className="container">
18           <Clients />
19         </div>
20       </ApolloProvider>
21     </>
22   );
23 }
24
25 export default App;
26
27
```

We import apollo references

We create our client along with our uri for our local server

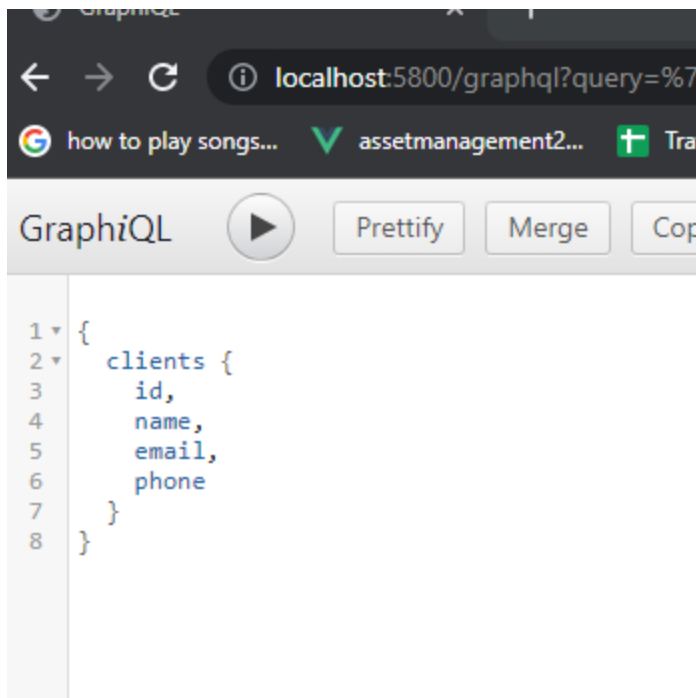
We wrap our component with the Apollo Provide and pass in our apolloclient reference

This exposes all components in our application to appollo

To query our data, in our client component

```
App.js U  Clients.jsx U X  ClientRow.jsx U
client > src > components > Clients.jsx > ...
1  import { gql, useQuery } from '@apollo/client'
2  import ClientRow from './ClientRow';
3
4  const GET_CLIENTS = gql`
5    query getClients {
6      clients {
7        id,
8        name,
9        email,
10       phone
11      }
12    }
13  `;
14
15  export default function Clients() {
16    const {loading,error, data} = useQuery(GET_CLIENTS);
17    if(loading) return <p>Loading....</p>
18    if(error) return <p>OPPS!!....</p>
19
20    return (
21      <>
22        {!loading && !error && (
23          <table className='table table-hover mt-3'>
24            <thead>
25              <tr>
26                <th>Name</th>
27                <th>Email</th>
28                <th>Phone</th>
29                <th></th>
30              </tr>
31            </thead>
32            <tbody>
33              {
34                data.clients.map(client => (
35                  <ClientRow key={client.id} client={client} />
36                ))
37              }
38            </tbody>
39          </table>
40        )
      </>
    )
  }
```

We grab the syntax from graphql



The build our query:

```
ent > src > components > Clients.jsx > ...  
1 import { gql, useQuery } from '@apollo/client'  
2 import ClientRow from './ClientRow';  
3  
4 const GET_CLIENTS = gql`  
5   query getClients {  
6     clients {  
7       id,  
8       name,  
9       email,  
10      phone  
11    }  
12  }  
13 `;  
14
```

We then use the useQuery hook

```
14  
15 export default function Clients() {  
16   const {loading,error, data} = useQuery(GET_CLIENTS);  
17   if(loading) return <p>Loading....</p>  
18   if(error) return <p>OPPS!!....</p>  
19
```

We then we render the results via a component (passing in our data as prop for the row)

```
Help ClientRow.jsx - ProjectMgmtApp - Visual Studio Code

client > src > components > Clients.jsx > ...
12 }
13 ;
14
15 export default function Clients() {
16   const {loading,error, data} = useQuery(GET_CLIENTS);
17   if(loading) return <p>Loading....</p>
18   if(error) return <p>OPPS!!....</p>
19
20   return (
21     <>
22       {!loading && !error && (
23         <table className='table table-hover mt-3'>
24           <thead>
25             <tr>
26               <th>Name</th>
27               <th>Email</th>
28               <th>Phone</th>
29             </tr>
30           </thead>
31           <tbody>
32             <tbody>
33               {
34                 data.clients.map(client => (
35                   <ClientRow key={client.id} client={client} />
36                 ))
37               }
38             </tbody>
39           </tbody>
40         </table>
41       )}
42     </>
43   )
44 }



client > src > components > ClientRow.jsx > ...
1 import {FaTrash} from 'react-icons/fa'
2
3 export default function ClientRow({client}) {
4   return (
5     <tr>
6       <td>{client.name}</td>
7       <td>{client.email}</td>
8       <td>{client.phone}</td>
9       <td>
10         <button className="btn-danger btn-sm" >
11           <FaTrash />
12         </button>
13       </td>
14     </tr>
15   )
16 }
17
```

Tip: Building a re-usable component for rows in a table

Tip: Using react-icons

Notice above how we built a re-usable component for our row

Also look at how we are using react-icons as well

Project Mgmt Graphql			
Name	Email	Phone	
Lionel Jones	lionel.jones@houstonisd.org	333-333-3333	
David Lee Jones	david.jones@optonline.org	444-333-3333	

Tip: Adding data with graphql with a MODAL dialog
See the AddClientModal.jsx

```
AddClientModal.jsx U X
client > src > components > AddClientModal.jsx > AddClientModal > onSubmit
import { ADD_CLIENT } from '../mutations/clientMutations';
import { GET_CLIENTS } from '../queries/clientQueries'
6
export default function AddClientModal() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [phone, setPhone] = useState('');
11
  const [addClient] = useMutation(ADD_CLIENT, {
12    variables: {
13      name,
14      email,
15      phone
16    },
17    refetchQueries: [{ query: GET_CLIENTS }]
18  })
19
  const onSubmit = (e) => {
20    e.preventDefault();
21
22    if(name === '' || email === '' || phone === '') {
23      return alert('Please fill in all relative fields')
24    }
25
26    addClient(name, email, phone)
27  }
28
  return (
29    <>
30
31    <button type="button"
32      className="btn btn-secondary"
33      data-bs-toggle="modal"
34      data-bs-target="#addClientModal">
35
36    <div className="d-flex align-items-center">
37      <FaUser className='icon' />
38    </div>
39  </button>
40  </>
  )
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS AZURE

You can now view `client` in the browser.

Local: http://localhost:3000
On Your Network: http://192.168.50.60:3000

Along with mutations

Help clientMutations.js - ProjectMgmtApp - Visu

AddClientModal.jsx U clientMutations.js M X

client > src > mutations > clientMutations.js > ...

```
1  import { gql } from '@apollo/client'
2
3
4  const ADD_CLIENT = gql`
5  mutation addClient($name: String!, $email: String!, $phone: String!) {
6    addClient(name: $name, email: $email, phone: $phone) {
7      id,
8      name,
9      email,
10     phone
11   }
12 }
13 `;
14
15
16 const DELETE_CLIENT = gql`
17 mutation deleteClient($id:ID!) {
18   deleteClient(id: $id) {
19     id,
20     name,
21     email,
22     phone
23   }
24 }
25 `;
26
27 export {DELETE_CLIENT, ADD_CLIENT};
```

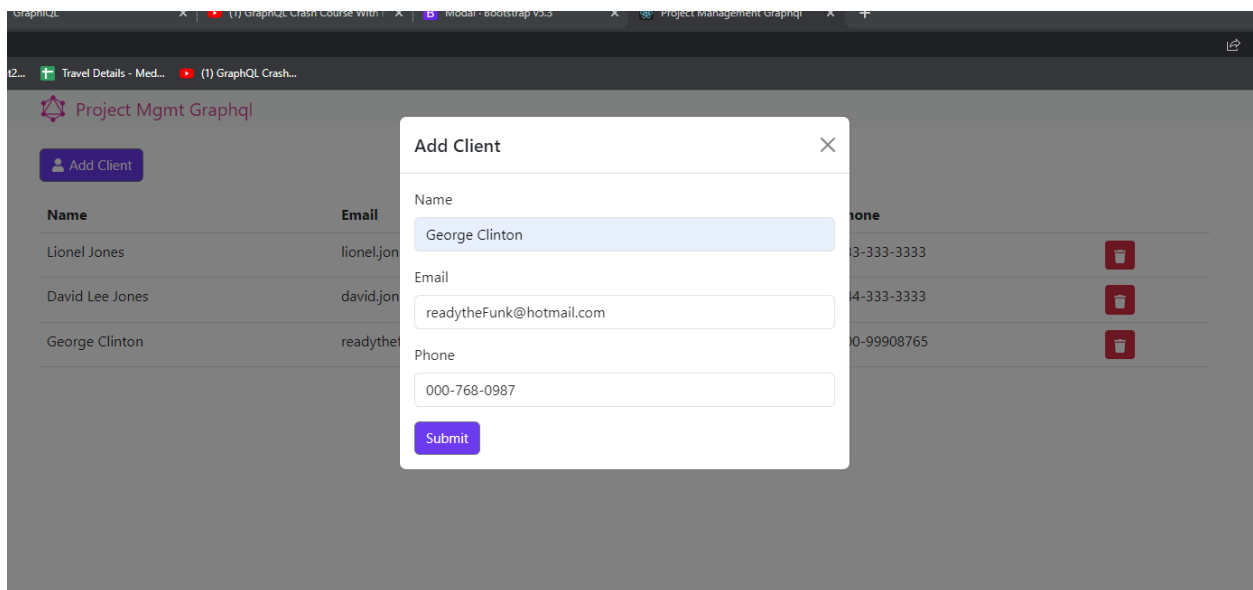
You, 1 second ago • Uncommitted changes

In app.js

```

33   }
34   );
35
36   function App() {
37     return (
38       <>
39         <ApolloProvider client={client}>
40           <Header />
41           <div className="container">
42             <AddClientModal />
43             <Clients />
44           </div>
45         </ApolloProvider>
46       </>
47     );
48   };
49 }
50
51 export default App;

```



2... Travel Details - Med... (1) GraphQL Crash...

Project Mgmt Graphql

Add Client

Name	Email	Phone	
Lionel Jones	lionel.jones@houstonisd.org	333-333-3333	
David Lee Jones	david.jones@optonline.org	444-333-3333	
George Clinton	readythefonk@hotmail.com	000-99908765	

Sweet

Tip: Creating a Spinner with built-in react spinner

Spinner.jsx - ProjectMgmtApp - Visual Studio

App.js U clientQueries.js U Clients.jsx U Spinner.jsx U X

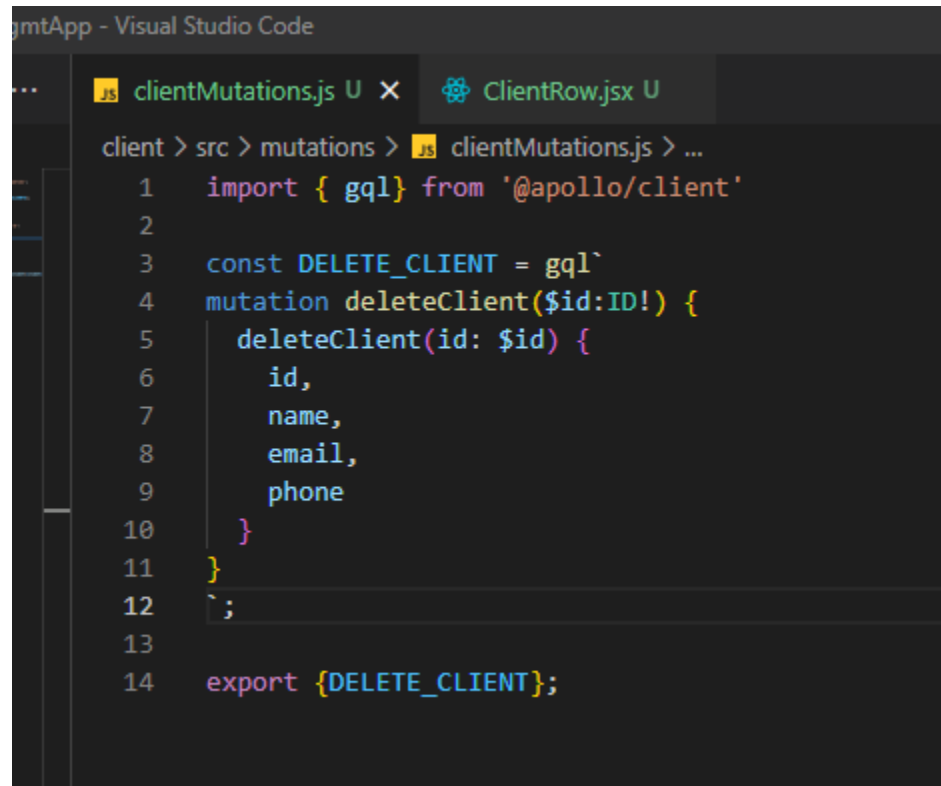
client > src > components > Spinner.jsx > ...

```
1 export default function Spinner() {
2   return (
3     <div className="d-flex justify-content-center">
4       <div className='spinner-border' role='status'>
5         <span className='rs-only'>Loading...</span>
6       </div>
7     </div>
8   )
9 }
10
```

```
Help Clients.jsx - ProjectMgmtApp - Visual Studio Code
App.js U clientQueries.js U Clients.jsx U X Spinner.jsx U
client > src > components > Clients.jsx > Clients
1 import { useQuery } from '@apollo/client'
2 import ClientRow from './ClientRow';
3 import Spinner from './Spinner';
4 import { GET_CLIENTS } from '../queries/clientQueries';
5
6
7 export default function Clients() {
8   const {loading,error, data} = useQuery(GET_CLIENTS);
9   if(loading) return <Spinner />
10  if(error) return <p>OPPS!!....</p>
11
12  return (
13    <>
14      <table className='table table-hover mt-3'>
15        <thead>
16          <tr>
17            <th>Name</th>
18            <th>Email</th>
19
```

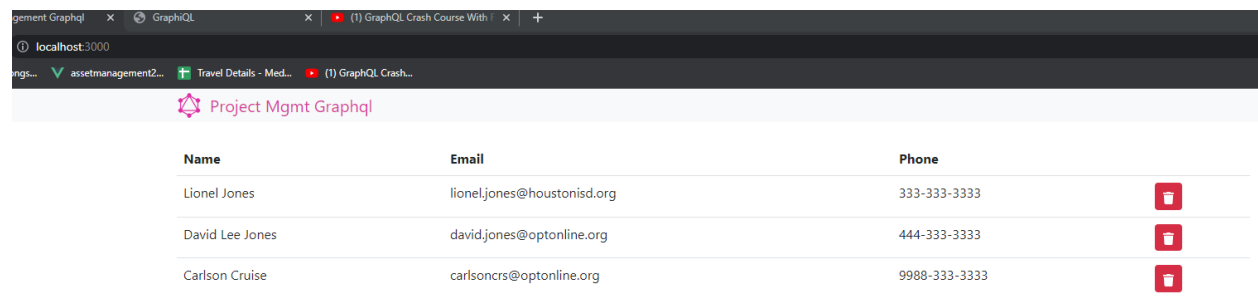
Tip: Deleting and re-fetching data with graphql




Create the mutation



```
client > src > mutations > clientMutations.js > ...
1  import { gql } from '@apollo/client'
2
3  const DELETE_CLIENT = gql`
4  mutation deleteClient($id:ID!) {
5    deleteClient(id: $id) {
6      id,
7      name,
8      email,
9      phone
10   }
11 }
12 `;
13
14 export {DELETE_CLIENT};
```

And below we can use the refetchQueries attribute to refresh the UI when a client record is deleted



Name	Email	Phone	
Lionel Jones	lionel.jones@houstonisd.org	333-333-3333	
David Lee Jones	david.jones@optonline.org	444-333-3333	
Carlson Cruise	carlsoncrs@optonline.org	9988-333-3333	



When we hit delete, the UI updates with the remaining records

GraphQL

(1) GraphQL Crash Course With

Travel Details - Med... (1) GraphQL Crash...

Project Mgmt GraphQL

Name	Email	Phone	
Lionel Jones	lionel.jones@houstonisd.org	333-333-3333	
David Lee Jones	david.jones@optonline.org	444-333-3333	

ClientRow.jsx - ProjectMgmtApp - Visual Studio Code

```

1  import { FaTrash } from 'react-icons/fa'
2  import { useMutation } from '@apollo/client'
3  import { DELETE_CLIENT } from '../mutations/clientMutations'
4  import { GET_CLIENTS } from '../queries/clientQueries'
5
6  export default function ClientRow({client}) {
7    const [deleteClient] = useMutation(DELETE_CLIENT, {
8      variables: {id: client.id},
9      refetchQueries:[{query:GET_CLIENTS}]
10   });
11   return (
12     <tr>
13       <td>{client.name}</td>
14       <td>{client.email}</td>
15       <td>{client.phone}</td>
16       <td>
17         <button className="btn btn-danger btn-sm"
18           onClick={deleteClient}>
19           <FaTrash />
20         </button>
21       </td>
22     </tr>
23   )
24 }
25
26 client={client}

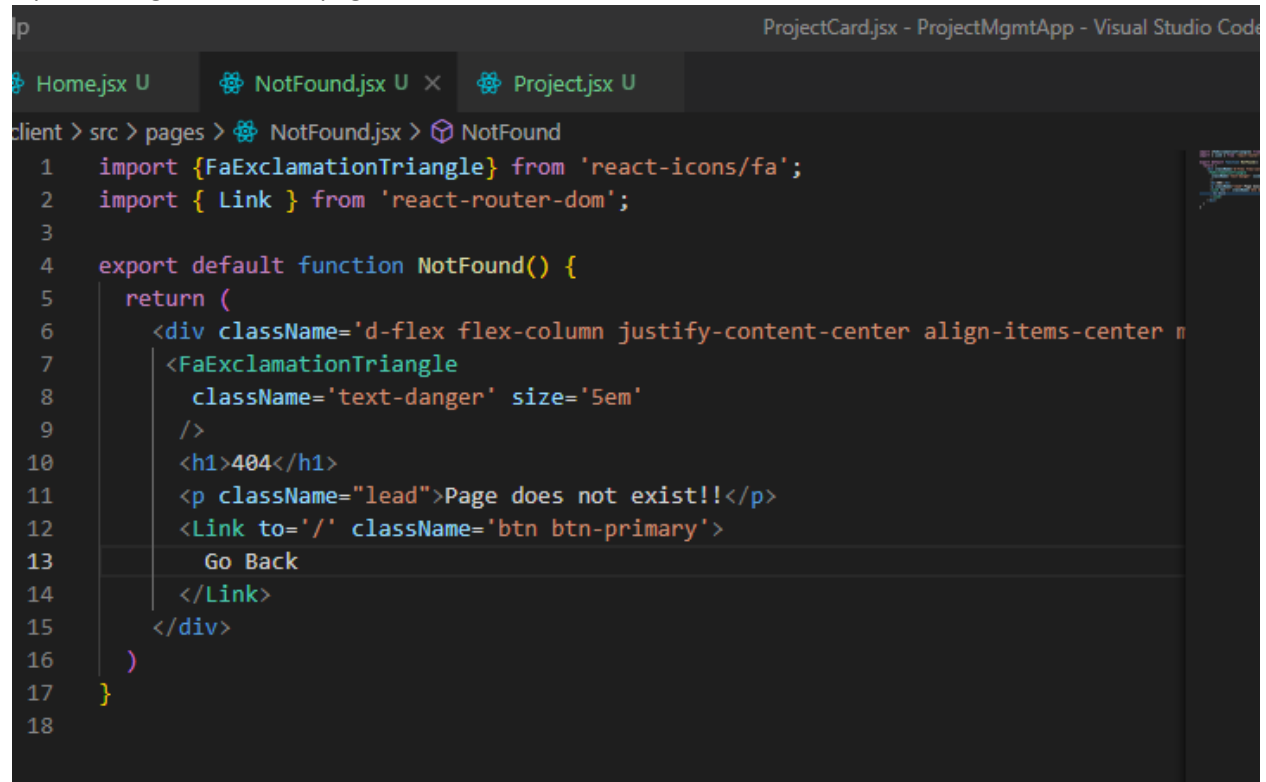
```

Second way:

```
client > src > components > ClientRow.jsx > ClientRow
1 import { FaTrash } from 'react-icons/fa'
2 import { useMutation } from '@apollo/client'
3 import { DELETE_CLIENT } from '../mutations/clientMutations'
4 import { GET_CLIENTS } from '../queries/clientQueries';
5
6 export default function ClientRow({ client }) {
7   const [deleteClient] = useMutation(DELETE_CLIENT, {
8     variables: { id: client.id },
9     refetchQueries: [{ query: GET_CLIENTS }]
10   /*
11     update(cache, { data: { deleteClient } }) {
12       const { clients } = cache.readQuery({ query: GET_CLIENTS });
13       cache.writeQuery({
14         query: GET_CLIENTS,
15         data: {
16           clients: clients.filter((client) => client.id !== deleteClient.id),
17         },
18       });
19     },
20   */
21 });
22   return (
23     <tr>
24       <td>{client.name}</td>
25       <td>{client.email}</td>
26       <td>{client.phone}</td>
27       <td>
28         <button className="btn btn-danger btn-sm"
29           onClick={deleteClient}>
30           <FaTrash />
31         </button>
32       </td>
33     </tr>
34   )
35 }
36
37
```

More code as shown above

Tip: Creating a not Found page



```
1 import {FaExclamationTriangle} from 'react-icons/fa';
2 import { Link } from 'react-router-dom';
3
4 export default function NotFound() {
5   return (
6     <div className='d-flex flex-column justify-content-center align-items-center m
7       <FaExclamationTriangle
8         className='text-danger' size='5em'
9       />
10      <h1>404</h1>
11      <p className="lead">Page does not exist!!</p>
12      <Link to="/" className='btn btn-primary'>
13        Go Back
14      </Link>
15    </div>
16  )
17 }
18
```

```
32
33 const client = new ApolloClient(
34   {
35     uri: 'http://localhost:5800/graphql',
36     cache: cache,
37   }
38 );
39
40 //the path with * needs to always be the last element
41 function App() {
42   return (
43     <>
44       <ApolloProvider client={client}>
45         <Router>
46           <Header />
47           <div className="container">
48             <Routes>
49               <Route path="/" element={<Home />}></Route>
50               <Route path="*" element={<NotFound />}></Route>
51             </Routes>
52           </div>
53         </Router>
54       </ApolloProvider>
55     </>
56   );
57 }
58
59 }
60
61 export default App;
62
```

You, 2 minutes ago • Uncommitted changes



404

Page does not exist!!

Go Back

Tip: GUI for MongoDB

<https://www.mongodb.com/products/compass>

×

Connect to Cluster0

✓ Setup connection security

✓ Choose a connection method

Connect

I do not have MongoDB Compass

I have MongoDB Compass

1 Select your operating system and download MongoDB Compass

macOS arm64 (M1) (11.0+) ▼

Download Compass (1.34.2)

 or

Copy download URL

2 Copy the connection string, then open MongoDB Compass.

mongodb+srv://lionel5116:<password>@cluster0.jwcnt.mongodb.net/test

You will be prompted for the password for the **lionel5116** user's (Database User) username. When entering your password, make sure that any special characters are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back

Close

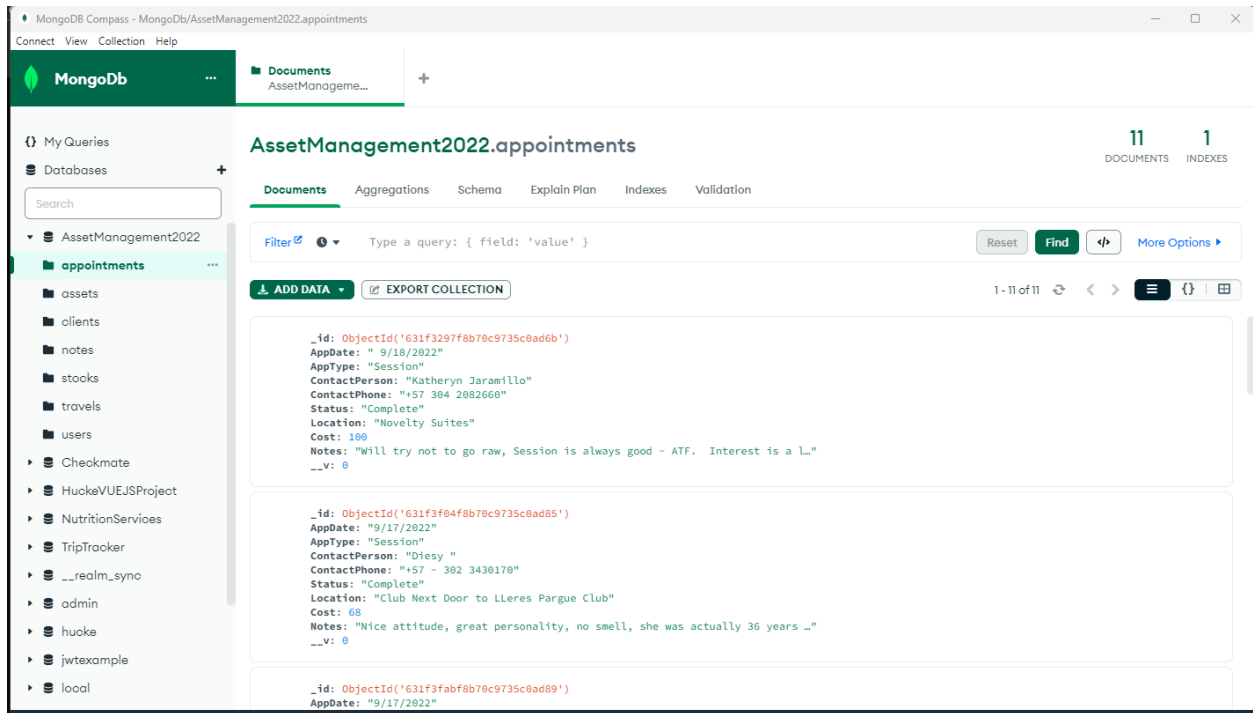
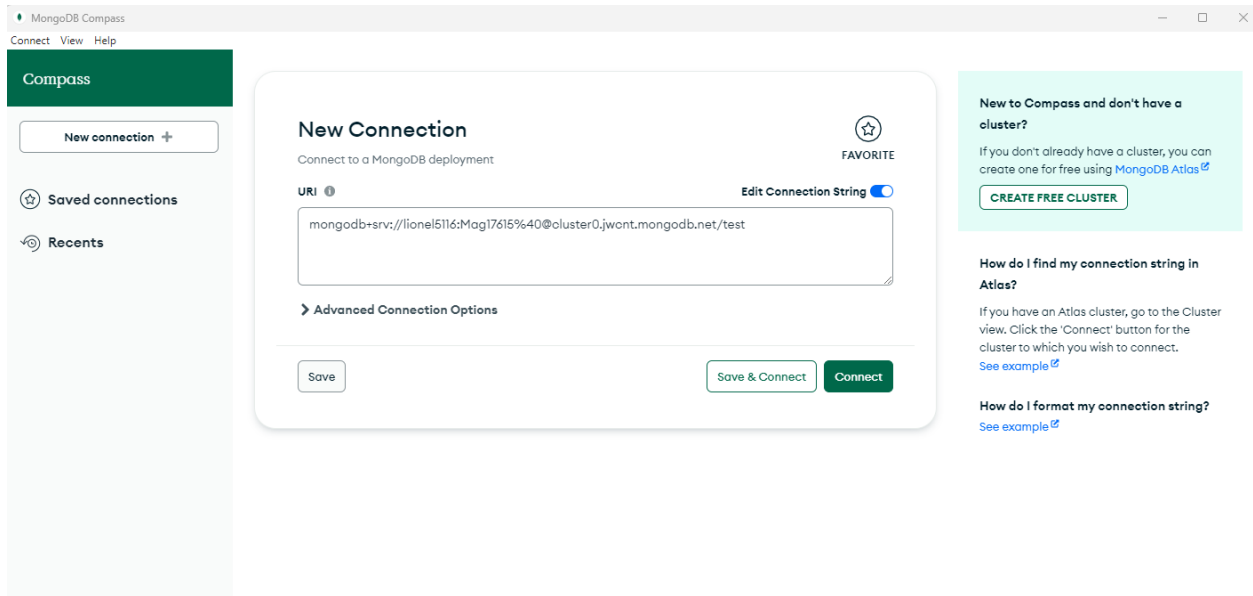
{

"mongoURI" :

"mongodb+srv://lionel5116:Mag17615%40@cluster0.jwcnt.mongodb.net/AssetManagement2022?retryWrites=true&w=majority",

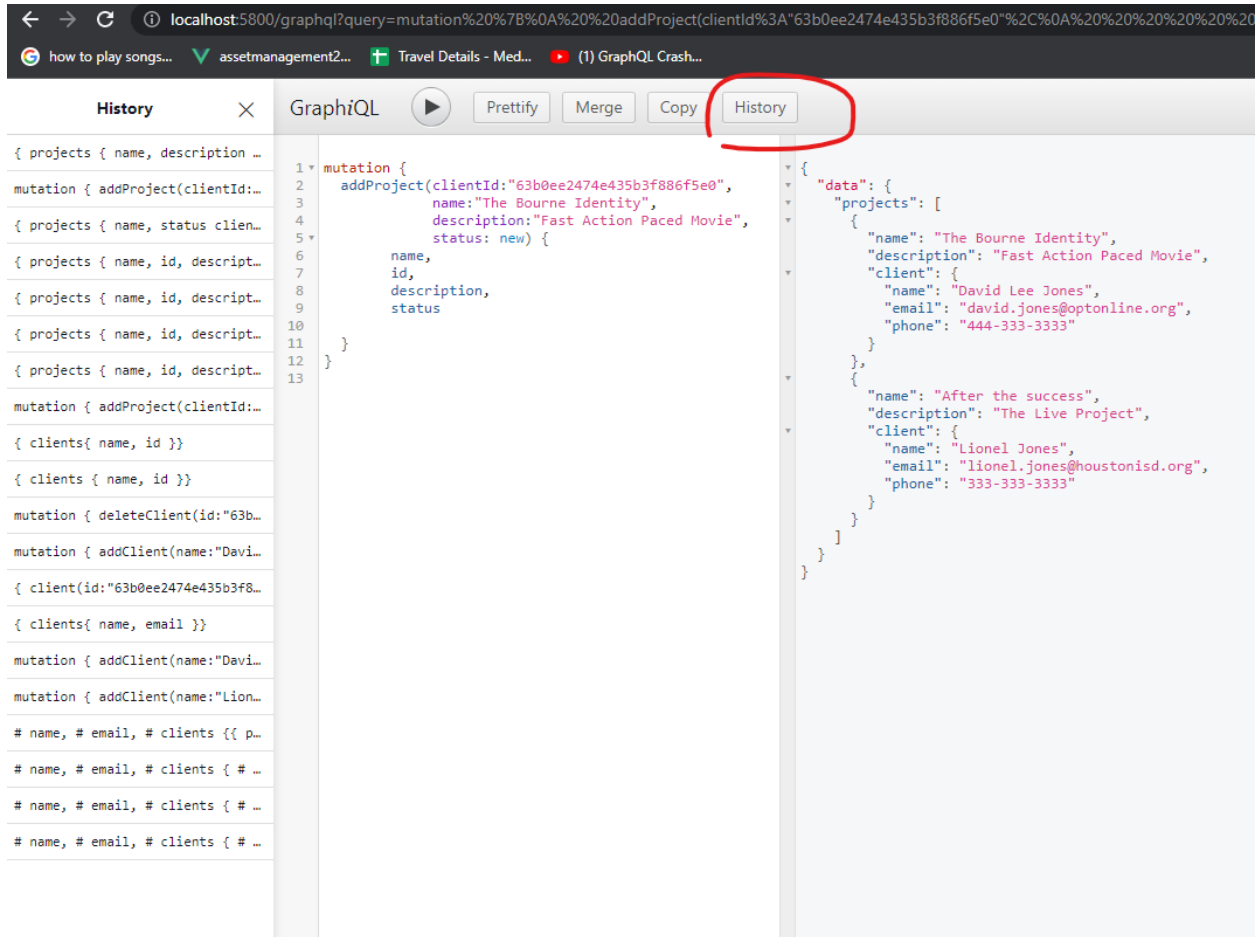
"jwtSecretToken":"mysecrettoken"

}



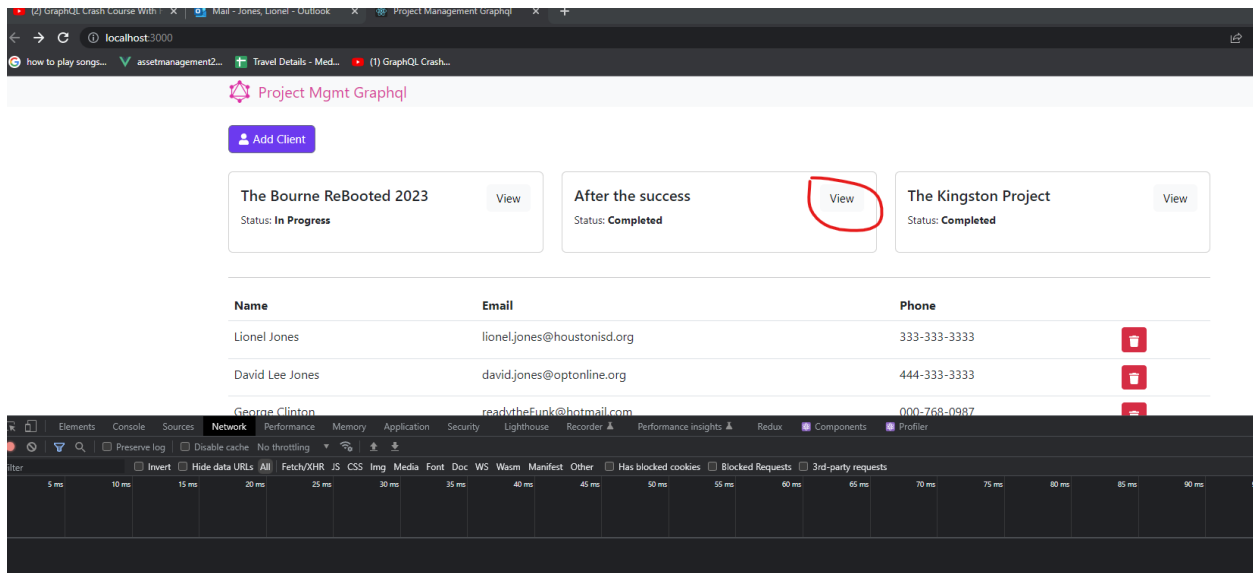
Tip: Getting History from graphql

To get the previous syntax, you can just use the history button, that way you don't have to comment out anything to re-enter syntax

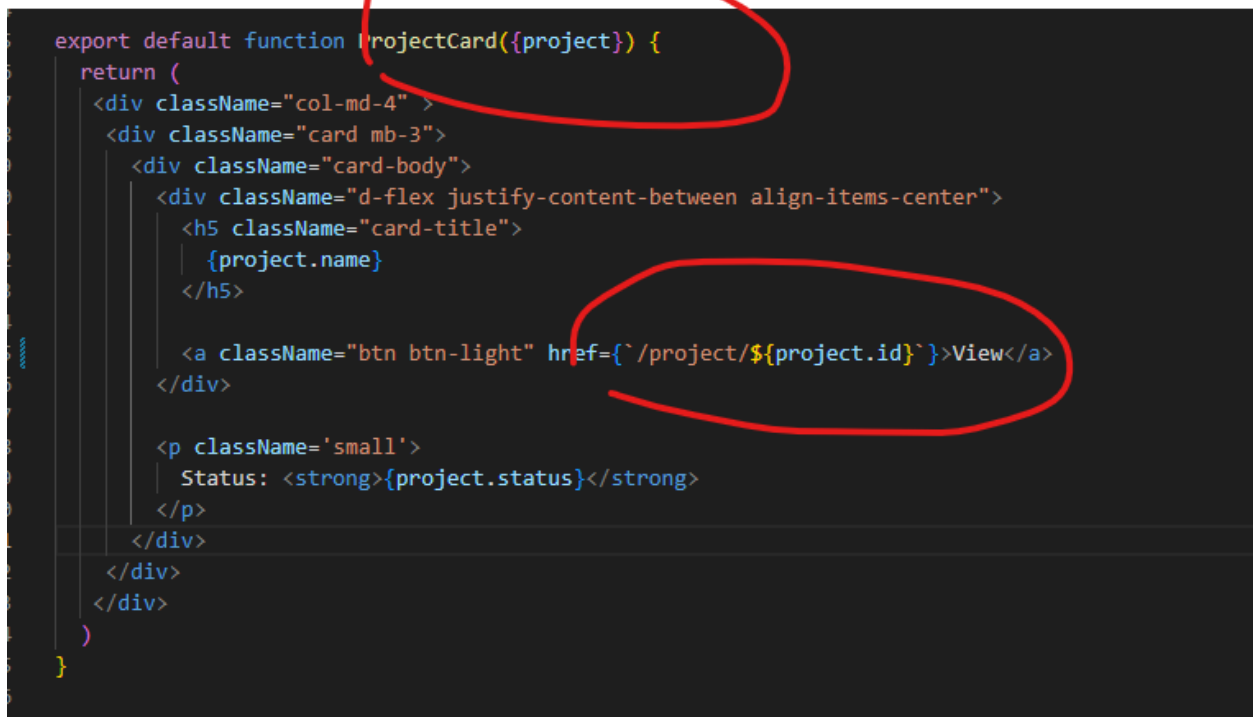


Tip: Grabbing the network response data with developer tools to view graphql data

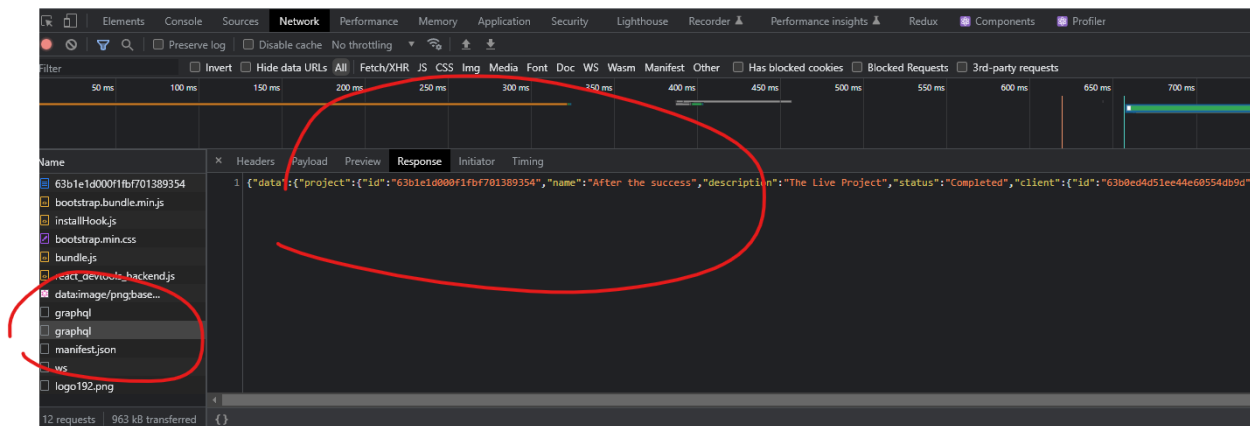
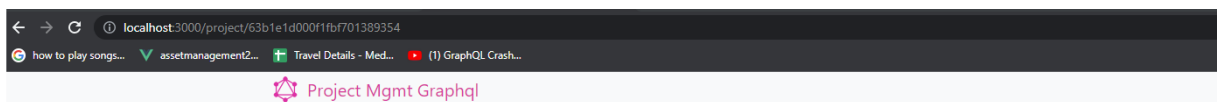
To view and check to see what data is being passed when you click on a url and it fetches graphql data, use the network and response tab:



Once you click on the view link:



```
Help ProjectCard.jsx - ProjectMgmtApp - Visual Studio Code
ProjectCard.jsx M X
client > src > components > ProjectCard.jsx > ProjectCard
1 //import { FaTrash } from 'react-icons/fa'
2 //import { useMutation } from '@apollo/client'
3 //import { GET_PROJECTS } from '../queries/projectQueries'
4
5 export default function ProjectCard({project}) {
6   return (
7     <div className="col-md-4">
8       <div className="card mb-3">
9         <div className="card-body">
10           <div className="d-flex justify-content-between align-items-center">
11             <h5 className="card-title">
12               {project.name}
13             </h5>
14             <a className="btn btn-light" href={`/${project.id}`}>View</a>
15           </div>
16           <p className="small">
17             Status: <strong>{project.status}</strong>
18           </p>
19         </div>
20       </div>
21     </div>
22   )
23 }
24
25
26
ProjectCard.jsx X
client > src > components > Projects.jsx > ...
1 You, 13 hours ago | 1 author (you)
2 import Spinner from './Spinner';
3 import { useQuery } from '@apollo/client';
4 import { GET_PROJECTS } from '../queries/projectQueries';
5 import ProjectCard from './ProjectCard';
6
7 export default function Projects() {
8   const { loading, error, data } = useQuery(GET_PROJECTS);
9   if (loading) return <Spinner />
10  if (error) return <p>OOPS!!...</p>
11  return (
12    <div>
13      {data.projects.length > 0 ? (
14        <div className="row mt-3">
15          {data.projects.map((project) => (
16            <ProjectCard key={project.id} project={project} />
17          ))}
18        </div>
19      ) : (
20        <p>No Projects</p>
21      )}
22    </div>
23  )
24 }
25
26
27
```



And you can see the data being passed in via the getProject by :\$id

Tip: CSS syntax and FontAwsome react
This is nice using react fontawsome

Back




The Kingston Project

Most Important Project to date

Project Status

Completed

Client Information

	Lionel Jones
	lionel.jones@houstonisd.org
	333-333-3333

```
Help ClientInfo.jsx - ProjectMgmtApp - Visu
Project.jsx M ClientInfo.jsx U X
client > src > components > ClientInfo.jsx > ClientInfo
1 import {FaEnvelope,FaPhone,FaIdBadge} from 'react-icons/fa';
2
3
4 export default function ClientInfo({client}) {
5   return (
6     <>
7       <h5 className="mt-5">Client Information</h5>
8       <ul className="list-group">
9         <li className="list-group-item">
10           <FaIdBadge className='icon' />{client.name}
11         </li>
12         <li className="list-group-item">
13           <FaEnvelope className='icon' />{client.email}
14         </li>
15         <li className="list-group-item">
16           <FaPhone className='icon' />{client.phone}
17         </li>
18       </ul>
19     </>
20   )
21 }
22
23
```

Also if you notice this syntax for css

```
7
8 export default function Project() {
9   const {id} = useParams()
10  const {loading,error, data} = useQuery(GET_PROJECT, {variables: {id}});
11  if(loading) return <Spinner />
12  if(error) return <p>OPPS!!....</p>
13  return (
14    <>
15    {!loading && !error && (
16      <div className="mx-auto w-75 card p-5">
17        <Link to="/" className="btn btn-light btn-sm w-25 d-inline ms-auto">
18          Back
19        </Link>
20
21        <h1>{data.project.name}</h1>
22        <p>{data.project.description}</p>
23        <h5 className="mt-3">Project Status</h5>
24        <p className="lead">{data.project.status}</p>
25
26        <ClientInfo client={data.project.client} />
27
28      </div>
29    )}
30    </>
31  )
32 }
```

Compiled successfully!

notice the w-25 p-5 syntax, this is taken from bootstrap

the w = width

the 25 is the pixels

This is really just shortcut syntax that is from bootstrap 4-5

Bootstrap crash course

<https://www.youtube.com/watch?v=5GcQtLDGXy8>

Tip: Working with Modals

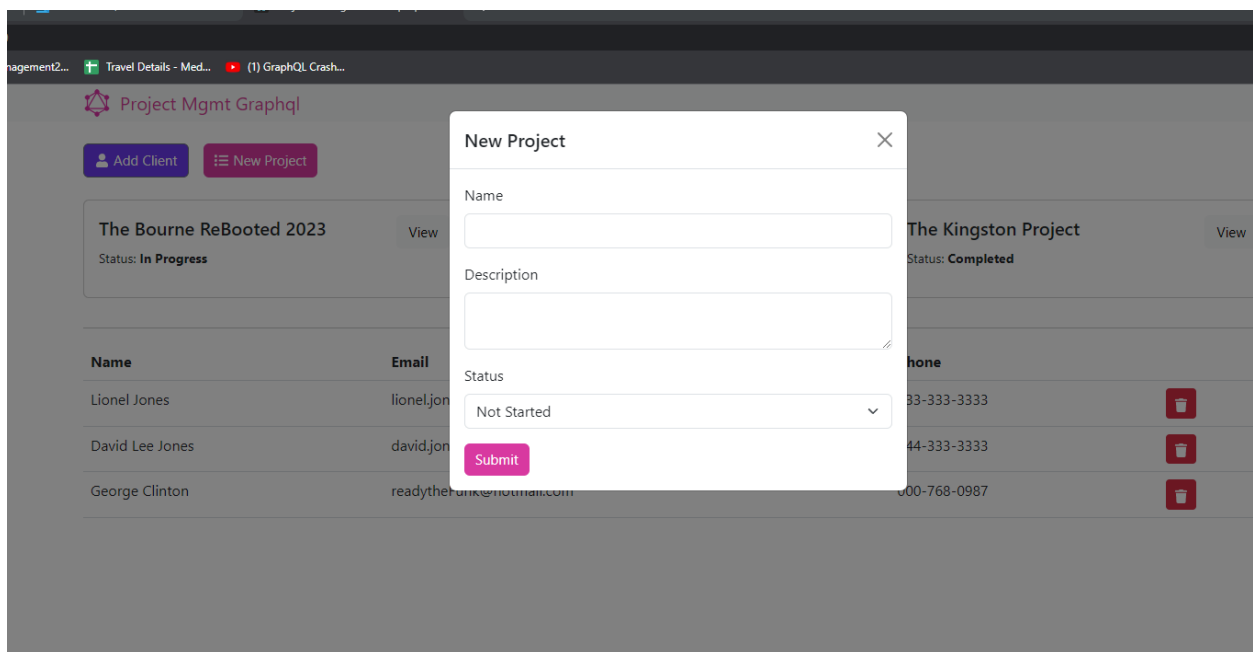
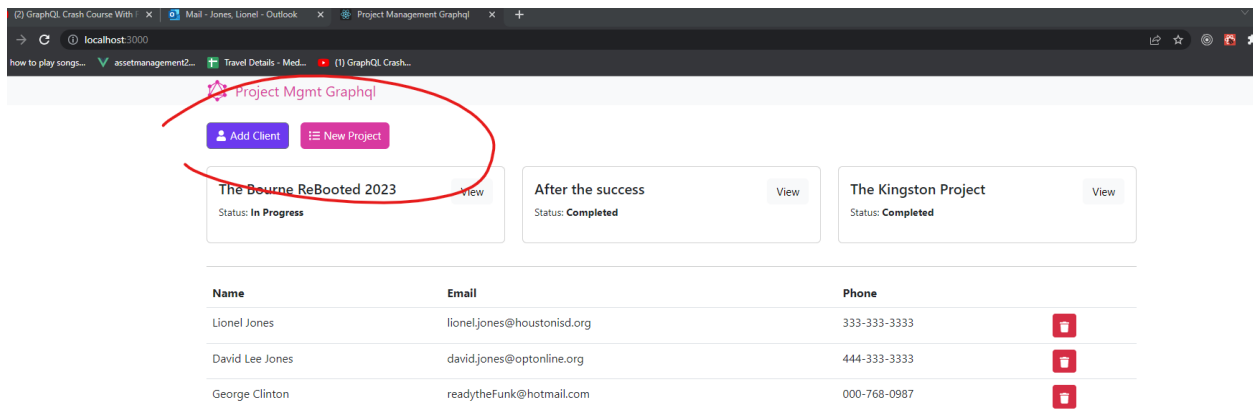
This is also clean as well, we create modals for adding data, the modals are re-usable components, we add them to our HomePage

```
client > src > pages > Home.jsx > Home
You, 2 minutes ago | 1 author (You)
1 import React from 'react';
2 import AddClientModal from '../components/AddClientModal';
3 import AddProjectModal from '../components/AddProjectModal';
4 import Clients from '../components/Clients';
5 import Projects from '../components/Projects';
6
7
8 export default function Home() {
9   return (
10     <>
11       <div className="d-flex gap-3 mb-4">
12         <AddClientModal />
13         <AddProjectModal />
14       </div>
15       <Projects />
16       <hr />
17       <Clients />
18     </>
19   )
20 }
21
```

And the modals when they are rendered, the render from left to right.

The modals render the buttons first, then when you press the button, the modals appear

```
client > src > components > AddProjectModal.jsx > AddProjectModal
33 addProject(name,description,status)
34 setName('');
35 setDescription('');
36 setStatus('');
37 }
38
39 return (
40   <>
41     <button type="button"
42       className="btn btn-primary"
43       data-bs-toggle="modal"
44       data-bs-target="#addProjectModal">
45
46       <div className="d-flex align-items-center">
47         <FaList className='icon' />
48         <div>New Project</div>
49       </div>
46
47     </button>
48
49     <div className="modal fade" id="addProjectModal" tabIndex="-1" aria-label="modal">
50       <div className="modal-dialog">
51         <div className="modal-content">
52           <div className="modal-header">
53             <h1 className="modal-title fs-5" id="addProjectModal">New Project
54             <button type="button" className="btn-close" data-bs-dismiss="modal" />
55           </div>
56           <div className="modal-body">
57             <form onSubmit={onSubmit}>
58
59           </div>
60         </div>
61       </div>
62     </div>
63   </>
64 )
```

Tip: 400 Bad Request with graphql

If you get a 400 bad request, remember on the mutation, you have to make sure that all of the fields are “exactly” matching

```
1  import { gql } from '@apollo/client';
2
3
4  const ADD_PROJECT = gql`
5    mutation AddProject(
6      $name: String!
7      $description: String!
8      $status: ProjectStatus!
9      $clientId: ID!
10   ) {
11     addProject(
12       name: $name
13       description: $description
14       status: $status
15       clientId: $clientId
16     ) {
17       id
18       name
19       description
20       status
21       client {
22         id
23         name
24         email
25         phone
26       }
27     }
28   }
29 `;
30
31
32
```

I made the mistake of having the field spelled:

Wrong: clientID

Correct: clientId

```
//Add a project
addProject: {
  type: ProjectType,
  args: {
    name: { type: GraphQLNonNull(GraphQLString) },
    description: { type: GraphQLNonNull(GraphQLString) },
    status: {
      type: new GraphQLEnumType(
        {
          name: 'ProjectStatus',
          values: {
            'new': { value: 'Not Started' },
            'progress': { value: 'In Progress' },
            'completed': { value: 'Completed' },
          }
        }
      ),
      defaultValue: 'Not Started'
    }, //status
    clientId: { type: GraphQLNonNull(GraphQLID) },
  }, //args
  resolve(parent, args) {
    const project = new Project({
      name: args.name,
      description: args.description,
      status: args.status,
      clientId: args.clientId,
    });

    return project.save();
  } //resolve
}, //addProject
```

You, 22 hours ago • added code for viewing cl