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)

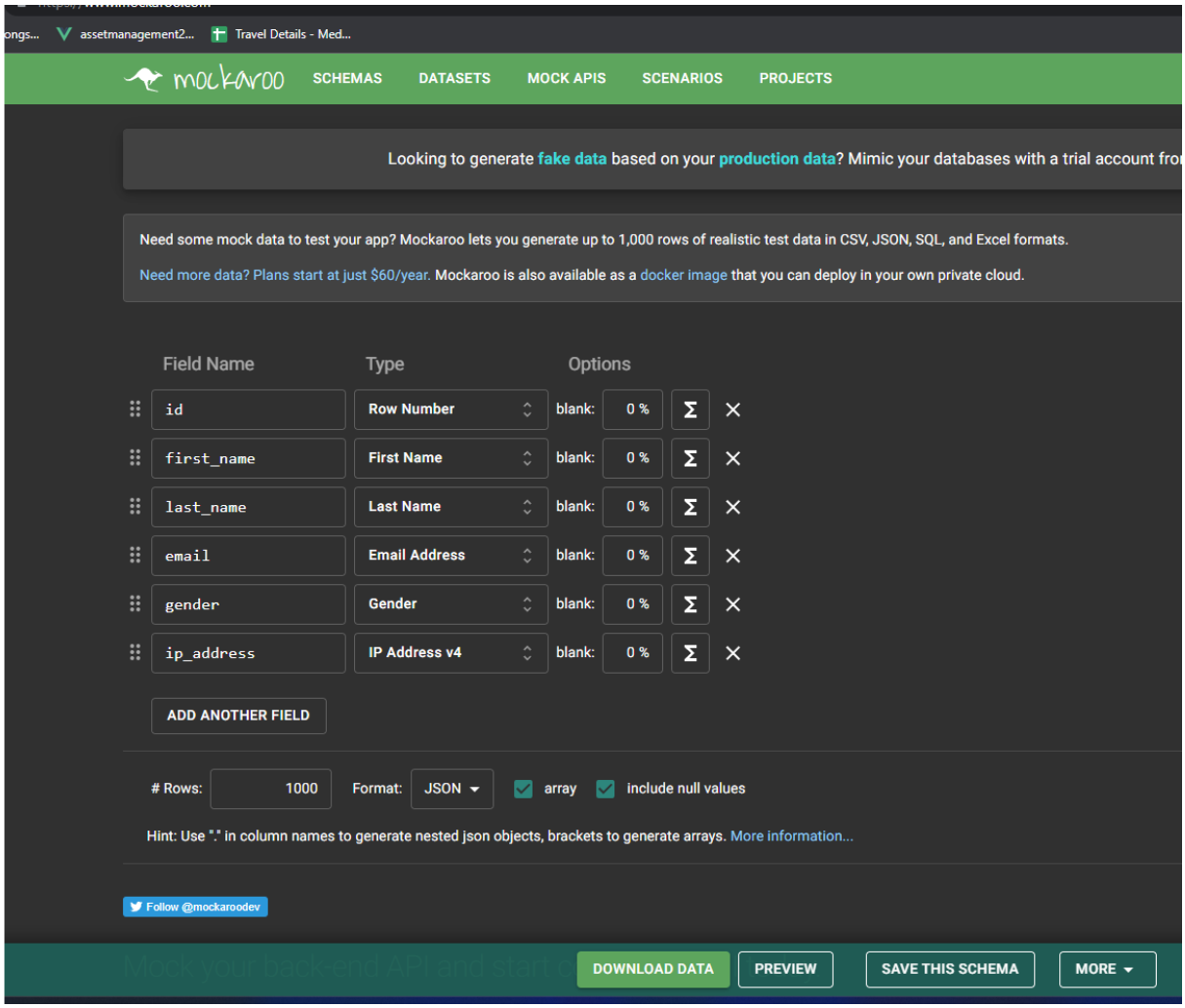
In traditional REST, your endpoints would look like:

/users
/travels
/books
etc…

In Graphql you only have "**one**" endpoint

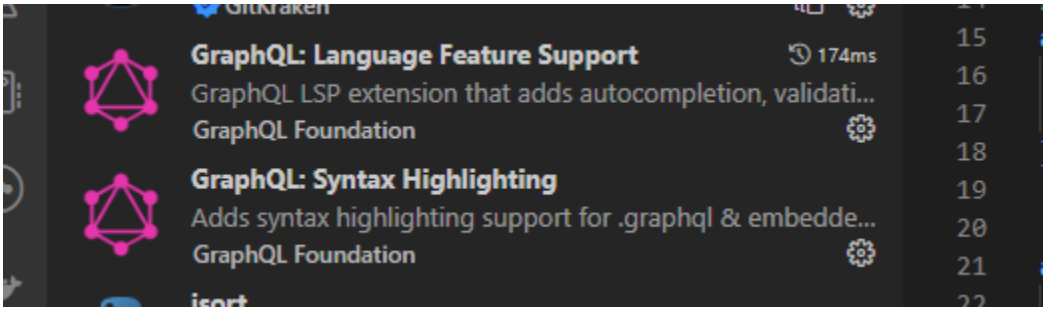
/graphql

Sample Data
To get some fake data to work with, you can go to a website called
https://www.mockaroo.com/



I changed the type to JSON

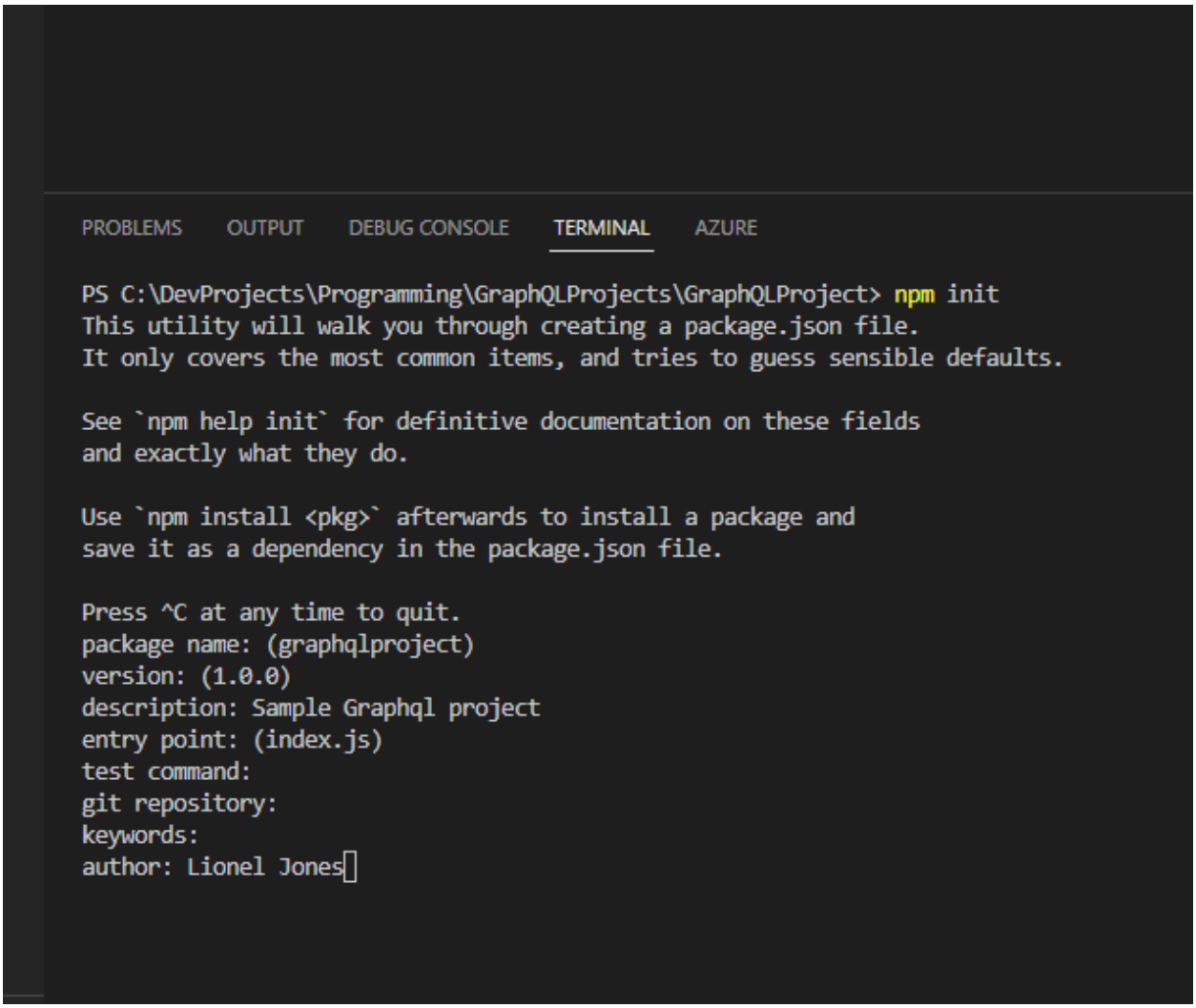
Add the VSCode Extensions to your VSCODE



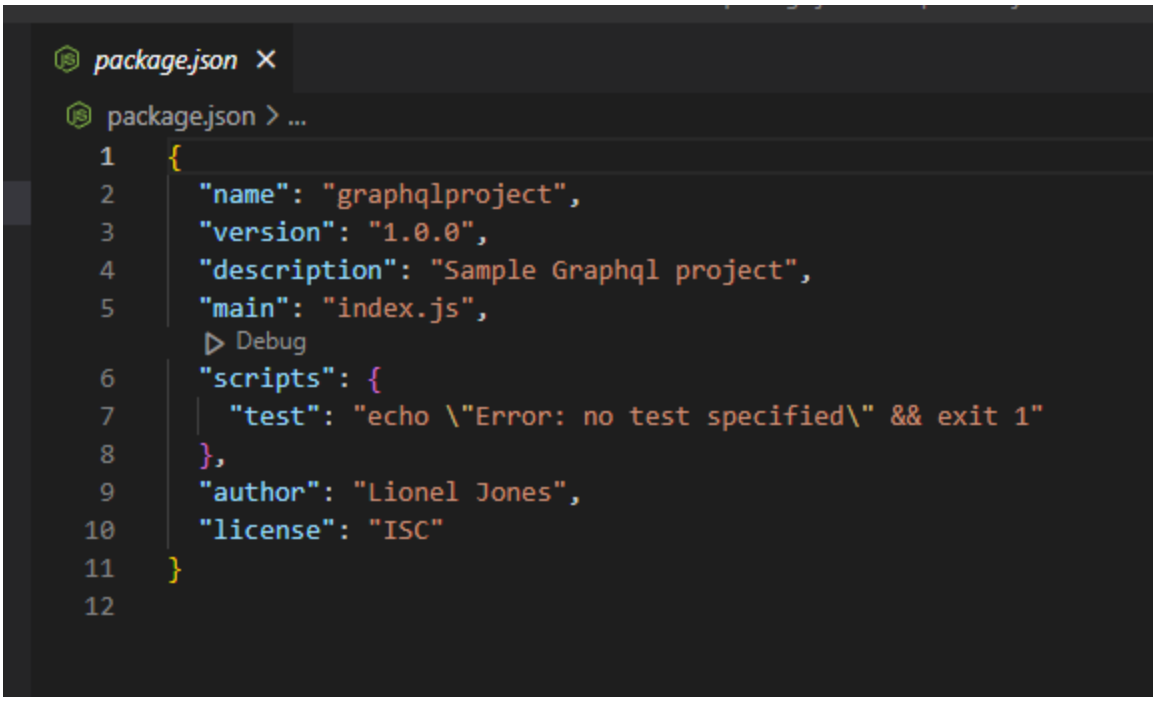
Tip: Creating a sample nodeJS Project Using VSCode

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

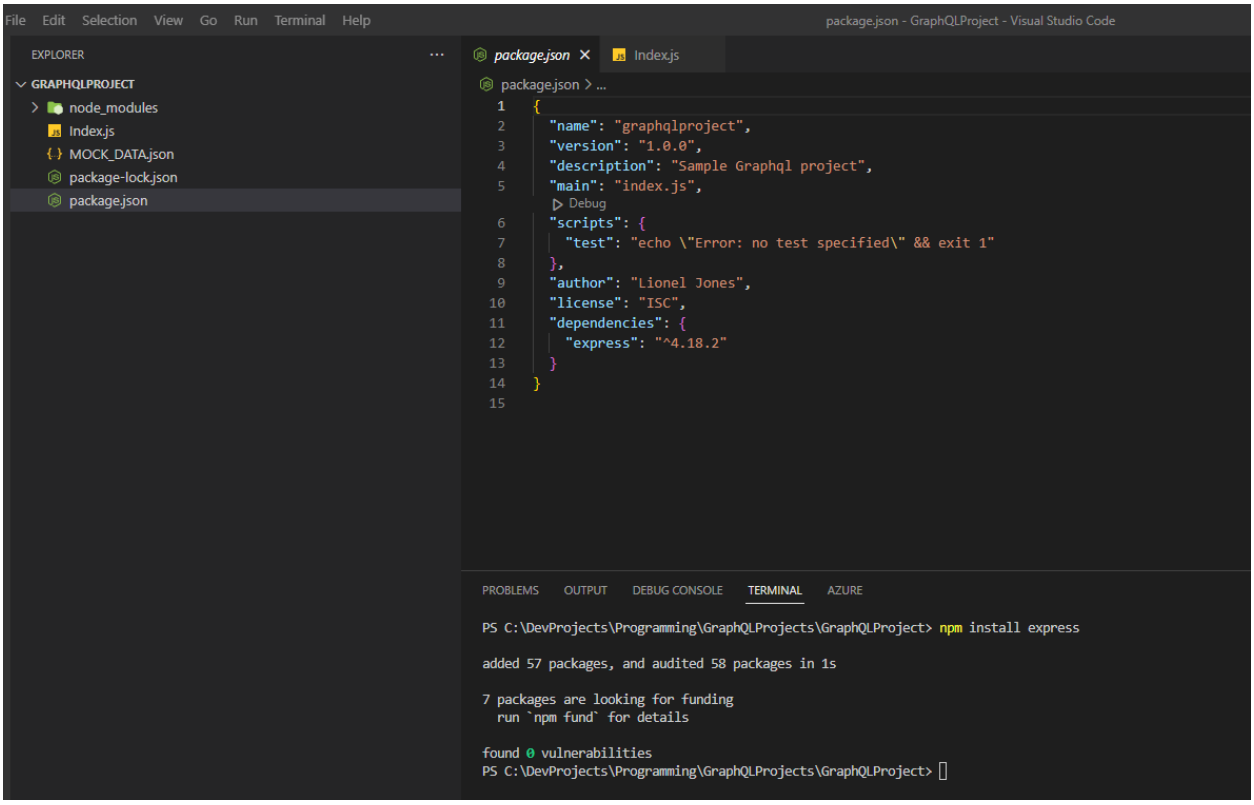
I just follow the prompts, enter in generic information



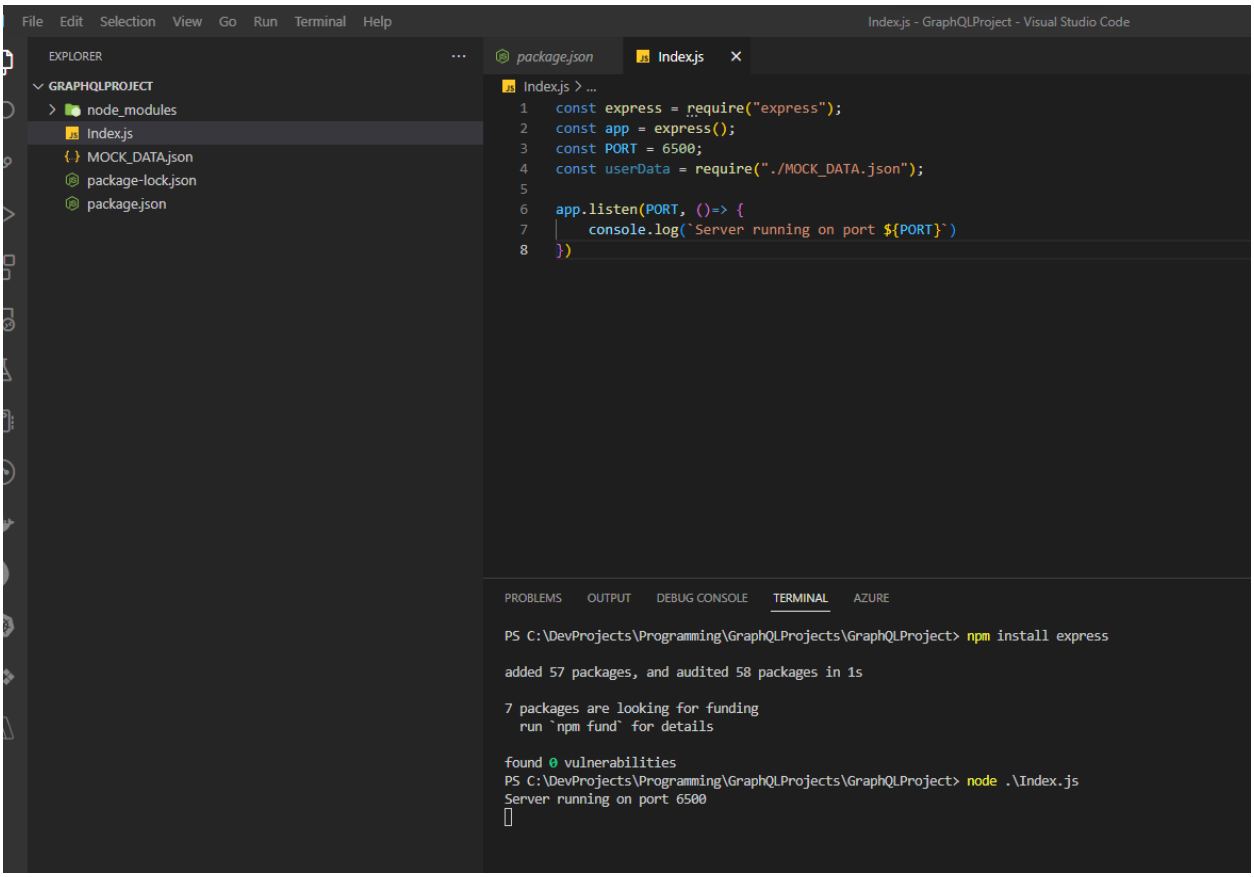
This will create my package.json file for my dependencies



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

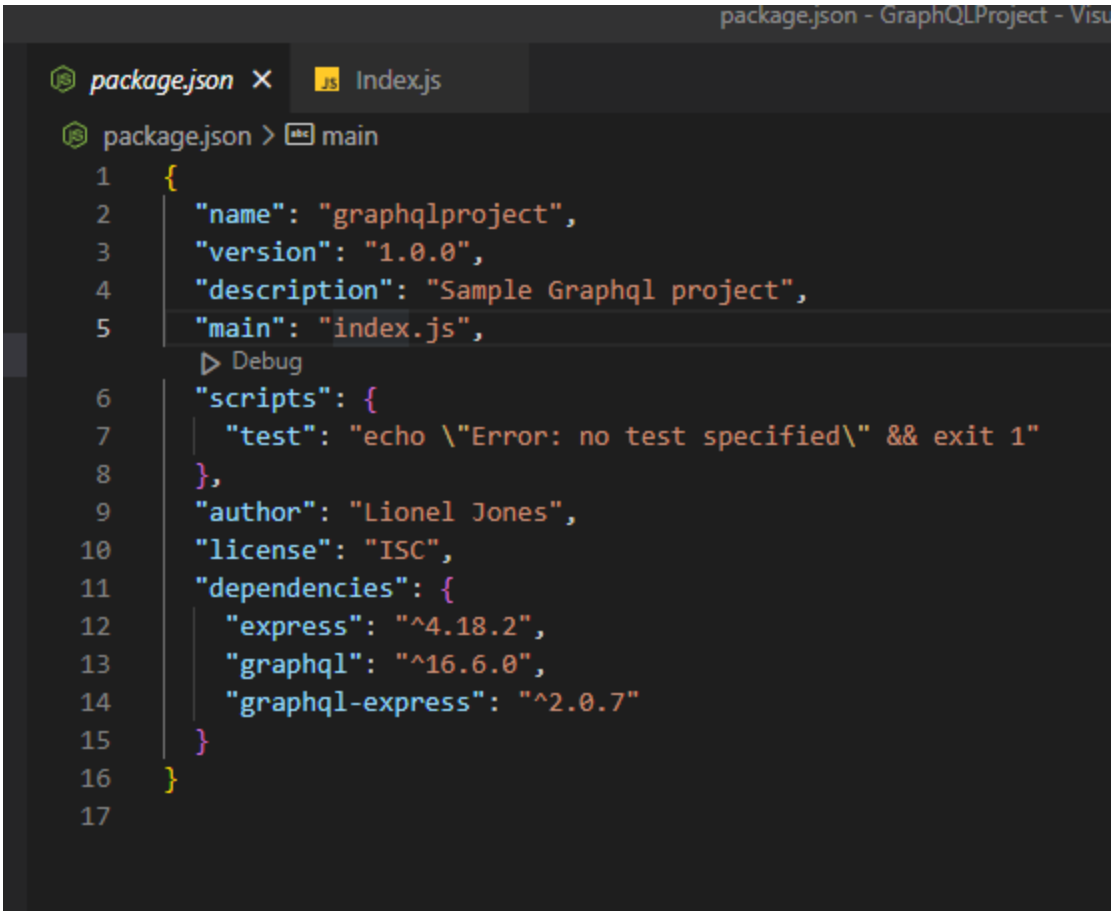


Next I create an index.js file and add the following lines of code to it



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

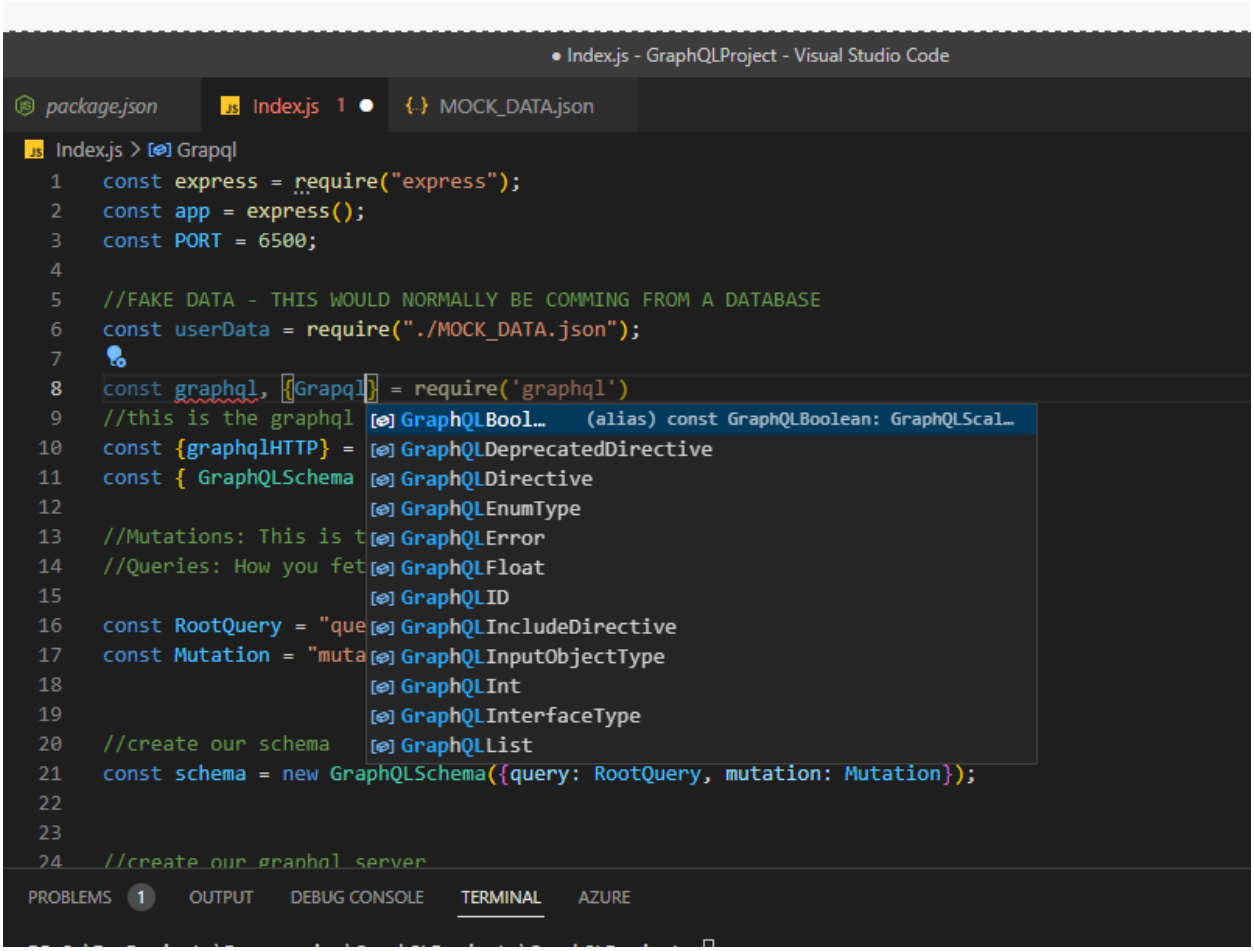


As you can see, as you install packages, it updates your package.json file

Graphql Concepts

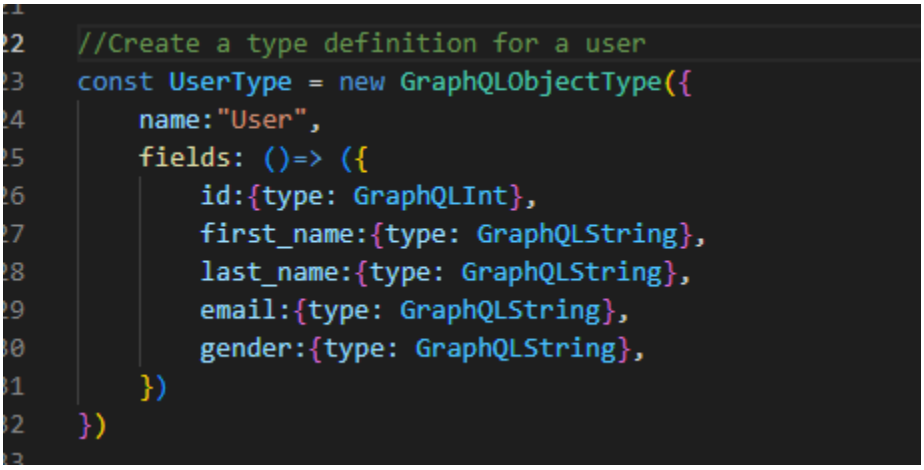
Mutations: This is the same as CRUD (Create, Read,Update, Delete)
Queries: How you fetch the data you need

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



Creating the type(s):

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



Creating the querie(s)

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

```
//CREATE QUERIE(S)
//Remember GraphQL only has "one" entpoint, 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

          }
      }
  });
```

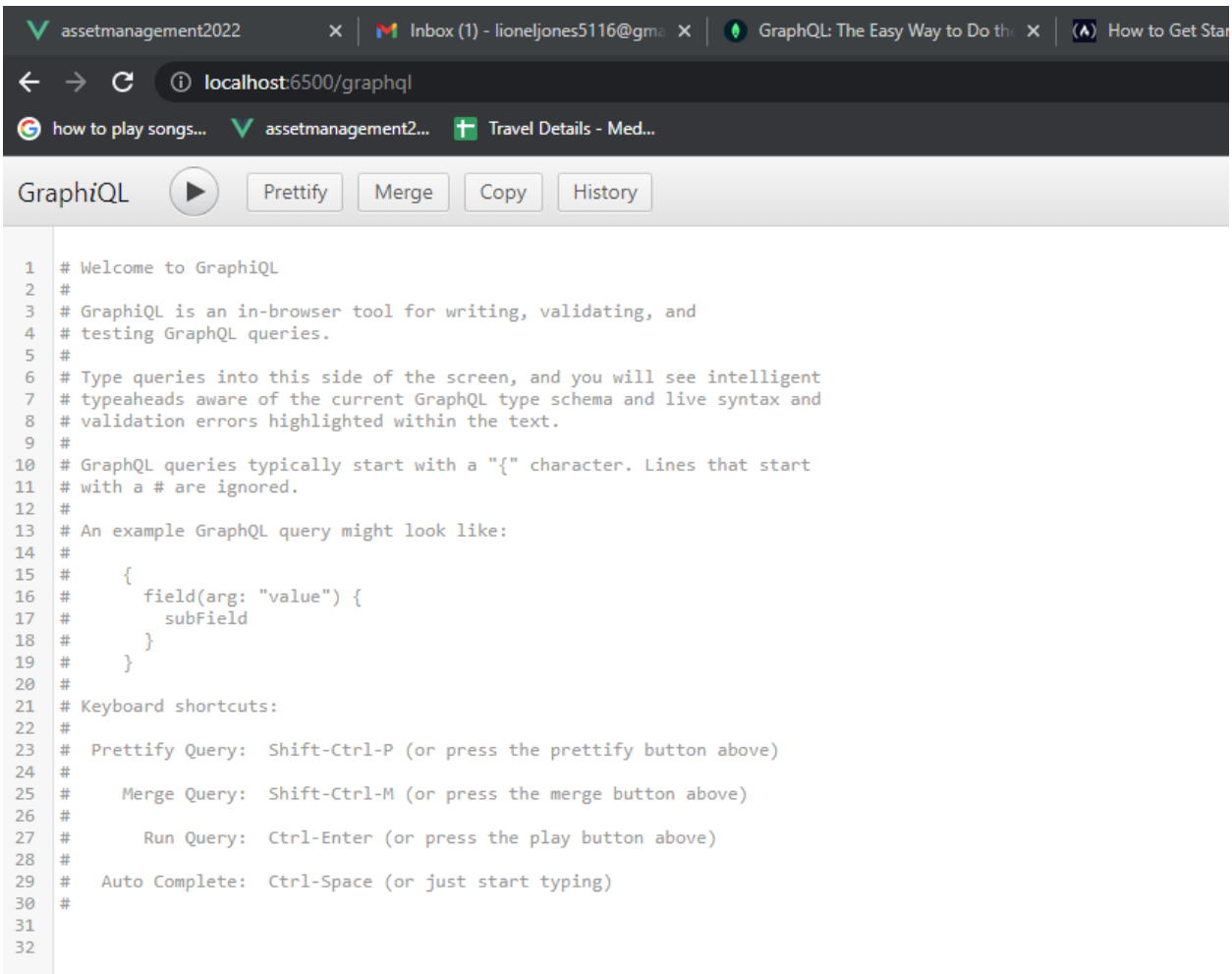
Next, let's fire up our graphql server
Enter the command at the terminal

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

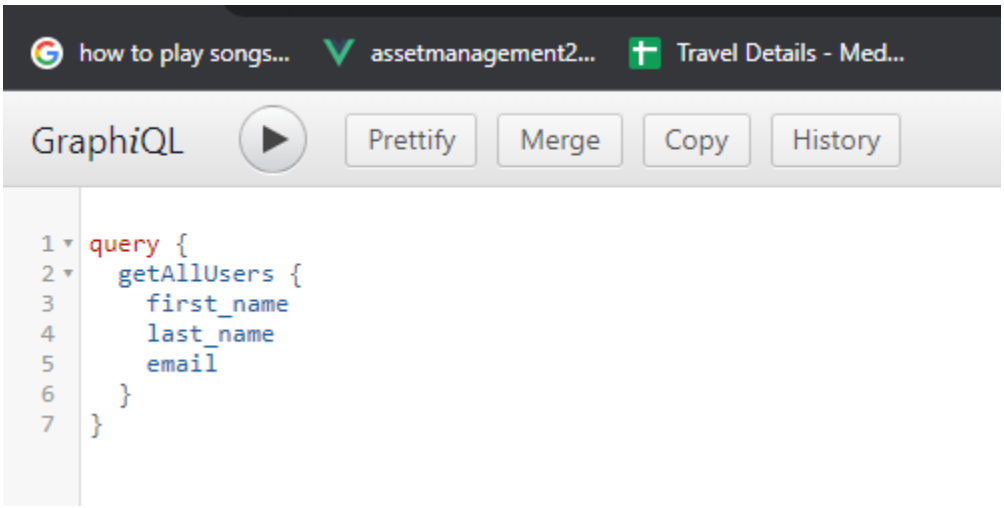
This is surfaced up via this entry:

```
23
24  //create our graphql server
25  //Remember GraphQL only has "one" entpoint, below is where you create your queries
26  app.use('/graphql',graphqlHTTP( {
27    schema,
28    graphiql:true      You, 21 hours ago • first commit …
29  }));
30
```

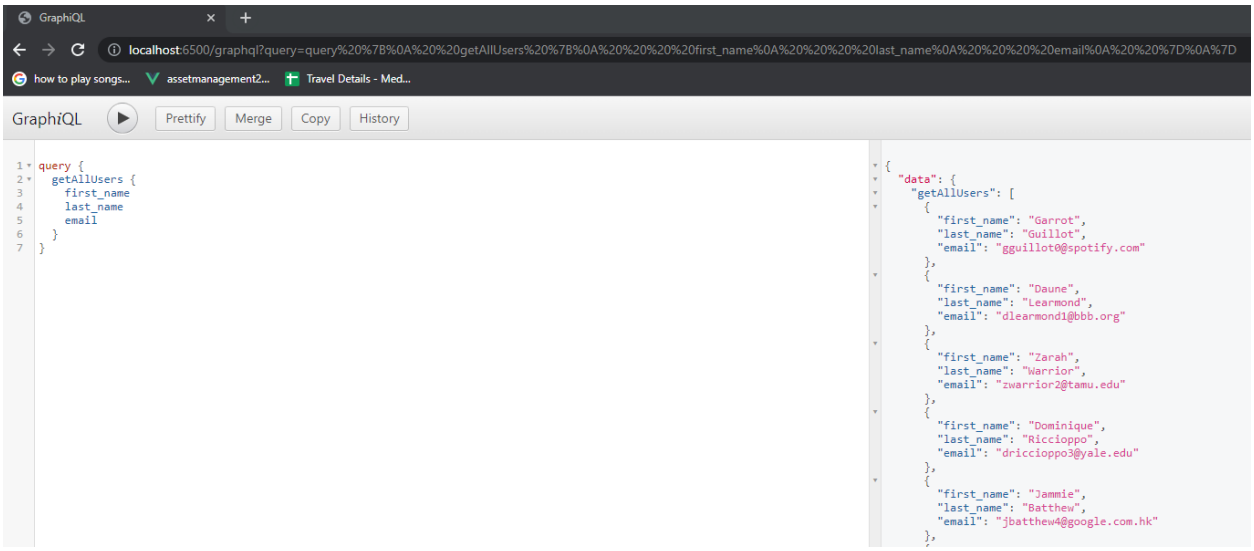
This is like using a API test like POSTMAN or any other API tester

Tip: Querying the data

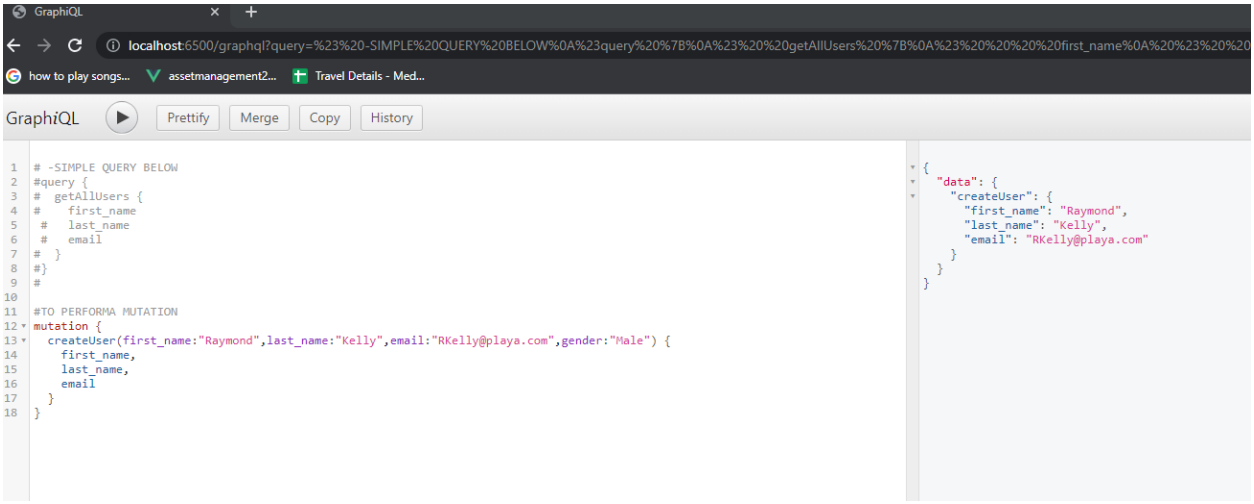
To fetch some data, just type below



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



To perform a mutation:



Run the query:

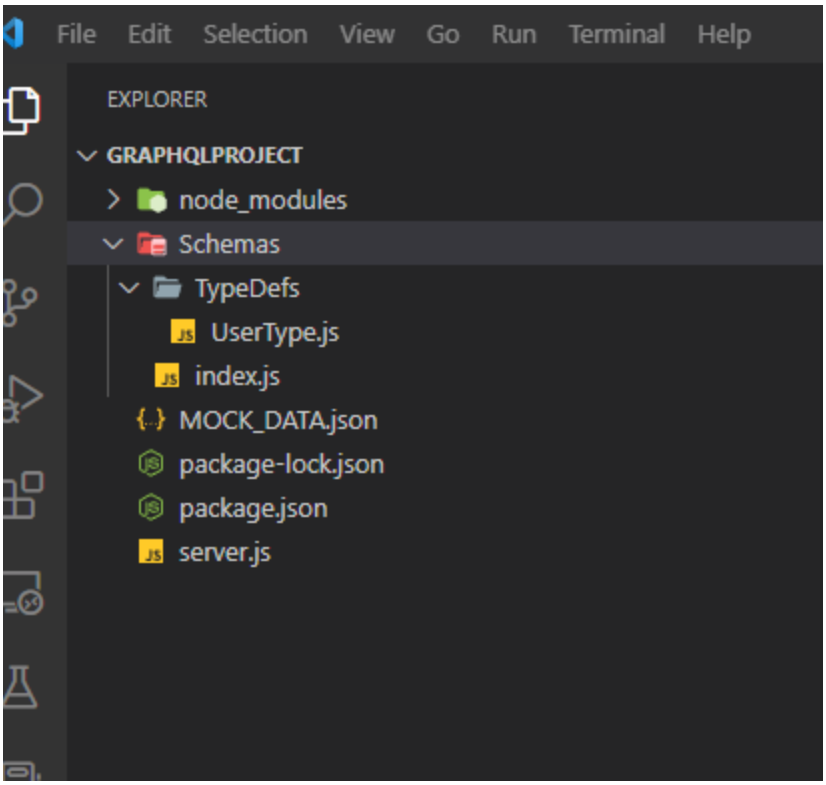
```
query {
 getAllUsers {
  id
  first_name
  last_name
  email
 }
}
```

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:



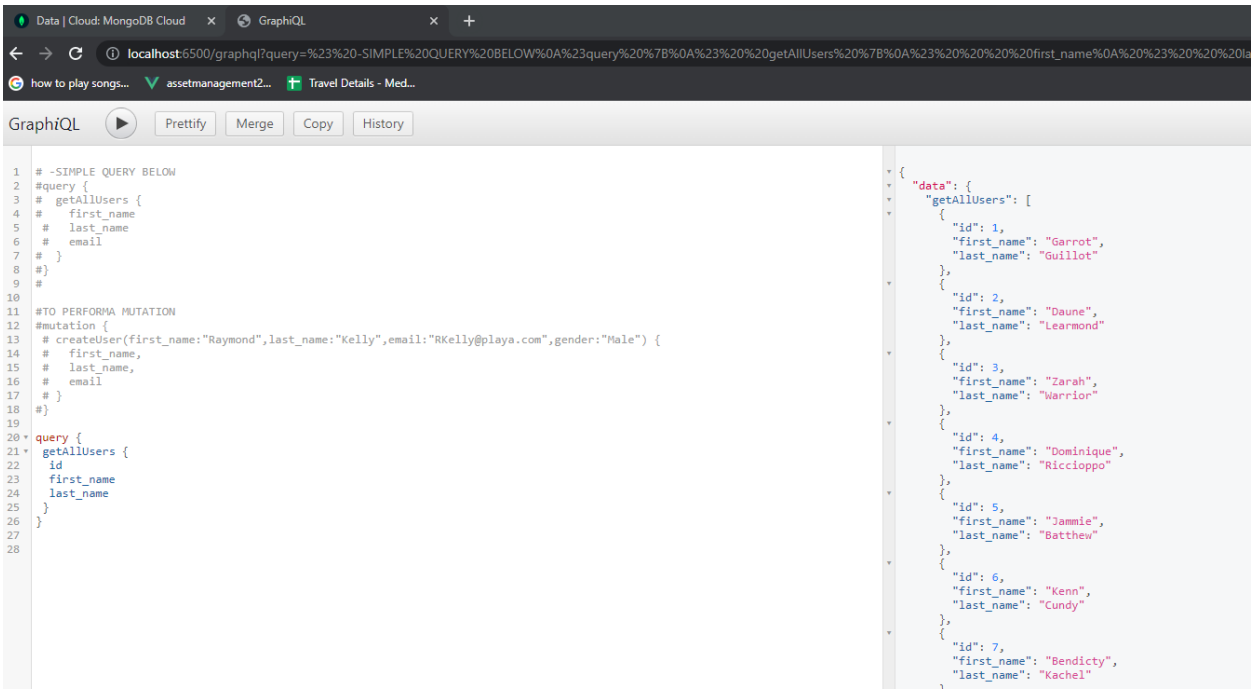
Rename index.js on our root to “Server.js”

We place our types inside of the typdefs folder
We place our mutations and queries in the index.js file under the schema folder

Then we execute
node server.js

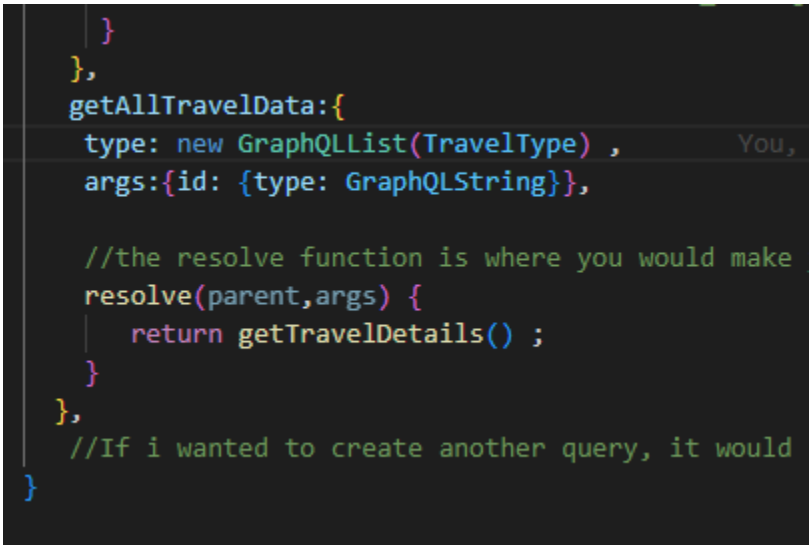
```
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> node .\server.js
Server running on port 6500
```

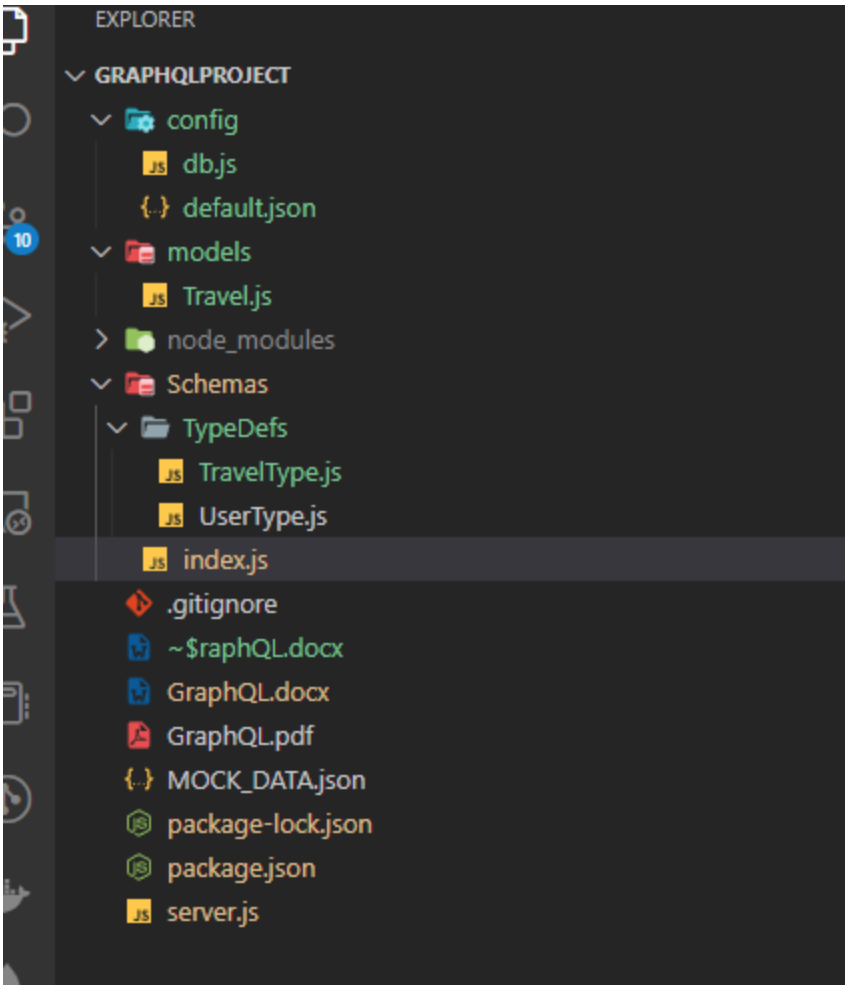
And everything works:

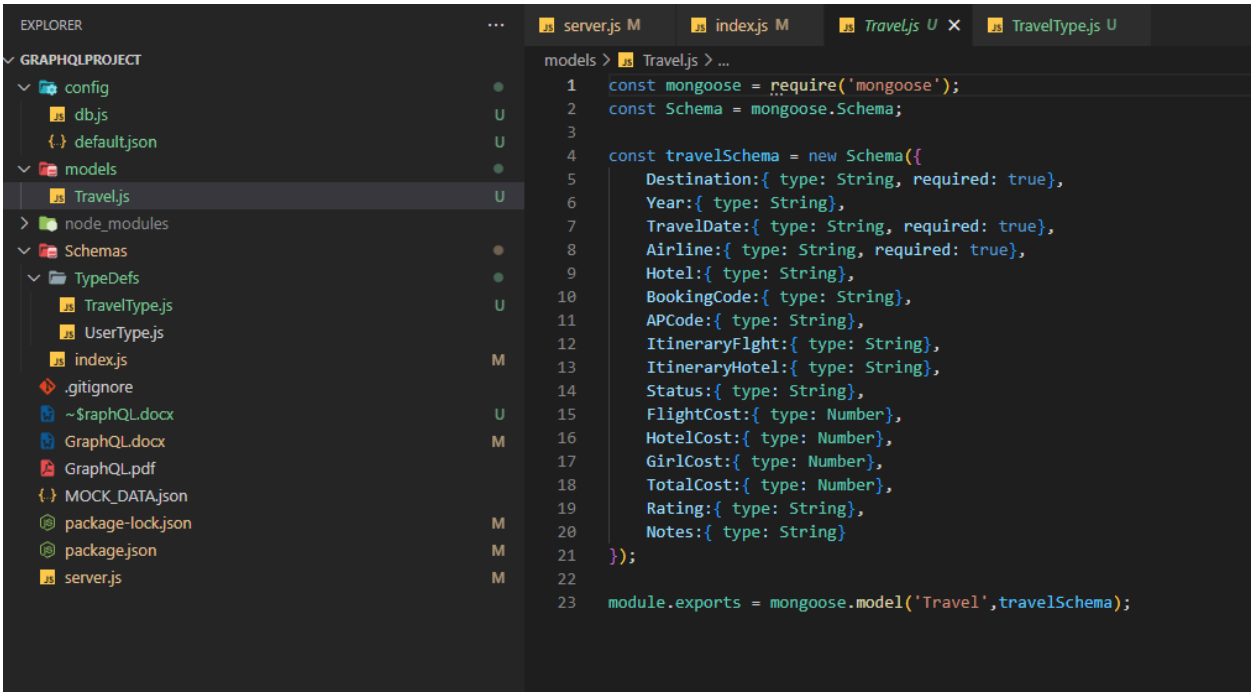


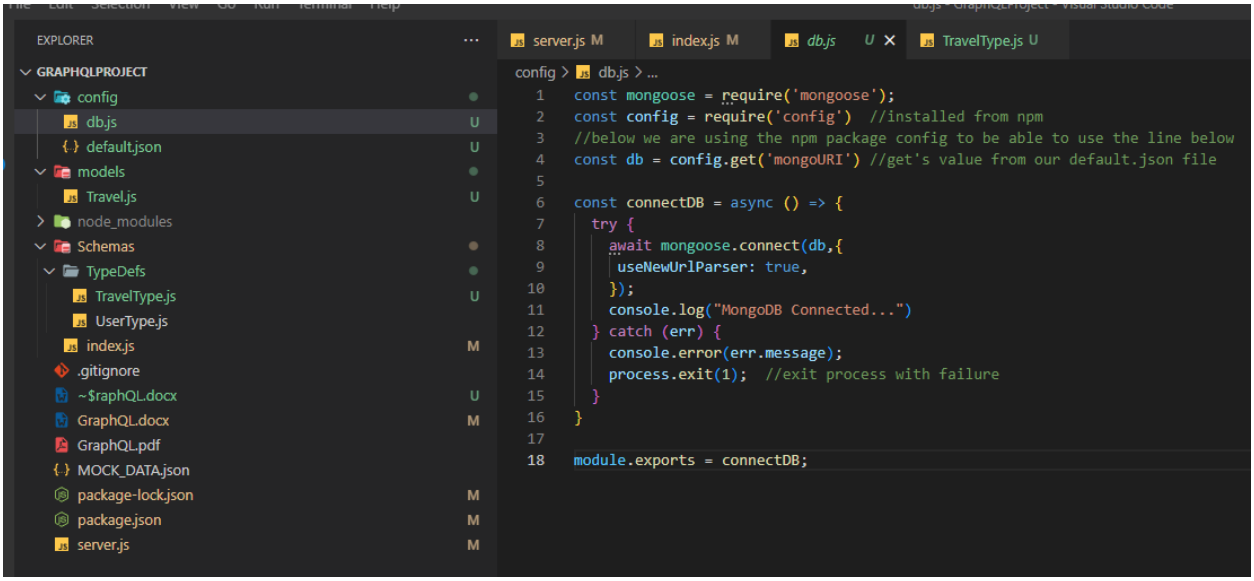
Tip: Fetching MongoDB Data with graphql

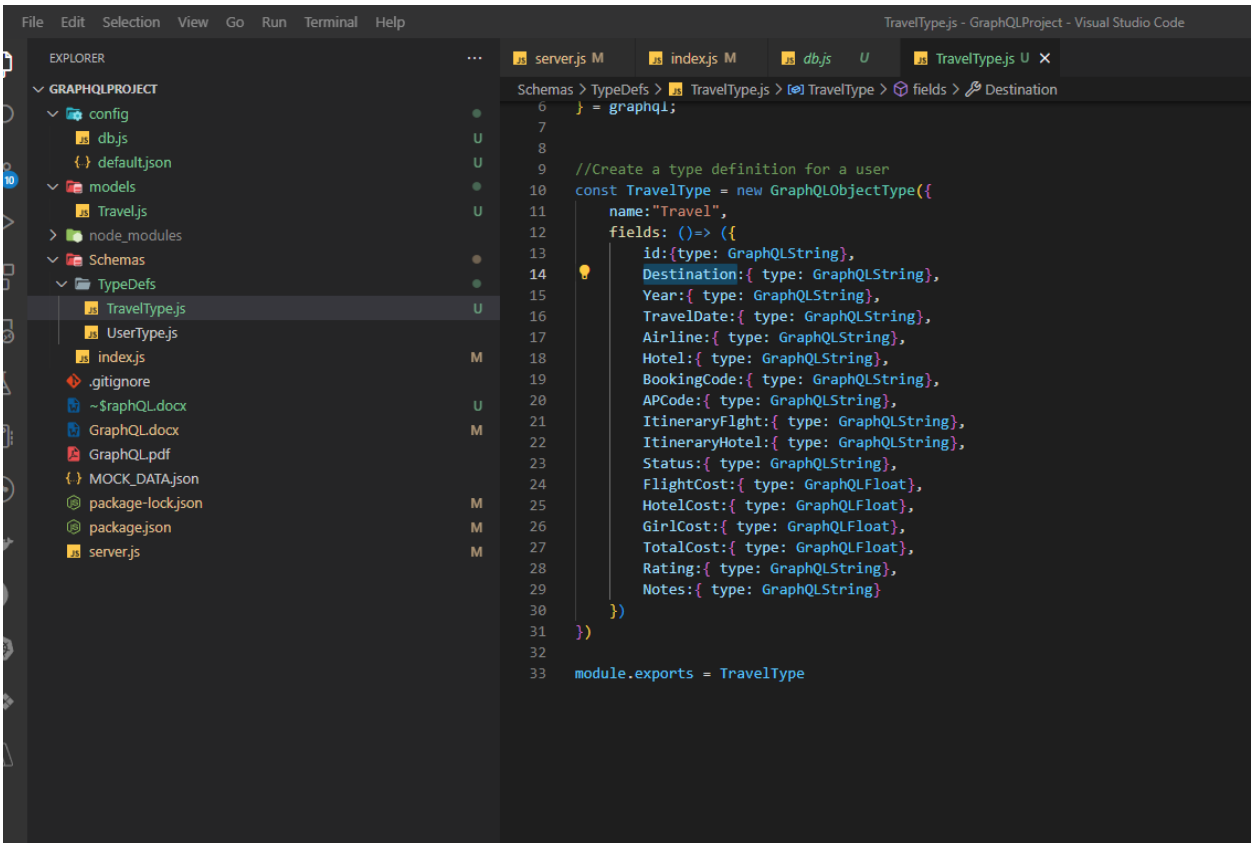
It was super easy, I just leveraged my code from my assetmgmt service using mongo express, connected to mongoDB, created my models and schema for graphql, then just did a mongodb call to grab travel records. The resolve function in your RootQuery method where you create your queries takes the data argument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

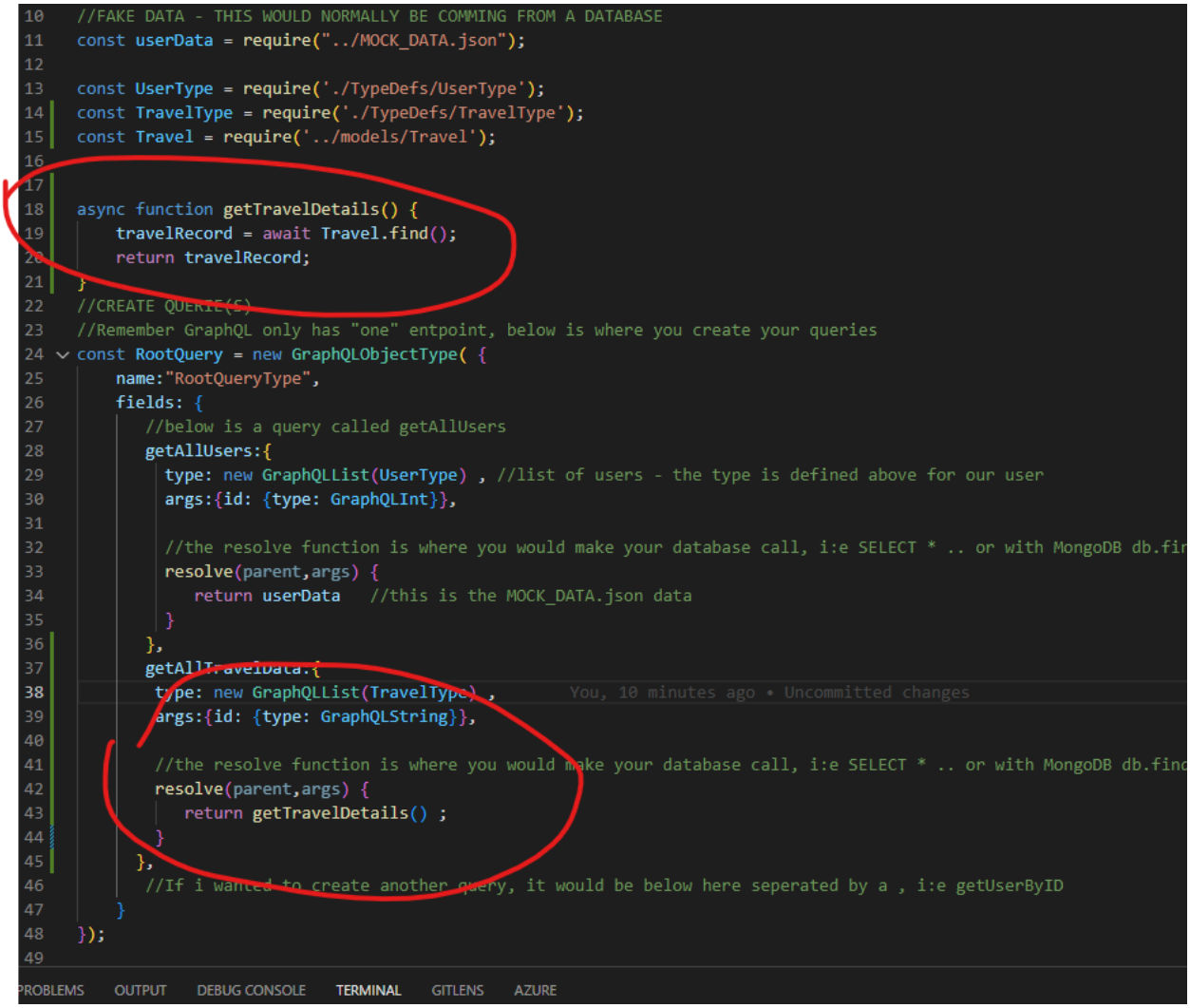


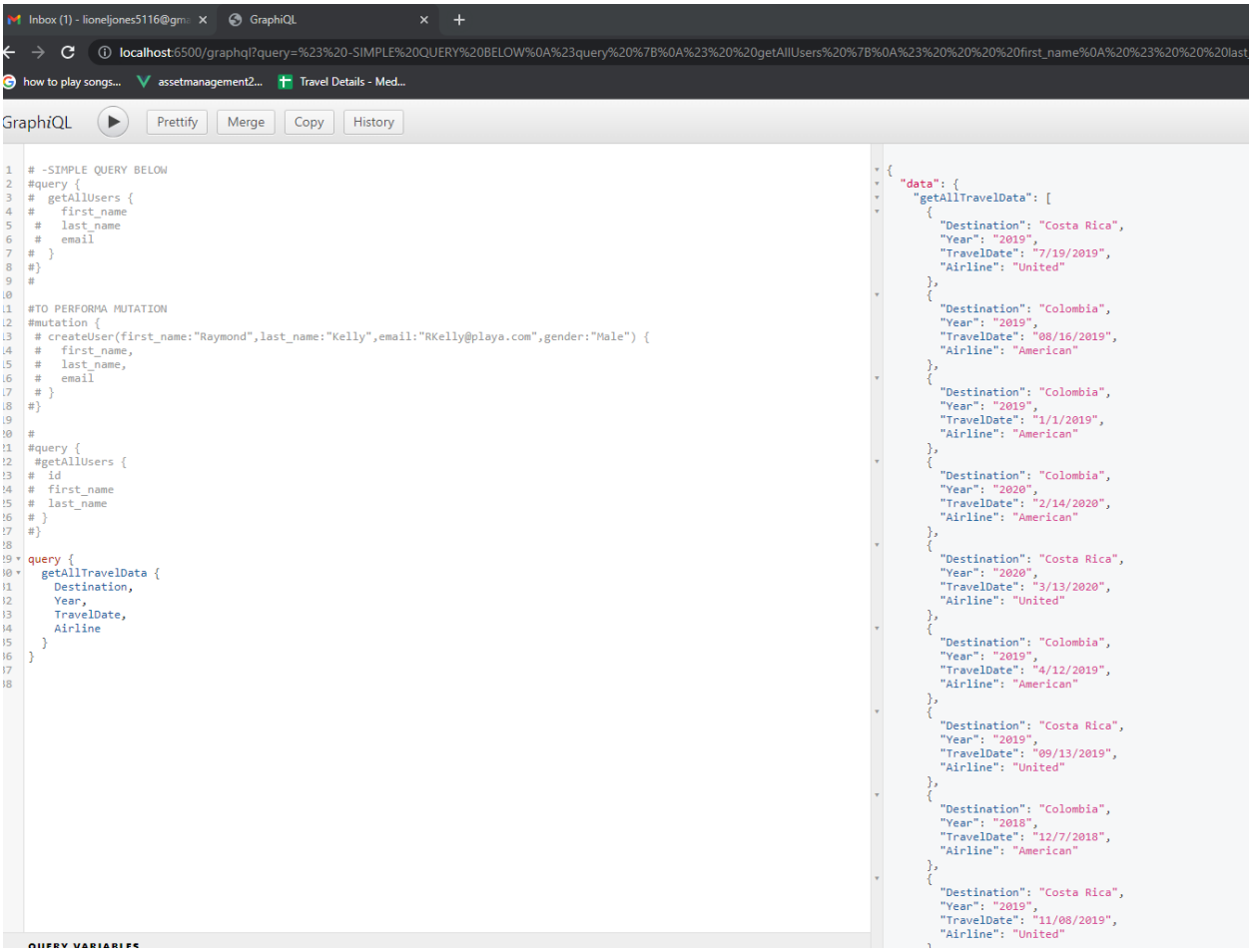

EXPLORER
GRAPHQLPROJECT
config
db.js
default.json
models
Travel.js
node_modules
Schemas
TypeDefs
TravelType.js
UserType.js
index.js
.gitignore
~$raphQL.docx
GraphQL.docx
GraphQL.pdf
MOCK_DATA.json
package-lock.json
package.json
server.js


EXPLORER
GRAPHQLPROJECT
config
db.js U
default.json U
models
Travel.js U
node_modules
Schemas
TypeDefs
TravelType.js U
UserType.js
index.js M
.gitignore
~$raphQL.docx U
GraphQL.docx M
GraphQL.pdf
MOCK_DATA.json
package-lock.json M
package.json M
server.js M
server.js M
index.js M
Travel.js U
TravelType.js U
models > Travel.js > ...
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const travelSchema = new Schema({
Destination:{ type: String, required: true},
Year:{ type: String},
TravelDate:{ type: String, required: true},
Airline:{ type: String, required: true},
Hotel:{ type: String},
BookingCode:{ type: String},
APCode:{ type: String},
ItineraryFlght:{ type: String},
ItineraryHotel:{ type: String},
Status:{ type: String},
FlightCost:{ type: Number},
HotelCost:{ type: Number},
GirlCost:{ type: Number},
TotalCost:{ type: Number},
Rating:{ type: String},
Notes:{ type: String}
});
module.exports = mongoose.model('Travel',travelSchema);


EXPLORER
GRAPHQLPROJECT
config
db.js
default.json
models
Travel.js
node_modules
Schemas
TypeDefs
TravelType.js
UserType.js
index.js
.gitignore
~$raphQL.docx
GraphQL.docx
GraphQL.pdf
MOCK_DATA.json
package-lock.json
package.json
server.js
server.js M
index.js M
db.js U
TravelType.js U
config > db.js > ...
const mongoose = require('mongoose');
const config = require('config') //installed from npm
//below we are using the npm package config to be able to use the line below
const db = config.get('mongoURI') //get's value from our default.json file
const connectDB = async () => {
try {
await mongoose.connect(db,{
useNewUrlParser: true,
});
console.log("MongoDB Connected...")
} catch (err) {
console.error(err.message);
process.exit(1); //exit process with failure
}
}
module.exports = connectDB;


File Edit Selection View Go Run Terminal Help
TravelType.js - GraphQLProject - Visual Studio Code
EXPLORER
GRAPHQLPROJECT
config
db.js
default.json
models
Travel.js
node_modules
Schemas
TypeDefs
TravelType.js
UserType.js
index.js
.gitignore
~$raphQL.docx
GraphQL.docx
GraphQL.pdf
MOCK_DATA.json
package-lock.json
package.json
server.js
server.js M
index.js M
db.js U
TravelType.js U
Schemas > TypeDefs > TravelType.js > TravelType > fields > Destination
} = graphql;
//Create a type definition for a user
const TravelType = new GraphQLObjectType({
name:"Travel",
fields: ()=> ({
id:{type: GraphQLString},
Destination:{ type: GraphQLString},
Year:{ type: GraphQLString},
TravelDate:{ type: GraphQLString},
Airline:{ type: GraphQLString},
Hotel:{ type: GraphQLString},
BookingCode:{ type: GraphQLString},
APCode:{ type: GraphQLString},
ItineraryFlght:{ type: GraphQLString},
ItineraryHotel:{ type: GraphQLString},
Status:{ type: GraphQLString},
FlightCost:{ type: GraphQLFloat},
HotelCost:{ type: GraphQLFloat},
GirlCost:{ type: GraphQLFloat},
TotalCost:{ type: GraphQLFloat},
Rating:{ type: GraphQLString},
Notes:{ type: GraphQLString}
})
})
module.exports = TravelType


//FAKE DATA - THIS WOULD NORMALLY BE COMMING FROM A DATABASE
const userData = require("../MOCK_DATA.json");
const UserType = require('./TypeDefs/UserType');
const TravelType = require('./TypeDefs/TravelType');
const Travel = require('../models/Travel');
async function getTravelDetails() {
travelRecord = await Travel.find();
return travelRecord;
}
//CREATE QUERIE(S)
//Remember GraphQL only has "one" entpoint, 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.fin
resolve(parent,args) {
return userData //this is the MOCK_DATA.json data
}
},
getAllTravelData:{
type: new GraphQLList(TravelType) ,
You, 10 minutes ago • Uncommitted changes
args:{id: {type: GraphQLString}},
//the resolve function is where you would make your database call, i:e SELECT * .. or with MongoDB db.find
resolve(parent,args) {
return getTravelDetails() ;
}
},
//If i wanted to create another query, it would be below here seperated by a , i:e getUserByID
}
});
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
GITLENS
AZURE


Inbox (1) - lioneljones5116@gm
GraphiQL
localhost:6500/graphql?query=%23%20-SIMPLE%20QUERY%20BELOW%0A%23query%20%7B%0A%23%20%20getAllUsers%20%7B%0A%23%20%20%20%20first_name%0A%20%20%23%20%20%20last
how to play songs...
assetmanagement2...
Travel Details - Med...
GraphiQL
Prettify
Merge
Copy
History
-SIMPLE QUERY BELOW
#query {
getAllUsers {
first_name
last_name
email
}
#}
#
#TO PERFORMA MUTATION
#mutation {
createUser(first_name:"Raymond",last_name:"Kelly",email:"RKelly@playa.com",gender:"Male") {
first_name,
last_name,
email
}
#}
#
#query {
#getAllUsers {
id
first_name
last_name
}
#}
query {
getAllTravelData {
Destination,
Year,
TravelDate,
Airline
}
}
{
"data": {
"getAllTravelData": [
{
"Destination": "Costa Rica",
"Year": "2019",
"TravelDate": "7/19/2019",
"Airline": "United"
},
{
"Destination": "Colombia",
"Year": "2019",
"TravelDate": "08/16/2019",
"Airline": "American"
},
{
"Destination": "Colombia",
"Year": "2019",
"TravelDate": "1/1/2019",
"Airline": "American"
},
{
"Destination": "Colombia",
"Year": "2020",
"TravelDate": "2/14/2020",
"Airline": "American"
},
{
"Destination": "Costa Rica",
"Year": "2020",
"TravelDate": "3/13/2020",
"Airline": "United"
},
{
"Destination": "Colombia",
"Year": "2019",
"TravelDate": "4/12/2019",
"Airline": "American"
},
{
"Destination": "Costa Rica",
"Year": "2019",
"TravelDate": "09/13/2019",
"Airline": "United"
},
{
"Destination": "Colombia",
"Year": "2018",
"TravelDate": "12/7/2018",
"Airline": "American"
},
{
"Destination": "Costa Rica",
"Year": "2019",
"TravelDate": "11/08/2019",
"Airline": "United"

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 atalas

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

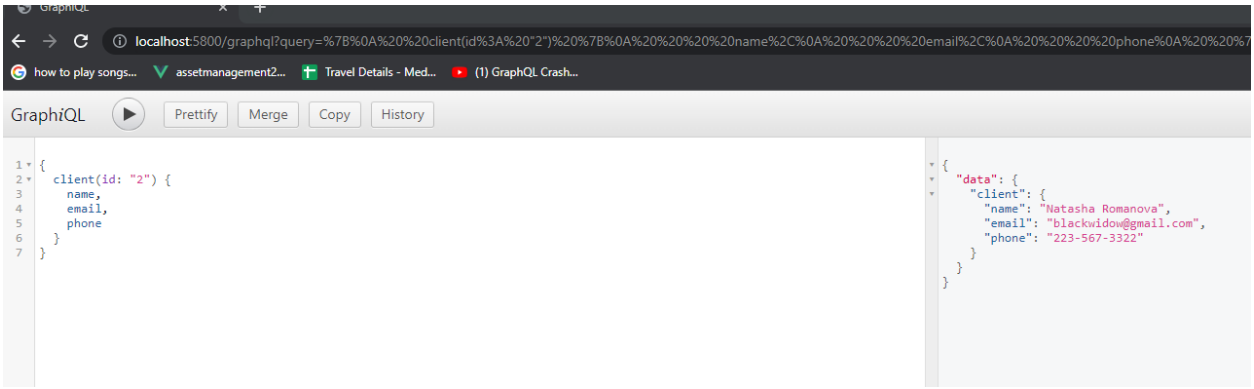
Install the packages below
npm i express express-graphql graphql mongoose cors colors –force

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

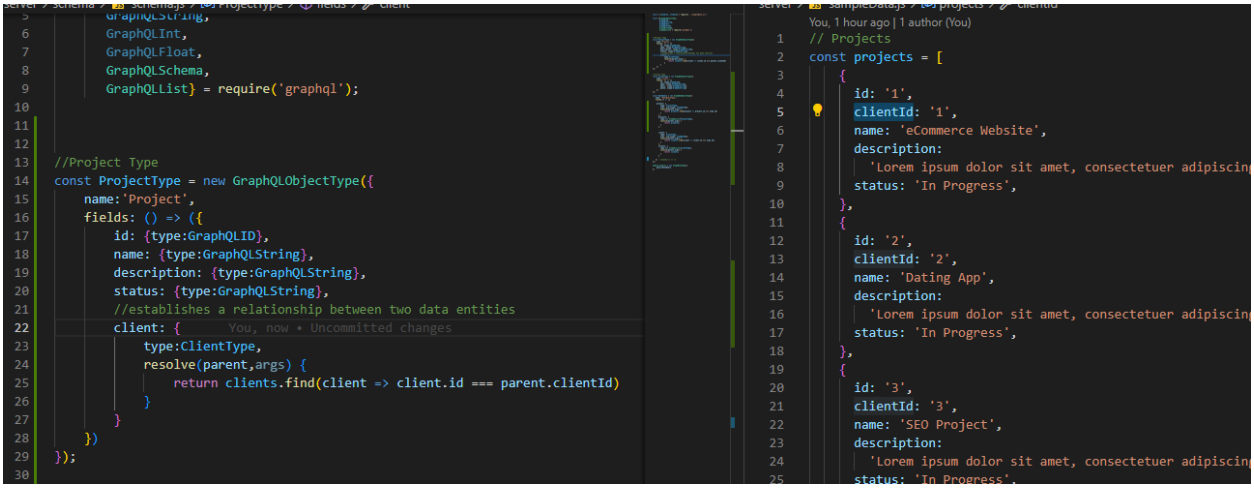
We make a call after we have wired up our code:

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',
```

Creating a relationship between two entities:



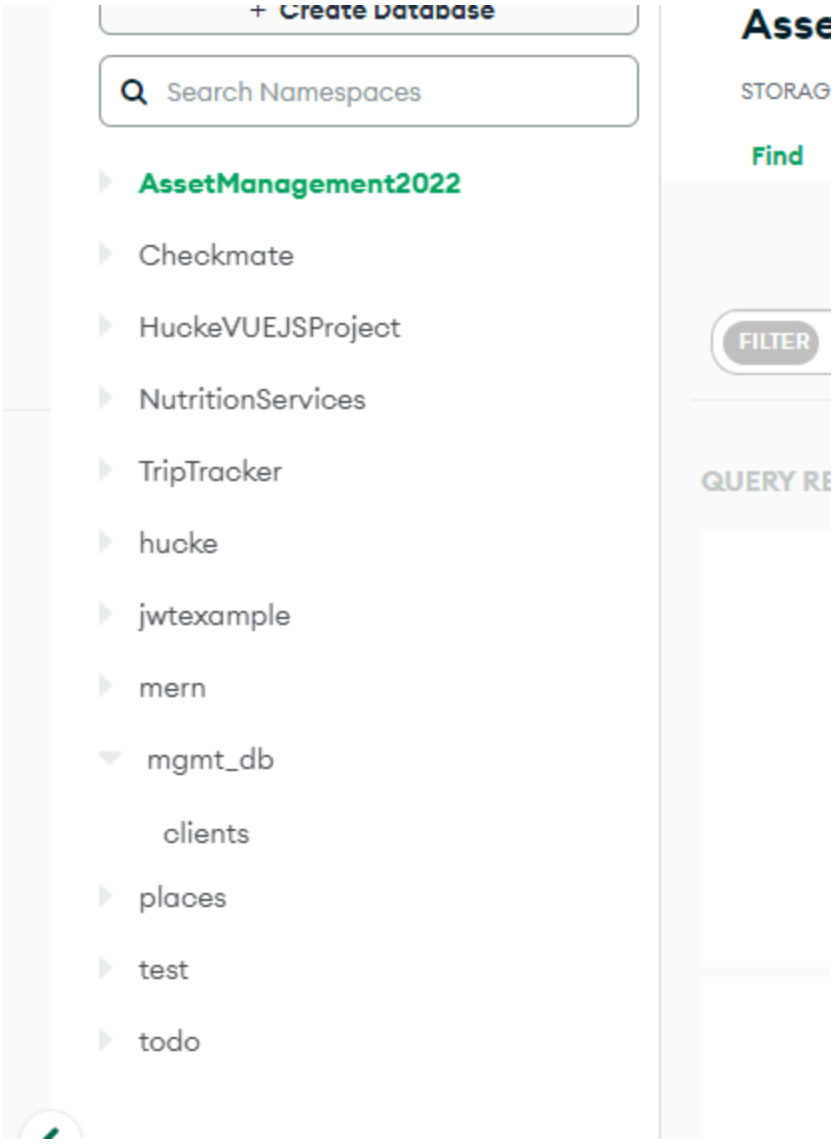
```
GraphQLString,
GraphQLInt,
GraphQLFloat,
GraphQLSchema,
GraphQLList} = require('graphql');


//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, consectetuer adipiscin
    status: 'In Progress',
  },
  {
    id: '2',
    clientId: '2',
    name: 'Dating App',
    description:
      'Lorem ipsum dolor sit amet, consectetuer adipiscin
    status: 'In Progress',
  },
  {
    id: '3',
    clientId: '3',
    name: 'SEO Project',
    description:
      'Lorem ipsum dolor sit amet, consectetuer adipiscin
    status: 'In Progress',
```

```
{
  project(id: "3") {
     id,
    name,
    description,
    status,
    client {
      name,
      phone
    }
  }
}
```


```
{
  "data": {
    "project": {
      "id": "3",
      "name": "SEO Project",
      "description": "Lorem ipsum dolor sit amet, consectetuer 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

```
//Mutations
const mutation = new GraphQLObjectType({
    name: 'Mutation',
    fields: {
        addClient: {
            type: ClientType,
            args: {
                name: { type: GraphQLNonNull(GraphQLString) },
                email: { type: GraphQLNonNull(GraphQLString) },
                phone: { type: GraphQLNonNull(GraphQLString) }
            },
            resolve(parent,args) {
                const client = new Client( {
                    name: args.name,
                    email: args.email,
                    phone: args.phone,
                });      You, 3 minutes ago • Uncommitted changes
                return client.save();
            }
        },

    }
});
```

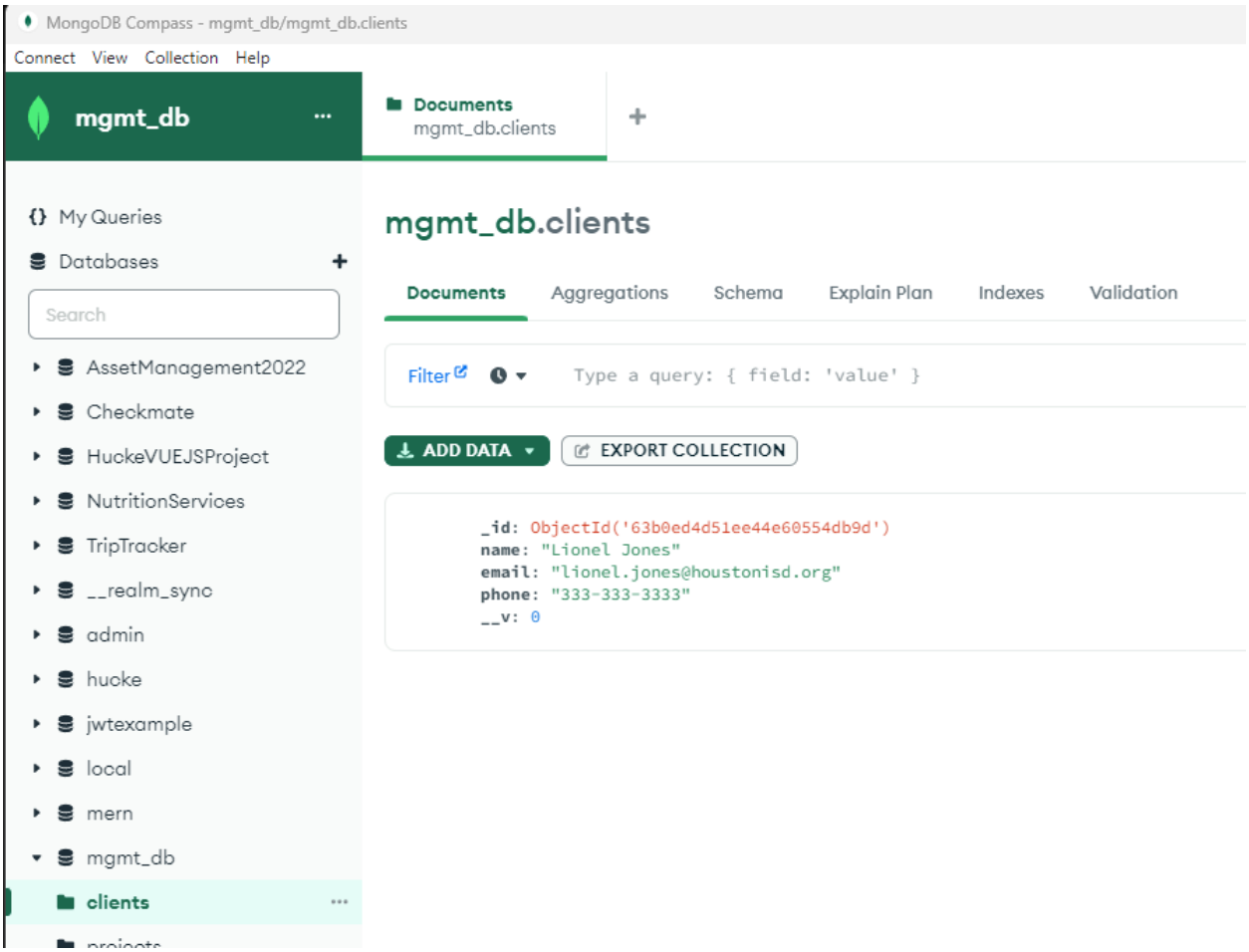
```
mutation {
  addClient(name:"David Lee Jones",email:"david.jones@optonline.org",phone:"444-333-3333") {
    id,
    name,
    email,
    phone
  }
}
```

GraphiQL ▶ Prettify Merge Copy History

```
mutation {
  addClient(name:"Lionel Jones",email:"lionel.jones@houstonisd.org",phone:"333-333-3333") {
    id,
    name,
    email,
    phone
  }
}
```

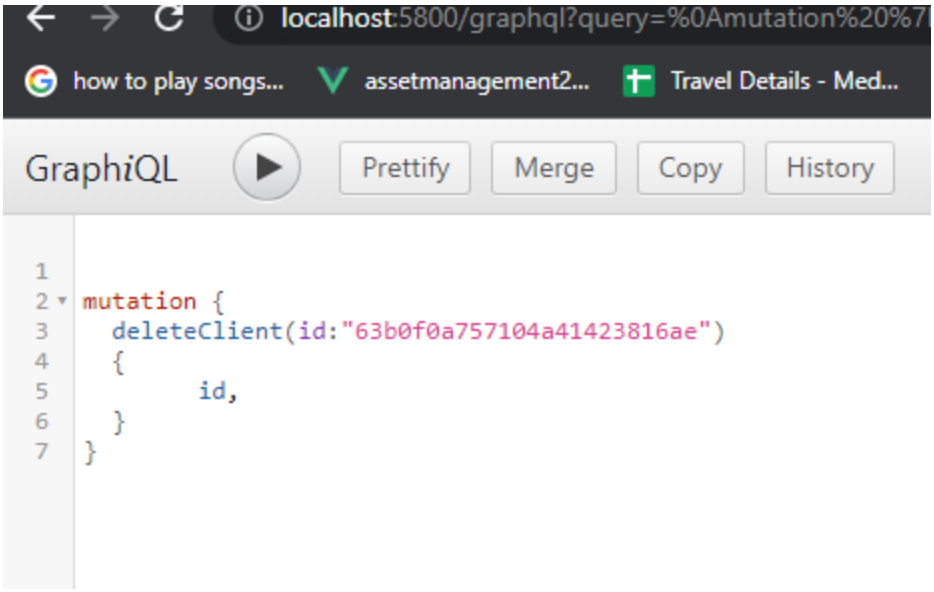

```
{
  "data": {
    "addClient": {
      "id": "63b0ed4d51ee44e60554db9d",
      "name": "Lionel Jones",
      "email": "lionel.jones@houstonisd.org",
      "phone": "333-333-3333"
    }
  }
}
```


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)
    }
},
```

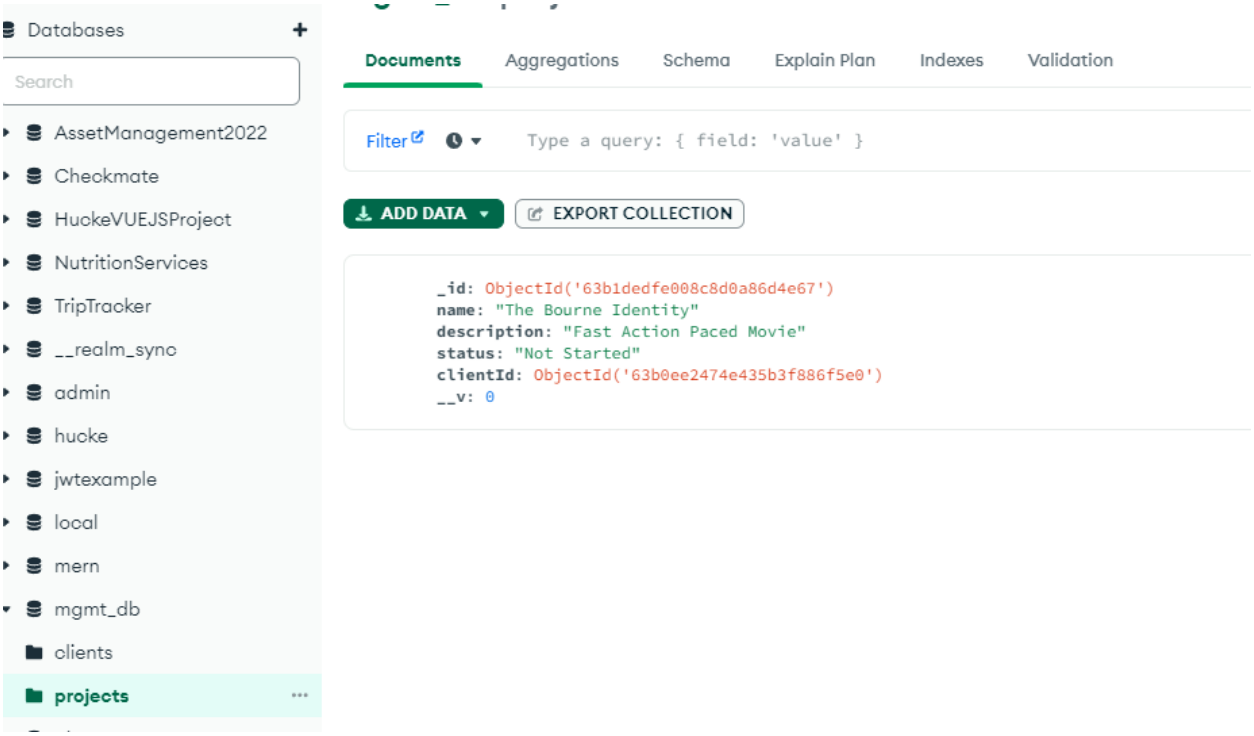


Stopped at 1:02 minutes – 12/31/2022

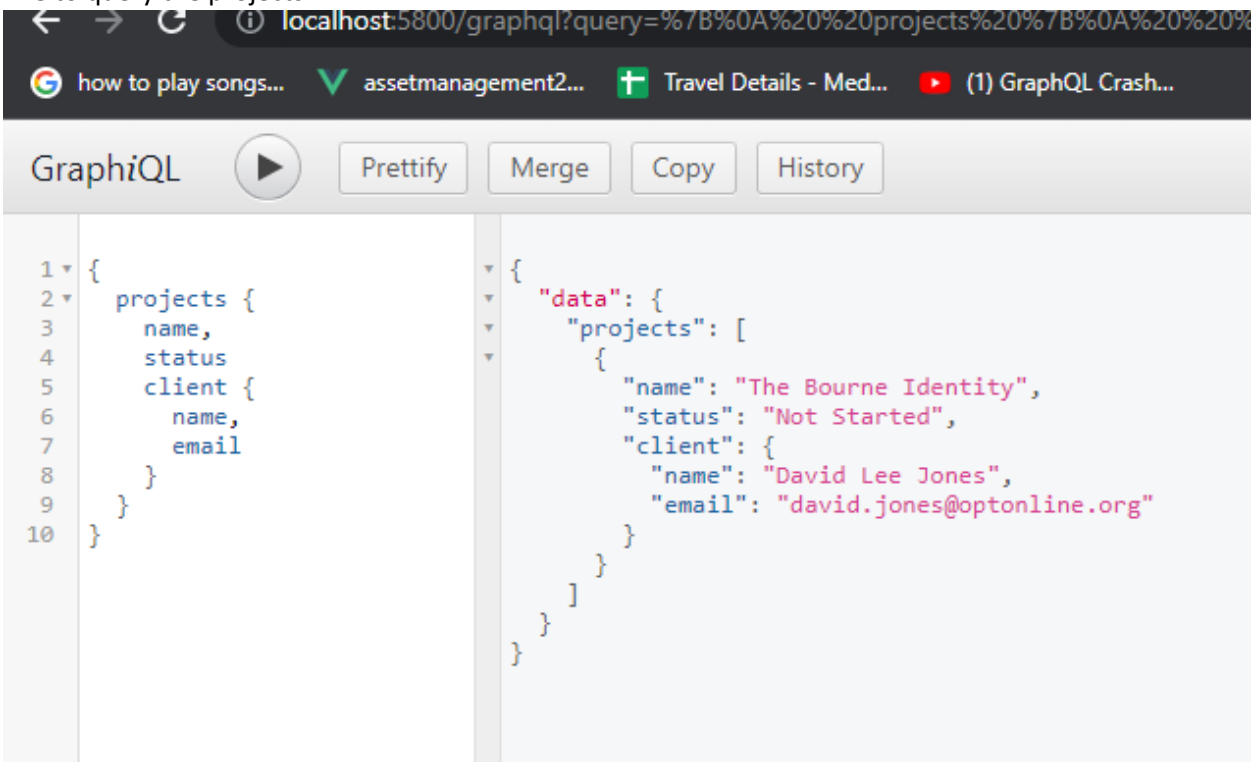
Mutation for Project Add

```
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



The to query the projects



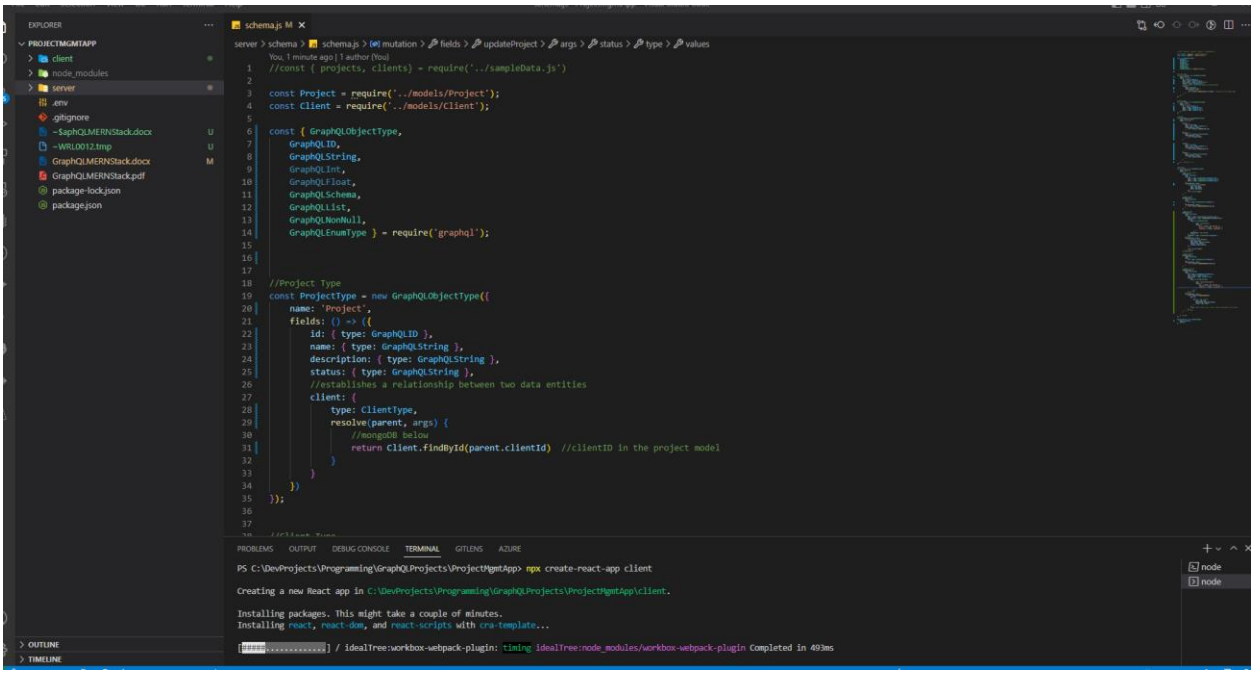
```
{
  projects {
    name,
    status
    client {
      name,
      email
    }
  }
}
```

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 sequal to original",
            status:progress) {
    name,
    description,
    status

  }
}
```

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

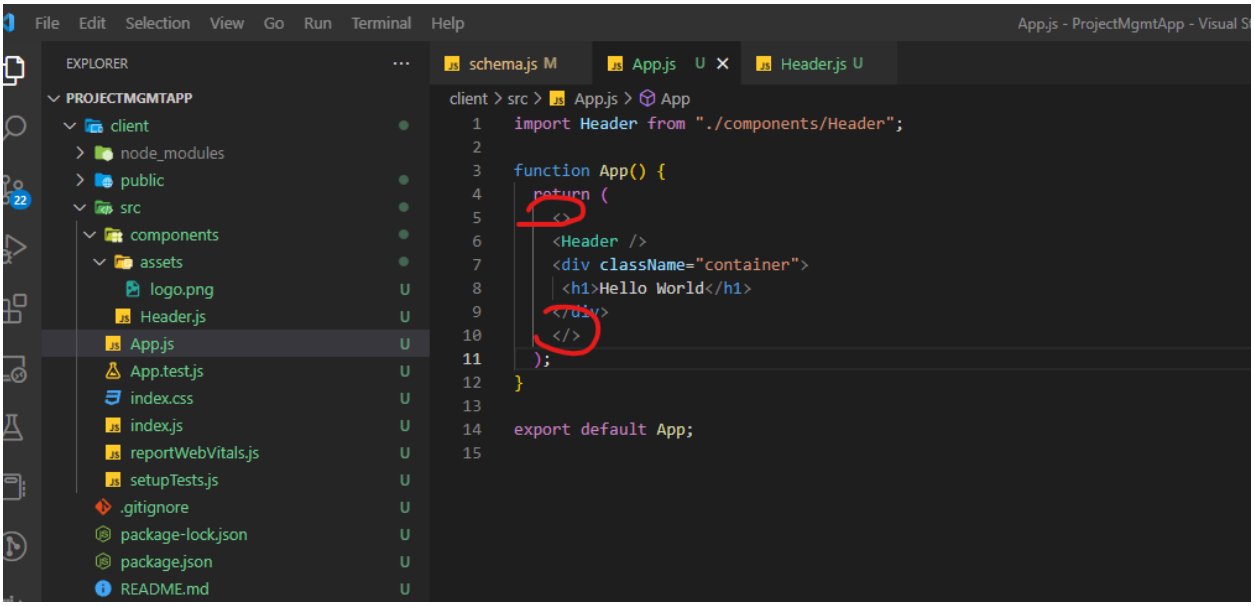


And as shown above, it creates a folder in the root called client (this is our front end)

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

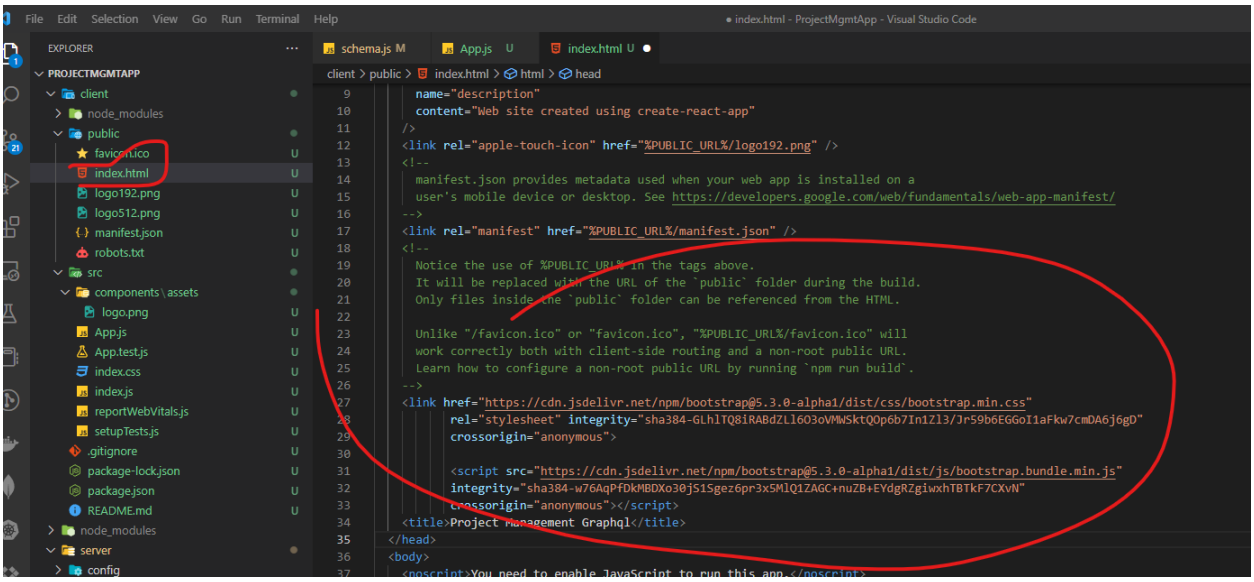
Above is adding the apollo client (to run queries against our graphql server)
Router and react icons (font-awesome icons) – from react

Tip: Using Fragment shorthand syntax
Shorthand syntax for a fragment:
<> </>



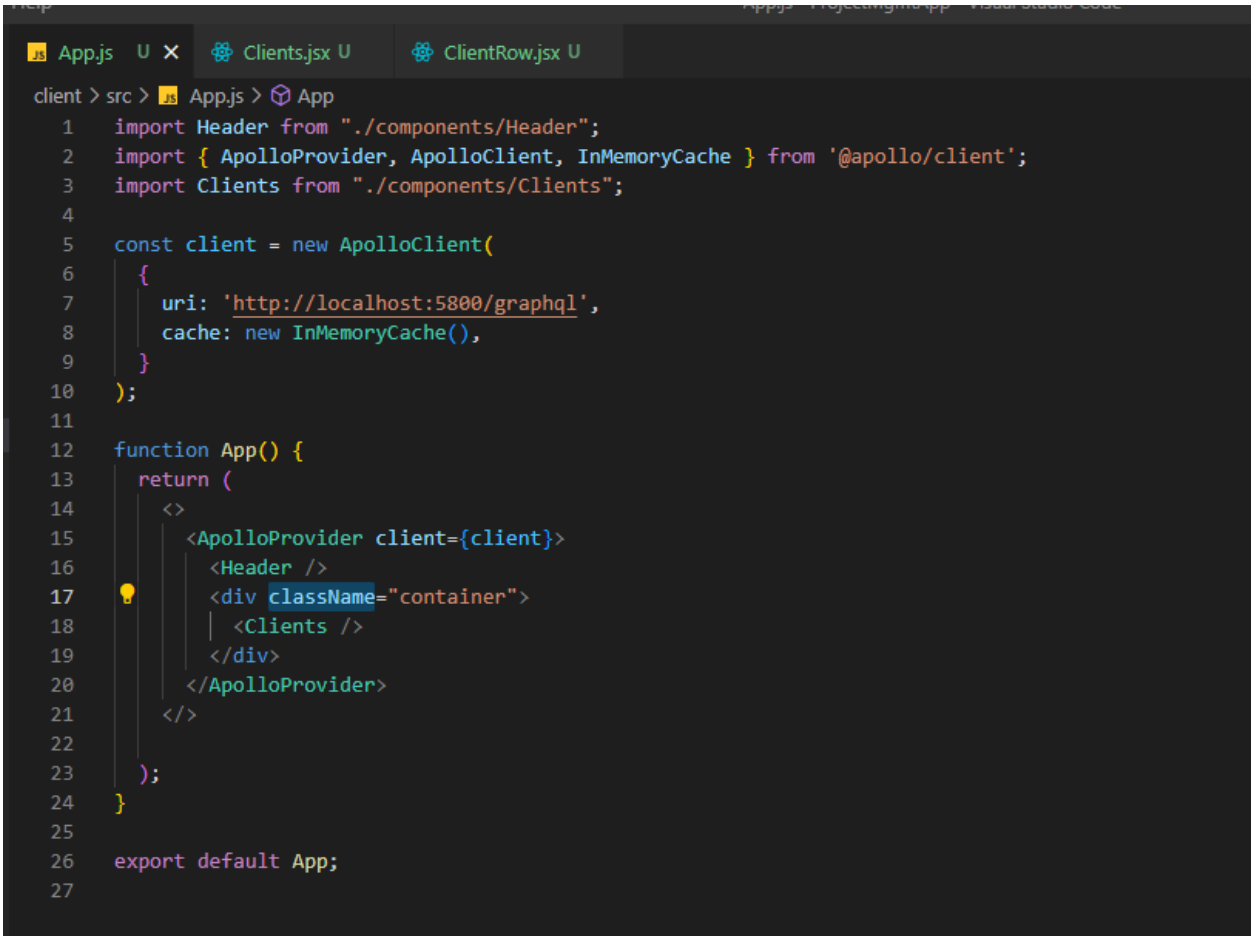
Tip: Adding bootstrap using the CDN
In this project he is not using bootstrap react via an npm, he is using the CDN
https://getbootstrap.com/

Add entries in the public/index.html file



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



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

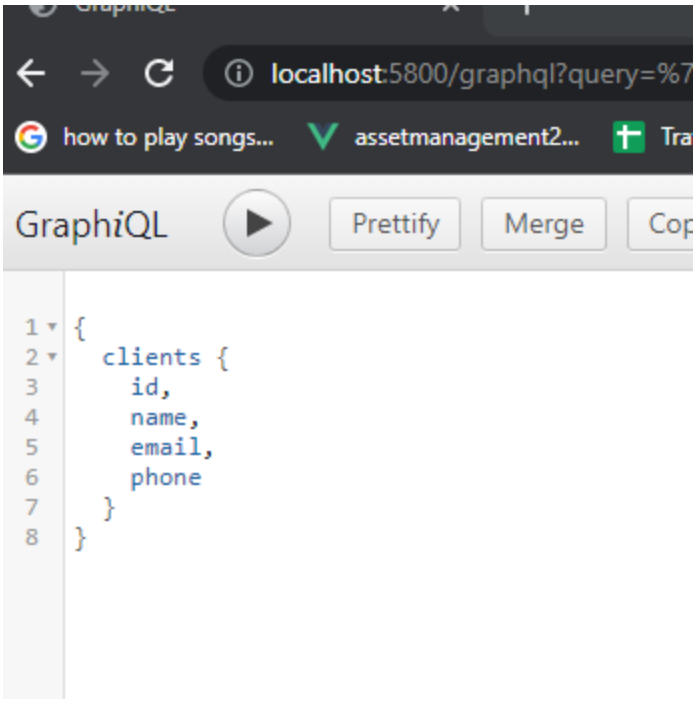
To query our data, in our client component

App.js U | Clients.jsx U | ClientRow.jsx U

client > src > components > Clients.jsx > ...

```jsx
import { gql, useQuery } from '@apollo/client'
import ClientRow from './ClientRow';

const GET_CLIENTS = gql`
  query getClients {
      clients {
          id,
          name,
          email,
          phone
      }
  }
`;

export default function Clients() {
    const {loading,error, data} = useQuery(GET_CLIENTS);
    if(loading) return <p>Loading....</p>
    if(error) return <p>OPPS!!....</p>

    return (
      <>
      {!loading && !error && (
       <table className='table table-hover mt-3'>
        <thead>
          <tr>
              <th>Name</th>
              <th>Email</th>
              <th>Phone</th>
              <th></th>
          </tr>
        </thead>
        <tbody>
          {
              data.clients.map(client => (
                  <ClientRow key={client.id} client={client} />
              ))
          }
        </tbody>
       </table>
      )}
```

We grab the syntax from graphql



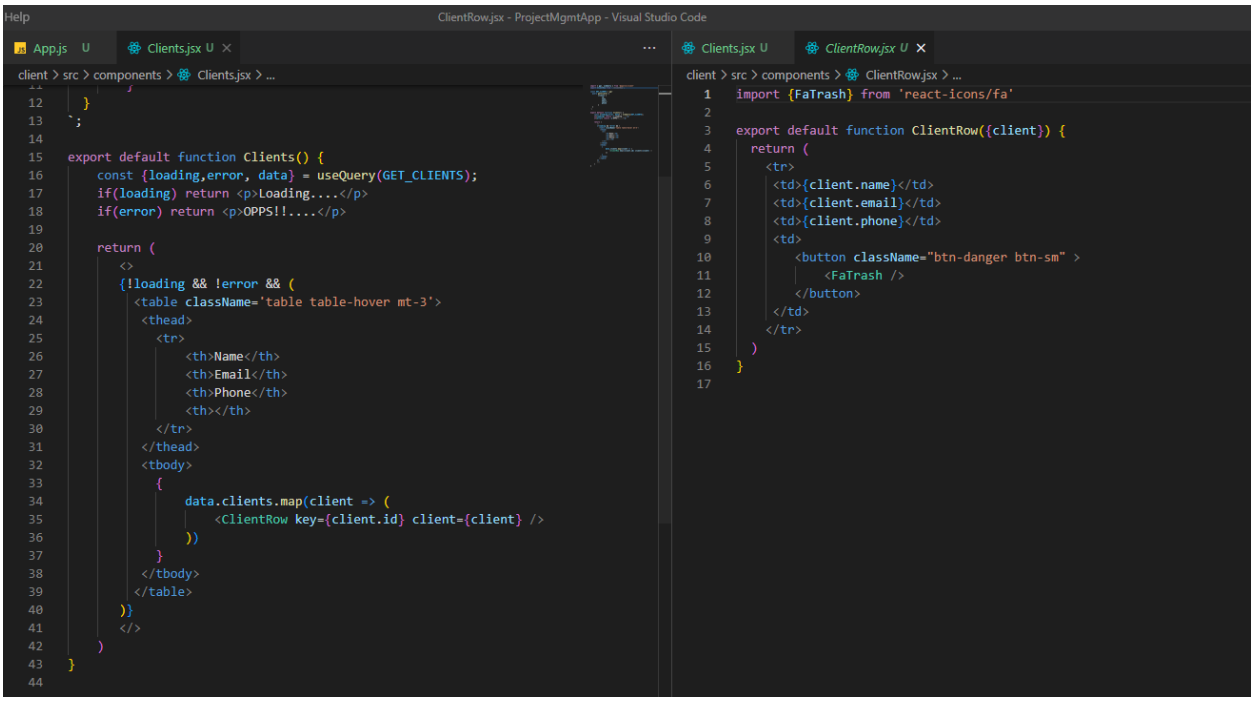
The build our query:

```
ent > src > components > Clients.jsx > ...
import { gql, useQuery } from '@apollo/client'
import ClientRow from './ClientRow';

const GET_CLIENTS = gql`
  query getClients {
        clients {
            id,
            name,
            email,
            phone
        }
  }
`;
```

We then use the useQuery hook

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

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

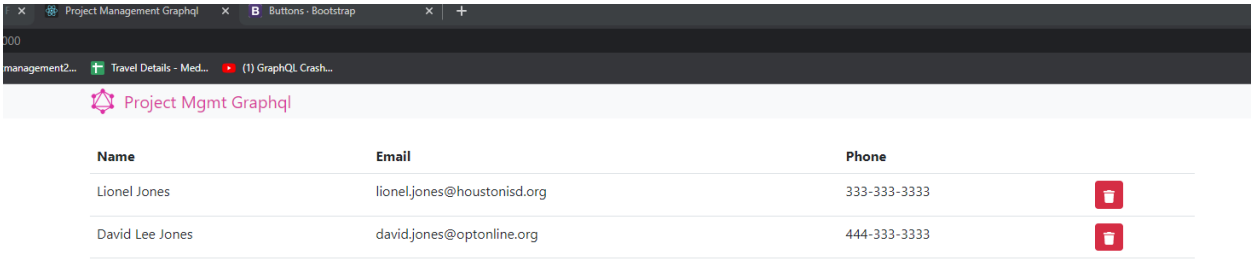


Tip: Building a re-usable component for rows in a table

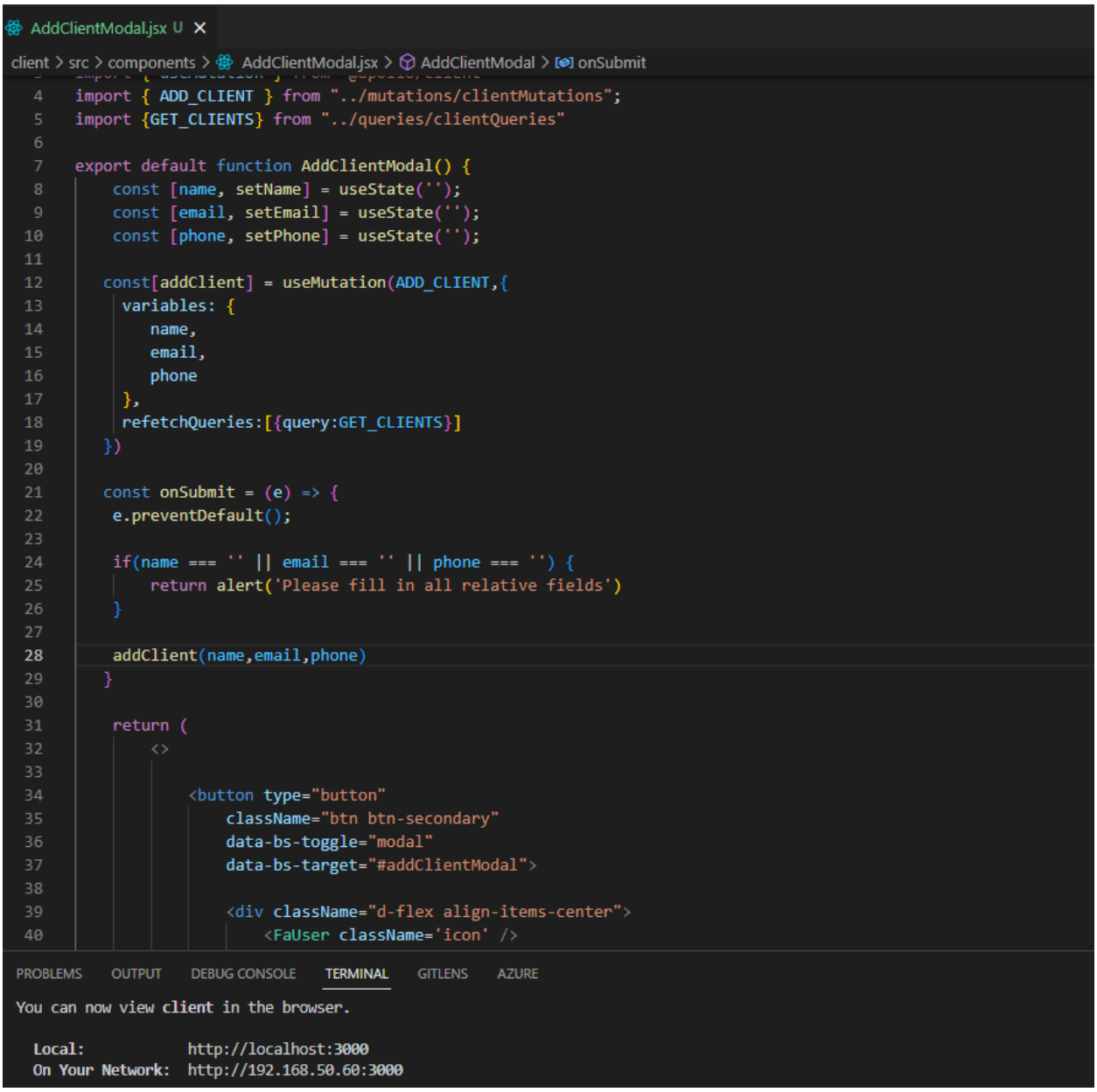
Tip: Using react-icons

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

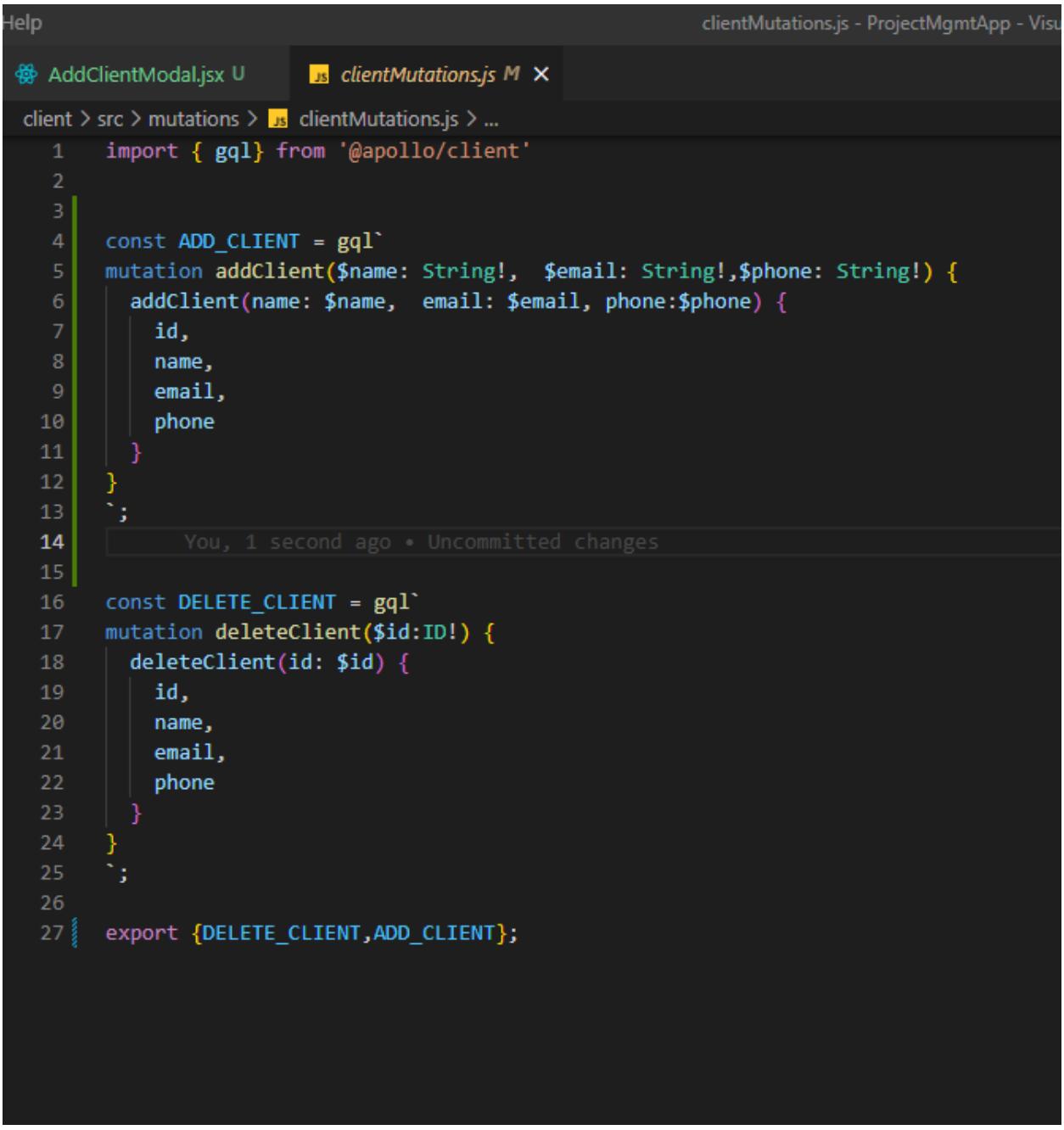
Also look at how we are using react-icons as well



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

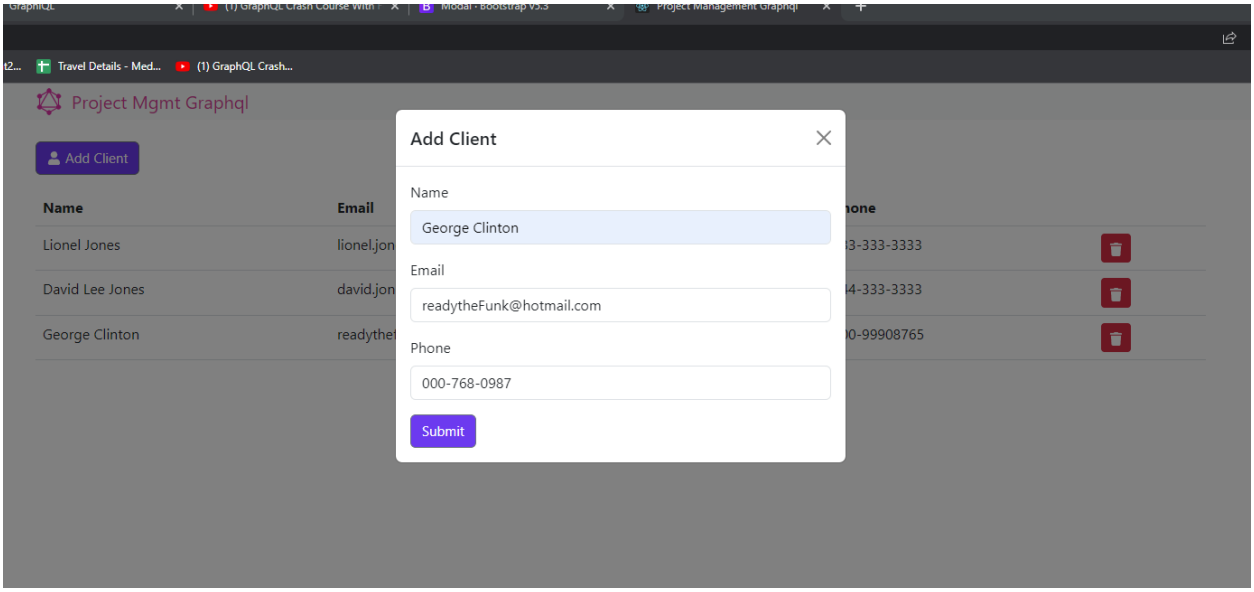


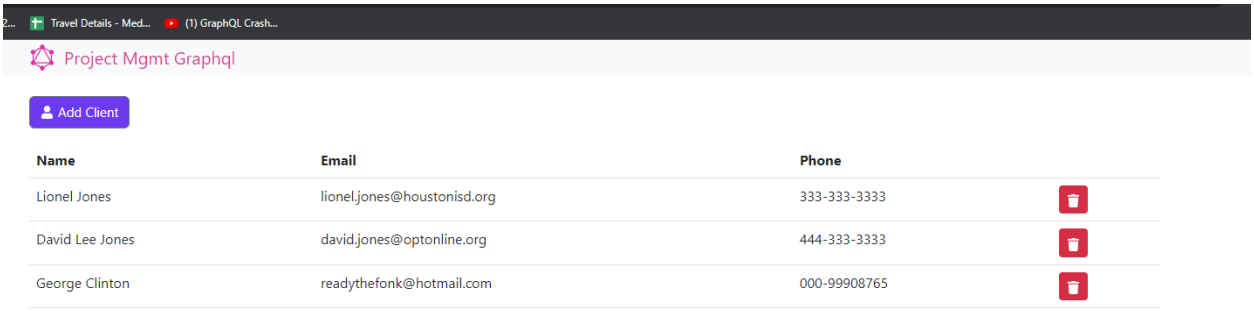
Along with mutations


Help
clientMutations.js - ProjectMgmtApp - Visu
AddClientModal.jsx U
clientMutations.js M
client > src > mutations > clientMutations.js > ...
import { gql} from '@apollo/client'
const ADD_CLIENT = gql`
mutation addClient($name: String!, $email: String!,$phone: String!) {
addClient(name: $name, email: $email, phone:$phone) {
id,
name,
email,
phone
}
}
`;
You, 1 second ago • Uncommitted changes
const DELETE_CLIENT = gql`
mutation deleteClient($id:ID!) {
deleteClient(id: $id) {
id,
name,
email,
phone
}
}
`;
export {DELETE_CLIENT,ADD_CLIENT};

In app.js

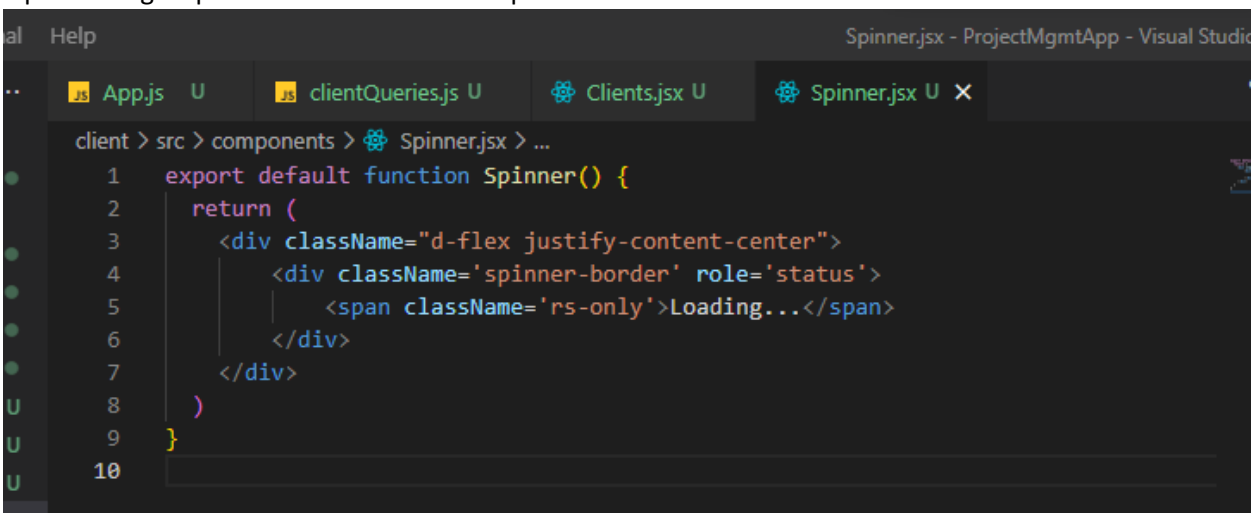
```
);

function App() {
  return (
    <>
      <ApolloProvider client={client}>
        <Header />
        <div className="container">
          <AddClientModal />
          <Clients />
        </div>      You, 3 hours ago • added code for viewing clients and de
      </ApolloProvider>
    </>

  );
}

export default App;
```

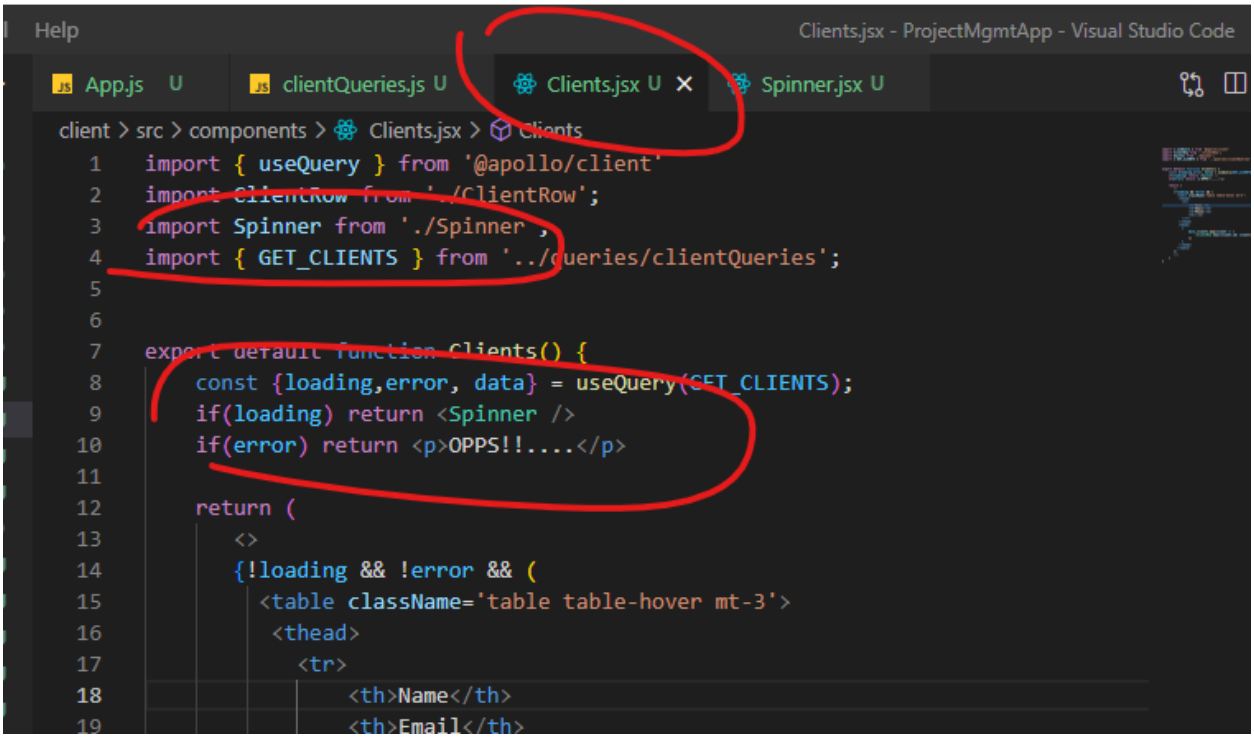




Sweet

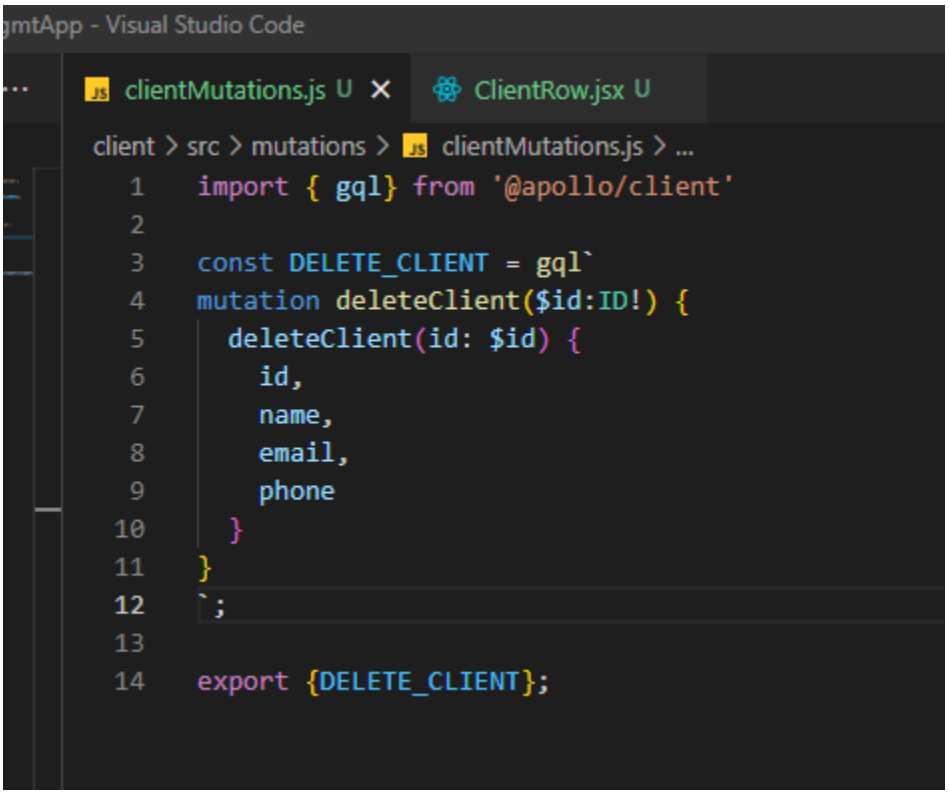
Tip: Creating a Spinner with built-in react spinner



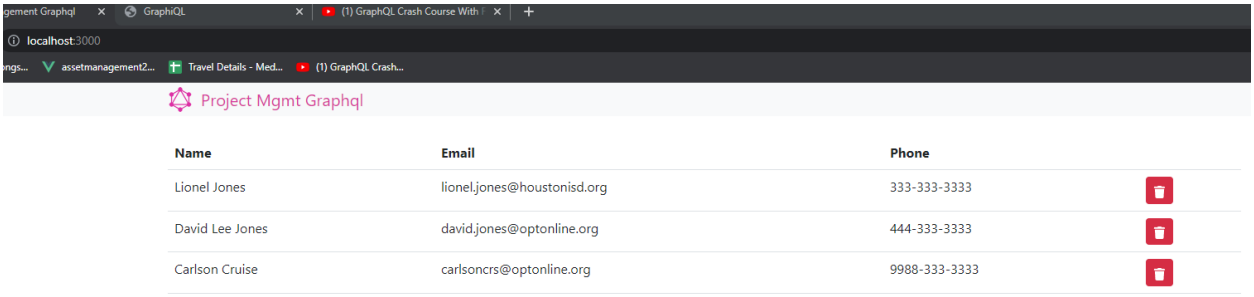

Help
Clients.jsx - ProjectMgmtApp - Visual Studio Code
App.js U
clientQueries.js U
Clients.jsx U
Spinner.jsx U
client > src > components > Clients.jsx > Clients
import { useQuery } from '@apollo/client'
import ClientRow from './ClientRow';
import Spinner from './Spinner';
import { GET_CLIENTS } from '../queries/clientQueries';
export default function Clients() {
const {loading,error, data} = useQuery(GET_CLIENTS);
if(loading) return <Spinner />
if(error) return OPPS!!....
return (
<>
{!loading && !error && (
<table className='table table-hover mt-3'>
<thead>
<tr>
<th>Name</th>
<th>Email</th>

Tip: Deleting and re-fetching data with graphiql

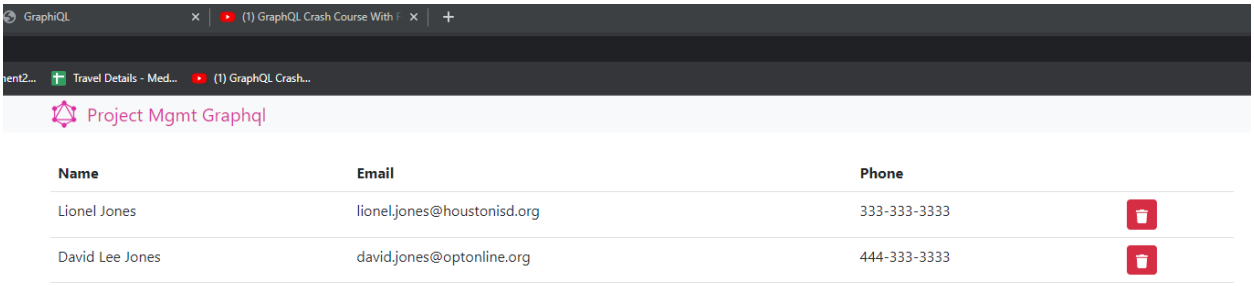
Create the mutation

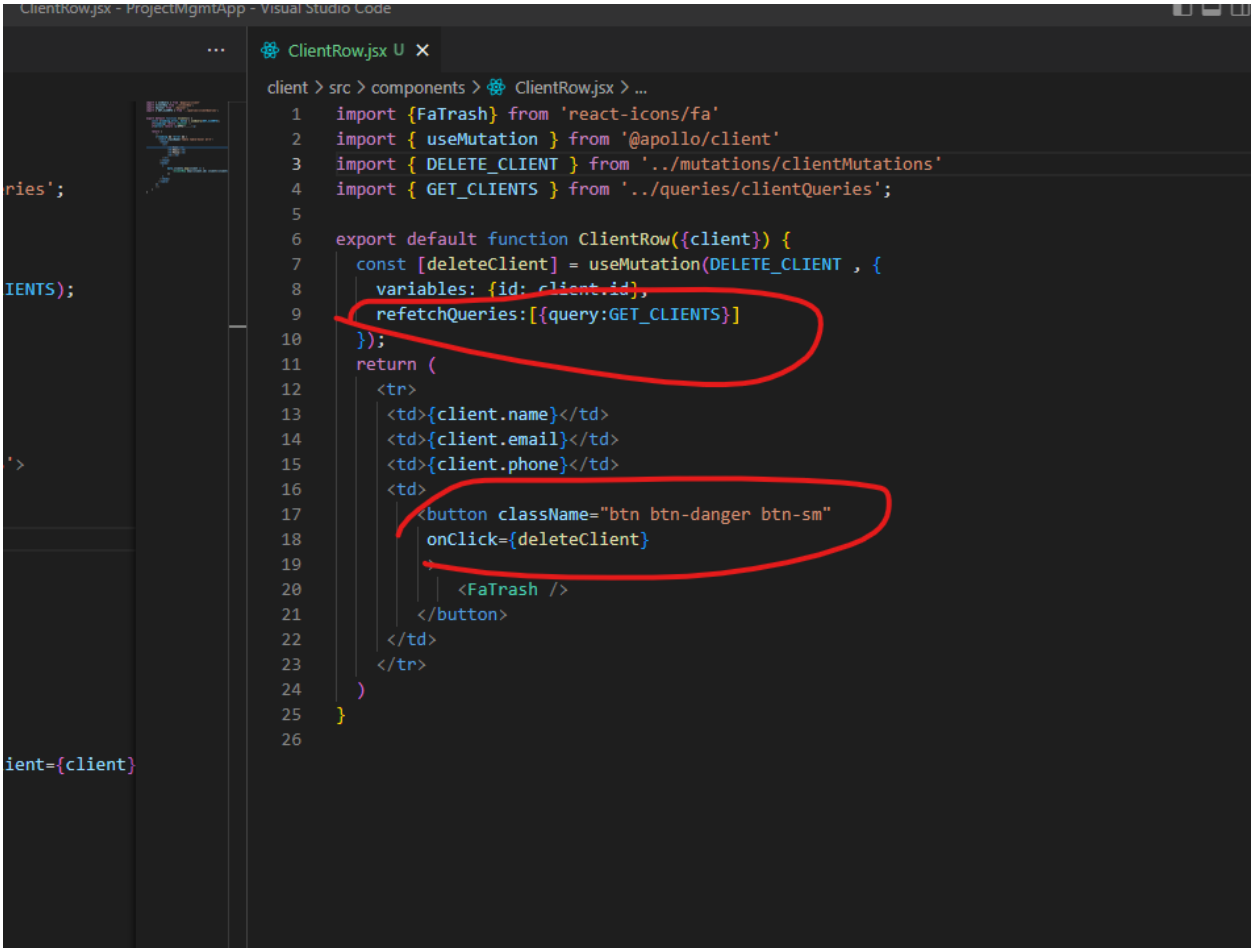


And below we can use the refetchQuieres attribute to refresh the UI when a client record is deleted

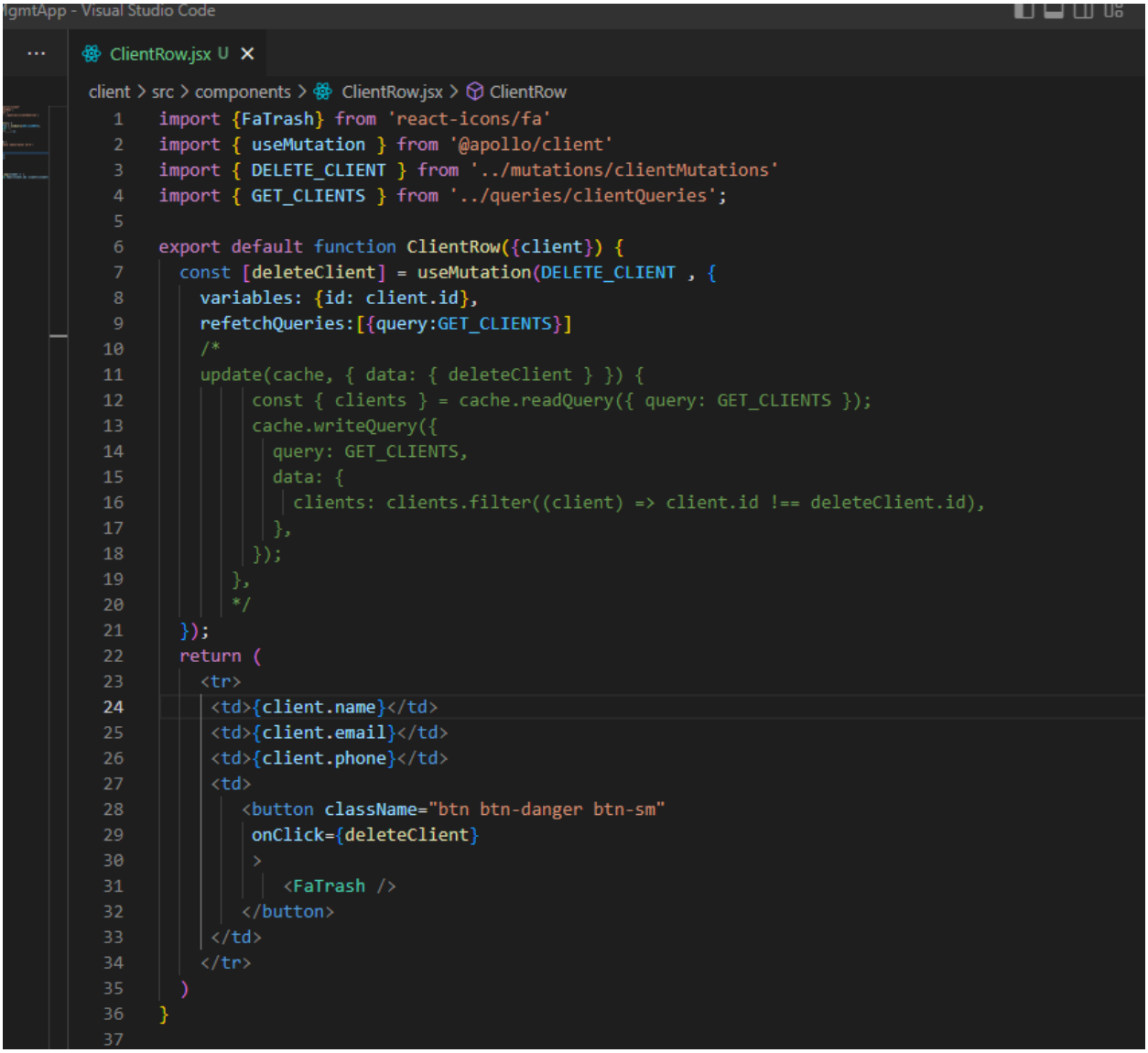


When we hit delete, the UI updates with the remaining records


GraphiQL
(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

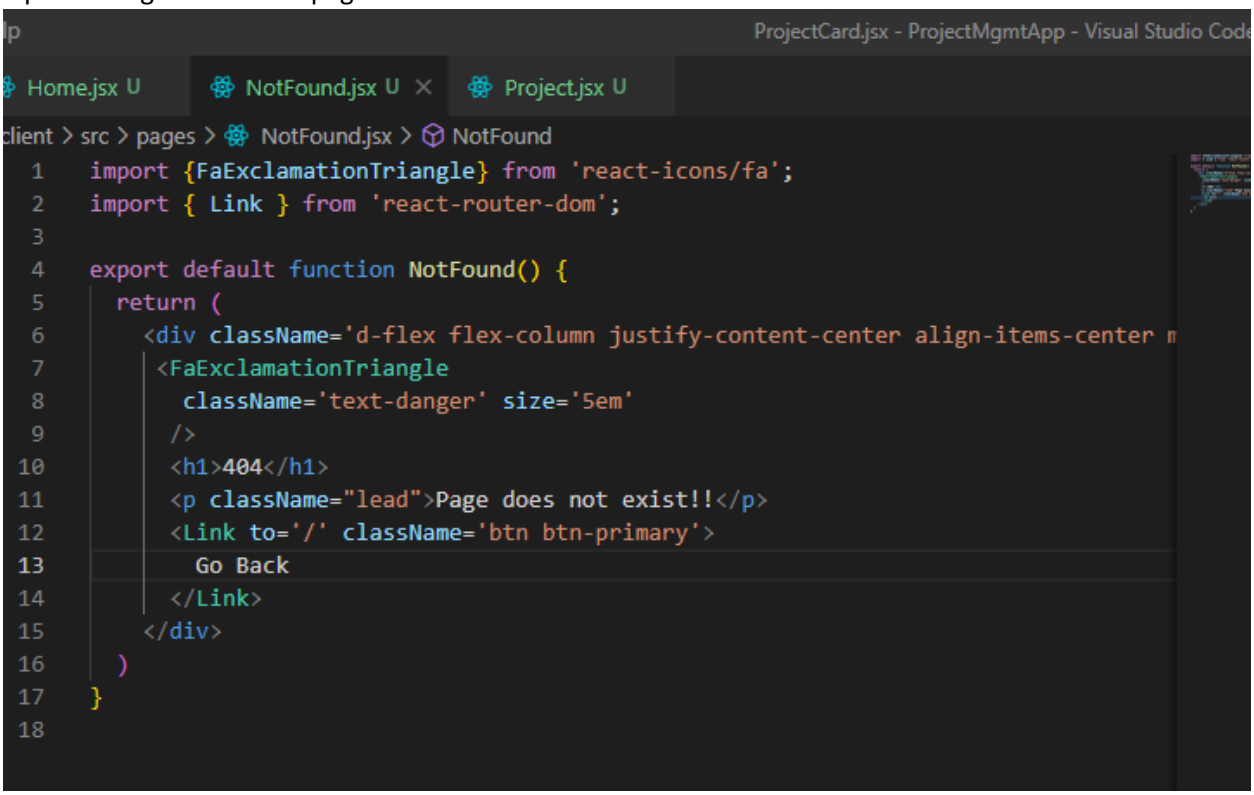

ClientRow.jsx - ProjectMgmtApp - Visual Studio Code
ClientRow.jsx U
client > src > components > ClientRow.jsx > ...
import {FaTrash} from 'react-icons/fa'
import { useMutation } from '@apollo/client'
import { DELETE_CLIENT } from '../mutations/clientMutations'
import { GET_CLIENTS } from '../queries/clientQueries';
export default function ClientRow({client}) {
const [deleteClient] = useMutation(DELETE_CLIENT , {
variables: {id: client.id},
refetchQueries:[{query:GET_CLIENTS}]
});
return (
<tr>
<td>{client.name}</td>
<td>{client.email}</td>
<td>{client.phone}</td>
<td>
<button className="btn btn-danger btn-sm"
onClick={deleteClient}
>
<FaTrash />
</button>
</td>
</tr>
)
}

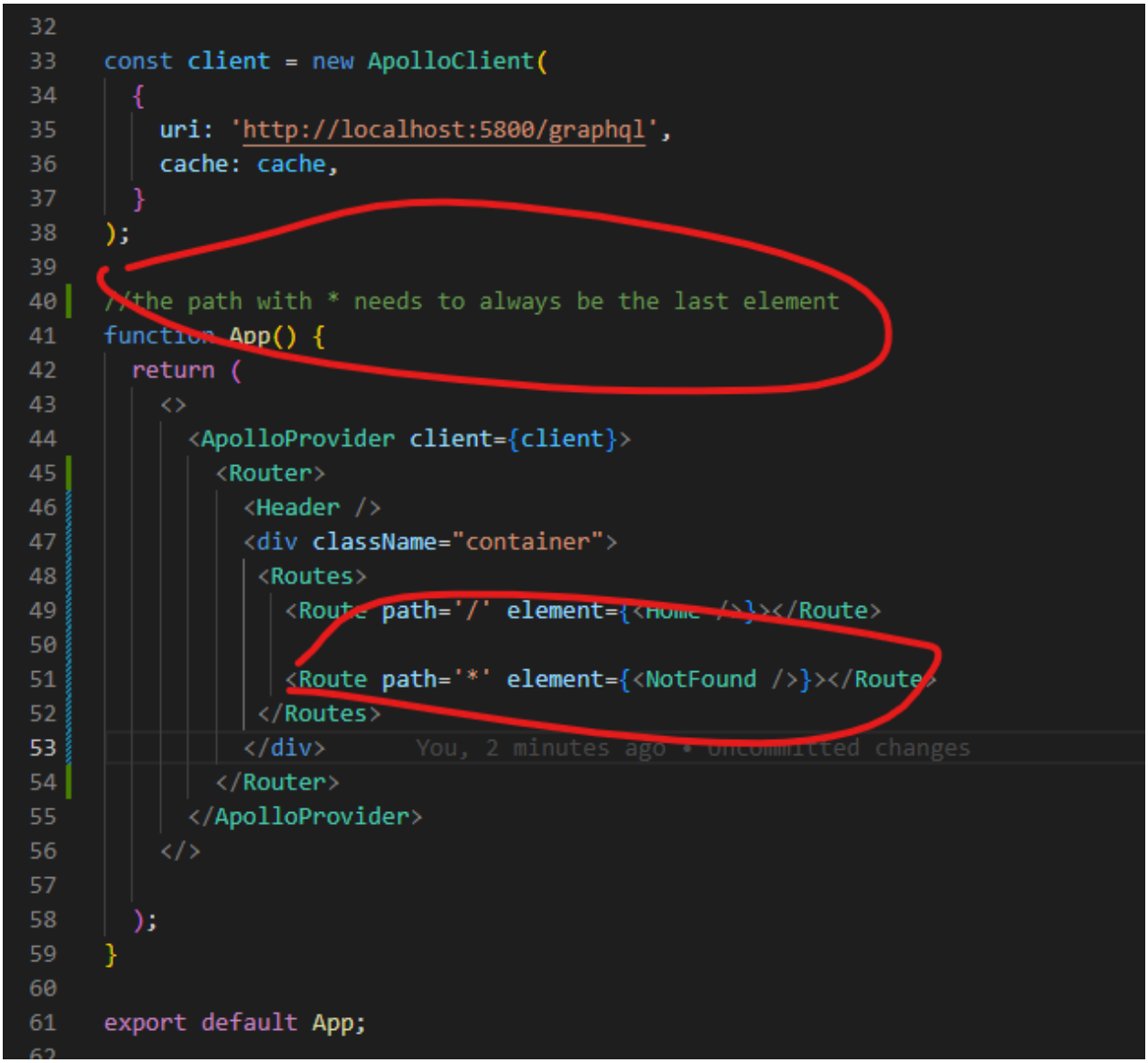
Second way:



More code as shown above

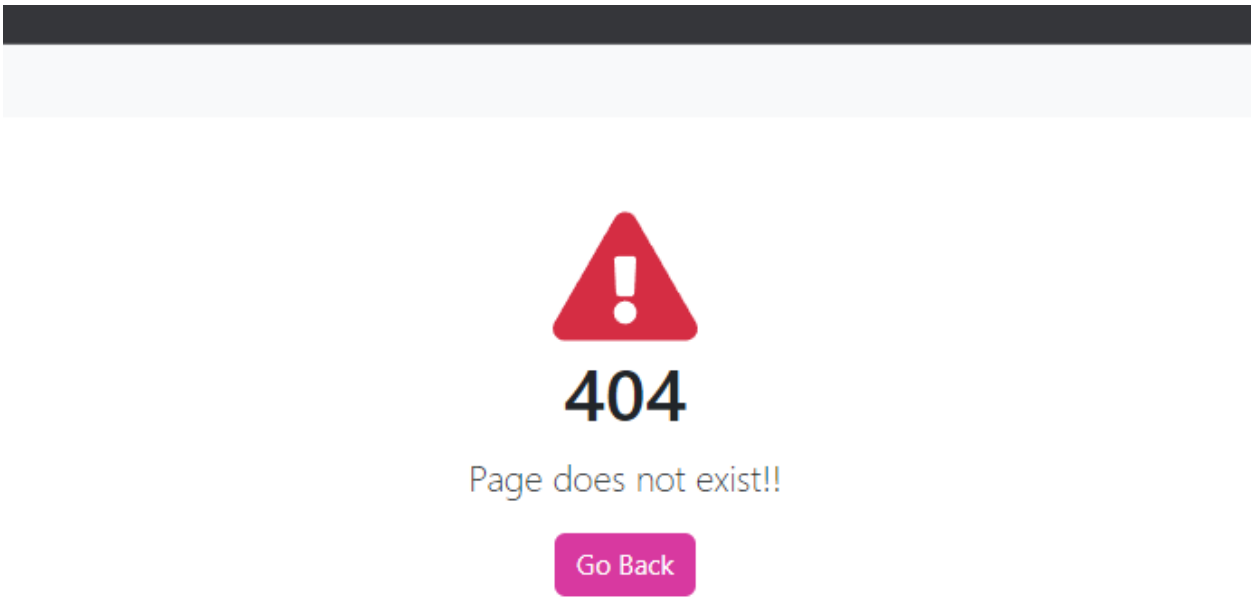
Tip: Creating a not Found page



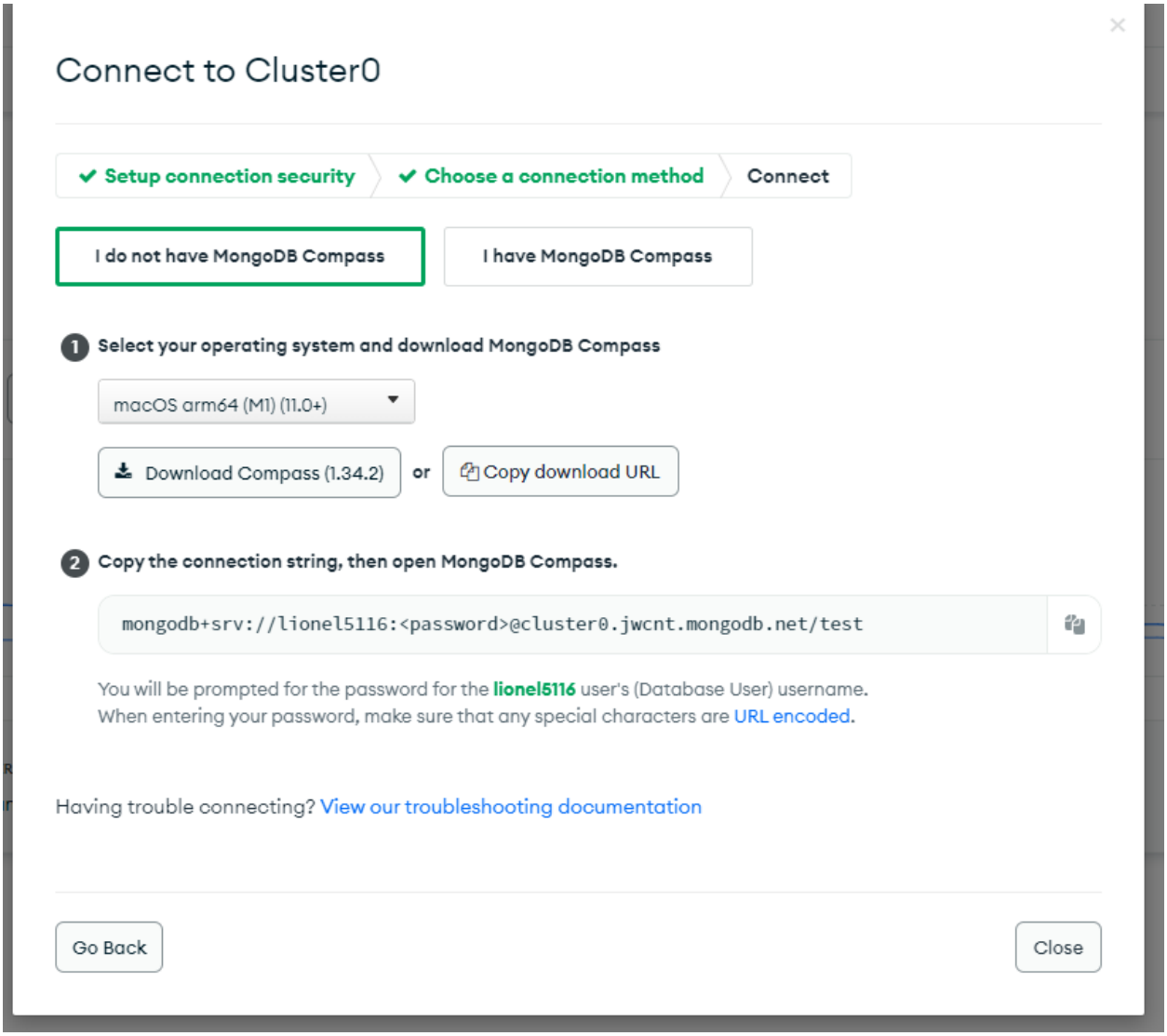

const client = new ApolloClient(
{
uri: 'http://localhost:5800/graphql',
cache: cache,
}
);
//the path with * needs to always be the last element
function App() {
return (
<>
<ApolloProvider client={client}>
<Router>
<Header />

<Routes>
<Route path='*' element={<NotFound />}></Route>
</Routes>

</Router>
</ApolloProvider>
</>
);
}
export default App;



Tip: GUI for MongoDB
https://www.mongodb.com/products/compass

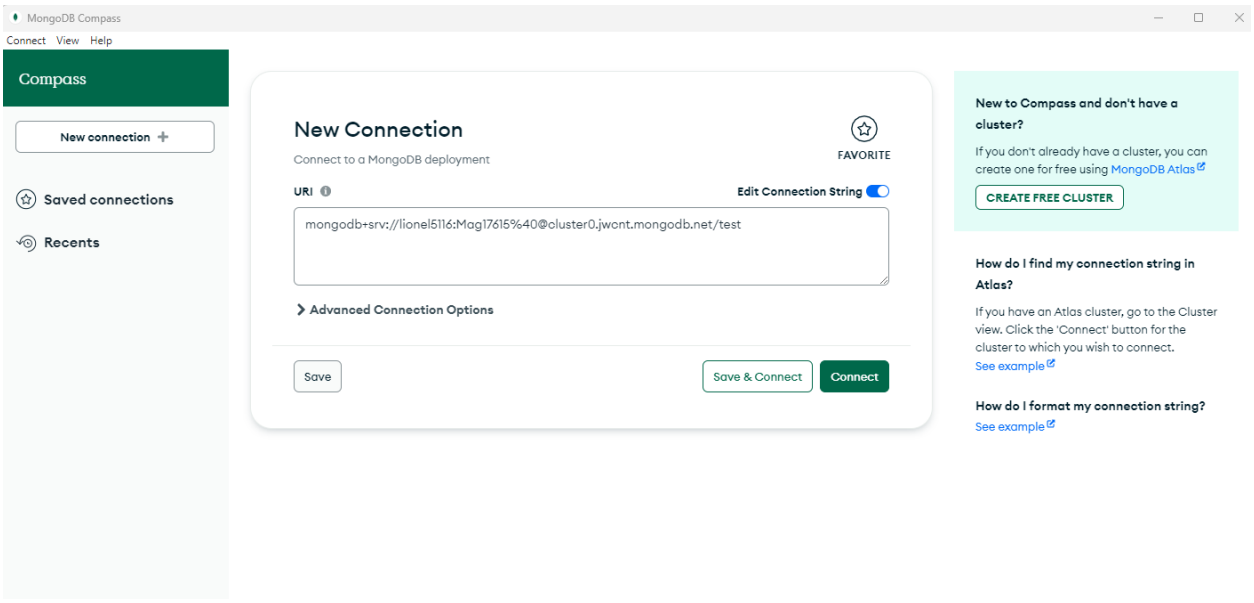


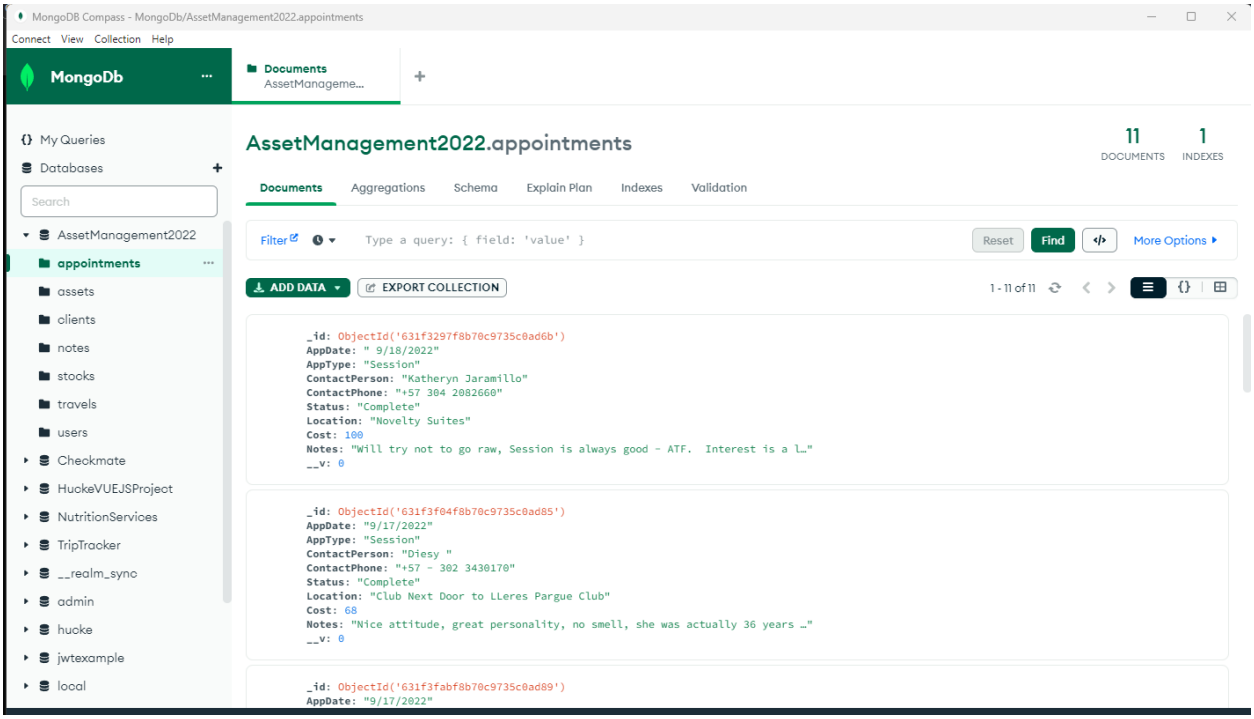
{

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

"jwtSecretToken":"mysecrettoken"

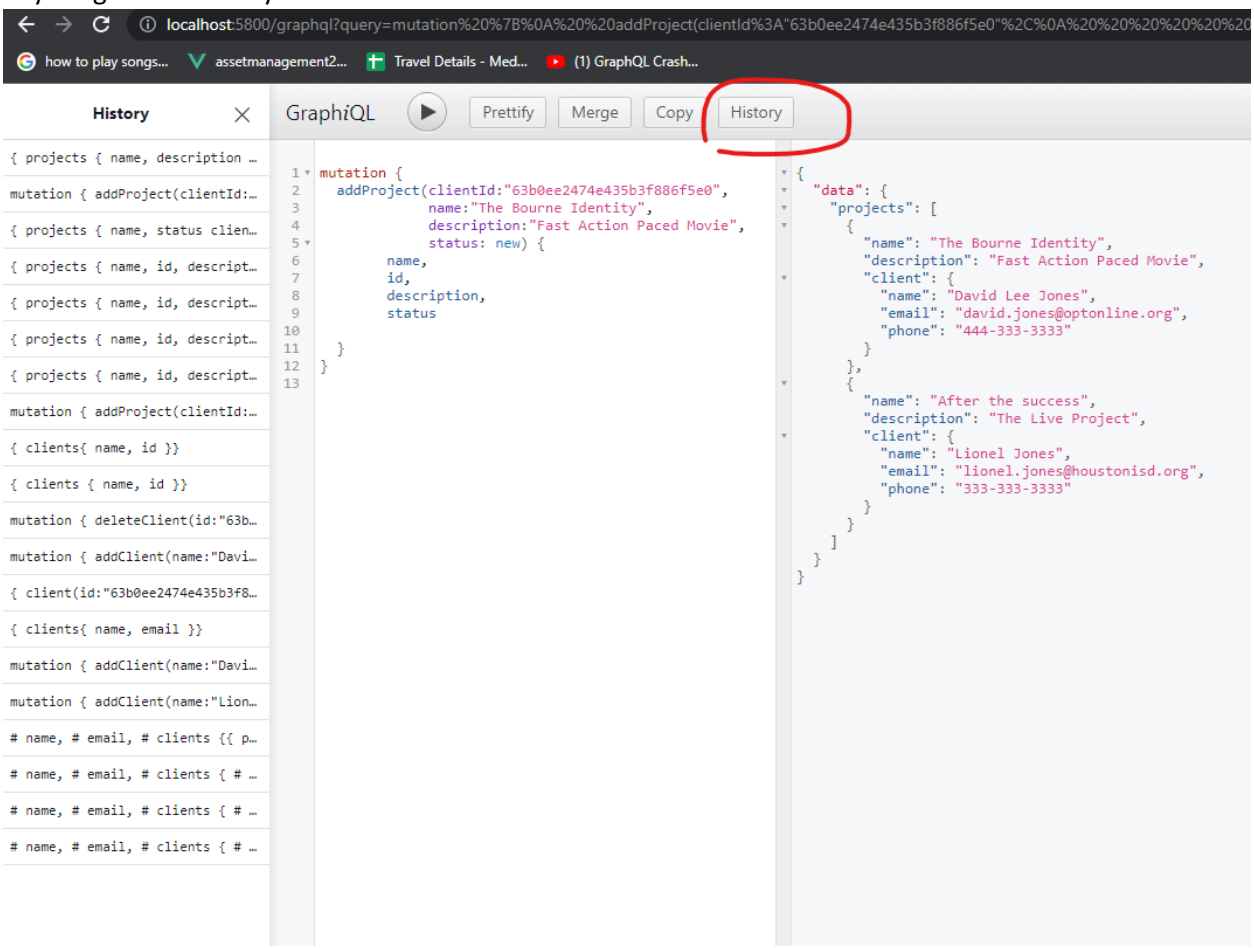
}


MongoDB Compass
Connect View Help
Compass
New connection
Saved connections
Recents
New Connection
Connect to a MongoDB deployment
FAVORITE
URI
Edit Connection String
mongodb+srv://lionel5116:Mag17615%40@cluster0.jwcnt.mongodb.net/test
Advanced Connection Options
Save
Save & Connect
Connect
New to Compass and don't have a cluster?
If you don't already have a cluster, you can create one for free using MongoDB Atlas
CREATE FREE CLUSTER
How do I find my connection string in Atlas?
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
See example
How do I format my connection string?
See example


MongoDB Compass - MongoDb/AssetManagement2022.appointments
Connect View Collection Help
MongoDb
Documents
AssetManageme...
My Queries
Databases
Search
AssetManagement2022
appointments
assets
clients
notes
stocks
travels
users
Checkmate
HuckeVUEJSProject
NutritionServices
TripTracker
__realm_sync
admin
hucke
jwtexample
local
AssetManagement2022.appointments
11
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 11 of 11
_id: ObjectId('631f3297f8b70c9735c0ad6b')
AppDate: " 9/18/2022"
AppType: "Session"
ContactPerson: "Katheryn Jaramillo"
ContactPhone: "+57 304 2082660"
Status: "Complete"
Location: "Novelty Suites"
Cost: 100
Notes: "Will try not to go raw, Session is always good - ATF. Interest is a l…"
__v: 0
_id: ObjectId('631f3f04f8b70c9735c0ad85')
AppDate: "9/17/2022"
AppType: "Session"
ContactPerson: "Diesy "
ContactPhone: "+57 - 302 3430170"
Status: "Complete"
Location: "Club Next Door to LLeres Pargue Club"
Cost: 68
Notes: "Nice attitude, great personality, no smell, she was actually 36 years …"
__v: 0
_id: ObjectId('631f3fabf8b70c9735c0ad89')
AppDate: "9/17/2022"

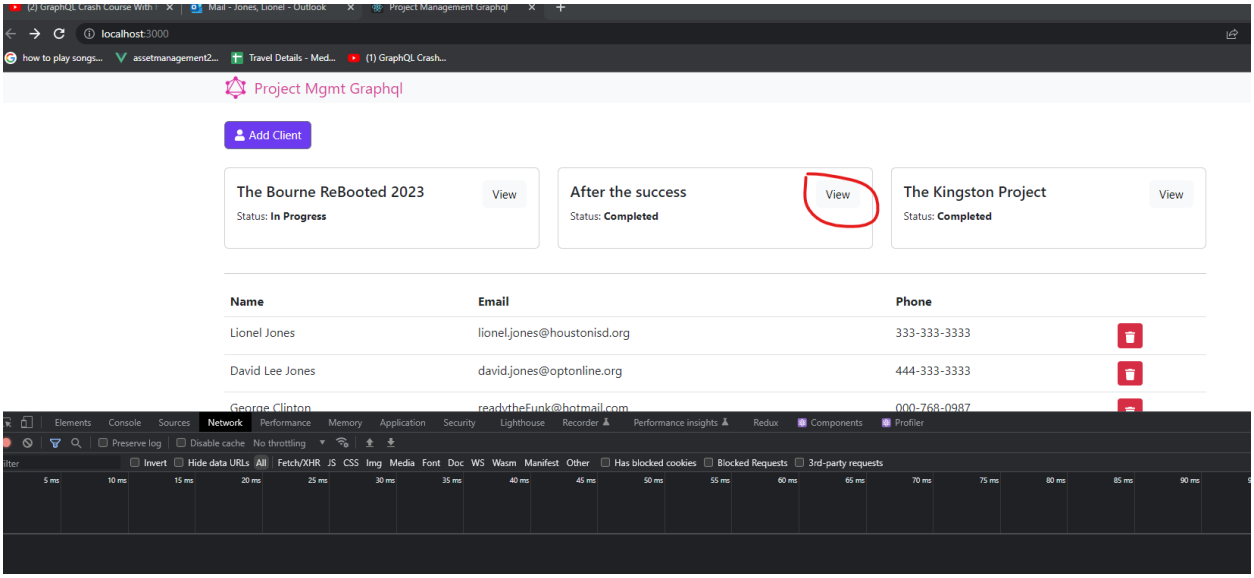
Tip: Getting History from graphiql

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



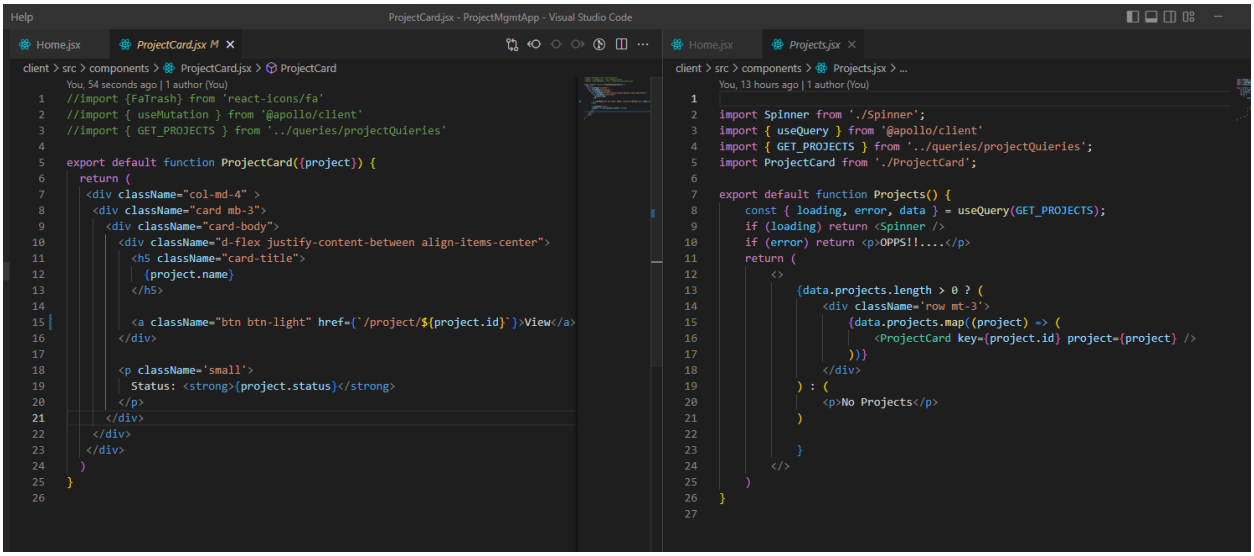
Tip: Grabbing the network response data with developer tools to view graphql data

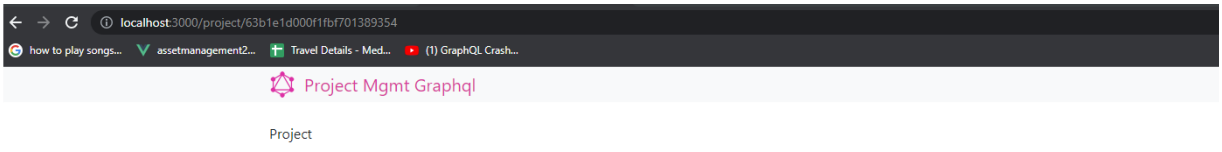
To view and check to see what data is being passed when you click on a url and it fetches graphql data, use the network and response tab:

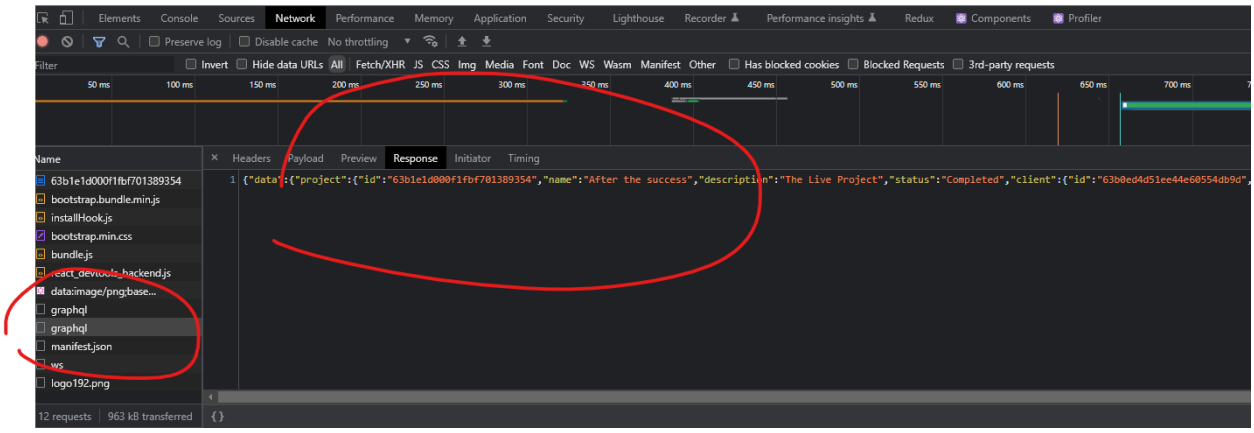


Once you click on the view link:

```
export default function ProjectCard({project}) {
  return (
   <div className="col-md-4" >
    <div className="card mb-3">
      <div className="card-body">
        <div className="d-flex justify-content-between align-items-center">
          <h5 className="card-title">
            {project.name}
          </h5>

          <a className="btn btn-light" href={`/project/${project.id}`}>View</a>
        </div>

        <p className='small'>
          Status: <strong>{project.status}</strong>
        </p>
      </div>
    </div>
   </div>
  )
}
```

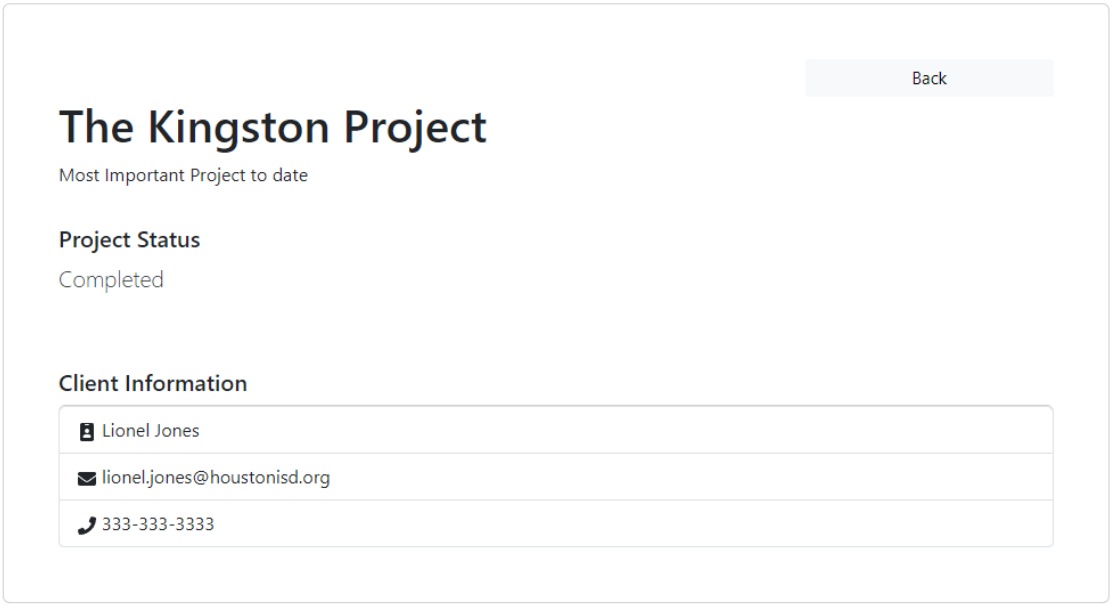


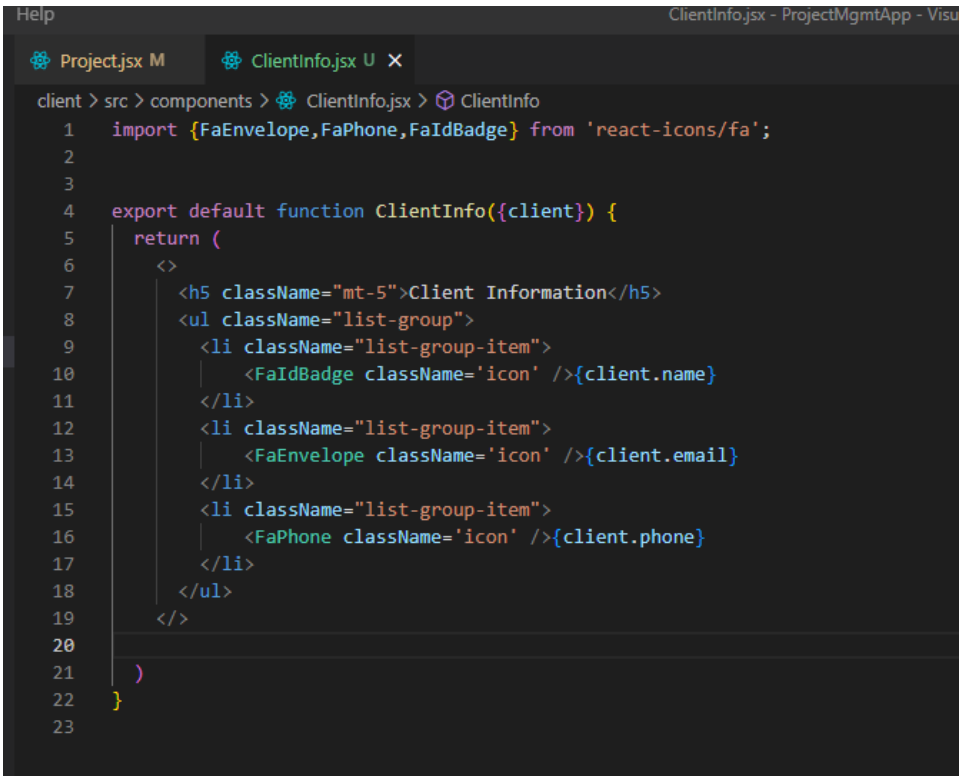




And you can see the data being passed in via the getProject by :$id

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





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>       You, 6 minutes ago • Uncommitted changes
29      )}
30      </>
31    )
32  }
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS   AZURE

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

AddProjectModal.jsx U Home.jsx M

client > src > pages > Home.jsx > Home

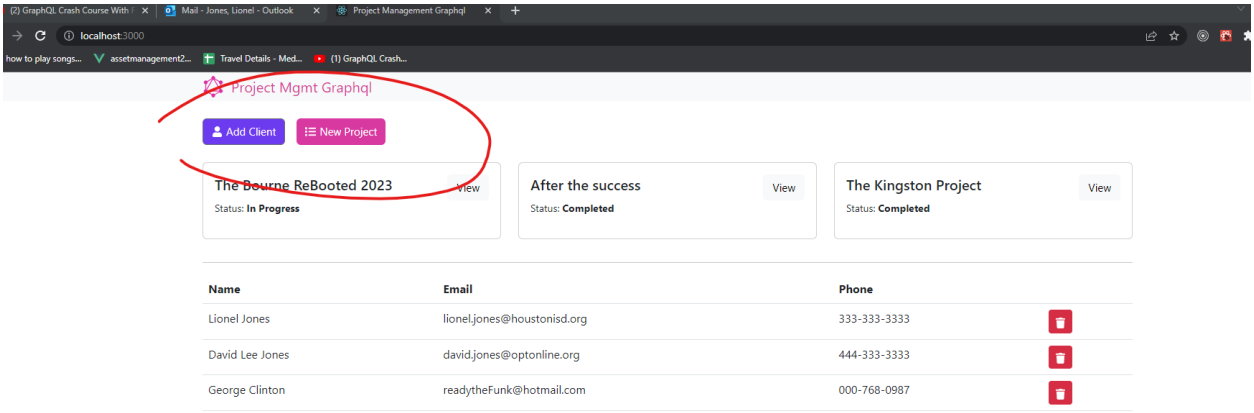
You, 2 minutes ago | 1 author (You)

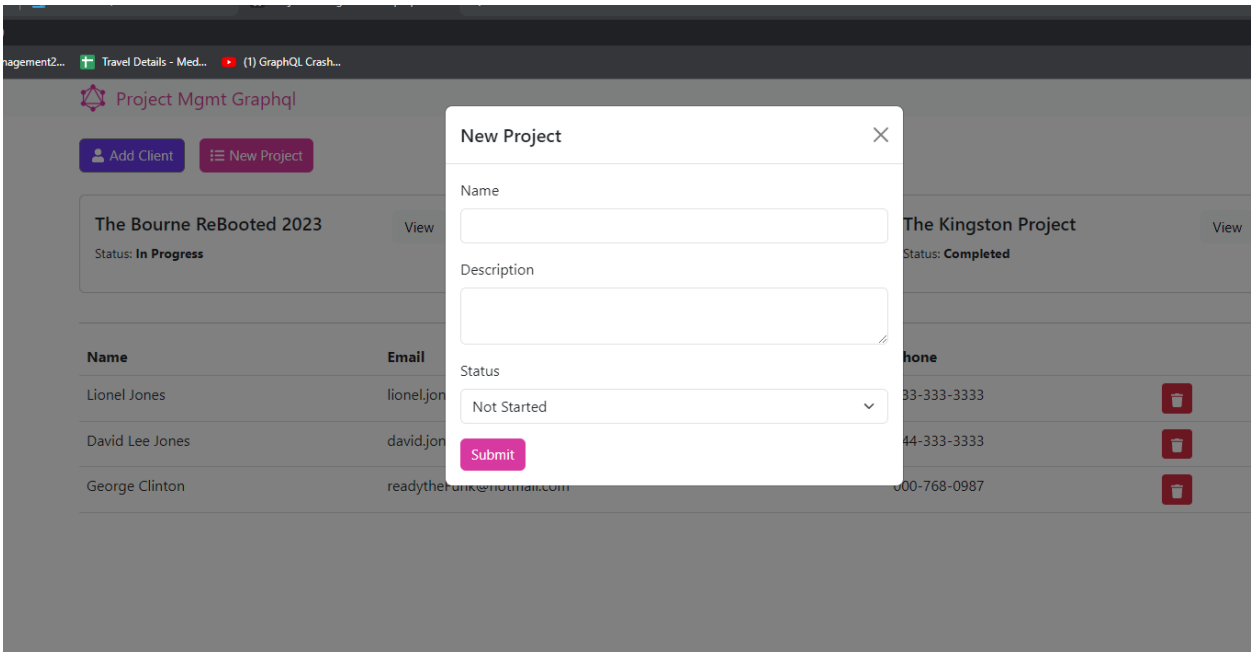
```jsx
import React from 'react';
import AddClientModal from '../components/AddClientModal';
import AddProjectModal from '../components/AddProjectModal';
import Clients from '../components/Clients';
import Projects from '../components/Projects';


export default function Home() {
  return (
   <>
    <div className="d-flex gap-3 mb-4">
     <AddClientModal />
     <AddProjectModal />
    </div>
      <Projects />
      <hr />
      <Clients />
   </>      You, 14 hours ago • added code for adding clients - project and not f…
  )
}

```

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

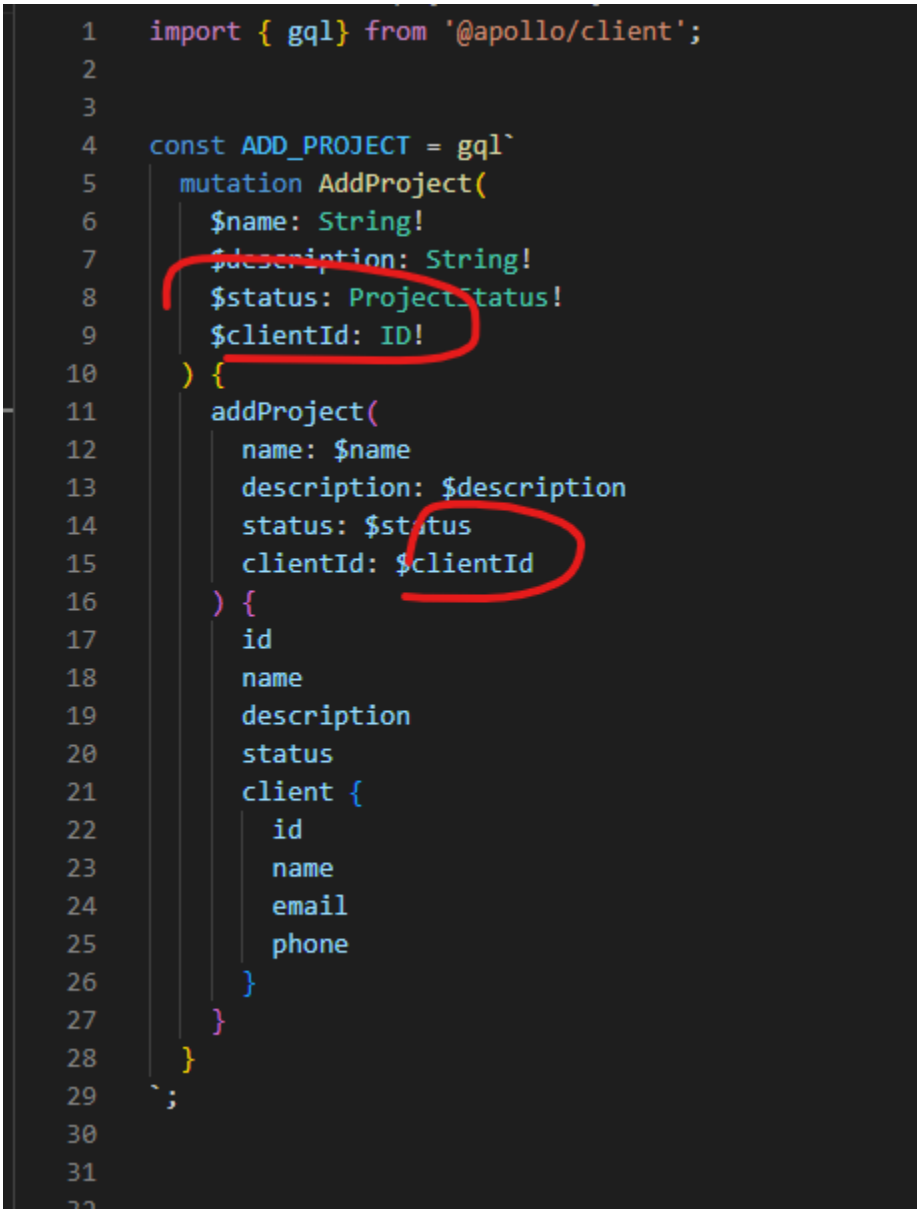
client > src > components > AddProjectModal.jsx > AddProjectModal

```jsx
    addProject(name,description,status)
    setName('');
    setDescription('');
    setStatus('');
   }

   return (
      <>

         <button type="button"
            className="btn btn-primary"
            data-bs-toggle="modal"
            data-bs-target="#addProjectModal">

            <div className="d-flex align-items-center">
               <FaList className='icon' />
               <div>New Project</div>
            </div>

         </button>

         <div className="modal fade" id="addProjectModal" tabIndex="-1" aria-labell
            <div className="modal-dialog">
               <div className="modal-content">
                  <div className="modal-header">
                     <h1 className="modal-title fs-5" id="addProjectModal">New
                     <button type="button" className="btn-close" data-bs-dismis
                  </div>
                  <div className="modal-body">
```


Project Mgmt Graphql
Add Client
New Project
The Bourne ReBooted 2023
View
Status: In Progress
After the success
View
Status: Completed
The Kingston Project
View
Status: Completed
Name
Email
Phone
Lionel Jones
lionel.jones@houstonisd.org
333-333-3333
David Lee Jones
david.jones@optonline.org
444-333-3333
George Clinton
readytheFunk@hotmail.com
000-768-0987


Project Mgmt Graphql
Add Client
New Project
New Project
Name
Description
Status
Not Started
Submit
The Bourne ReBooted 2023
View
Status: In Progress
The Kingston Project
View
Status: Completed
Name
Email
Lionel Jones
David Lee Jones
George Clinton

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



```
import { gql} from '@apollo/client';


const ADD_PROJECT = gql`
  mutation AddProject(
    $name: String!
    $description: String!
    $status: ProjectStatus!
    $clientId: ID!
  ) {
    addProject(
      name: $name
      description: $description
      status: $status
      clientId: $clientId
    ) {
      id
      name
      description
      status
      client {
        id
        name
        email
        phone
      }
    }
  }
`;


```

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,       You, 22 hours ago • added code for viewing cl
            description: args.description,
            status: args.status,
            clientId: args.clientId,
        });

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