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

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)

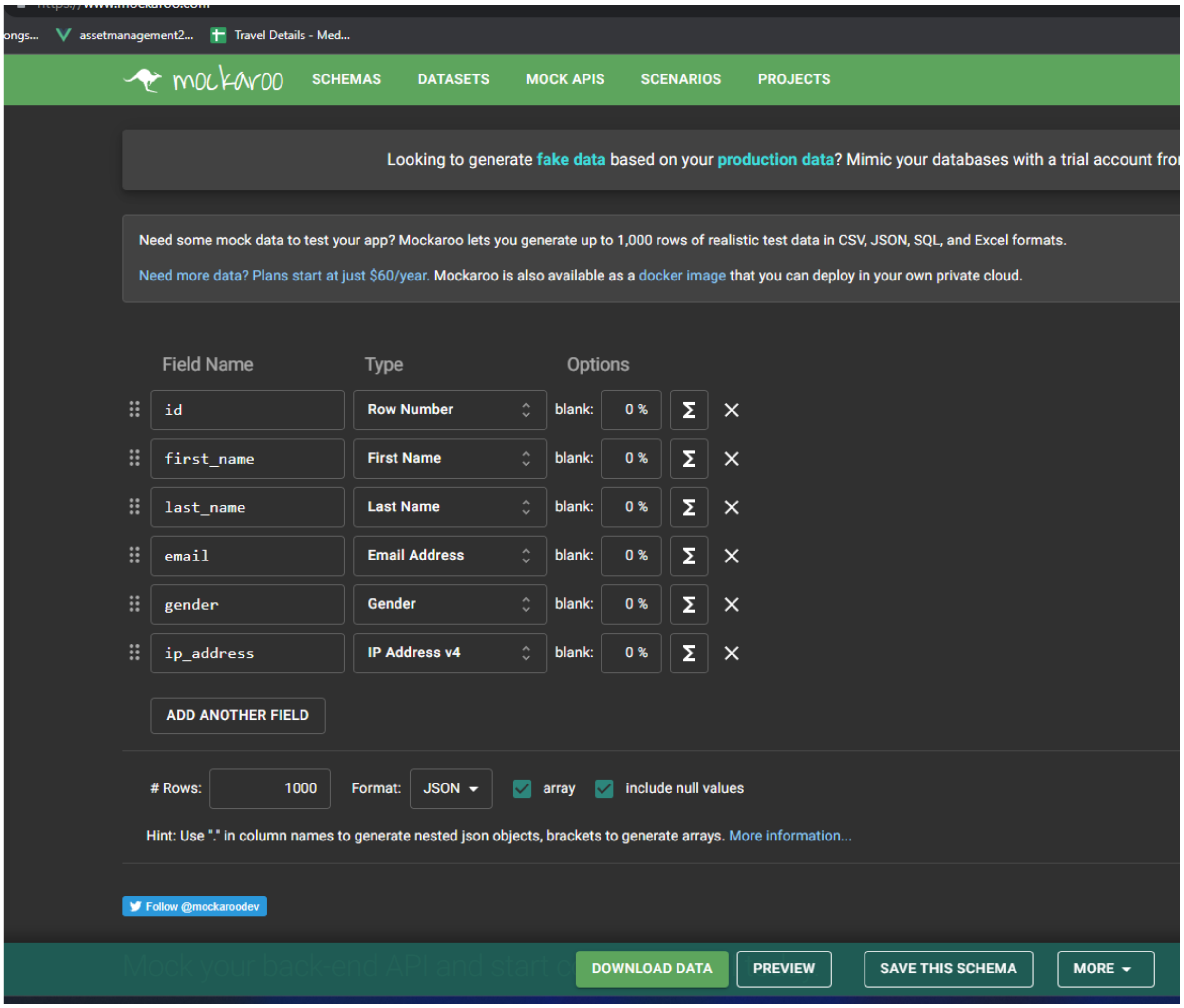
In traditional REST, your endpoints would look like:

/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/



Looking to generate fake data based on your production data? Mimic your databases with a trial account fro

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.

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

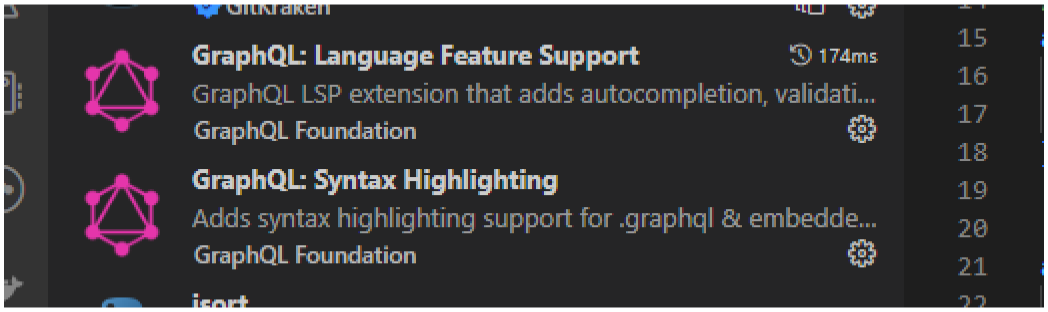
ADD ANOTHER FIELD

Rows: 1000 Format: JSON ✓ array ✓ include null values

Hint: Use "." in column names to generate nested json objects, brackets to generate arrays. More information...

I changed the type to JSON

Add the VSCode Extensions to your VSCODE

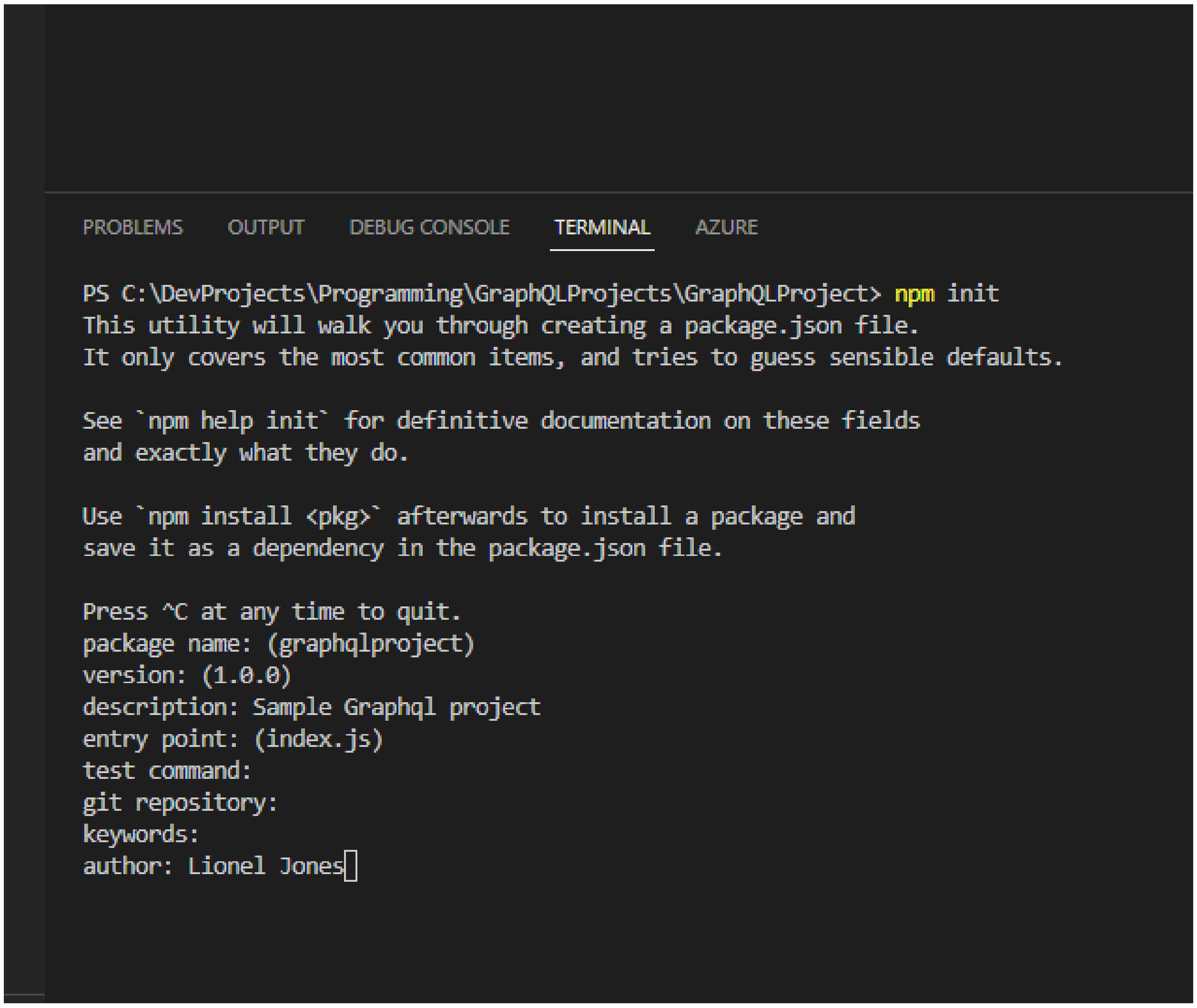

GitKraken
GraphQL: Language Feature Support
174ms
GraphQL LSP extension that adds autocompletion, validati...
GraphQL Foundation
GraphQL: Syntax Highlighting
Adds syntax highlighting support for .graphql & embedde...
GraphQL Foundation
isort

Tip: Creating a sample nodeJS Project Using VSCode

First I create a directory called GraphqlProject

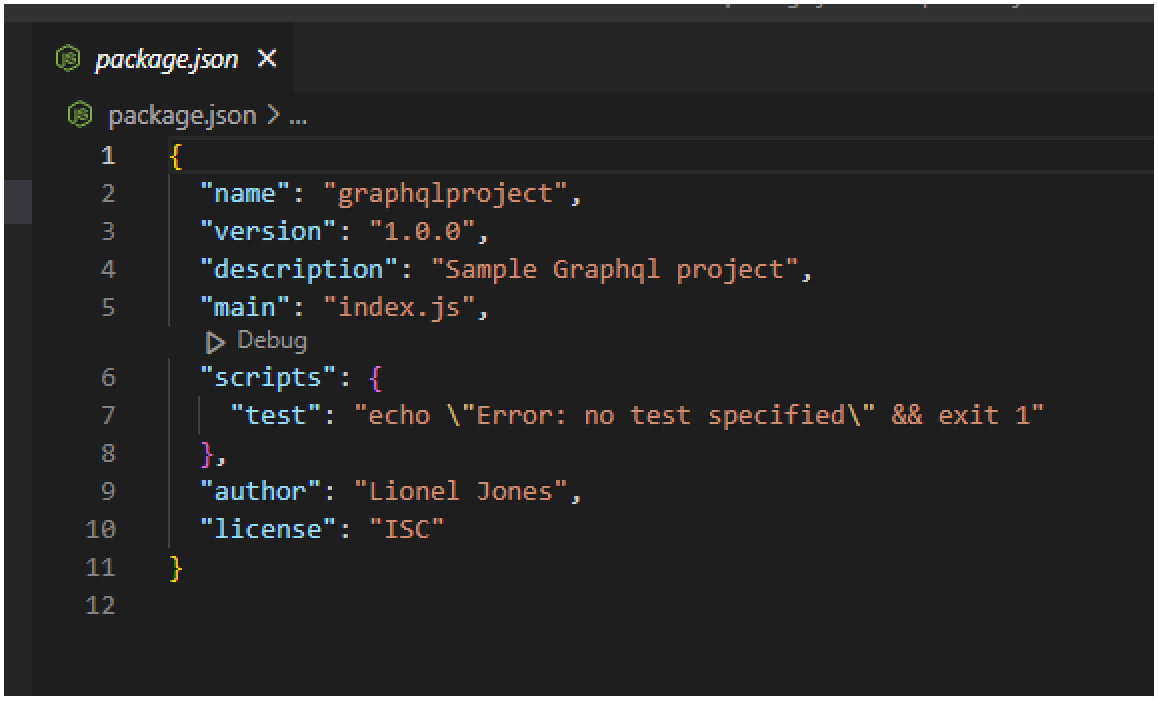
I download the fake data and add it to my directory
Then to initialize the project, I open a new terminal, type
npm init

I just follow the prompts, enter in generic information



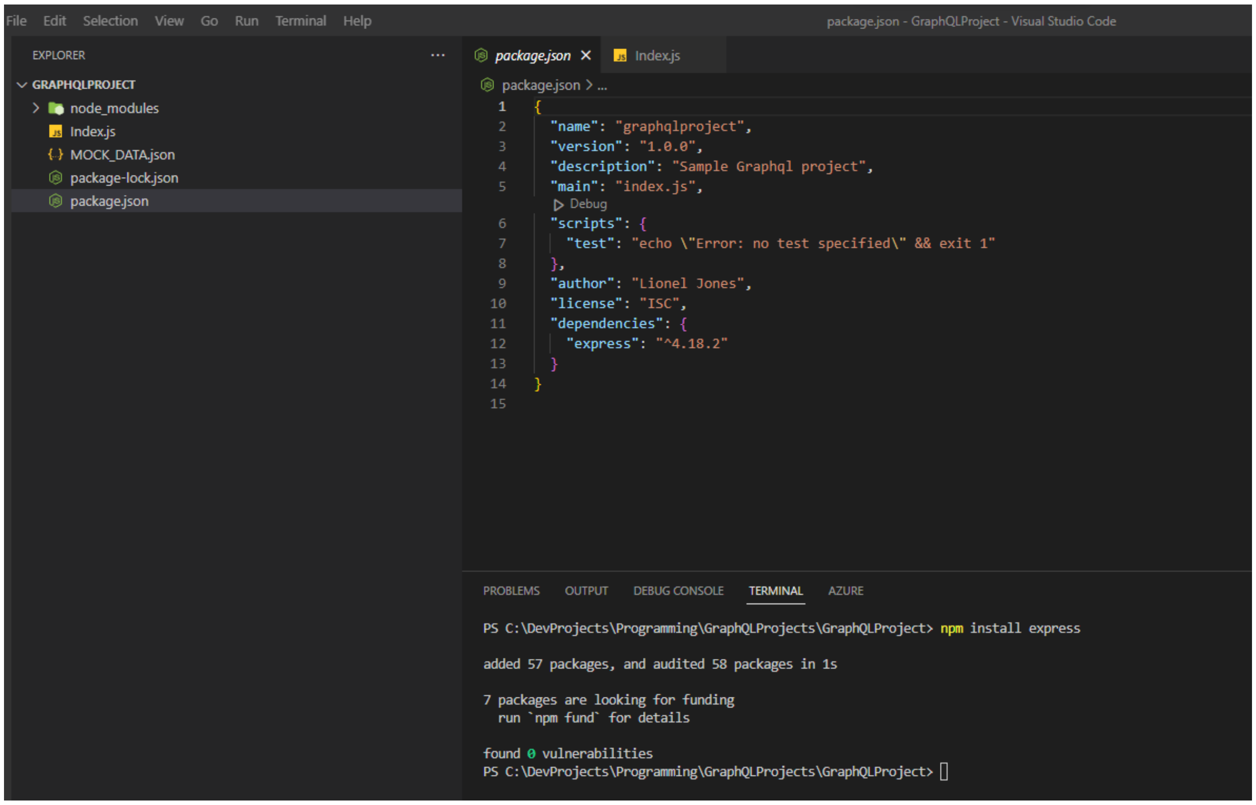
```
PS C:\DevProjects\Programming\GraphQLProjects\GraphQLProject> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

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



```json
{
  "name": "graphqlproject",
  "version": "1.0.0",
  "description": "Sample Graphql project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lionel Jones",
  "license": "ISC"
}
```

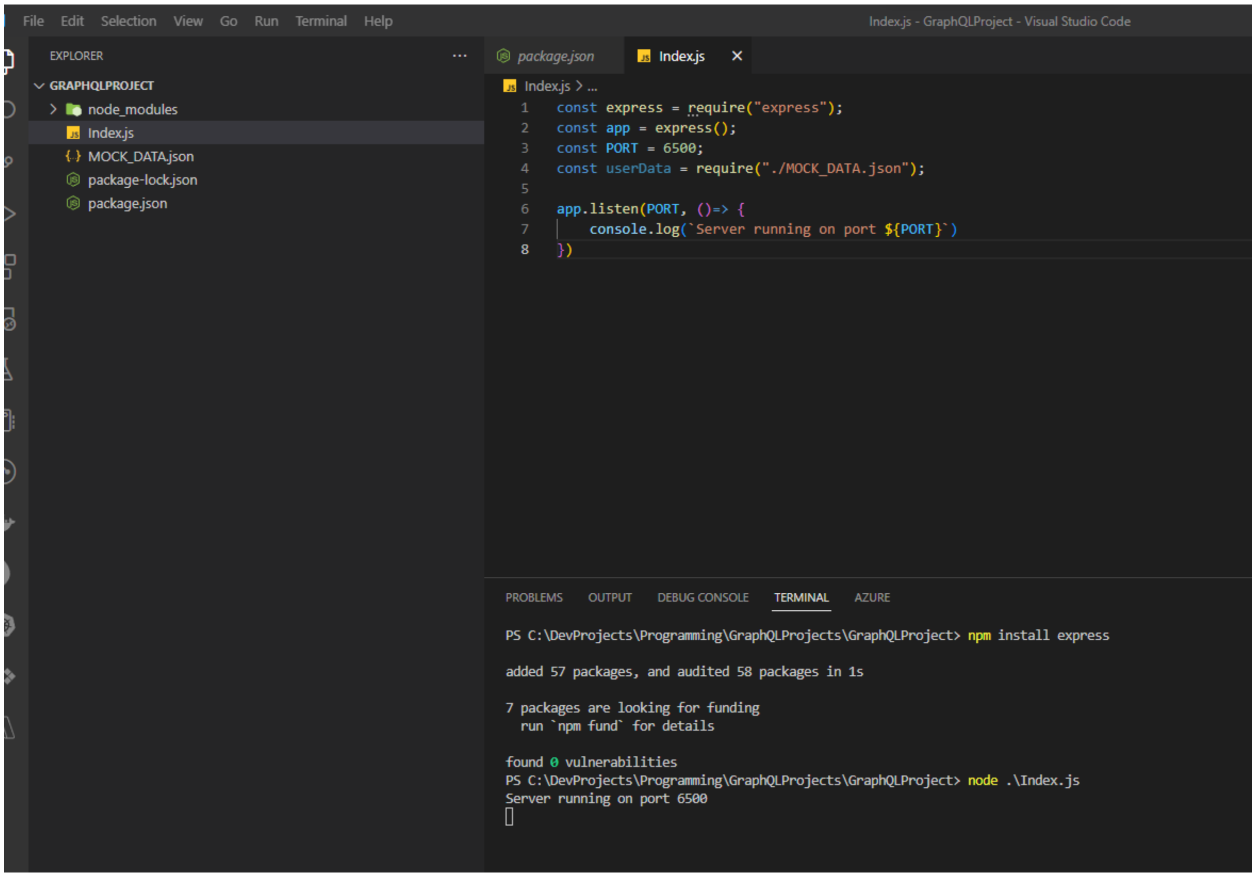
Next, type the following command to install expressjs
npm install express


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
Index.js
Debug
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE

```json
{
  "name": "graphqlproject",
  "version": "1.0.0",
  "description": "Sample Graphql project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lionel Jones",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

```
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

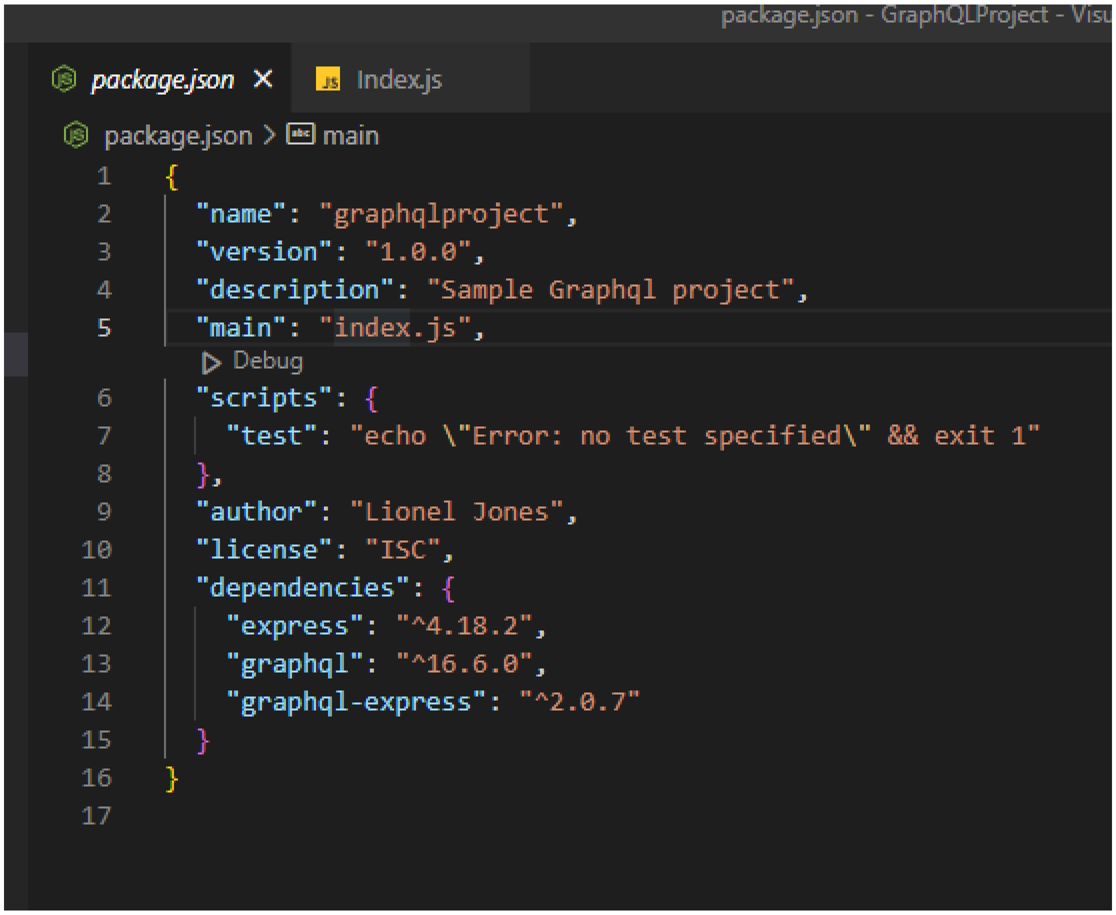

File Edit Selection View Go Run Terminal Help
Index.js - GraphQLProject - Visual Studio Code
EXPLORER
GRAPHQLPROJECT
node_modules
Index.js
MOCK_DATA.json
package-lock.json
package.json
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE

```
const express = require("express");
const app = express();
const PORT = 6500;
const userData = require("./MOCK_DATA.json");

app.listen(PORT, ()=> {
    console.log(`Server running on port ${PORT}`)
})
```

```
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


package.json - GraphQLProject - Visu

package.json ✕ Index.js

package.json > main

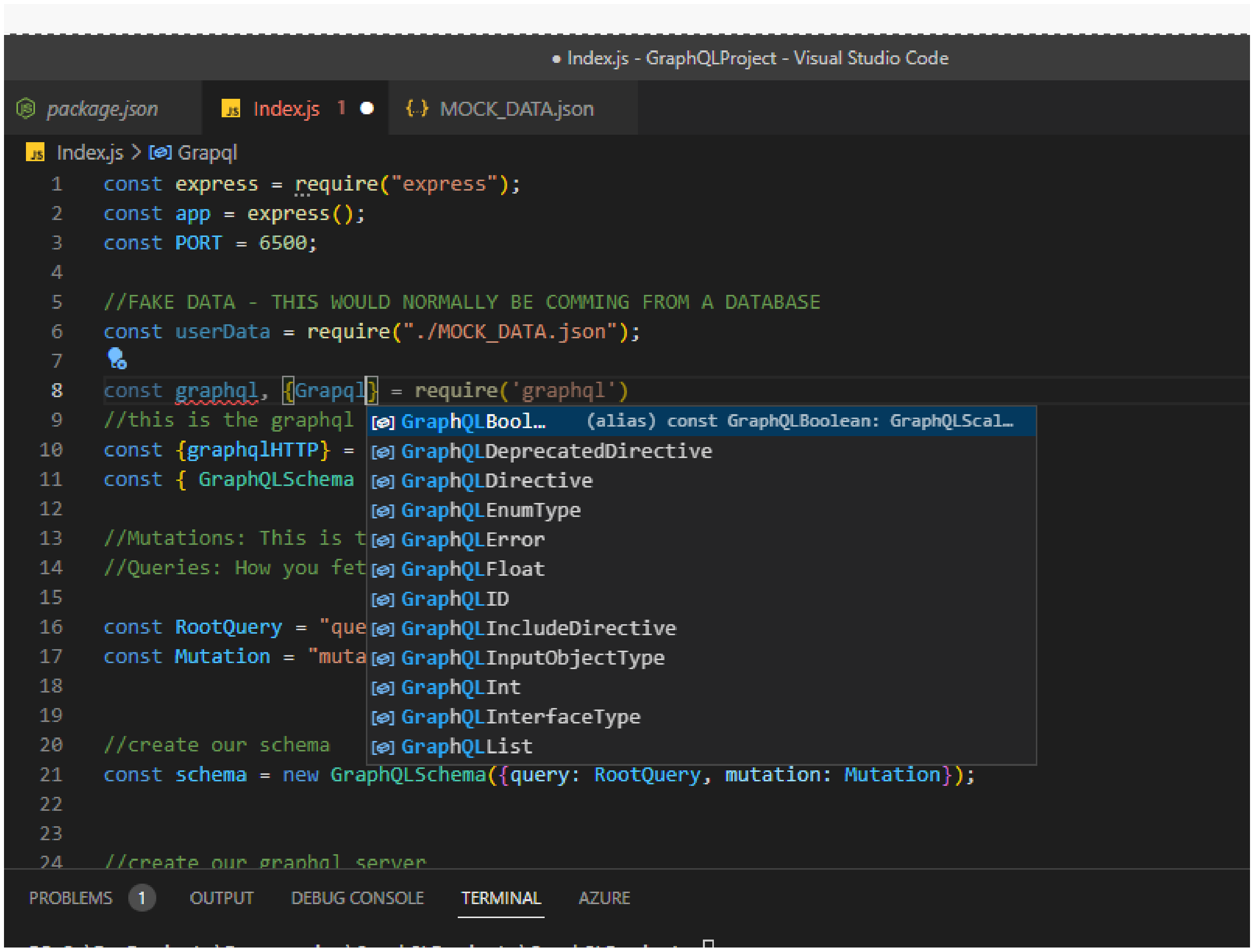
```json
{
  "name": "graphqlproject",
  "version": "1.0.0",
  "description": "Sample Graphql project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lionel Jones",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "graphql": "^16.6.0",
    "graphql-express": "^2.0.7"
  }
}
```

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

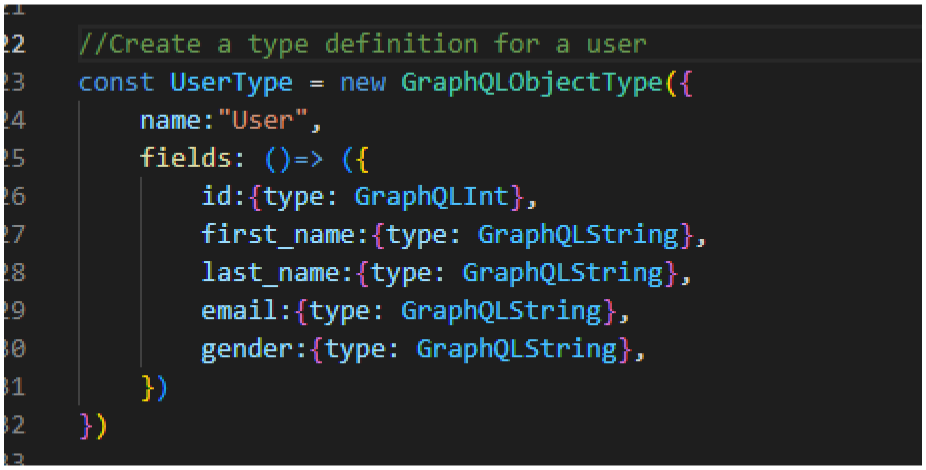
Object types, before we create our schema, we need import the different object types



```javascript
const express = require("express");
const app = express();
const PORT = 6500;

//FAKE DATA - THIS WOULD NORMALLY BE COMMING FROM A DATABASE
const userData = require("./MOCK_DATA.json");

const graphql, {Grapql} = require('graphql')
//this is the graphql
const {graphqlHTTP} =
const { GraphQLSchema

//Mutations: This is t
//Queries: How you fet

const RootQuery = "que
const Mutation = "muta


//create our schema
const schema = new GraphQLSchema({query: RootQuery, mutation: Mutation});


//create our graphql server
```

Creating the type(s):

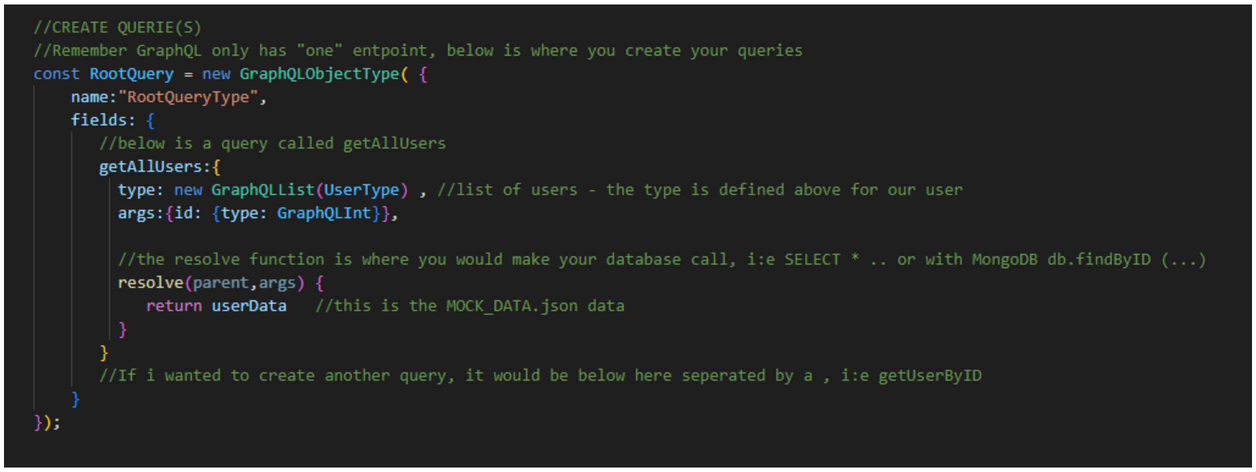
In Graphql, you interact with your data by first defining your type(s)



```javascript
//Create a type definition for a user
const UserType = new GraphQLObjectType({
    name:"User",
    fields: ()=> ({
        id:{type: GraphQLInt},
        first_name:{type: GraphQLString},
        last_name:{type: GraphQLString},
        email:{type: GraphQLString},
        gender:{type: GraphQLString},
    })
})
```

Creating the querie(s)

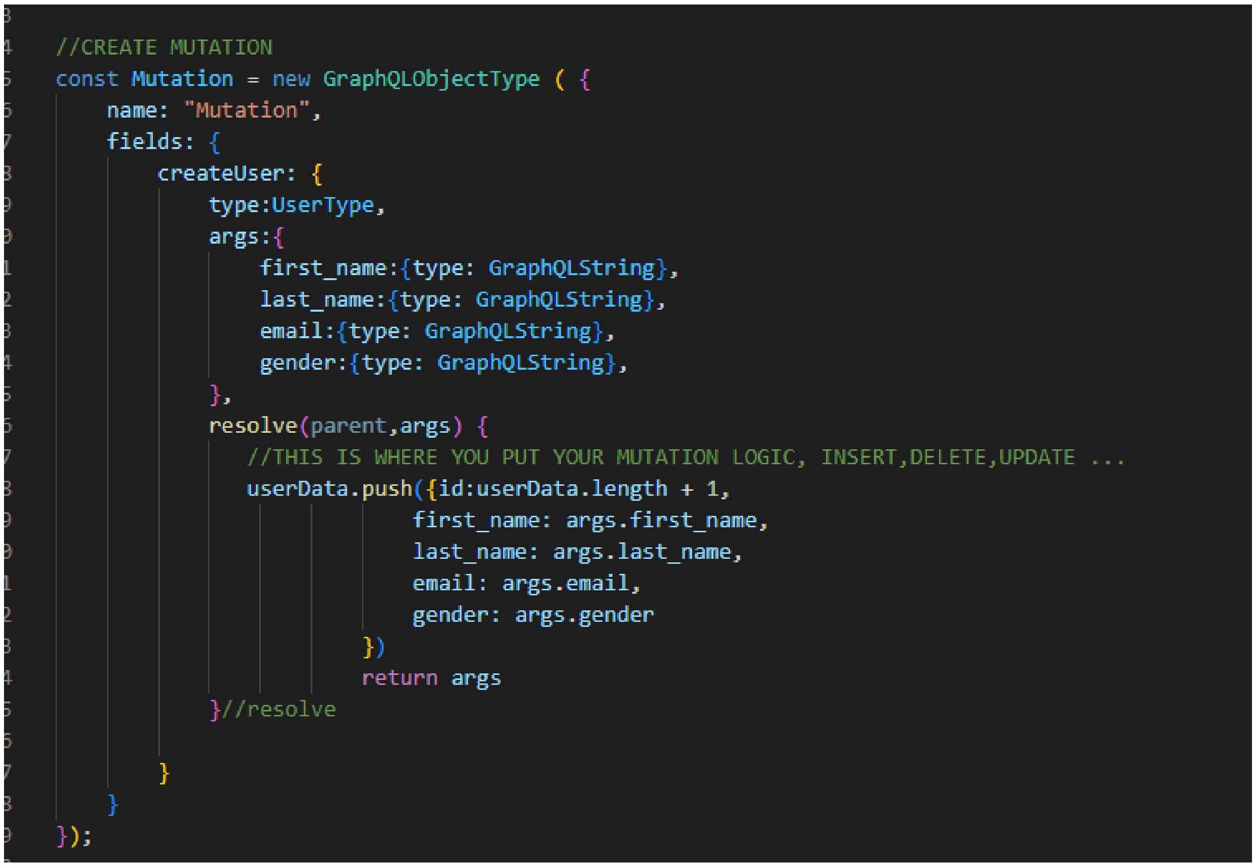
As explained above, you create your mutations (queries as coded below)



```
//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

        }
    }
});
```

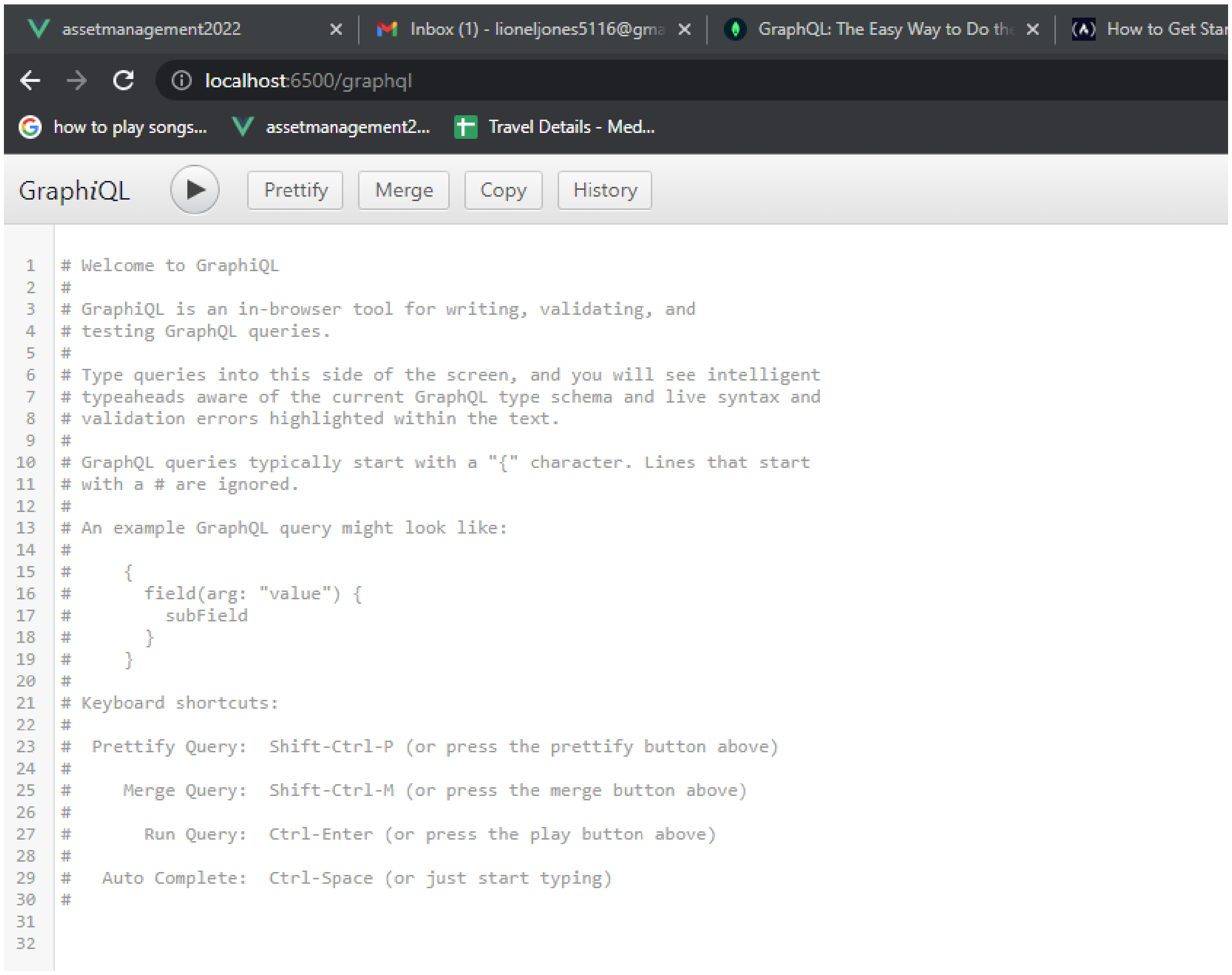
Next, let's fire up our graphql server
Enter the command at the terminal

node index.js

Then in the browser type:

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

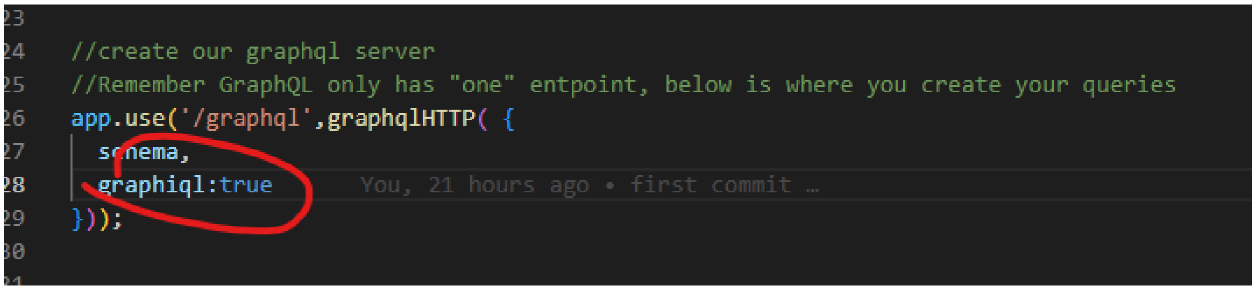
When you do this:



```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that start
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#


```

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

This is surfaced up via this entry:

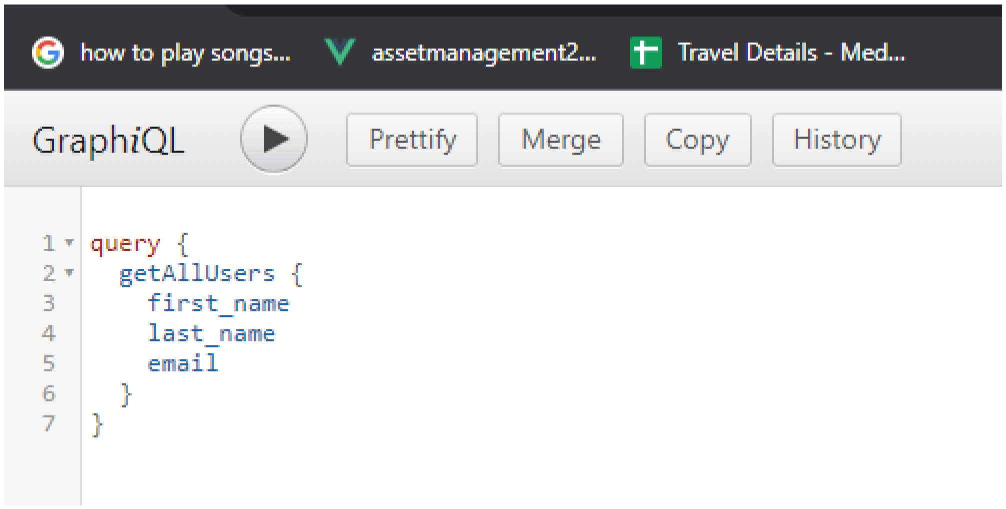


```
//create our graphql server
//Remember GraphQL only has "one" entpoint, below is where you create your queries
app.use('/graphql',graphqlHTTP( {
  schema,
  graphiql:true        You, 21 hours ago • first commit …
}));
```

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

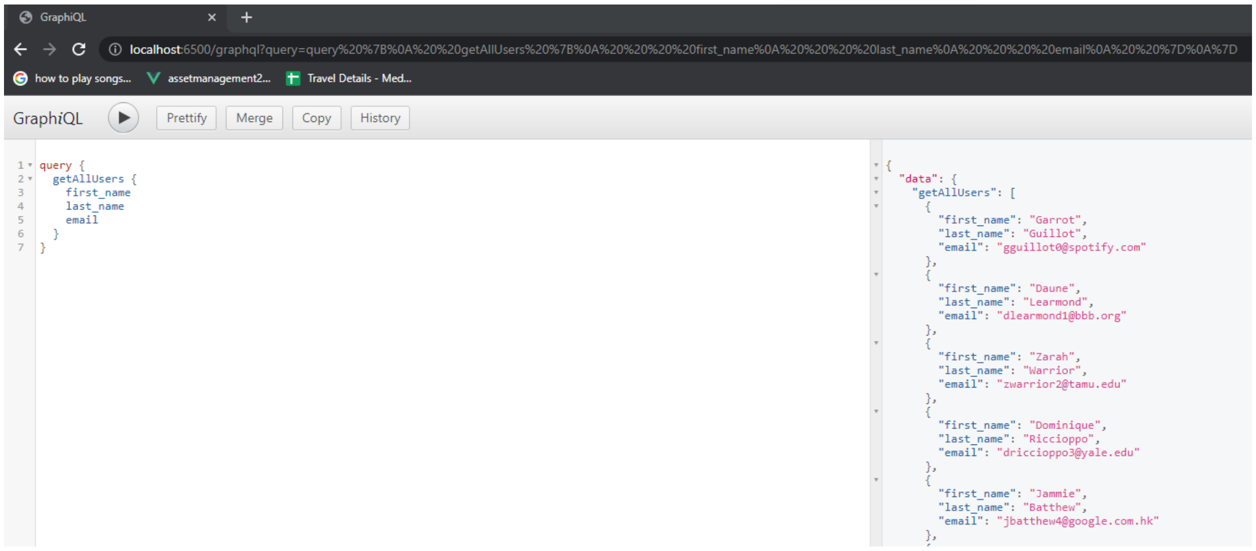
Tip: Querying the data

To fetch some data, just type below



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

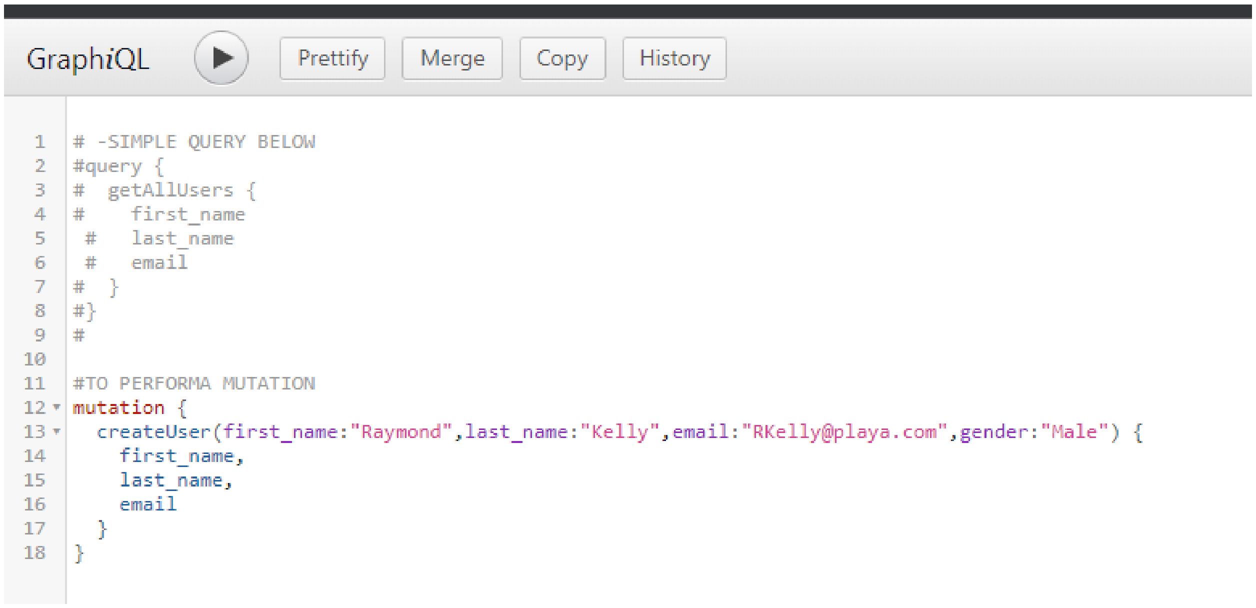
As you type your information inside of the query, it will autocomplete the query you defined in your code.
Then just hit the run button and you will see the results in the right pane


GraphiQL
localhost:6500/graphql?query=query%20%7B%0A%20%20getAllUsers%20%7B%0A%20%20%20%20first_name%0A%20%20%20%20last_name%0A%20%20%20%20email%0A%20%20%7D%0A%7D
how to play songs...
assetmanagement2...
Travel Details - Med...
GraphiQL
Prettify
Merge
Copy
History

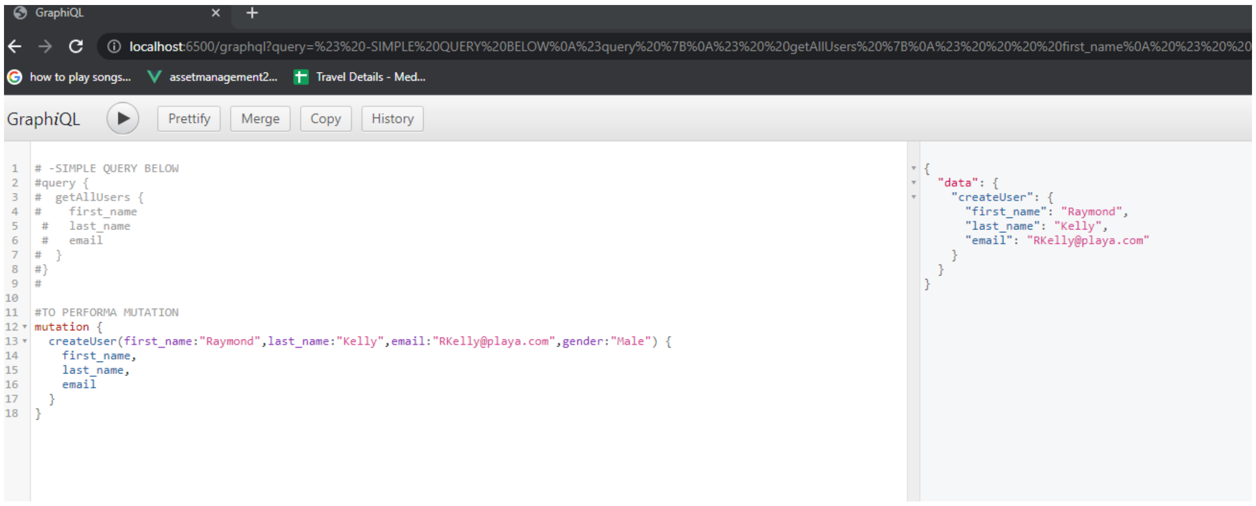
```
query {
  getAllUsers {
    first_name
    last_name
    email
  }
}
```

```
{
  "data": {
    "getAllUsers": [
      {
        "first_name": "Garrot",
        "last_name": "Guillot",
        "email": "gguillot0@spotify.com"
      },
      {
        "first_name": "Daune",
        "last_name": "Learmond",
        "email": "dlearmond1@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:



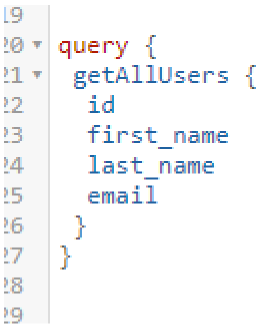
```
# -SIMPLE QUERY BELOW
#query {
#  getAllUsers {
#    first_name
 #    last_name
 #    email
#  }
#}
#

#TO PERFORMA MUTATION
mutation {
  createUser(first_name:"Raymond",last_name:"Kelly",email:"RKelly@playa.com",gender:"Male") {
    first_name,
    last_name,
    email
  }
}
```


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%23%20%20
how to play songs...
assetmanagement2...
Travel Details - Med...
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
  }
}
```

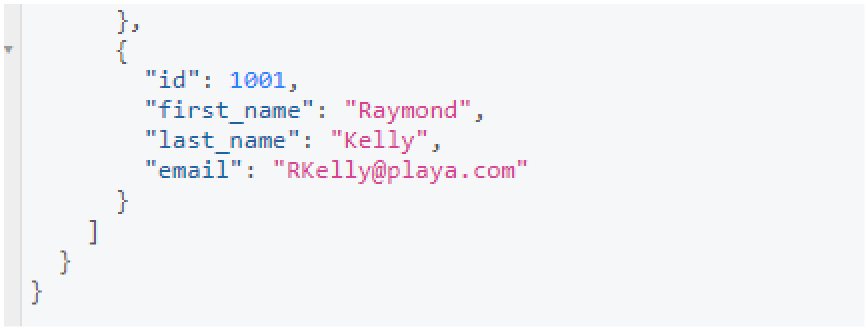
```
{
  "data": {
    "createUser": {
      "first_name": "Raymond",
      "last_name": "Kelly",
      "email": "RKelly@playa.com"
    }
  }
}
```

Run the query:



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

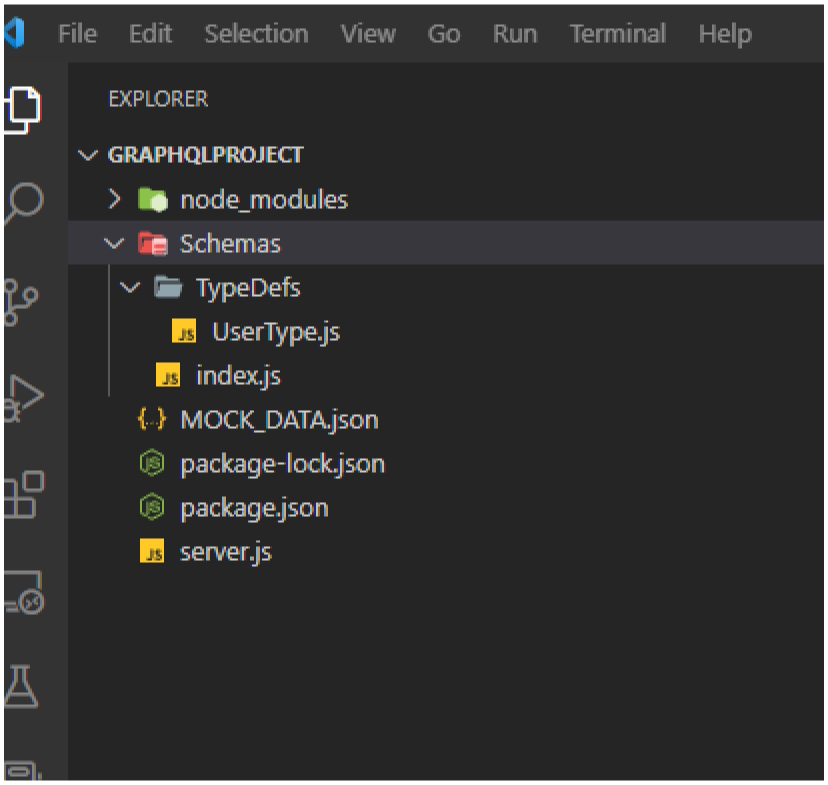
And you will see the item added to the array:



```
        },
        {
          "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

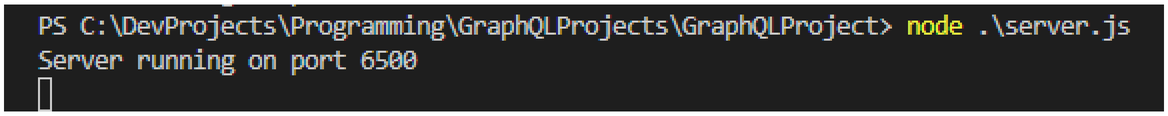
To clean up the project structure:



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

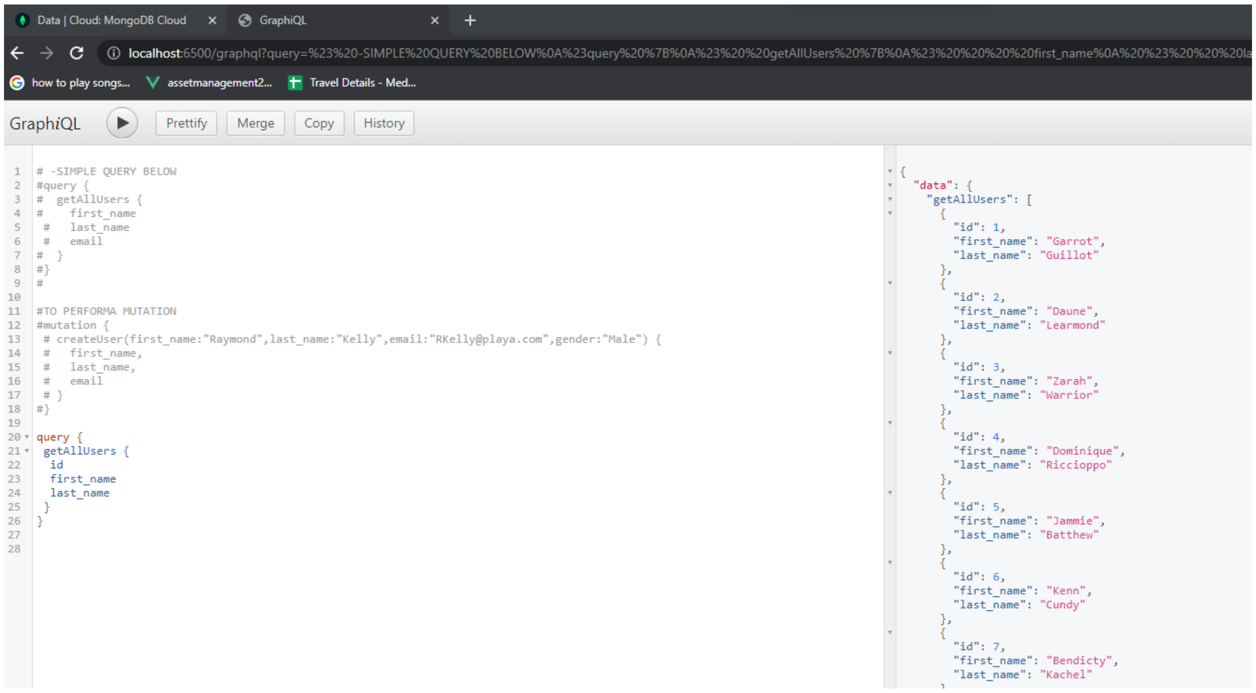
We place our types inside of the typdefs folder
We place our mutations and queries in the index.js file under the schema folder

Then we execute
node server.js



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

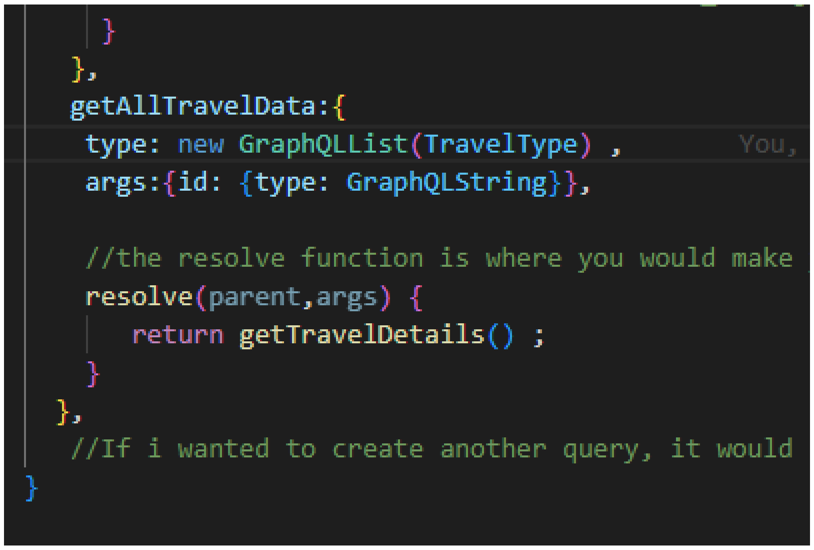
And everything works:


Data | Cloud: MongoDB Cloud
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%23%20%20%20%20la
how to play songs...
assetmanagement2...
Travel Details - Med...
GraphiQL
Prettify
Merge
Copy
History

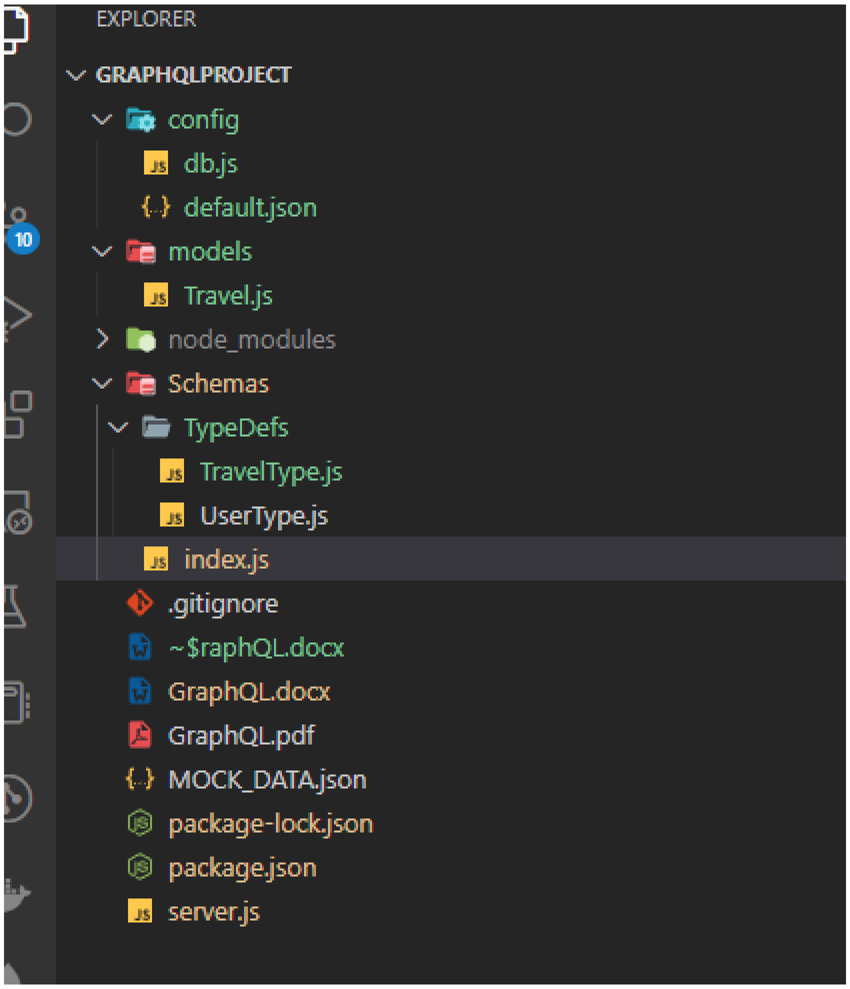
```
# -SIMPLE QUERY BELOW
#query {
#  getAllUsers {
#    first_name
 #   last_name
 #   email
#  }
#}
#

#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
 }
}
```

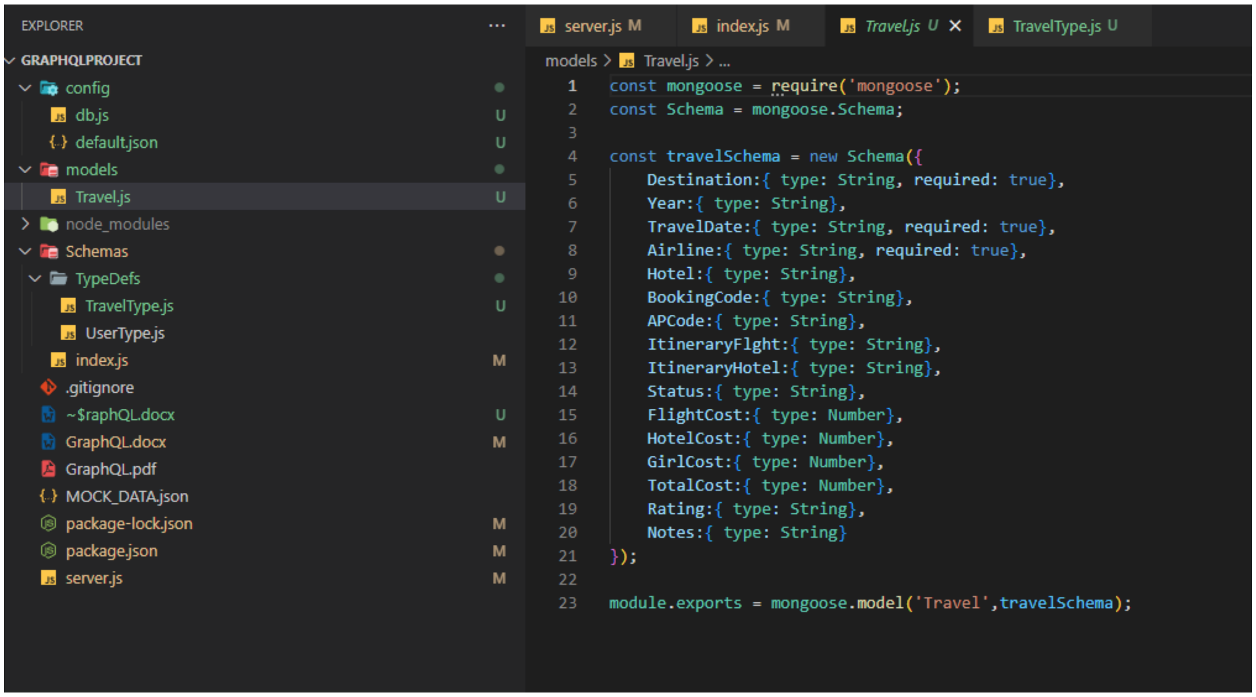
```
{
  "data": {
    "getAllUsers": [
      {
        "id": 1,
        "first_name": "Garrot",
        "last_name": "Guillot"
      },
      {
        "id": 2,
        "first_name": "Daune",
        "last_name": "Learmond"
      },
      {
        "id": 3,
        "first_name": "Zarah",
        "last_name": "Warrior"
      },
      {
        "id": 4,
        "first_name": "Dominique",
        "last_name": "Riccioppo"
      },
      {
        "id": 5,
        "first_name": "Jammie",
        "last_name": "Batthew"
      },
      {
        "id": 6,
        "first_name": "Kenn",
        "last_name": "Cundy"
      },
      {
        "id": 7,
        "first_name": "Bendicty",
        "last_name": "Kachel"
```

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.

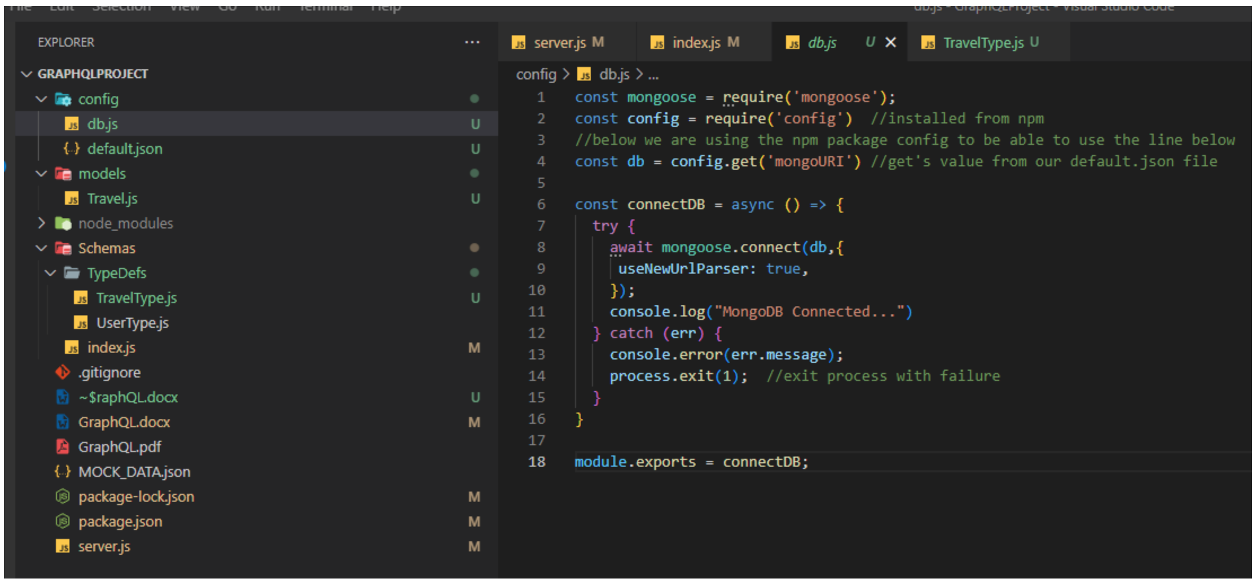


```
      }
    },
    getAllTravelData:{
     type: new GraphQLList(TravelType) ,          You,
     args:{id: {type: GraphQLString}},

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

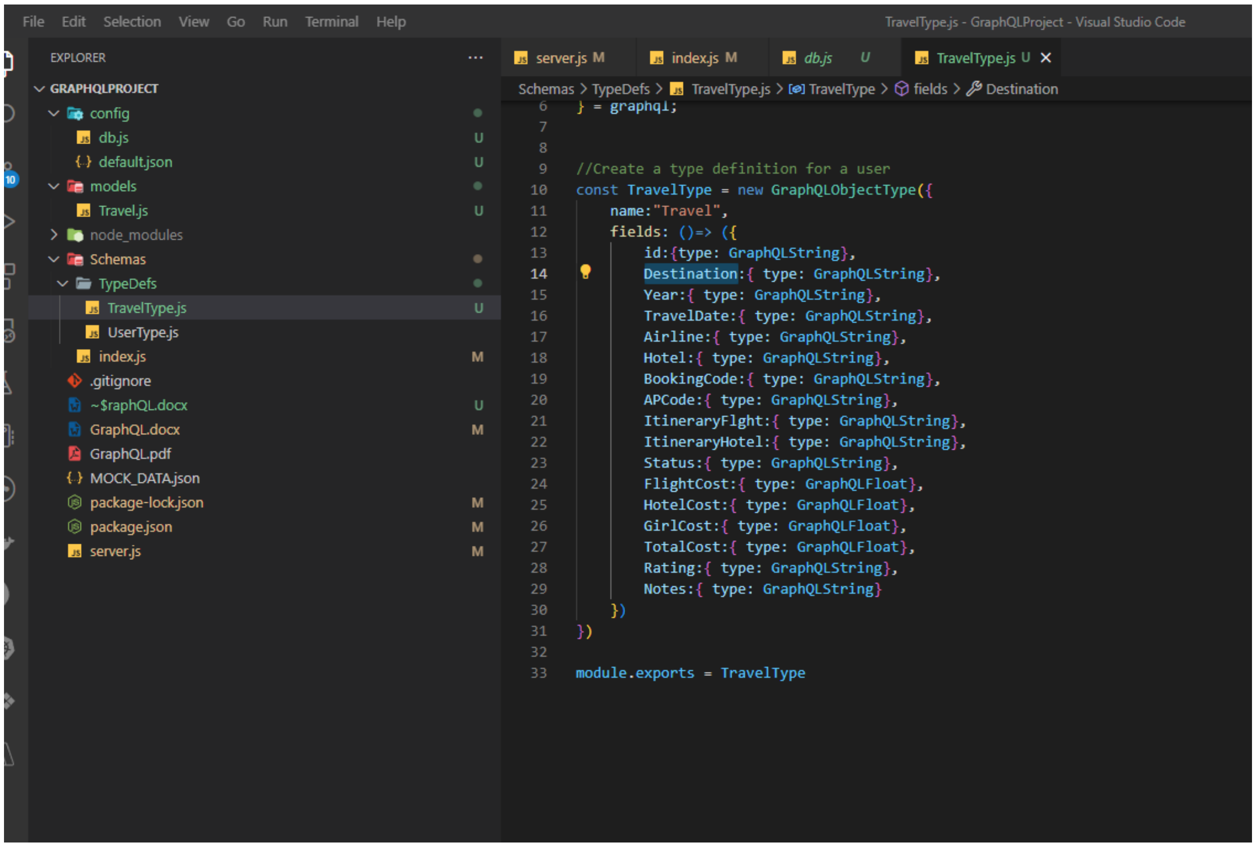




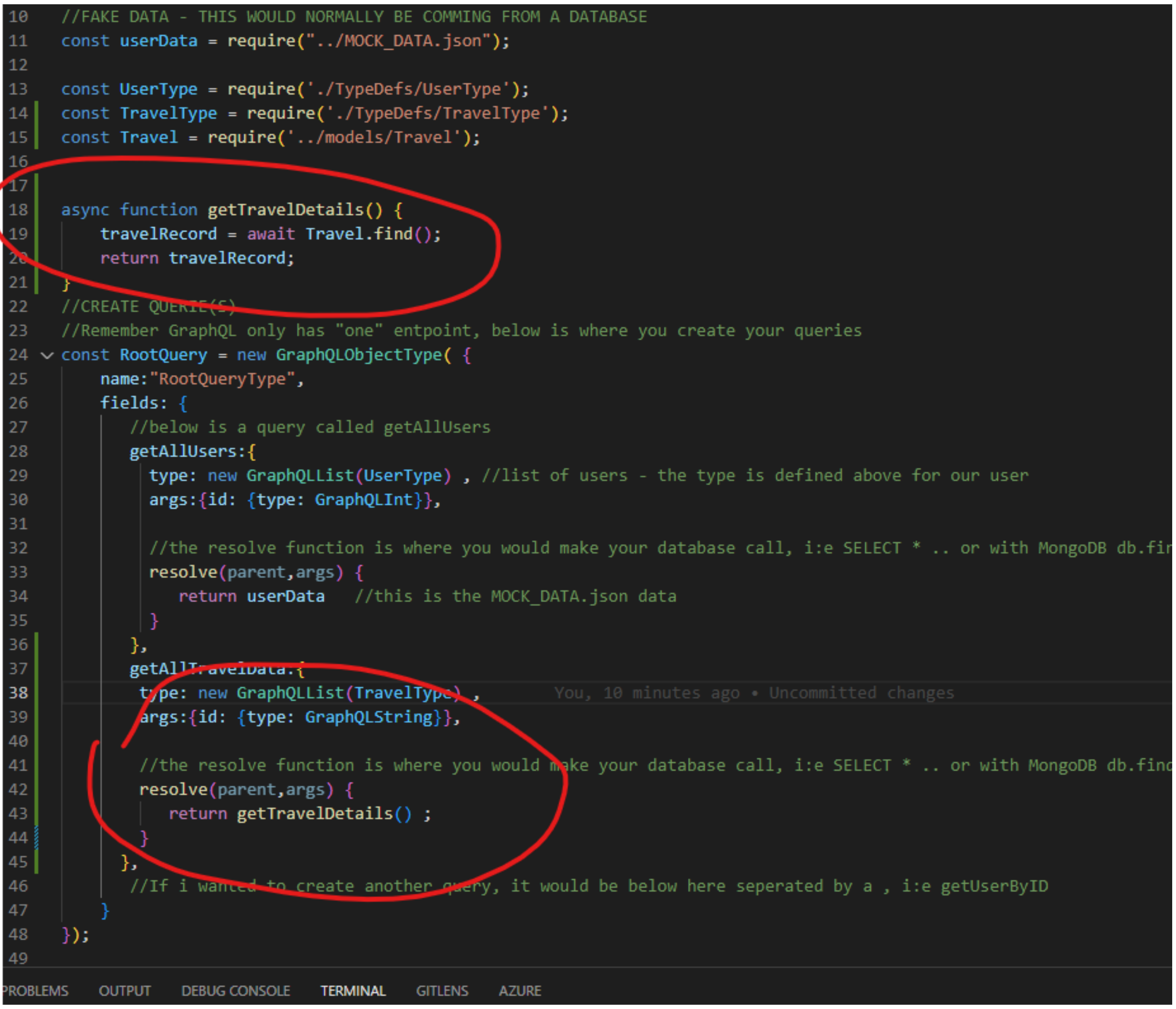
```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

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);
```



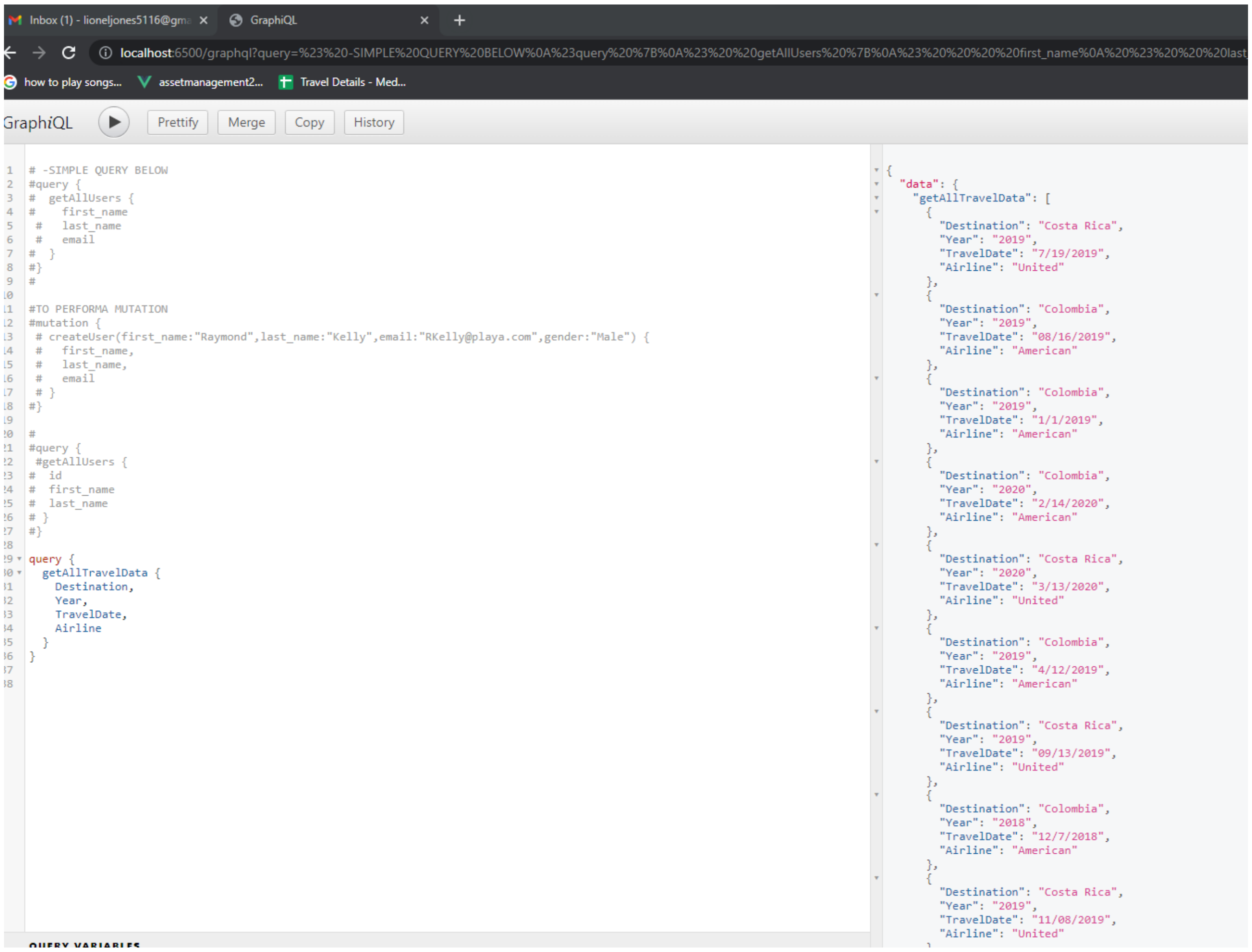
```
const mongoose = require('mongoose');
const config = require('config')  //installed from npm
//below we are using the npm package config to be able to use the line below
const db = config.get('mongoURI') //get's value from our default.json file

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;
```



```
} = 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.fi
         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.fin
        resolve(parent,args) {
           return getTravelDetails() ;
        }
      },
       //If i wanted to create another query, it would be below here seperated by a , i:e getUserByID
    }
});

```


Inbox (1) - lioneljones5116@gma
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%23%20%20%20last
how to play songs...
assetmanagement2...
Travel Details - Med...
GraphiQL
Prettify
Merge
Copy
History
QUERY VARIABLES

```
# -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

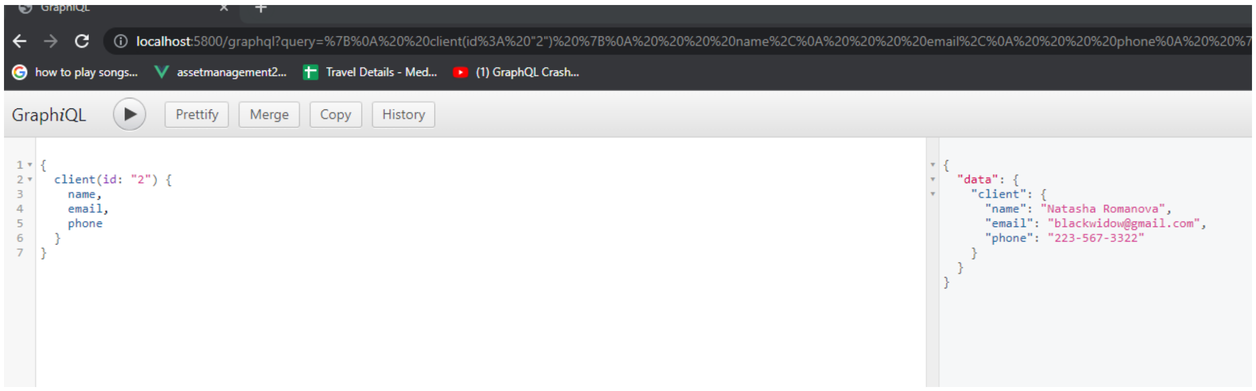
Install the packages below
npm i express express-graphql graphql mongoose cors colors –force

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

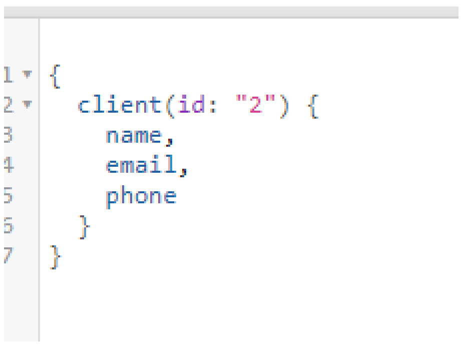
We make a call after we have wired up our code:

http://localhost:5800/graphql

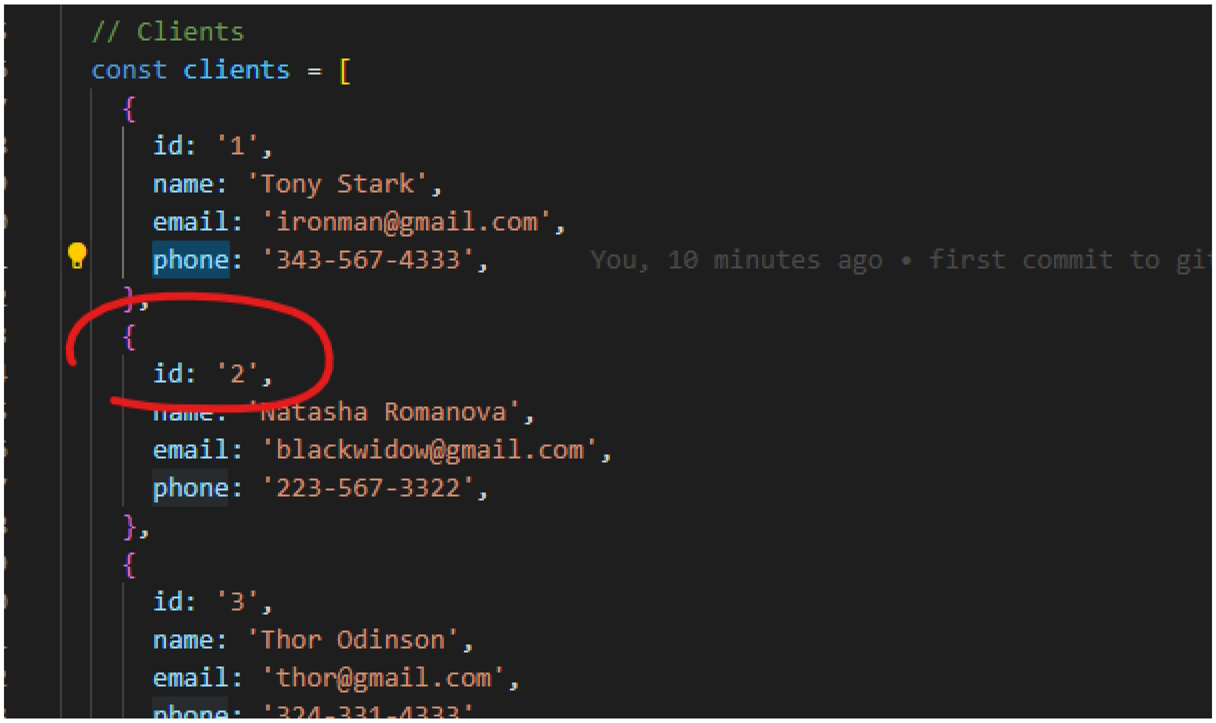

GraphiQL
localhost:5800/graphql?query=%7B%0A%20%20client(id%3A%20"2")%20%7B%0A%20%20%20%20name%2C%0A%20%20%20%20email%2C%0A%20%20%20%20phone%0A%20%20%20%20%7
how to play songs...
assetmanagement2...
Travel Details - Med...
(1) GraphQL Crash...
GraphiQL
Prettify
Merge
Copy
History

```
{
  client(id: "2") {
    name,
    email,
    phone
  }
}
```

```
{
  "data": {
    "client": {
      "name": "Natasha Romanova",
      "email": "blackwidow@gmail.com",
      "phone": "223-567-3322"
    }
  }
}
```

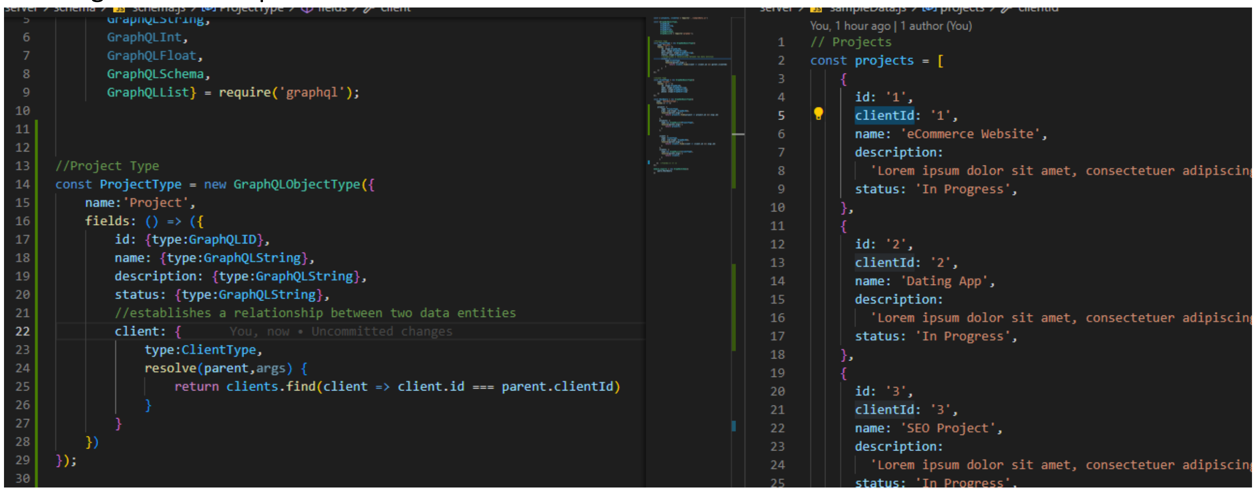


```
{
  client(id: "2") {
    name,
    email,
    phone
  }
}
```



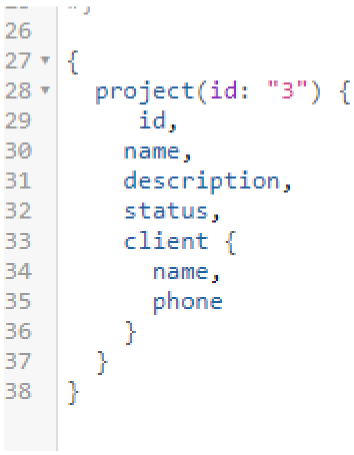
```
// Clients
const clients = [
  {
    id: '1',
    name: 'Tony Stark',
    email: 'ironman@gmail.com',
    phone: '343-567-4333',      You, 10 minutes ago • first commit to gi
  },
  {
    id: '2',
    name: 'Natasha Romanova',
    email: 'blackwidow@gmail.com',
    phone: '223-567-3322',
  },
  {
    id: '3',
    name: 'Thor Odinson',
    email: 'thor@gmail.com',
    phone: '324-331-4333'
```

Creating a relationship between two entities:

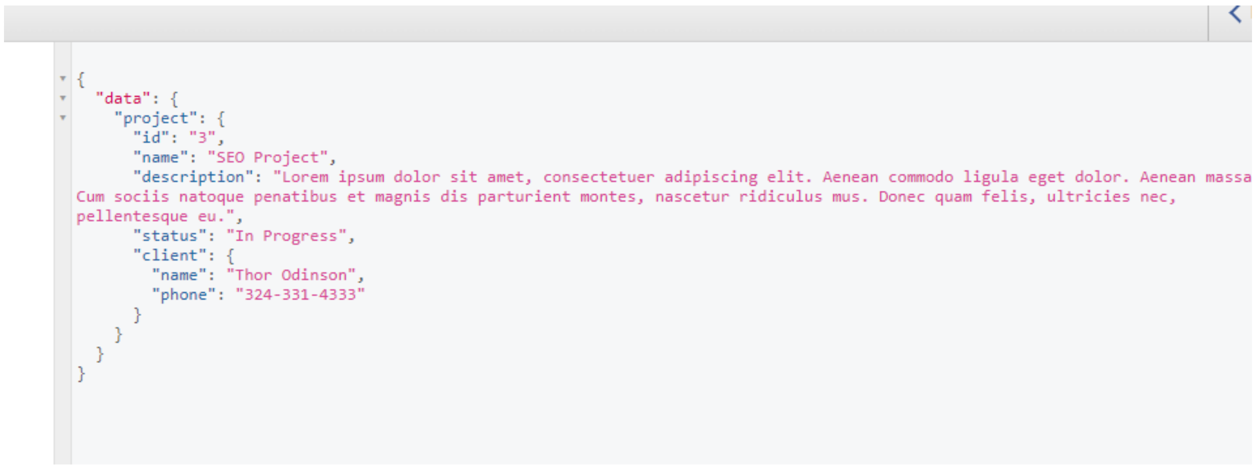


```
       GraphQLString,
       GraphQLInt,
       GraphQLFloat,
       GraphQLSchema,
       GraphQLList} = require('graphql');



//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: {       You, now • Uncommitted changes
            type:ClientType,
            resolve(parent,args) {
                return clients.find(client => client.id === parent.clientId)
            }
        }
    })
});

```

```
You, 1 hour ago | 1 author (You)
// Projects
const projects = [
    {
      id: '1',
      clientId: '1',
      name: 'eCommerce Website',
      description:
        'Lorem ipsum dolor sit amet, consectetuer adipiscing
      status: 'In Progress',
    },
    {
      id: '2',
      clientId: '2',
      name: 'Dating App',
      description:
        'Lorem ipsum dolor sit amet, consectetuer adipiscing
      status: 'In Progress',
    },
    {
      id: '3',
      clientId: '3',
      name: 'SEO Project',
      description:
        'Lorem ipsum dolor sit amet, consectetuer adipiscing
      status: 'In Progress',
```

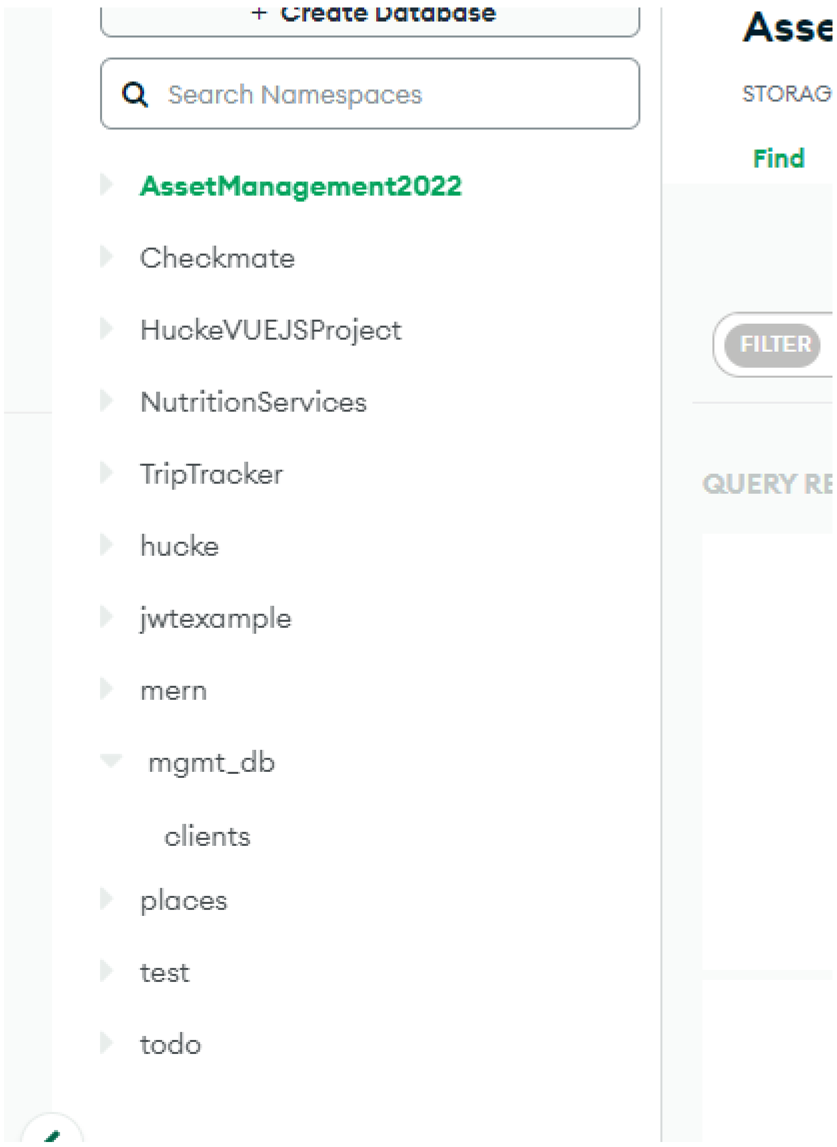


```
{
  project(id: "3") {
     id,
    name,
    description,
    status,
    client {
      name,
      phone
    }
  }
}
```



```
{
  "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


+ Create Database

Search Namespaces

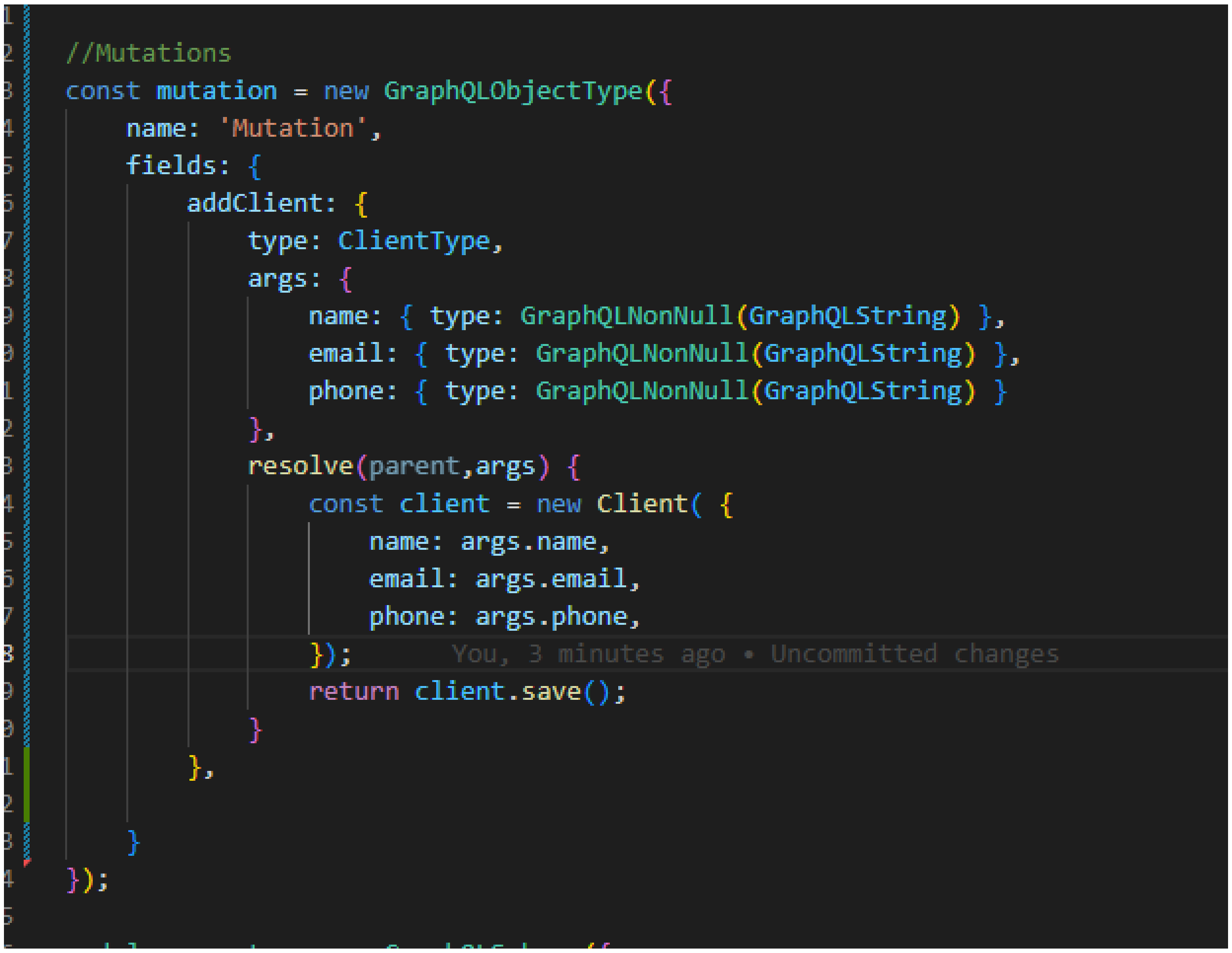
AssetManagement2022
Checkmate
HuckeVUEJSProject
NutritionServices
TripTracker
hucke
jwtexample
mern
mgmt_db
clients
places
test
todo

Asse
STORAG
Find
FILTER
QUERY RE

See the tip below about MongoDB Atlas GUI (Free)

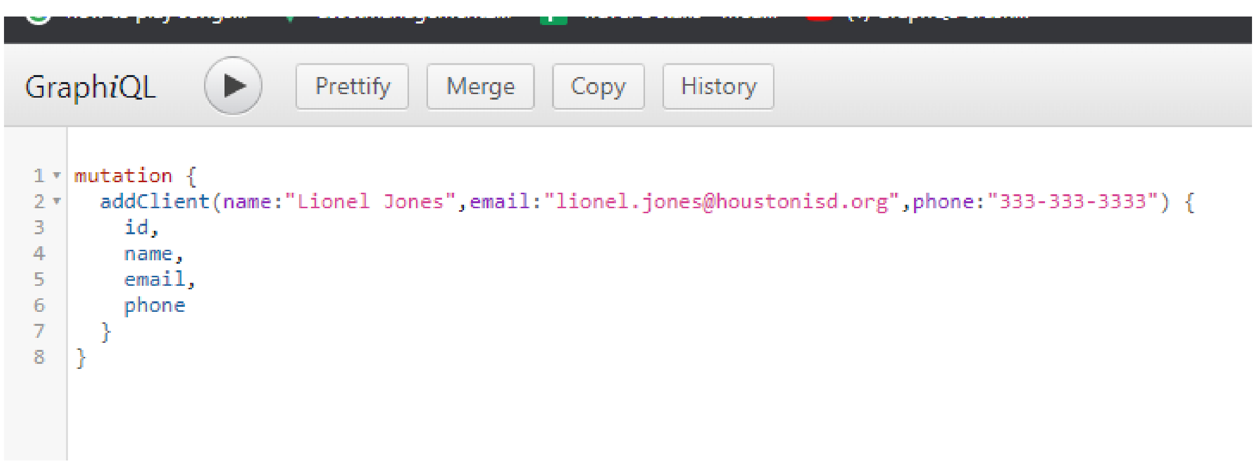
Tip: Adding a record to MongoDB using GraphQL

This is done via a mutation

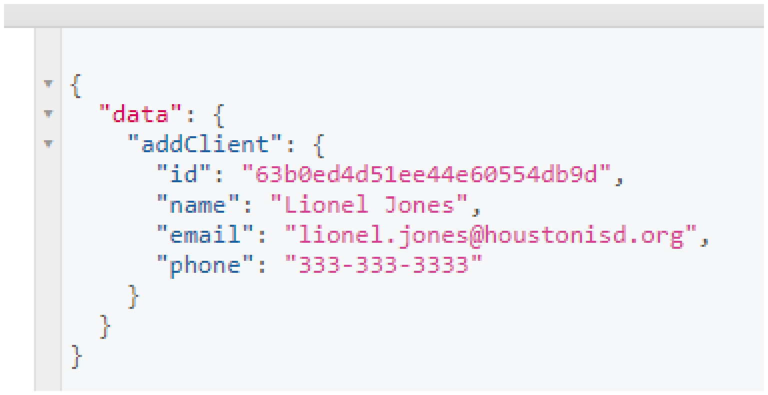


```
//Mutations
const mutation = new GraphQLObjectType({
    name: 'Mutation',
    fields: {
        addClient: {
            type: ClientType,
            args: {
                name: { type: GraphQLNonNull(GraphQLString) },
                email: { type: GraphQLNonNull(GraphQLString) },
                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
  }
}
```

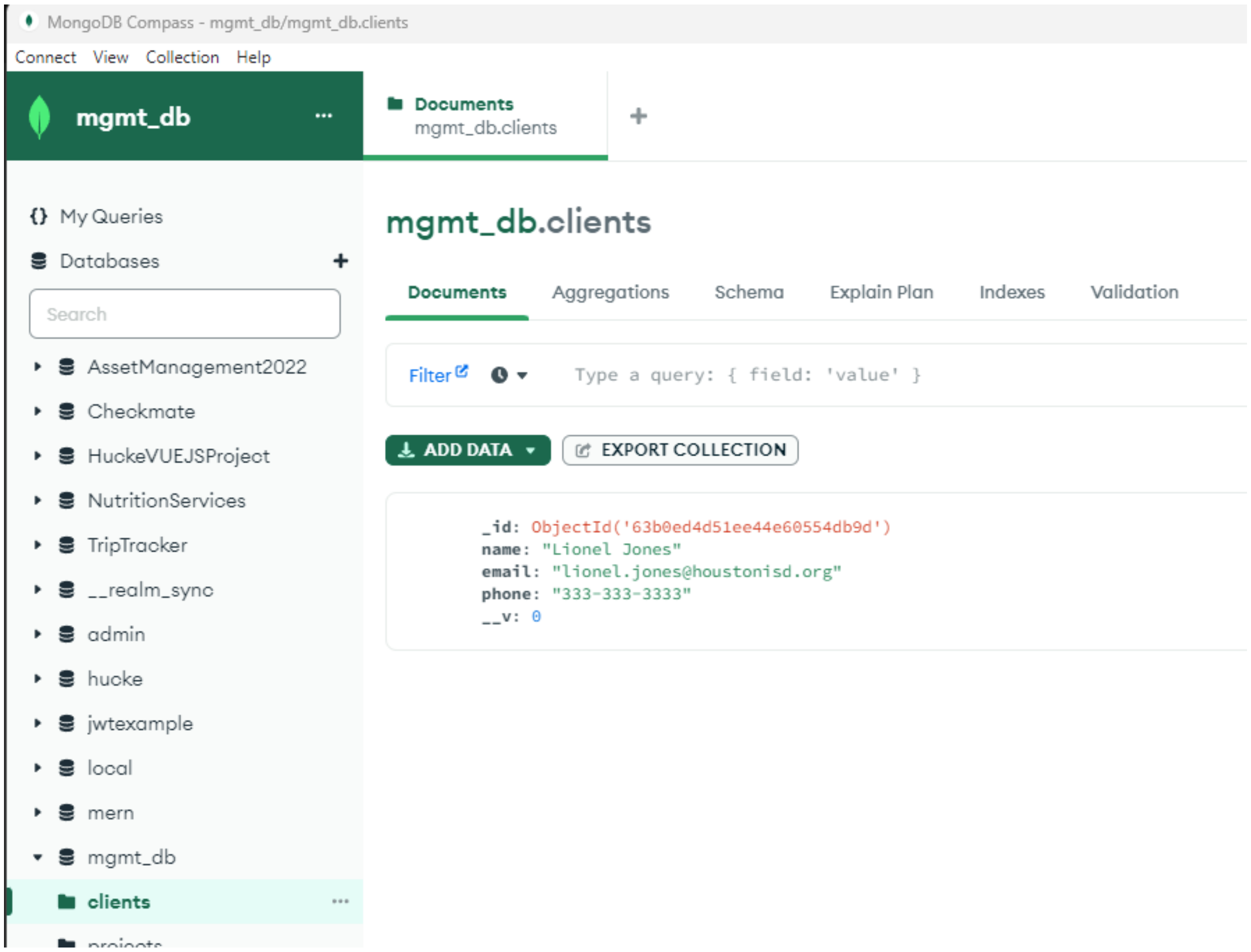


```
mutation {
  addClient(name:"Lionel Jones",email:"lionel.jones@houstonisd.org",phone:"333-333-3333") {
    id,
    name,
    email,
    phone
  }
}
```



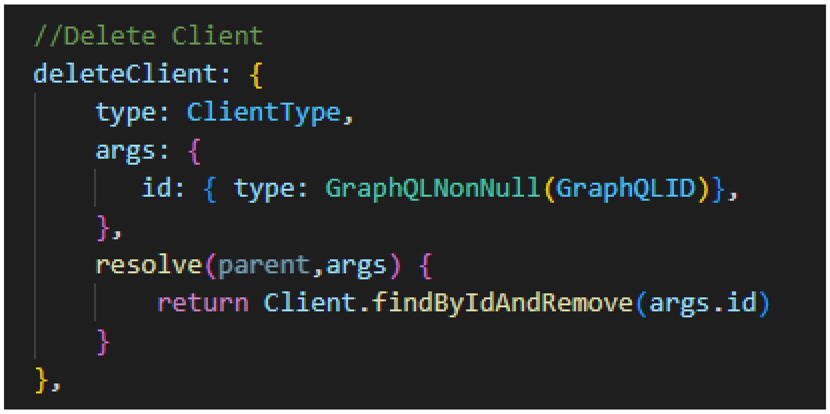
```
{
  "data": {
    "addClient": {
      "id": "63b0ed4d51ee44e60554db9d",
      "name": "Lionel Jones",
      "email": "lionel.jones@houstonisd.org",
      "phone": "333-333-3333"
    }
  }
}
```

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

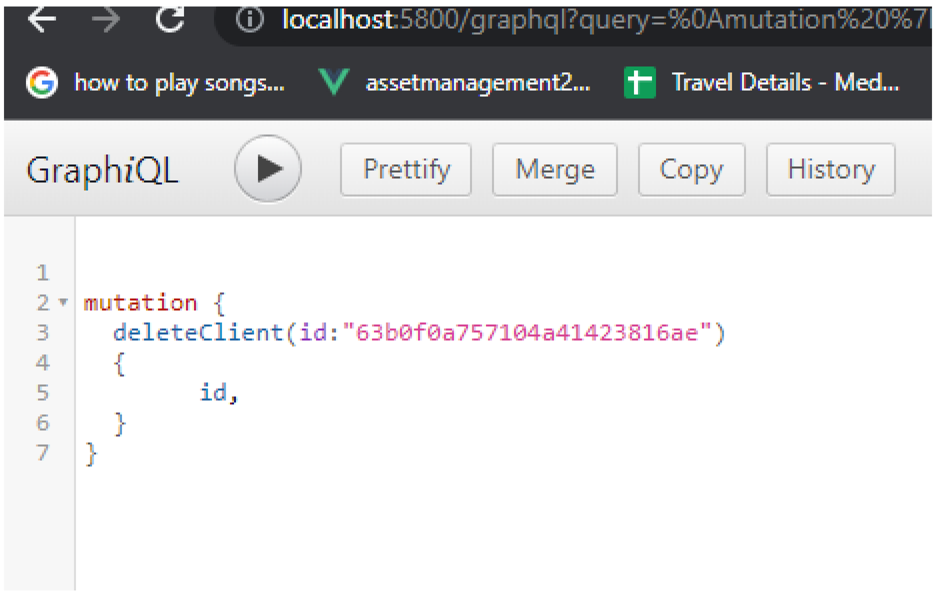


```
_id: ObjectId('63b0ed4d51ee44e60554db9d')
name: "Lionel Jones"
email: "lionel.jones@houstonisd.org"
phone: "333-333-3333"
__v: 0
```

To delete a client



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



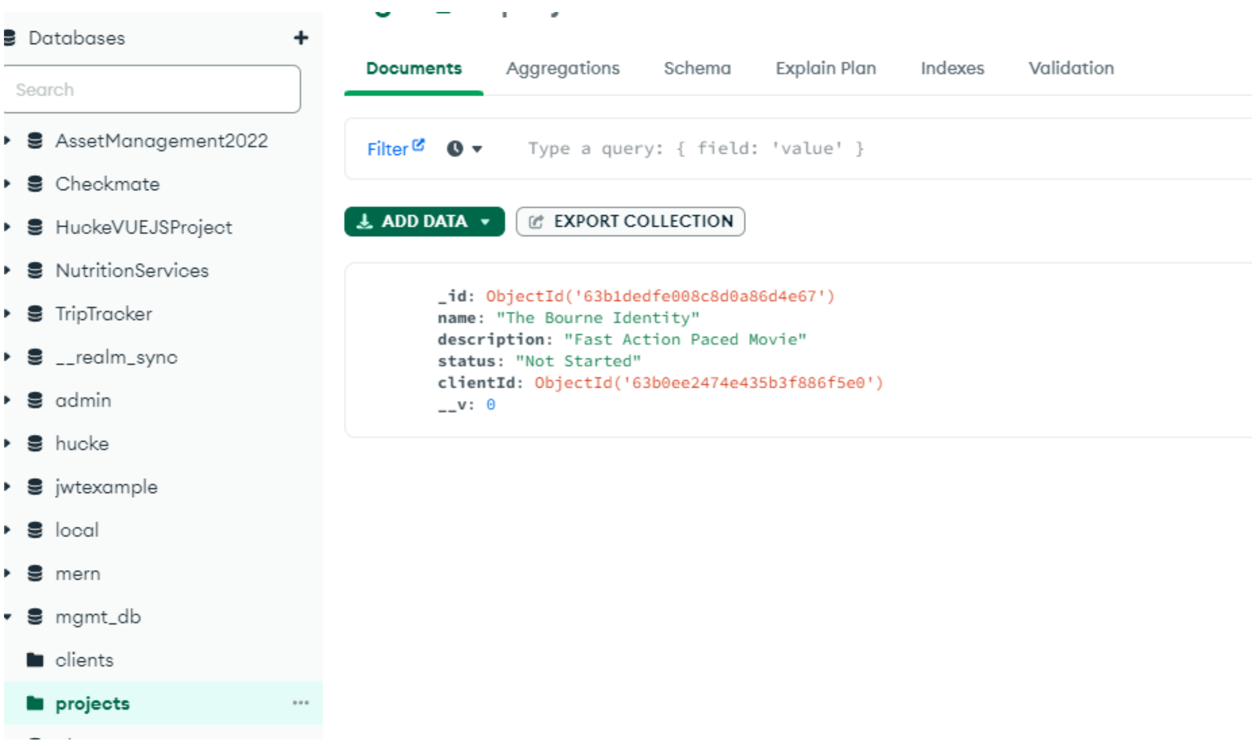
```

mutation {
  deleteClient(id:"63b0f0a757104a41423816ae")
  {
        id,
  }
}
```

Stopped at 1:02 minutes – 12/31/2022

Mutation for Project Add

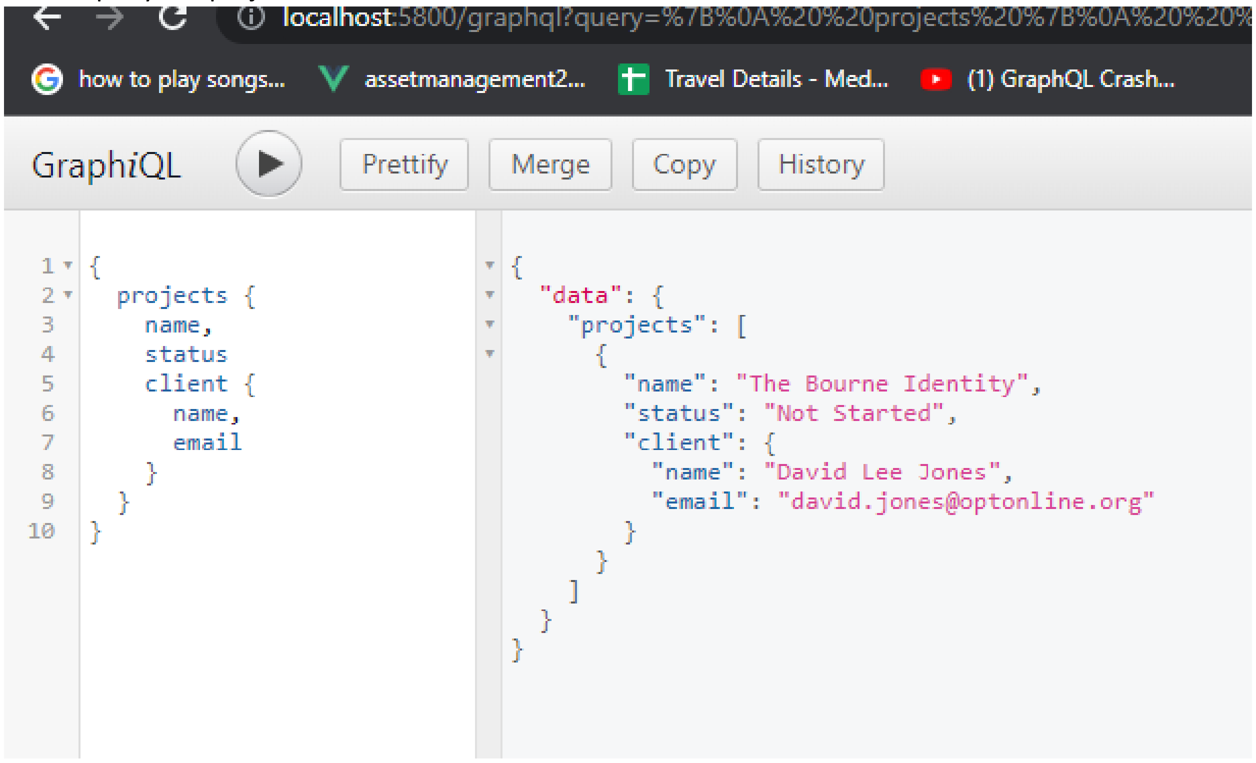
```
mutation {
 addProject(clientId:"63b0ee2474e435b3f886f5e0",
           name:"The Bourne Identity",
           description:"Fast Action Paced Movie",
           status: new) {
    name,
    id,
    description,
    status

 }
}
```

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



```
_id: ObjectId('63b1dedfe008c8d0a86d4e67')
name: "The Bourne Identity"
description: "Fast Action Paced Movie"
status: "Not Started"
clientId: ObjectId('63b0ee2474e435b3f886f5e0')
__v: 0
```

The to query the projects

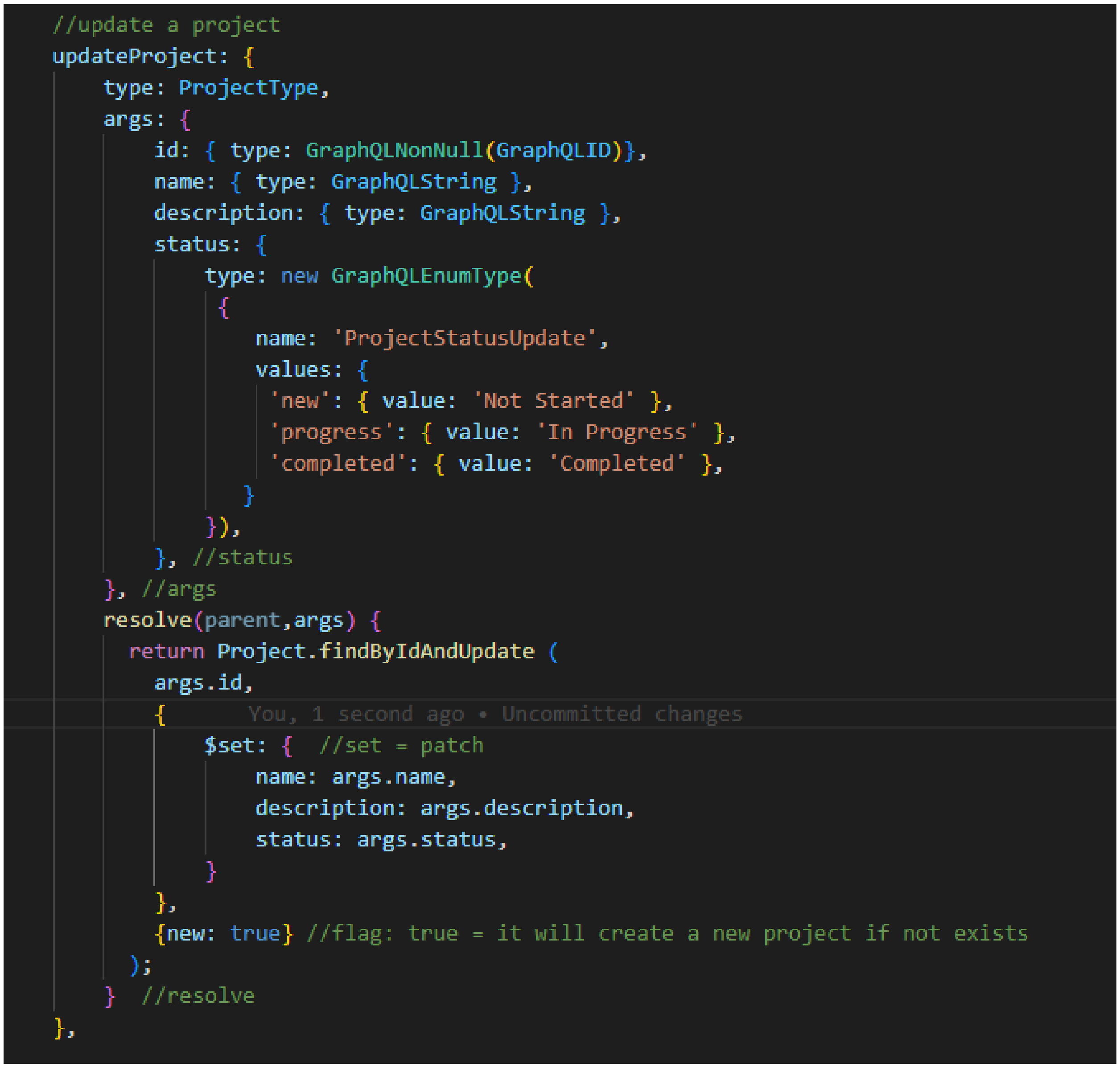


```
{
  projects {
    name,
    status
    client {
      name,
      email
    }
  }
}
```

```
{
  "data": {
    "projects": [
      {
        "name": "The Bourne Identity",
        "status": "Not Started",
        "client": {
          "name": "David Lee Jones",
          "email": "david.jones@optonline.org"
        }
      }
    ]
  }
}
```

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

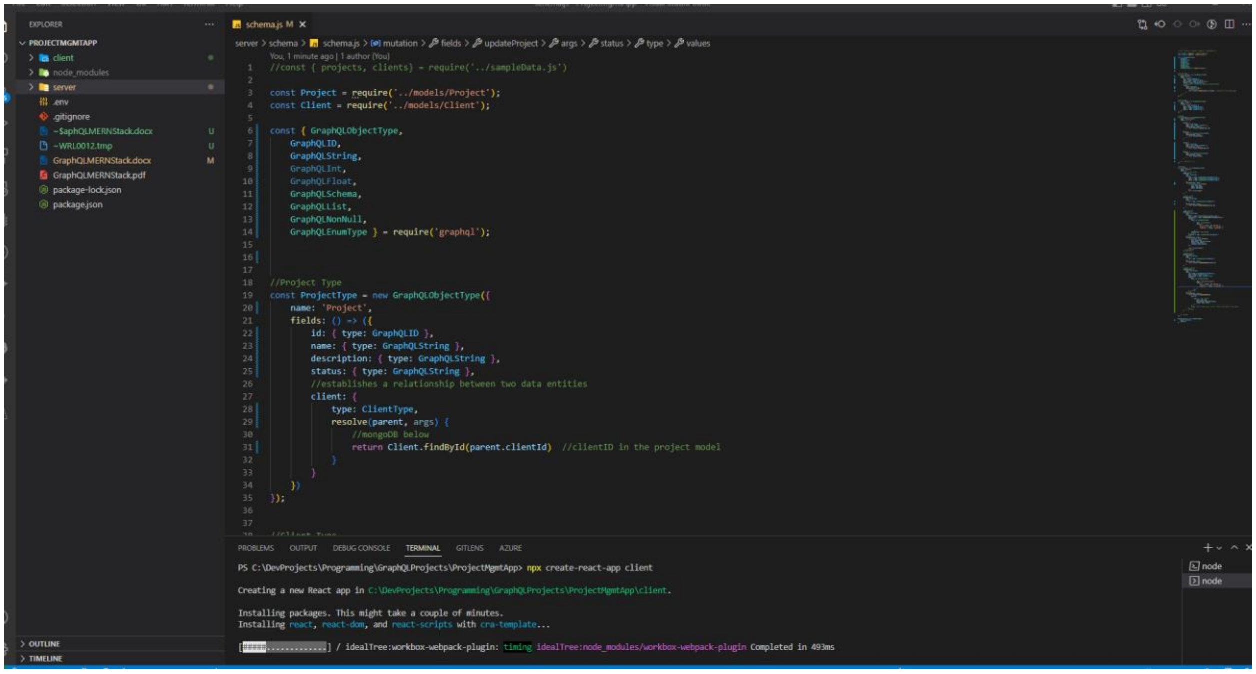
Update project



```
//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
```



```
//const { projects, clients} = require('../sampleData.js')

const Project = require('../models/Project');
const Client = require('../models/Client');

const { GraphQLObjectType,
    GraphQLID,
    GraphQLString,
    GraphQLInt,
    GraphQLFloat,
    GraphQLSchema,
    GraphQLList,
    GraphQLNonNull,
    GraphQLEnumType } = 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) {
                //mongoDB below
                return Client.findById(parent.clientId)  //clientID in the project model
            }
        }
    })
});
```

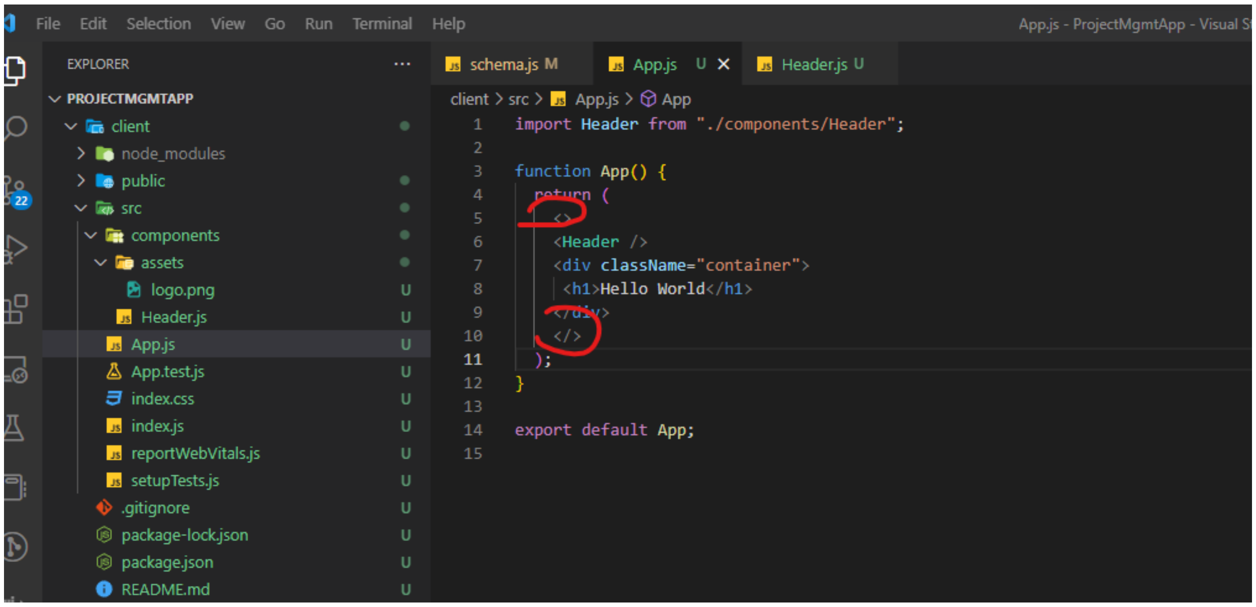
```
PS C:\DevProjects\Programming\GraphQLProjects\ProjectMgmtApp> npx create-react-app client

Creating a new React app in C:\DevProjects\Programming\GraphQLProjects\ProjectMgmtApp\client.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[#####.............] / idealTree:workbox-webpack-plugin: timing idealTree:node_modules/workbox-webpack-plugin Completed in 493ms
```

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

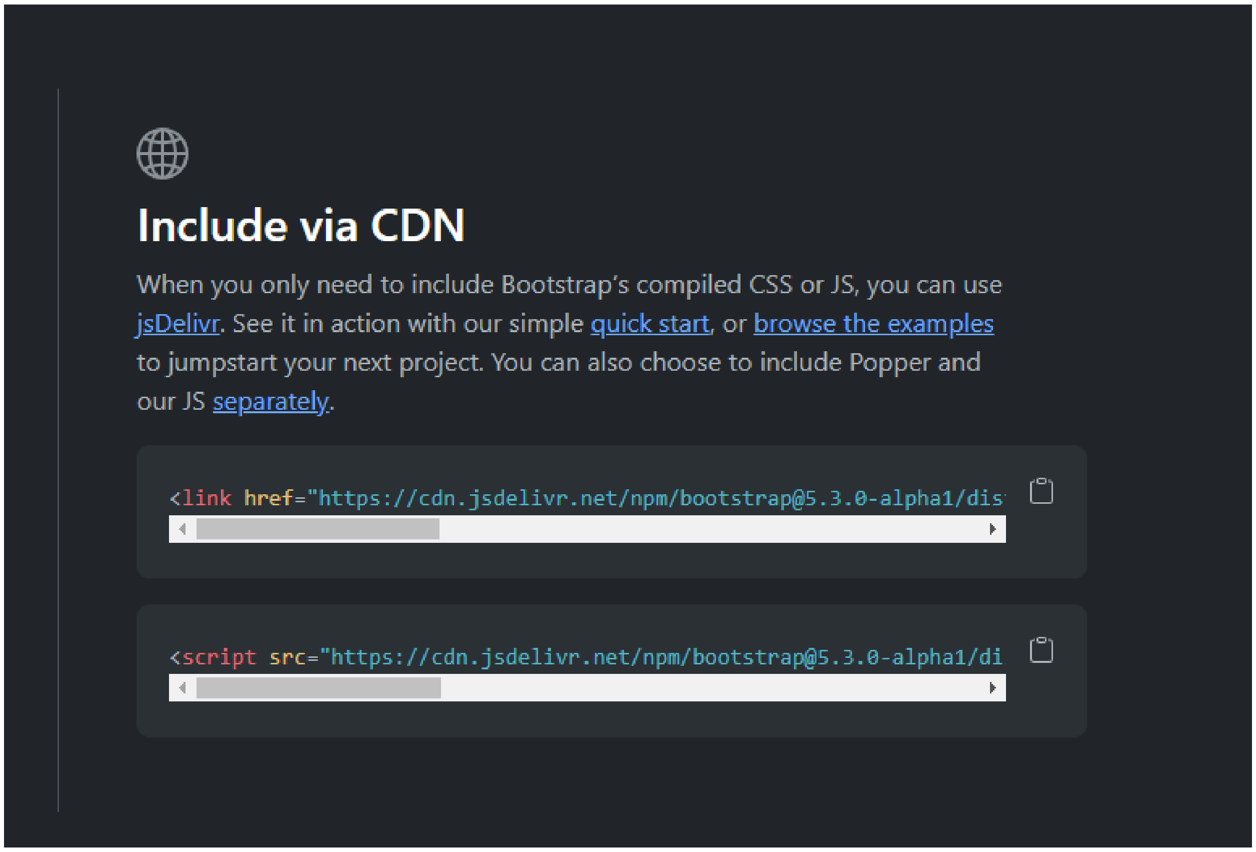
Above is adding the apollo client (to run queries against our graphql server)
Router and react icons (font-awesome icons) – from react

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



```
import Header from "./components/Header";

function App() {
  return (
    <>
     <Header />
     <div className="container">
      <h1>Hello World</h1>
     </div>
    </>
  );
}

export default App;
```

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/



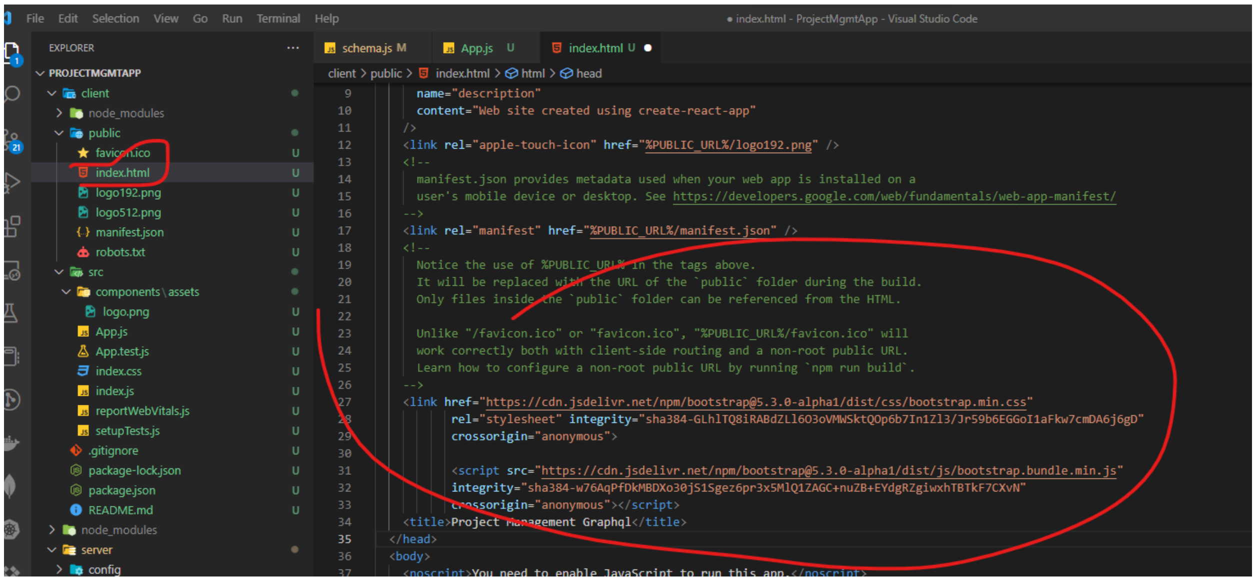
Include via CDN

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/dis
```

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

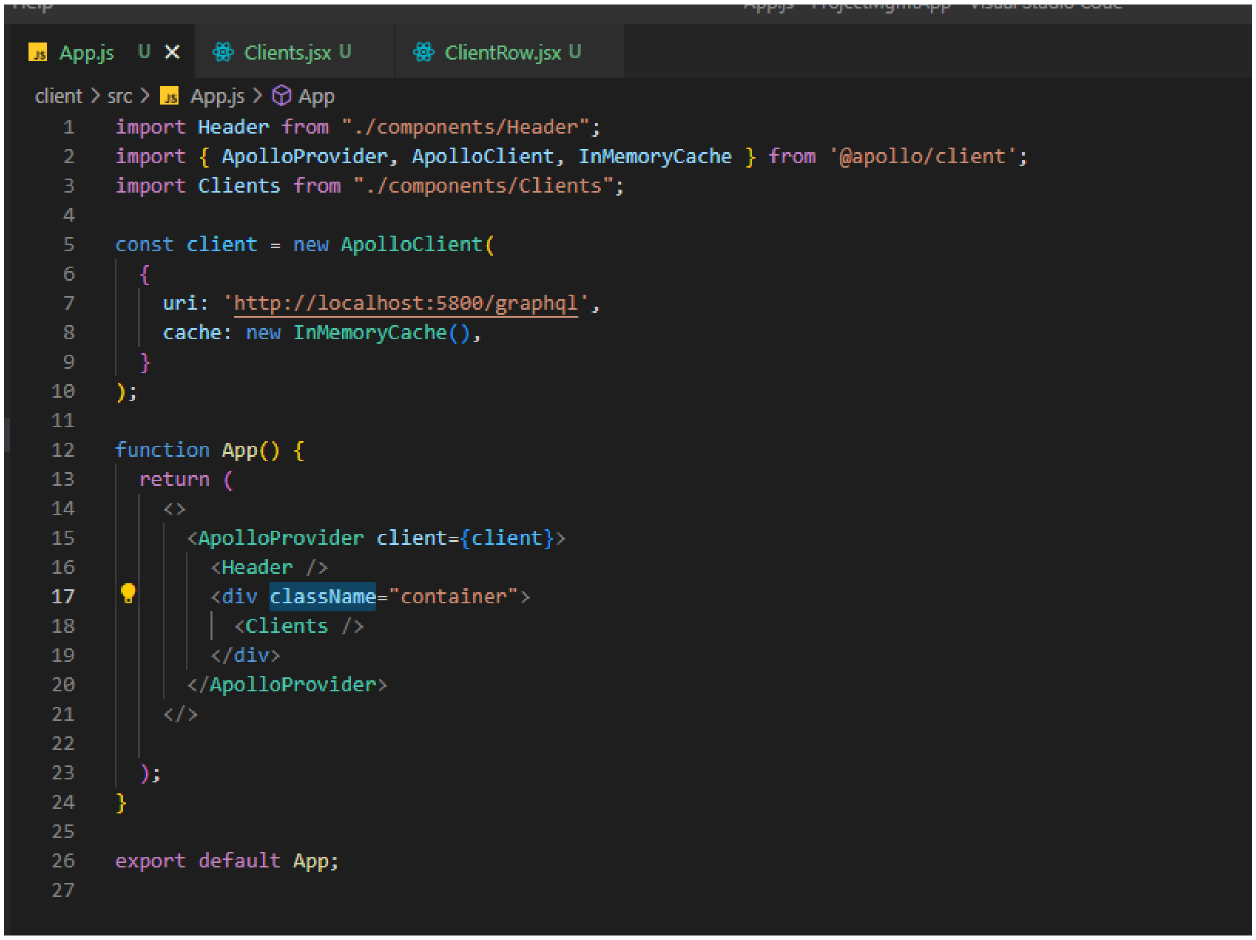
Add entries in the public/index.html file



```
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
          crossorigin="anonymous">

          <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
          integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN"
          crossorigin="anonymous"></script>
    <title>Project Management Graphql</title>
  </head>
  <body>
    <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)

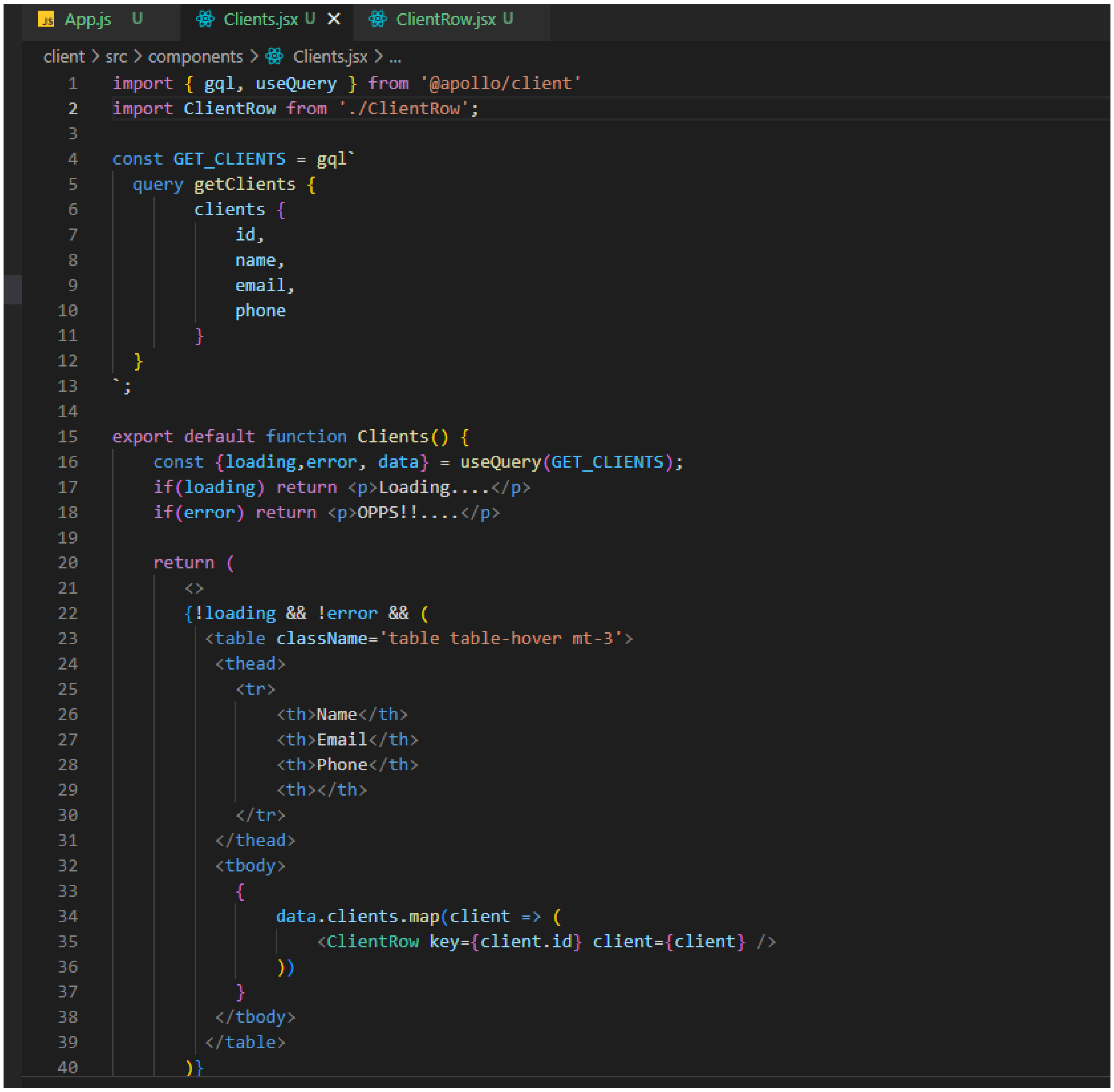
In your app.js file



```
import Header from "./components/Header";
import { ApolloProvider, ApolloClient, InMemoryCache } from '@apollo/client';
import Clients from "./components/Clients";

const client = new ApolloClient(
  {
    uri: 'http://localhost:5800/graphql',
    cache: new InMemoryCache(),
  }
);

function App() {
  return (
    <>
      <ApolloProvider client={client}>
        <Header />
        <div className="container">
          <Clients />
        </div>
      </ApolloProvider>
    </>

  );
}

export default App;
```

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

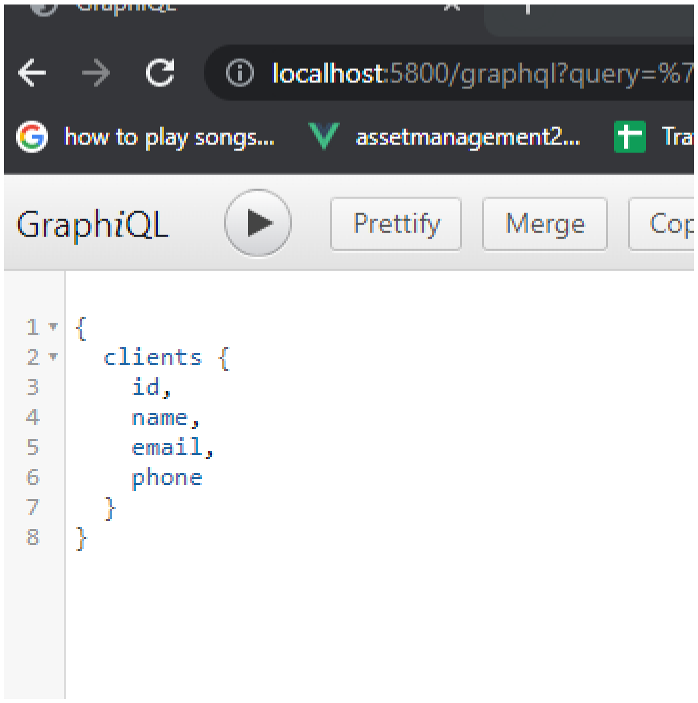


App.js U   Clients.jsx U ×   ClientRow.jsx U

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

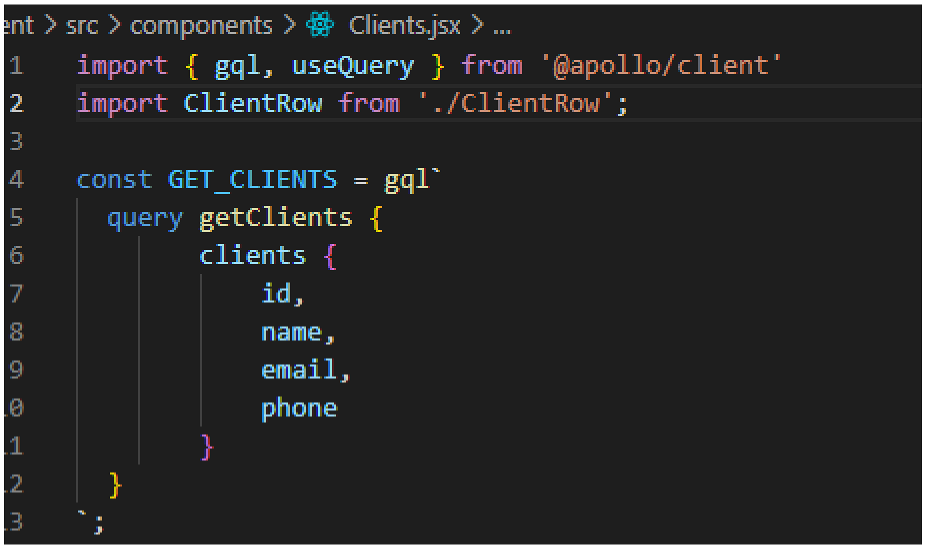
```
import { gql, useQuery } from '@apollo/client'
import ClientRow from './ClientRow';

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



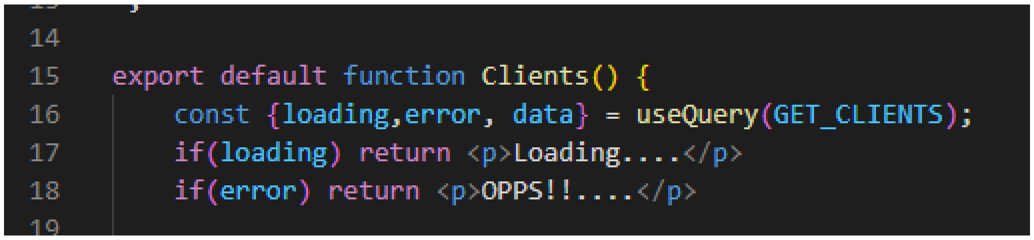
```
{
  clients {
    id,
    name,
    email,
    phone
  }
}
```

The build our query:



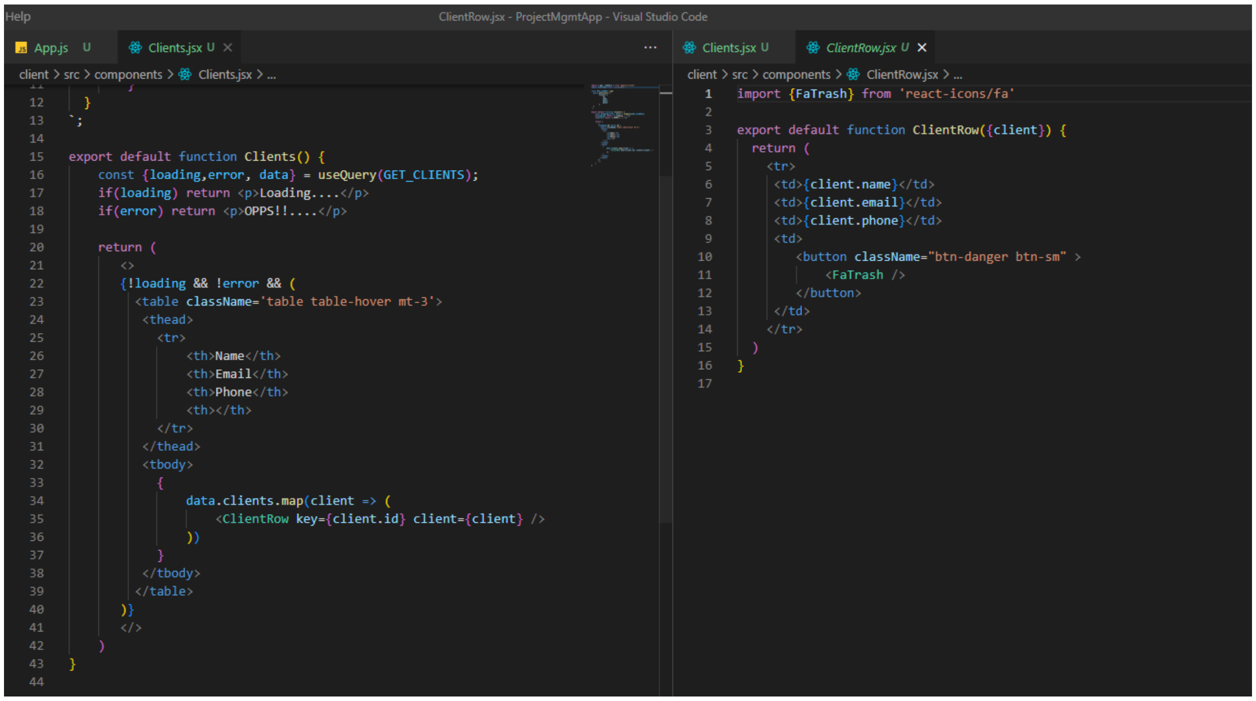
```
import { gql, useQuery } from '@apollo/client'
import ClientRow from './ClientRow';

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>
```

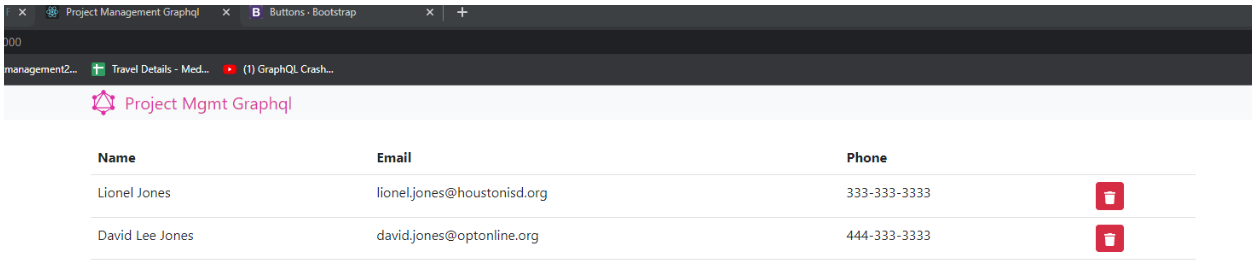
We then we render the results via a component (passing in our data as prop for the row)


Help
ClientRow.jsx - ProjectMgmtApp - Visual Studio Code
App.js U
Clients.jsx U
ClientRow.jsx U
client > src > components > Clients.jsx > ...
client > src > components > ClientRow.jsx > ...

```
  }
`;

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>
       )}
       </>
    )
}
```

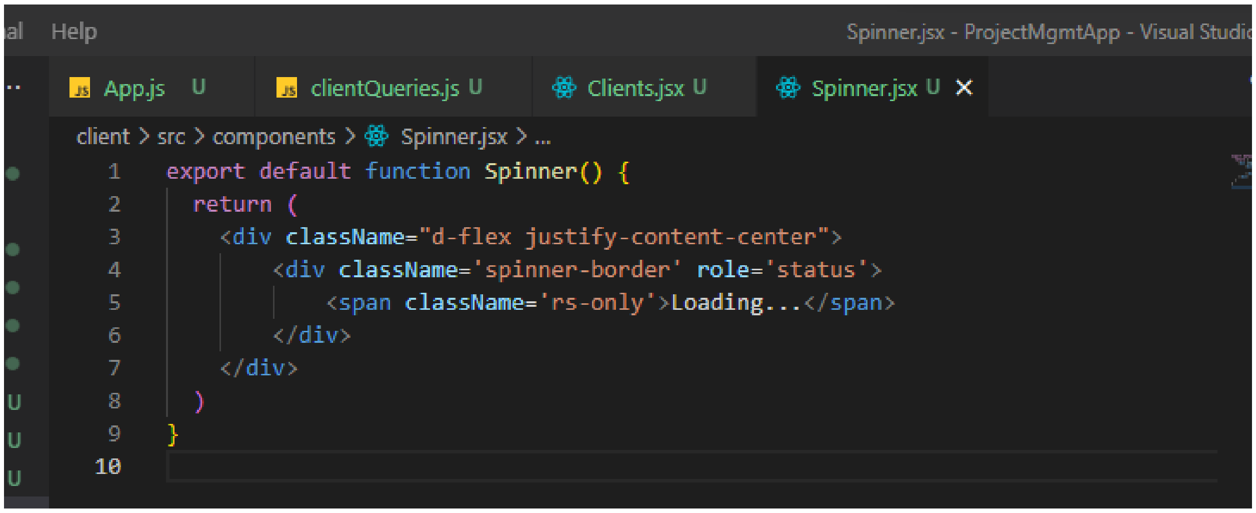
```
import {FaTrash} from 'react-icons/fa'

export default function ClientRow({client}) {
  return (
    <tr>
     <td>{client.name}</td>
     <td>{client.email}</td>
     <td>{client.phone}</td>
     <td>
        <button className="btn-danger btn-sm" >
            <FaTrash />
        </button>
     </td>
    </tr>
  )
}
```

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

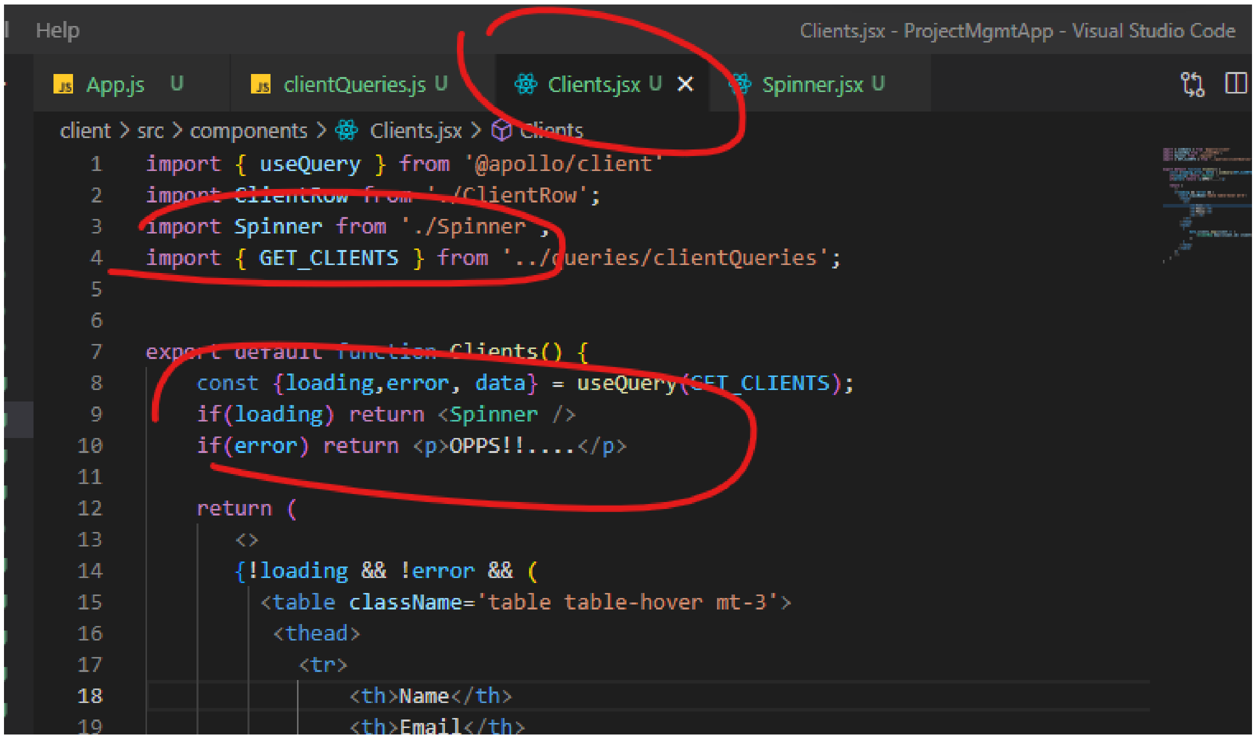


Name	Email	Phone	
Lionel Jones	lionel.jones@houstonisd.org	333-333-3333	🗑
David Lee Jones	david.jones@optonline.org	444-333-3333	🗑

Tip: Creating a Spinner with built-in react spinner



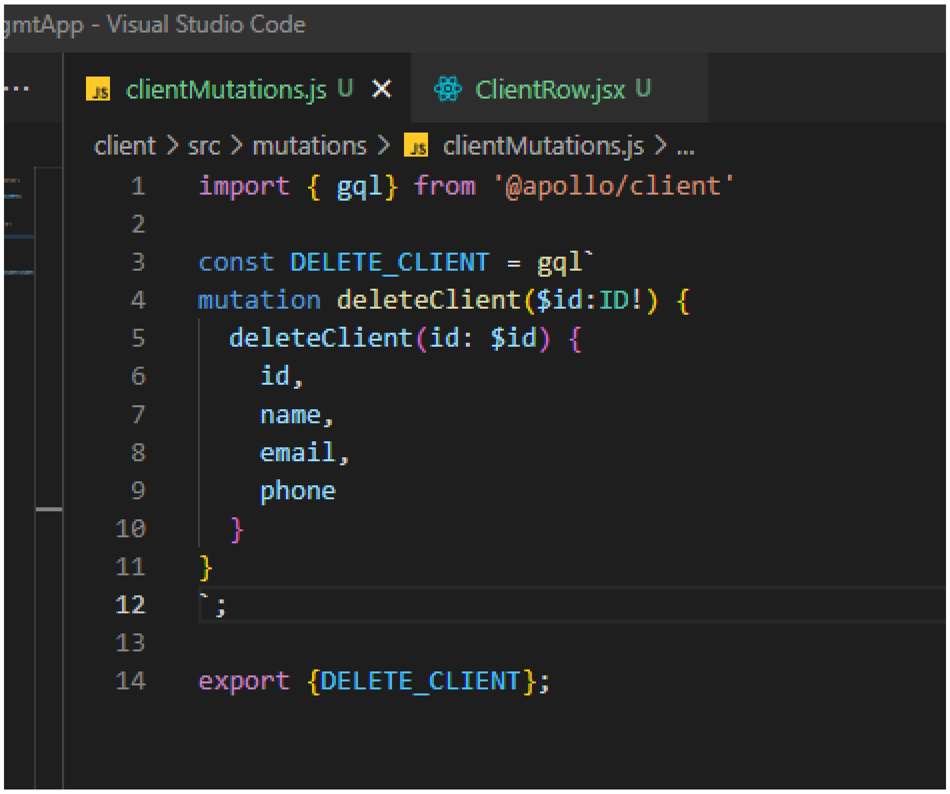
```
export default function Spinner() {
  return (
    <div className="d-flex justify-content-center">
        <div className='spinner-border' role='status'>
            <span className='rs-only'>Loading...</span>
        </div>
    </div>
  )
}
```



```
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 <p>OPPS!!....</p>

    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



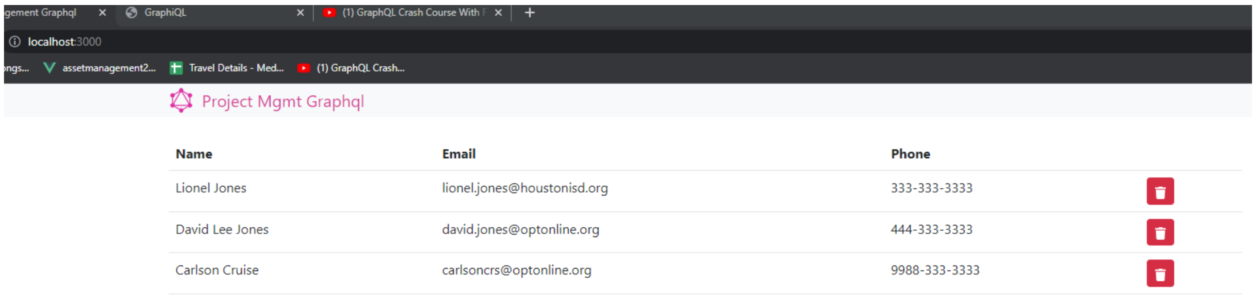
gmtApp - Visual Studio Code

clientMutations.js U | ClientRow.jsx U

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

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

const DELETE_CLIENT = gql`
mutation deleteClient($id:ID!) {
  deleteClient(id: $id) {
    id,
    name,
    email,
    phone
  }
}
`;

export {DELETE_CLIENT};
```

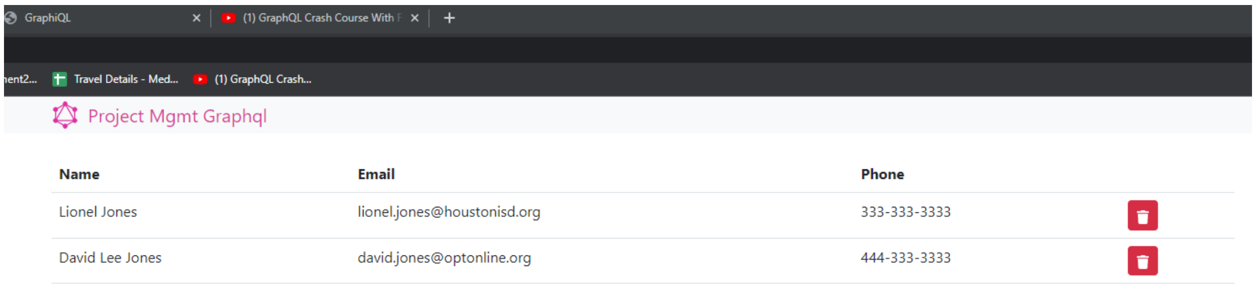
And below we can use the refetchQuieres attribute to refresh the UI when a client record is deleted



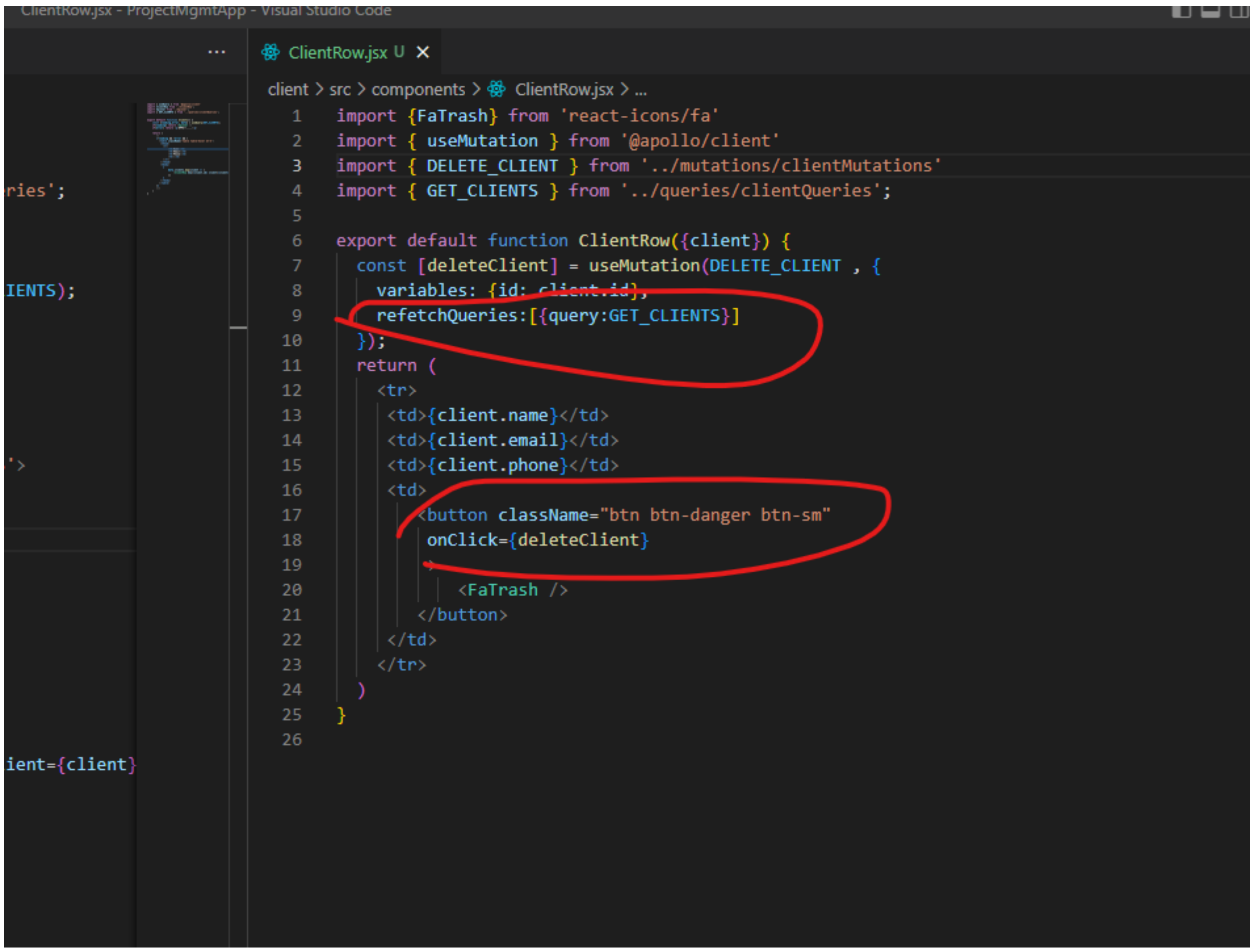
Project Mgmt Graphql

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

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

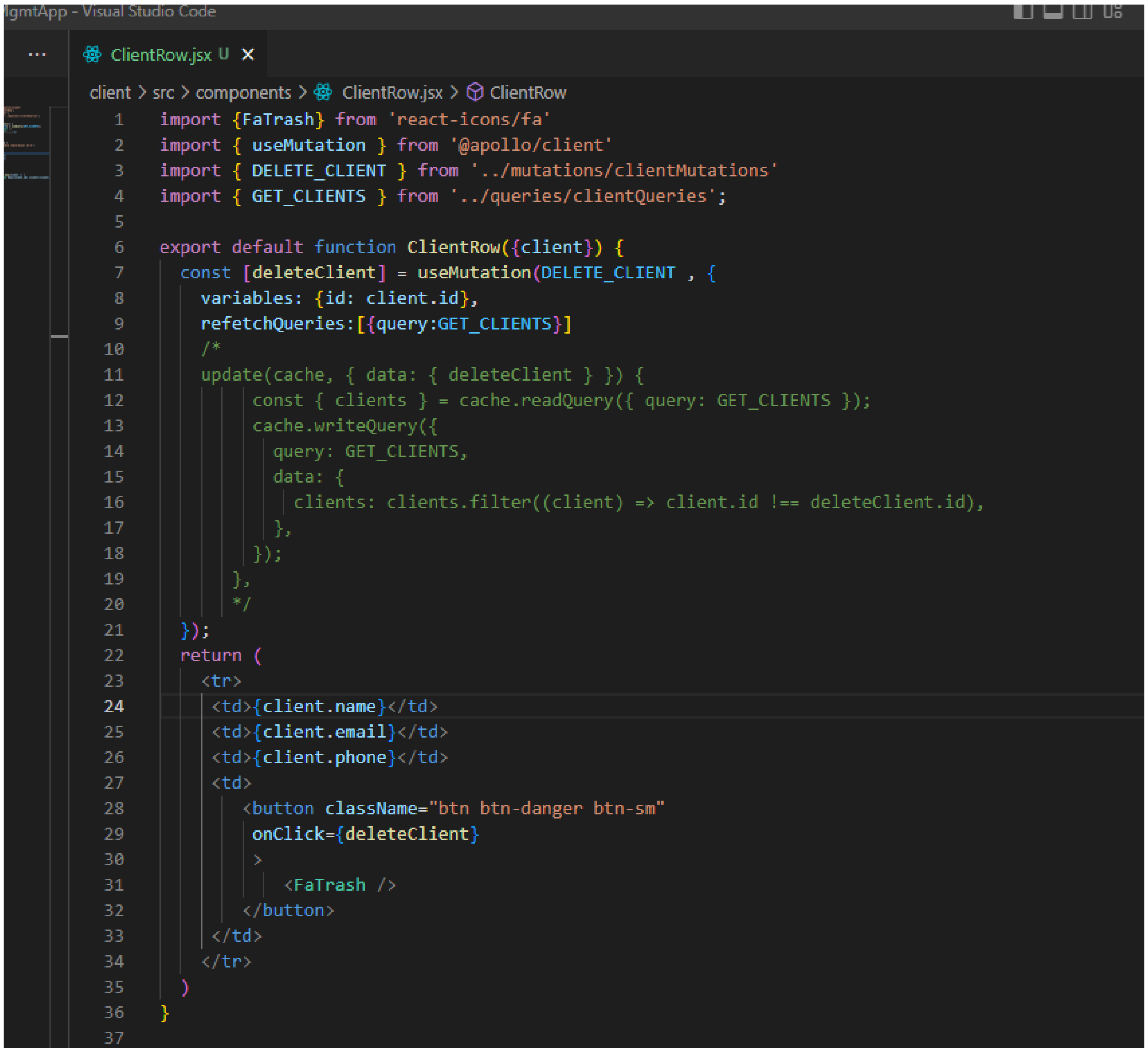


Name	Email	Phone	
Lionel Jones	lionel.jones@houstonisd.org	333-333-3333	
David Lee Jones	david.jones@optonline.org	444-333-3333	


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:

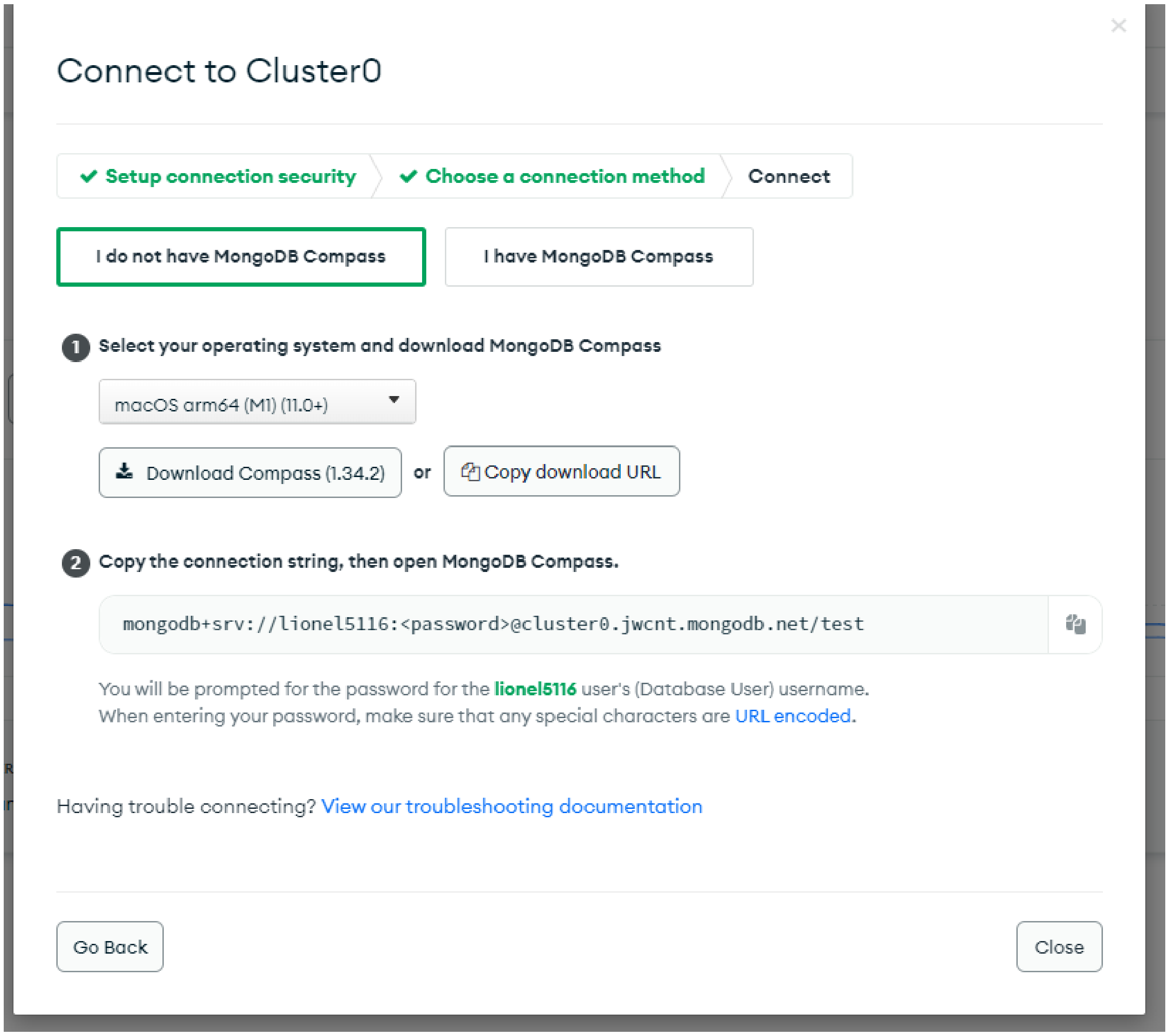


lgmtApp - Visual Studio Code
ClientRow.jsx U
client > src > components > ClientRow.jsx > ClientRow

```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}]
    /*
    update(cache, { data: { deleteClient } }) {
        const { clients } = cache.readQuery({ query: GET_CLIENTS });
        cache.writeQuery({
          query: GET_CLIENTS,
          data: {
            clients: clients.filter((client) => client.id !== deleteClient.id),
          },
        });
      },
      */
  });
  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>
  )
}
```

More code as shown above

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
```

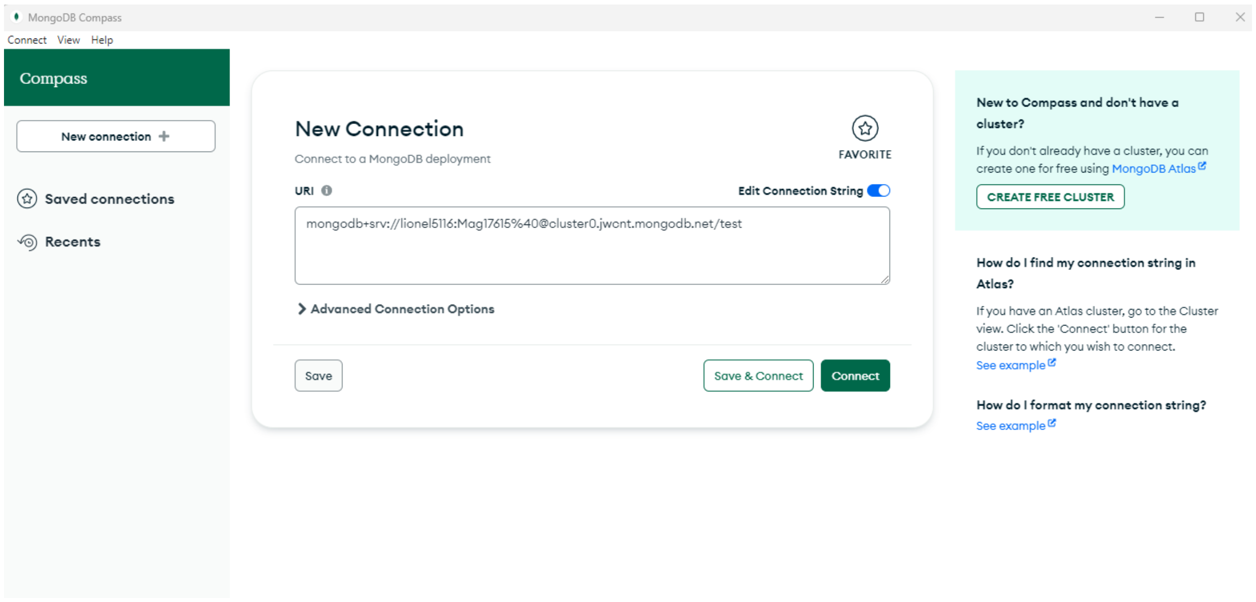
You will be prompted for the password for the **lionel5116** user's (Database User) username. When entering your password, make sure that any special characters are URL encoded.

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"
```

}



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

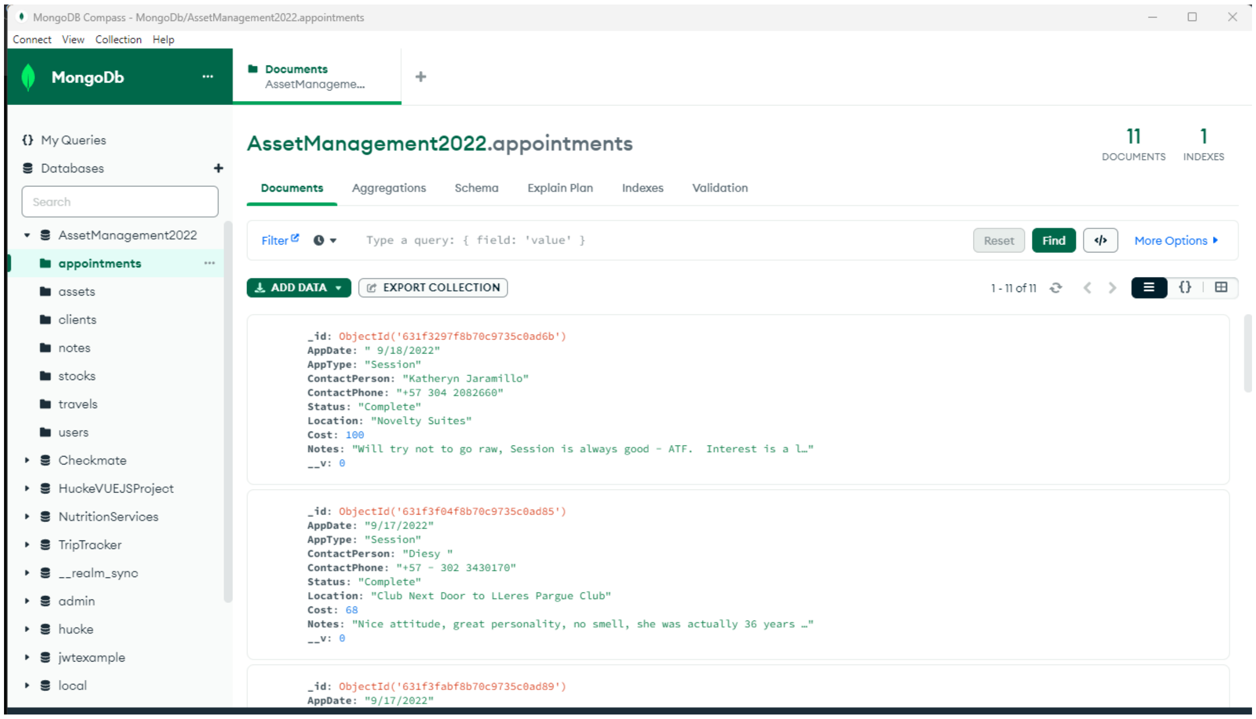
How do I find my connection string in Atlas?

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

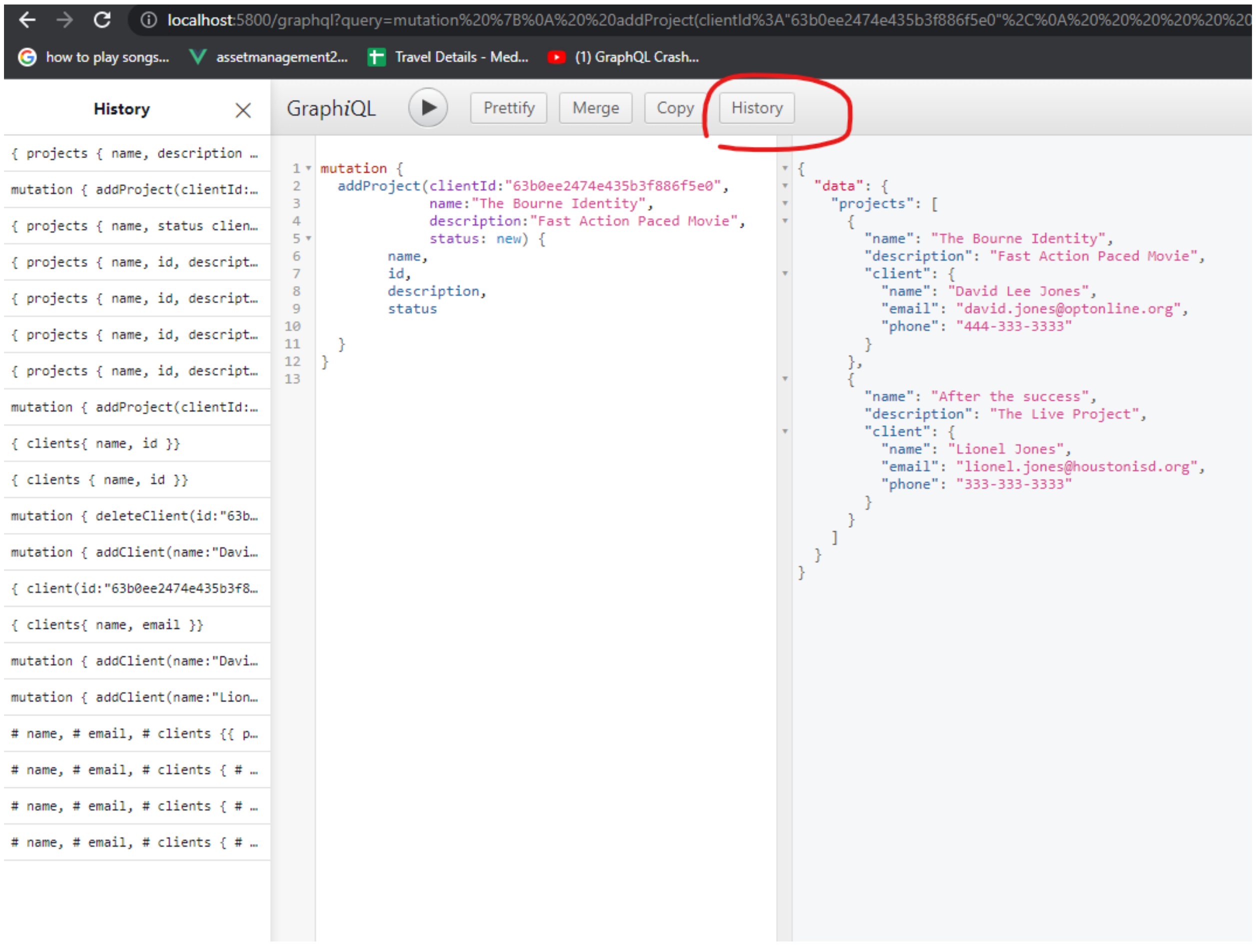
ADD DATA EXPORT COLLECTION 1 - 11 of 11

```
_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


localhost:5800/graphql?query=mutation%20%7B%0A%20%20addProject(clientId%3A"63b0ee2474e435b3f886f5e0"%2C%0A%20%20%20%20%20%20%2
how to play songs...
assetmanagement2...
Travel Details - Med...
(1) GraphQL Crash...
History
GraphiQL
Prettify
Merge
Copy
History

```
{ projects { name, description …
mutation { addProject(clientId:…
{ projects { name, status clien…
{ projects { name, id, descript…
{ projects { name, id, descript…
{ projects { name, id, descript…
{ projects { name, id, descript…
mutation { addProject(clientId:…
{ clients{ name, id }}
{ clients { name, id }}
mutation { deleteClient(id:"63b…
mutation { addClient(name:"Davi…
{ client(id:"63b0ee2474e435b3f8…
{ clients{ name, email }}
mutation { addClient(name:"Davi…
mutation { addClient(name:"Lion…
# name, # email, # clients {{ p…
# name, # email, # clients { # …
# name, # email, # clients { # …
# name, # email, # clients { # …
```

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

  }
}
```

```
{
  "data": {
    "projects": [
      {
        "name": "The Bourne Identity",
        "description": "Fast Action Paced Movie",
        "client": {
          "name": "David Lee Jones",
          "email": "david.jones@optonline.org",
          "phone": "444-333-3333"
        }
      },
      {
        "name": "After the success",
        "description": "The Live Project",
        "client": {
          "name": "Lionel Jones",
          "email": "lionel.jones@houstonisd.org",
          "phone": "333-333-3333"
        }
      }
    ]
  }
}
```