

GraphQL

Tip: Creating a sample nodeJS Project Using VSCode

Tip: Querying the data

Tip: Exposing the GUI for graphiql

Tip: Fetching MongoDB Data with graphql

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 from

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 % Σ X
first_name	First Name	blank: 0 % Σ X
last_name	Last Name	blank: 0 % Σ X
email	Email Address	blank: 0 % Σ X
gender	Gender	blank: 0 % Σ X
ip_address	IP Address v4	blank: 0 % Σ X

[ADD ANOTHER FIELD](#)

Rows: Format: ☒ array ☒ include null values

Hint: Use "*" in column names to generate nested json objects, brackets to generate arrays. [More information...](#)

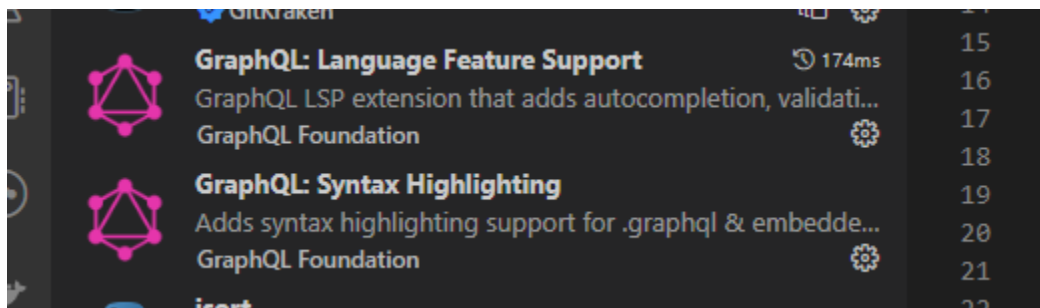
[Follow @mockaroo](#)

Mock your back-end API and start

[DOWNLOAD DATA](#) [PREVIEW](#) [SAVE THIS SCHEMA](#) [MORE](#)

I changed the type to JSON

Add the VSCode Extensions to your VSCODE

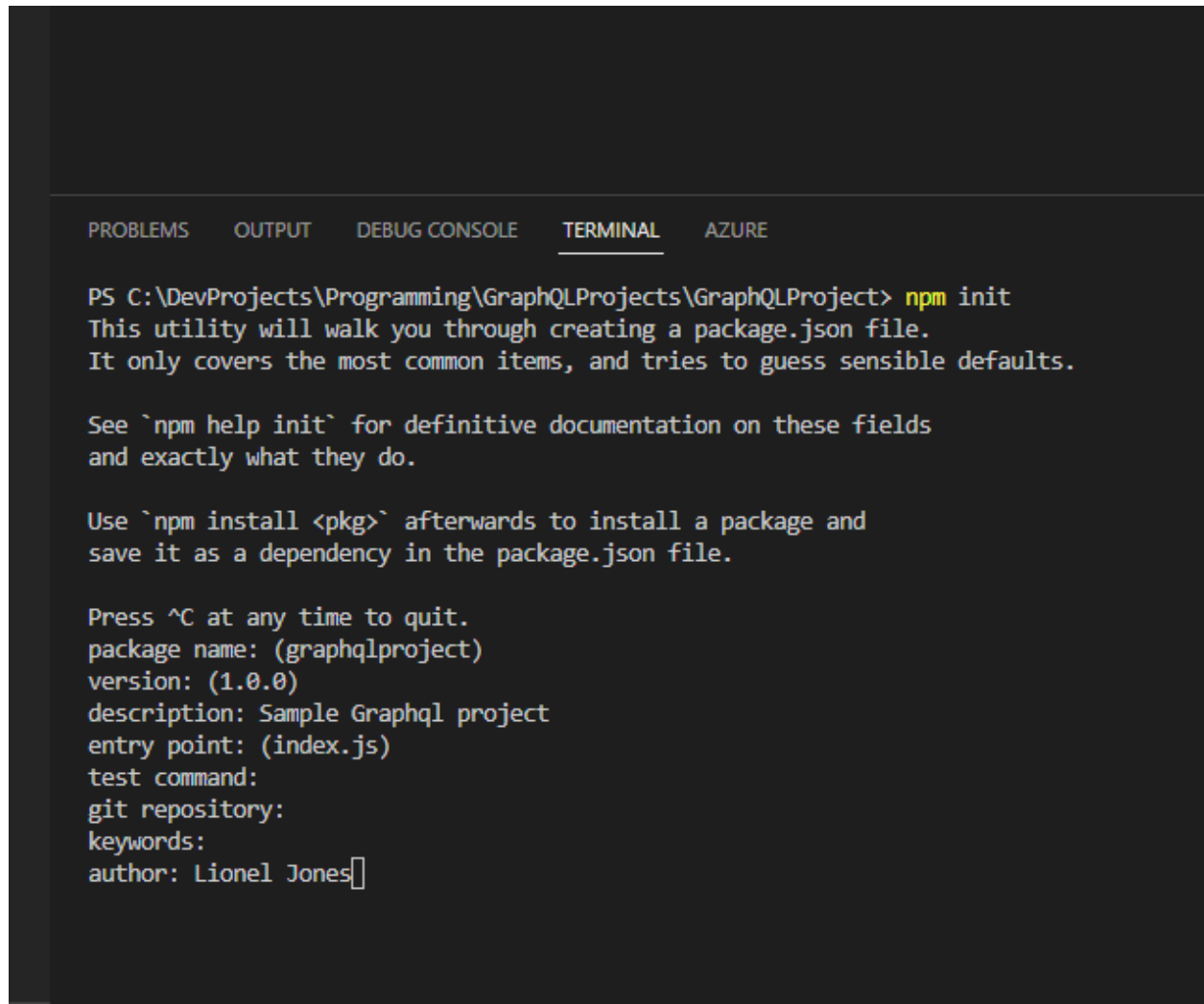


Tip: Creating a sample nodeJS Project Using VSCode

First I create a directory called GraphQLProject

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

I just follow the prompts, enter in generic information



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  AZURE

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

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

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

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

This will create my package.json file for my dependencies

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

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

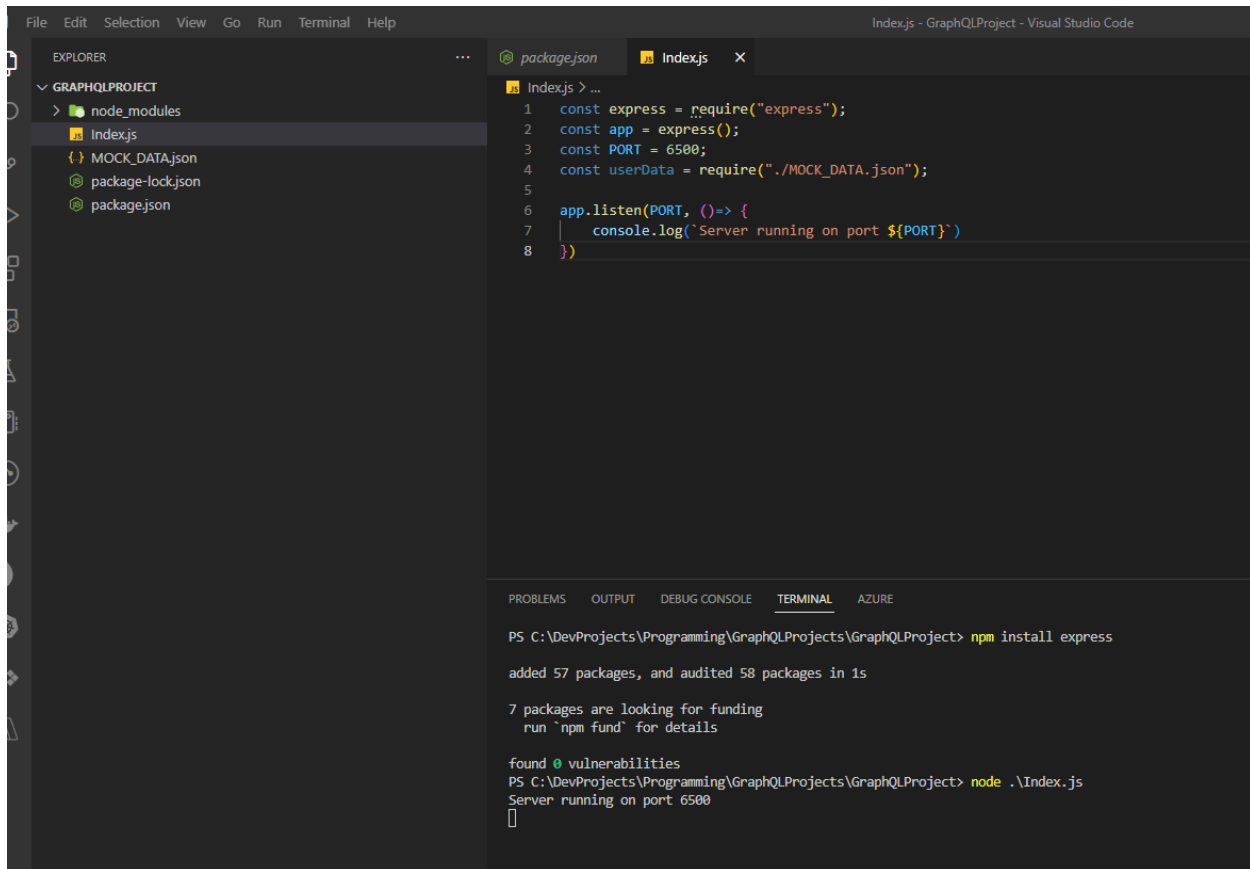
```
File Edit Selection View Go Run Terminal Help
package.json - GraphQLProject - Visual Studio Code

EXPLORER
  GRAPHQLPROJECT
    > node_modules
      Indexjs
      MOCK_DATA.json
      package-lock.json
      package.json

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

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

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



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the project structure: GRAPHQLPROJECT, node_modules, and a new file index.js. The index.js file is open in the editor, showing the following code:

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

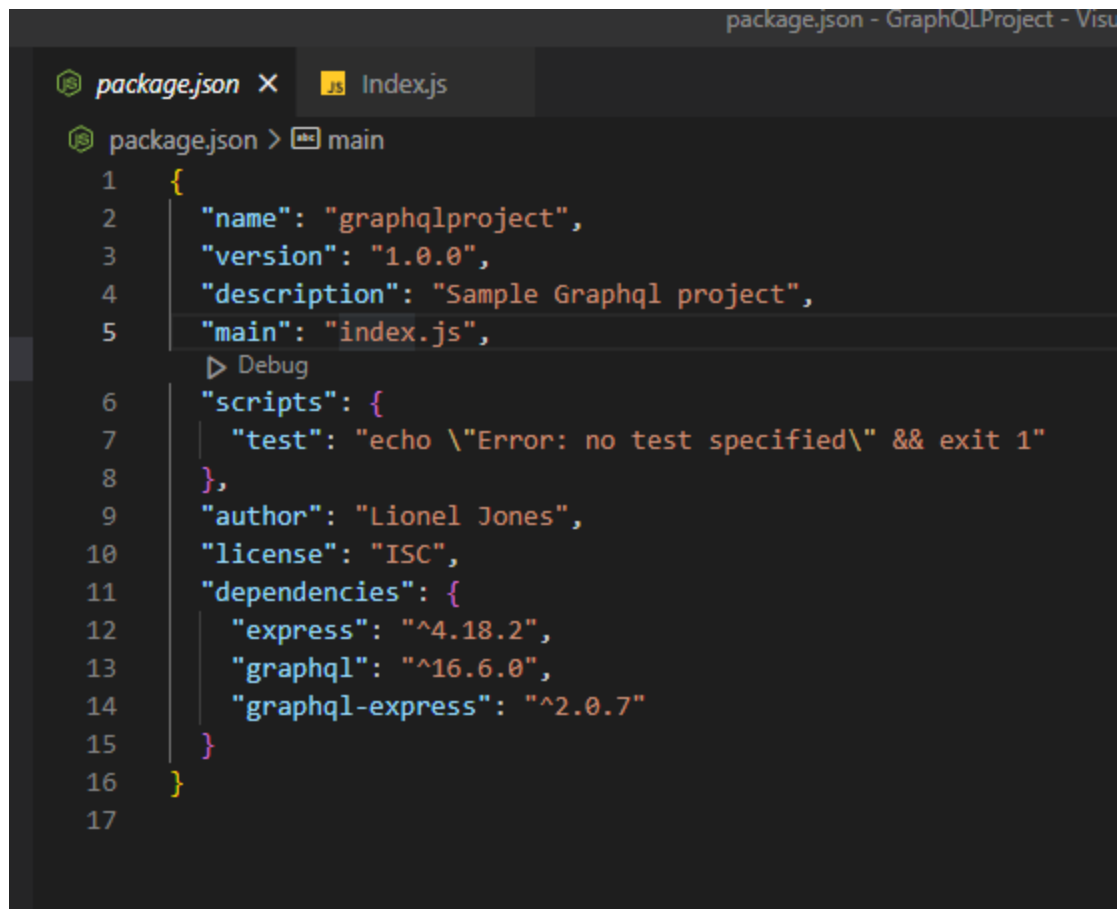
The Terminal panel at the bottom shows the following output:

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

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

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

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

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

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

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

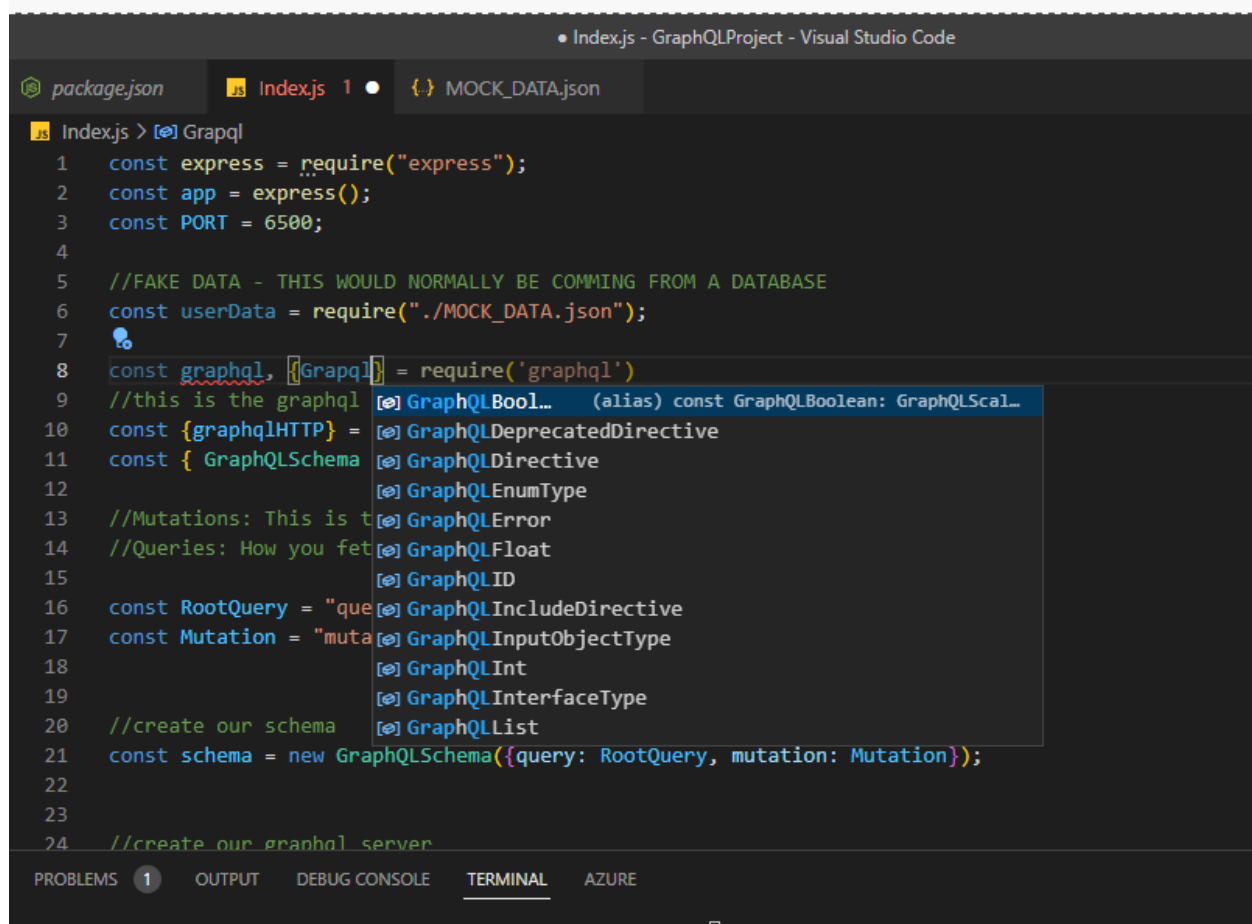
As you can see, as you install packages, it updates your package.json file

GraphQL Concepts

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

Queries: How you fetch the data you need

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

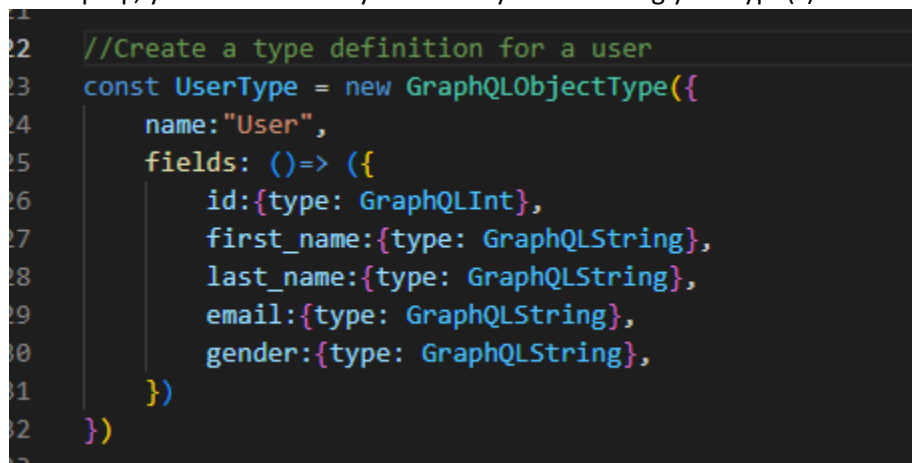


```
Index.js - GraphQLProject - Visual Studio Code
package.json  index.js 1  MOCK_DATA.json

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

Creating the type(s):

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



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

Creating the query(s)

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

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

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

Creating the mutation

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

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

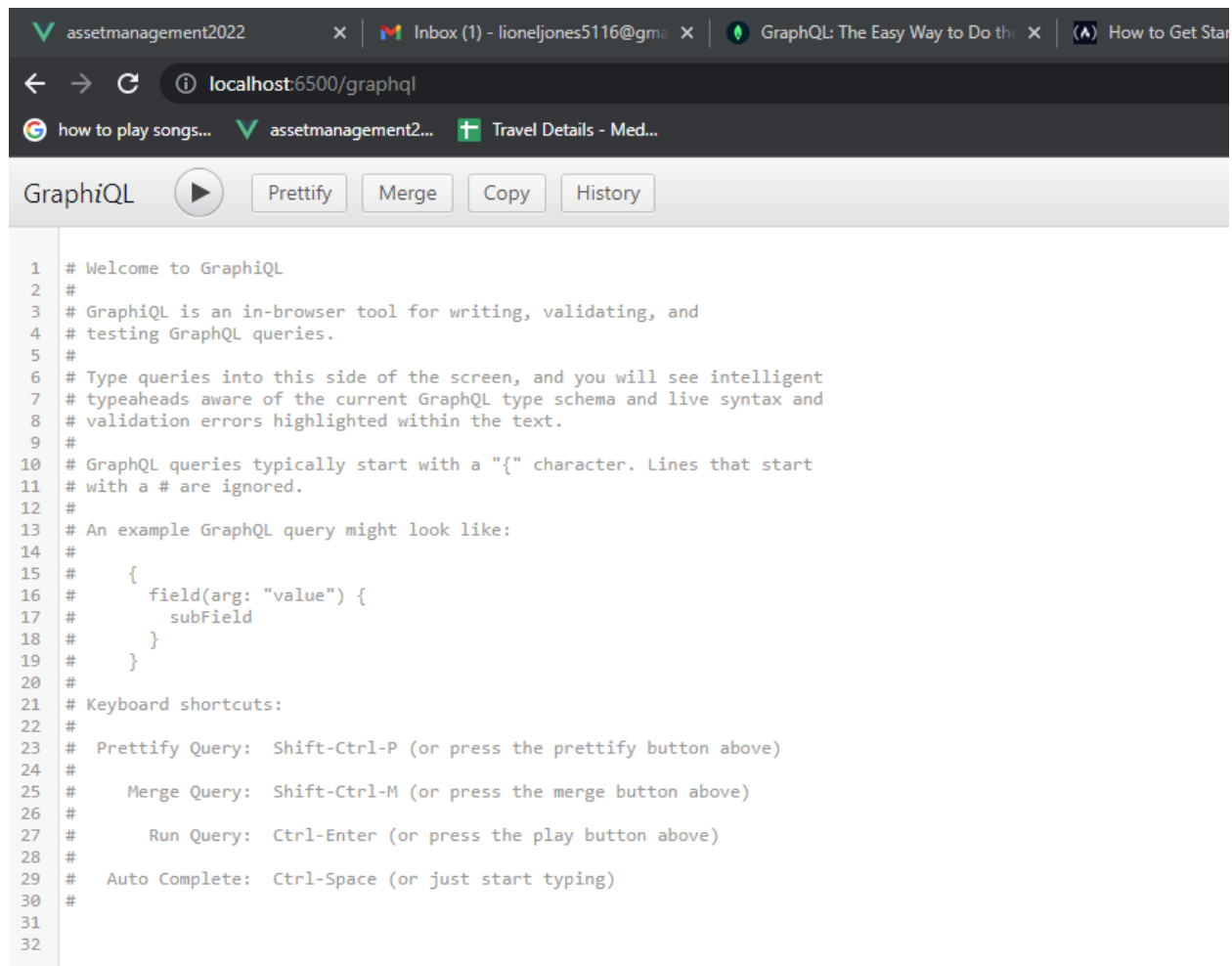

Next, let's fire up our graphql server
Enter the command at the terminal

node index.js

Then in the browser type:

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

When you do this:



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

This is surfaced up via this entry:

```

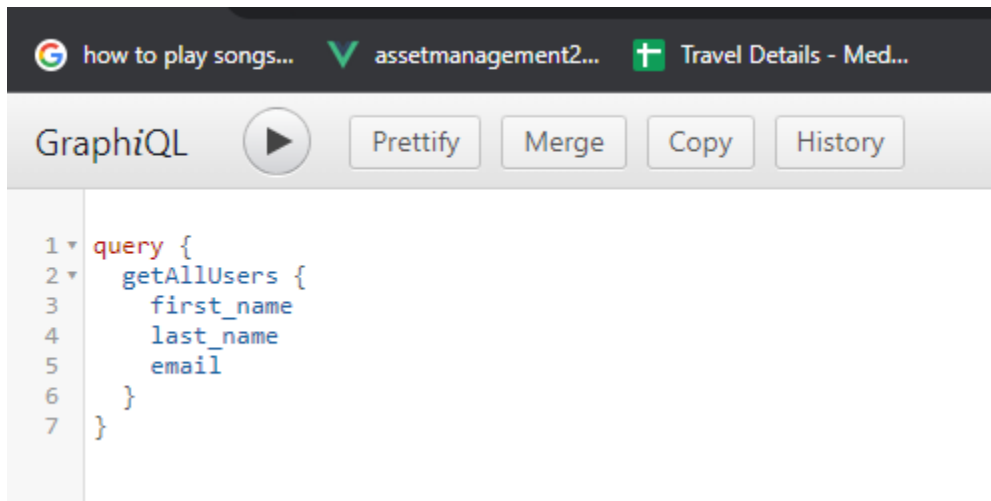
23
24 //create our graphql server
25 //Remember GraphQL only has "one" endpoint, below is where you create your queries
26 app.use('/graphql',graphqlHTTP( {
27   schema,
28   graphiql:true
29 }));
30
31

```

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

Tip: Querying the data

To fetch some data, just type below



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

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

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

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

. The right pane displays the JSON response:

```
{  
  "data": {  
    "getAllUsers": [  
      {  
        "first_name": "Garrot",  
        "last_name": "Guillot",  
        "email": "gguillot@spotify.com"  
      },  
      {  
        "first_name": "Daune",  
        "last_name": "Learnmond",  
        "email": "dlearnmond1@bbb.org"  
      },  
      {  
        "first_name": "Zarah",  
        "last_name": "Warrior",  
        "email": "zwarrior2@tamu.edu"  
      },  
      {  
        "first_name": "Dominique",  
        "last_name": "Riccioppo",  
        "email": "dr Riccioppo3@yale.edu"  
      },  
      {  
        "first_name": "Jammie",  
        "last_name": "Batthew",  
        "email": "jbatthew4@google.com.hk"  
      }  
    ]  
  }  
}
```

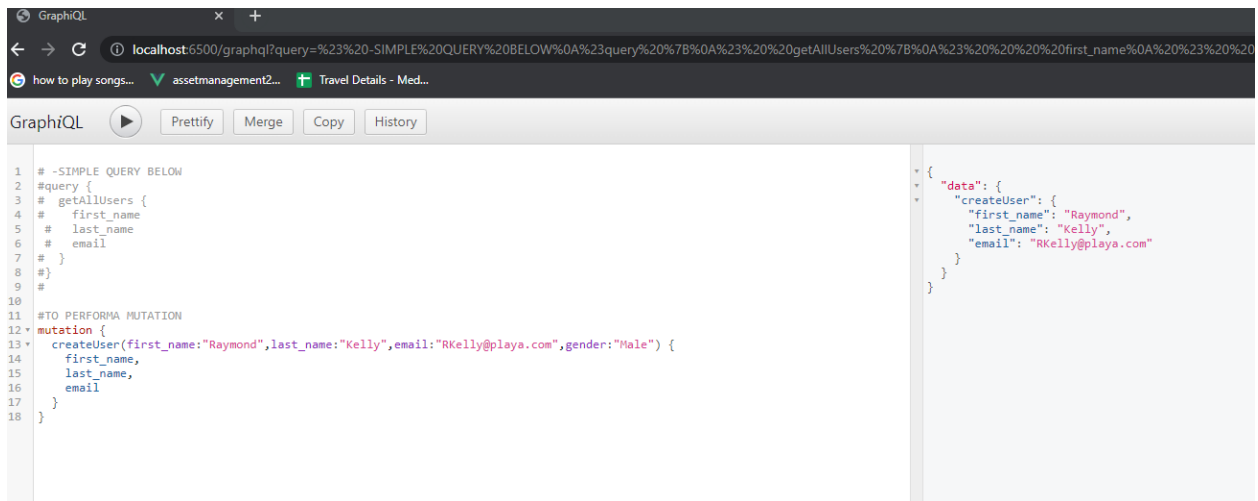
. The top bar shows the URL: `localhost:6500/graphql?query=query%20%7B%0A%20getAllUsers%20%7B%0A%20%20%20first_name%0A%20%20%20last_name%0A%20%20%20email%0A%20%20%7D%0A%7D`.

To perform a mutation:

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

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

. The right pane is empty. The top bar shows the GraphQL IDE controls: a play button, and buttons for Prettify, Merge, Copy, and History.



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

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

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

Run the query:

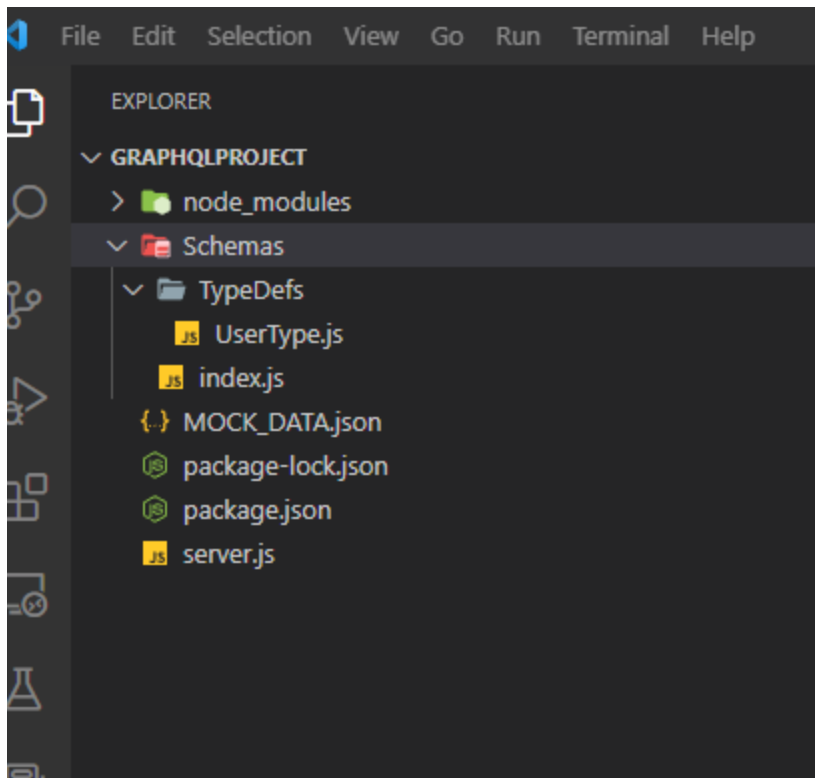
```
19
20 query {
21   getAllUsers {
22     id
23     first_name
24     last_name
25     email
26   }
27 }
28
29
```

And you will see the item added to the array:

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

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

To clean up the project structure:



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

We place our types inside of the typedefs folder

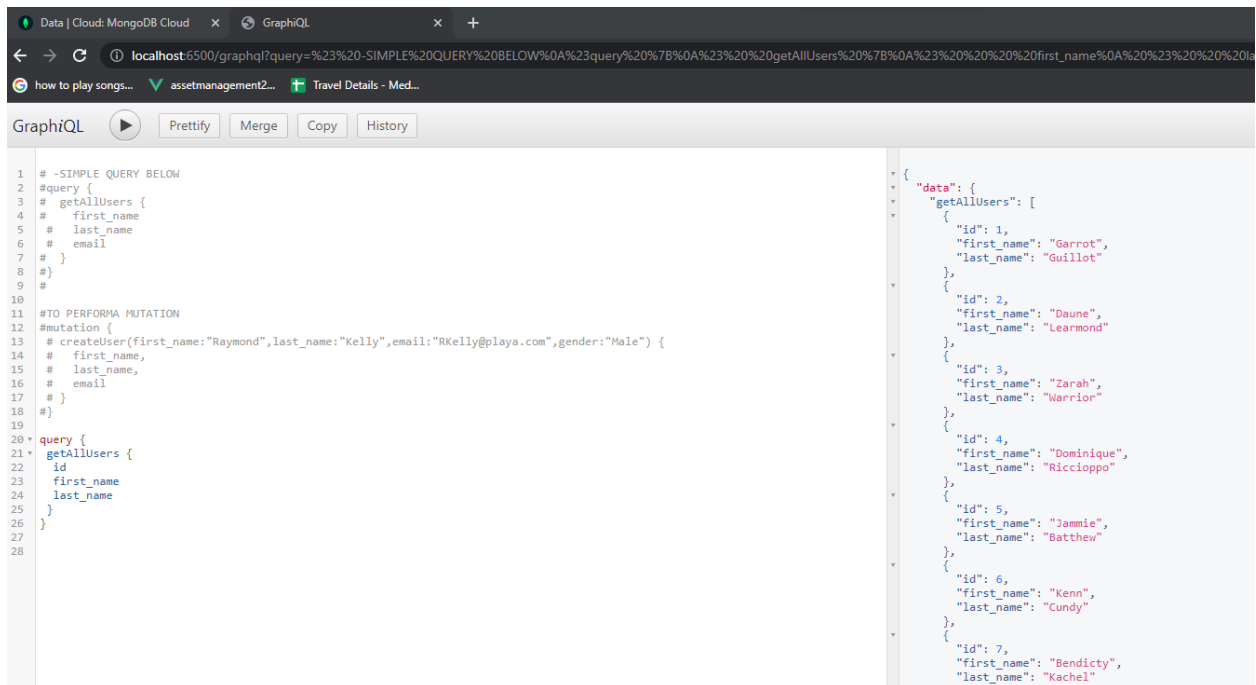
We place our mutations and queries in the index.js file under the schema folder

Then we execute

node server.js

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

And everything works:



Tip: Fetching MongoDB Data with graphql

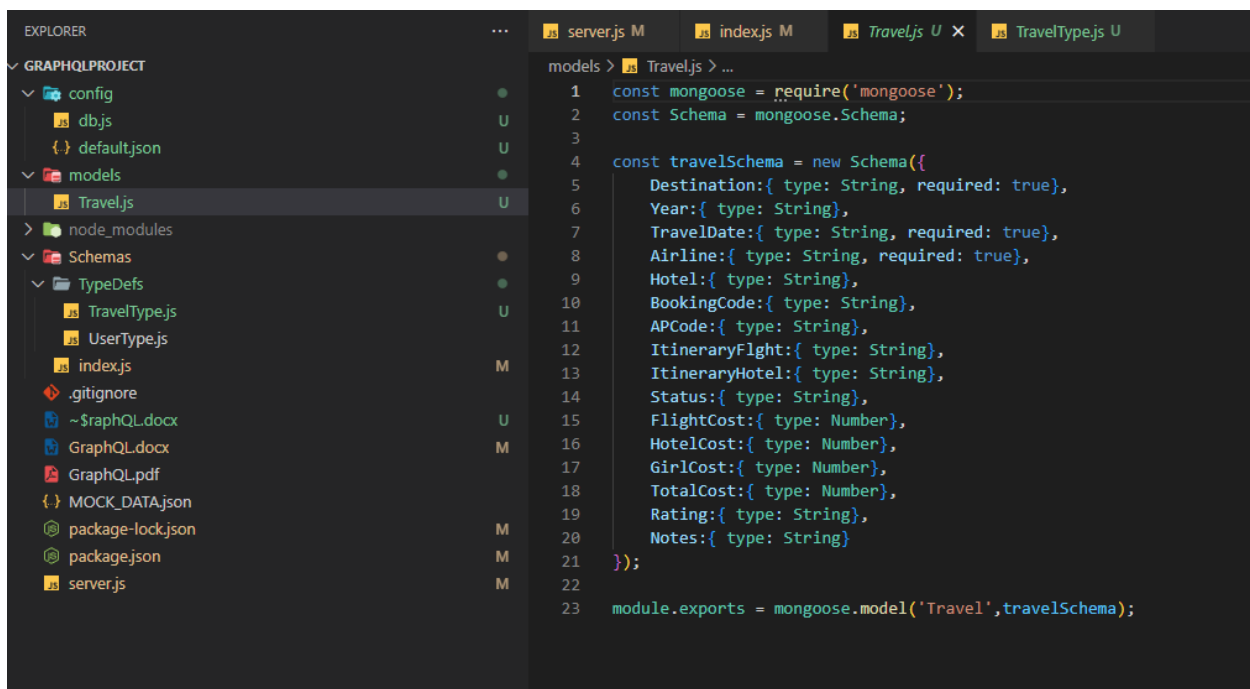
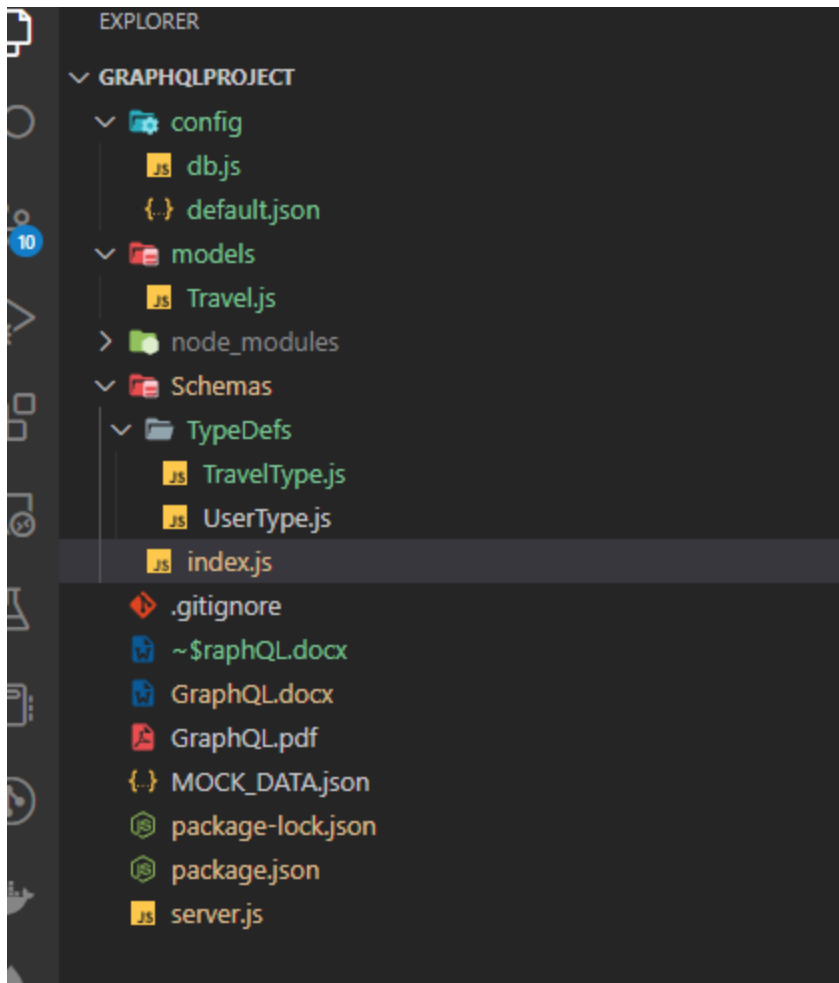
It was super easy, I just leveraged my code from my assetmgmt service using mongo express, connected to mongoDB, created my models and schema for graphql, then just did a mongoddb call to grab travel records. The resolve function in your RootQuery method where you create your queries takes the data argument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

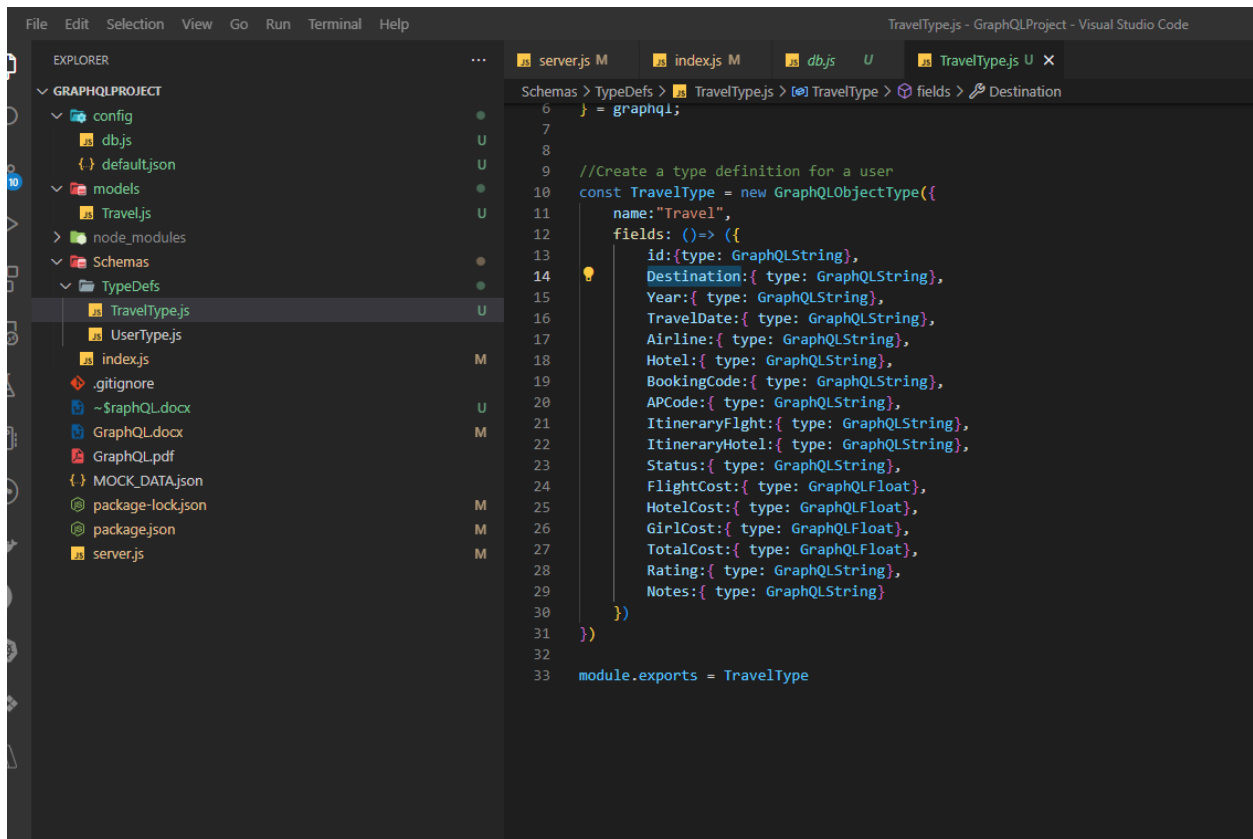
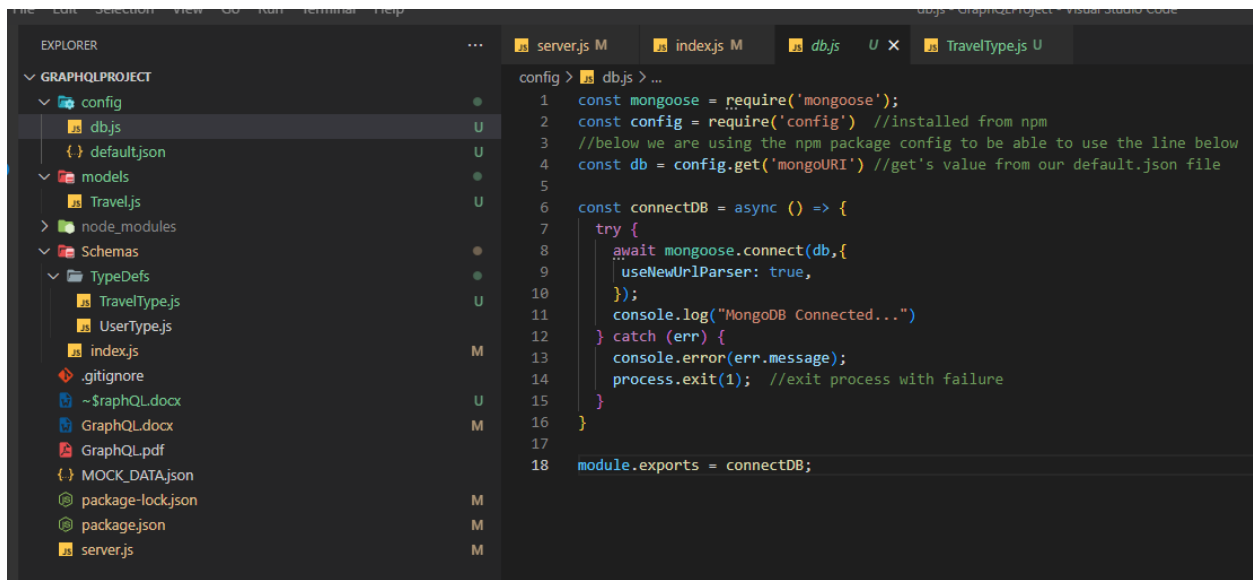
```

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

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

```






```

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS AZURE



AND IT WORKS!!!!