**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

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/>

Graphical user interface

Description automatically generated  
  
I changed the type to JSON  
  
Add the VSCode Extensions to your VSCODE  
  
Text

Description automatically generated  
  
  
**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  
  
Text

Description automatically generated  
  
This will create my package.json file for my dependencies  
  
Text

Description automatically generated  
  
Next, type the following command to install expressjs  
npm install express

Text

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

Description automatically generated  
  
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  
  
Text

Description automatically generated  
  
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  
Text

Description automatically generated

**Creating the type(s):**  
In Graphql, you interact with your data by first defining your type(s)  
Text

Description automatically generated

**Creating the querie(s)**   
As explained above, you create your mutations (queries as coded below)  
  
Text

Description automatically generated  
  
**Creating the mutation**Also explained before, you create (CRUD OPERATIONS) via “Mutations”  
Text

Description automatically generated  
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:

Graphical user interface, text, website

Description automatically generated  
  
You get a graphical UI to view your results of your query against  
  
This is surfaced up via this entry:  
A screenshot of a computer

Description automatically generated with medium confidence  
  
  
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

Graphical user interface, application

Description automatically generated  
  
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  
  
Graphical user interface, application, Word

Description automatically generated  
  
To perform a mutation:  
Graphical user interface, text, application, email

Description automatically generated  
  
Graphical user interface, text, application

Description automatically generated  
  
Run the query:

Graphical user interface, text, application

Description automatically generated  
  
And you will see the item added to the array:  
Text

Description automatically generated  
  
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:  
Graphical user interface, text, application

Description automatically generated  
  
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  
  
Text

Description automatically generated  
  
And everything works:

Graphical user interface, text, application

Description automatically generated  
  
  
  
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.  
A screenshot of a computer

Description automatically generated with medium confidence  
Graphical user interface, text, application

Description automatically generated  
  
Graphical user interface, text

Description automatically generated  
  
Text

Description automatically generated  
  
Text

Description automatically generated  
  
Text

Description automatically generated  
  
Graphical user interface, text

Description automatically generated  
  
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  
  
Graphical user interface, Word

Description automatically generated  
  
Graphical user interface, text, application, Word

Description automatically generated  
  
Text

Description automatically generated  
  
Creating a relationship between two entities:  
Text

Description automatically generated  
  
Text

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated  
  
Created a database and collection Atlas

Graphical user interface, text, application, email

Description automatically generated  
  
**See the tip below about MongoDB Atlas GUI (Free)**

**Tip**: Adding a record to MongoDB using GraphQL

This is done via a mutation  
Text

Description automatically generated  
  
mutation {

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

id,

name,

email,

phone

}

}  
  
Graphical user interface, text, application, Word

Description automatically generated  
  
Graphical user interface, text

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

Graphical user interface, text, application, email

Description automatically generated

To delete a client **Text

Description automatically generated  
  
Graphical user interface, application

Description automatically generated  
  
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**

Graphical user interface, text, application, chat or text message, email

Description automatically generated

The to query the projects  
**Graphical user interface, text, application

Description automatically generated**  
  
{

projects {

name,

status

client {

name,

email

}

}

}  
  
 **Update project  
  
Text

Description automatically generated**

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

A screenshot of a computer

Description automatically generated with medium confidence

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:  
<> </>

Text

Description automatically generated  
  
  
  
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/  
  
Graphical user interface, text, application

Description automatically generated  
Add entries in the public/index.html file  
Text

Description automatically generated

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

Text

Description automatically generated  
  
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  
Text

Description automatically generated  
  
We grab the syntax from graphql  
Graphical user interface, text, application

Description automatically generated  
  
The build our query:

A screenshot of a computer

Description automatically generated with medium confidence  
  
We then use the useQuery hook  
Text

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

Description automatically generated

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  
  
Graphical user interface, text, application

Description automatically generated

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

Description automatically generated  
  
Along with mutations  
Text

Description automatically generated  
  
In app.js

Text

Description automatically generated  
  
Graphical user interface, application, Teams

Description automatically generated  
  
Graphical user interface, text, application

Description automatically generated  
  
Sweet

Tip: Creating a Spinner with built-in react spinner  
A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Tip: Deleting and re-fetching data with graphiql  
Create the mutation  
**Text

Description automatically generated**

And below we can use the refetchQuieres attribute to refresh the UI when a client record is deleted  
  
Graphical user interface, text, application

Description automatically generated  
  
When we hit delete, the UI updates with the remaining records  
Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Second way:

Text

Description automatically generated  
  
More code as shown above

Tip: Creating a not Found page  
**Text

Description automatically generated**  
  
**A picture containing diagram

Description automatically generated**  
  
**Graphical user interface, application, Teams

Description automatically generated**

**Tip:** GUI for MongoDB  
<https://www.mongodb.com/products/compass>  
  
Graphical user interface, text, application, email

Description automatically generated  
{

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

"jwtSecretToken":"mysecrettoken"

}  
Graphical user interface, application, Teams

Description automatically generated  
  
  
Graphical user interface, text, application, email

Description automatically generated

**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  
Graphical user interface, text

Description automatically generated