**Exercise 13**

*Create a simple GraphQL server in Node.js using Mongo as a database server*

**Prior Knowledge**

Unix Command Line Shell

Some simple JavaScript (node.js)

**Learning Objectives**

Understand GraphQL

**Software Requirements**

Node.js

npm/yarn

Mongo

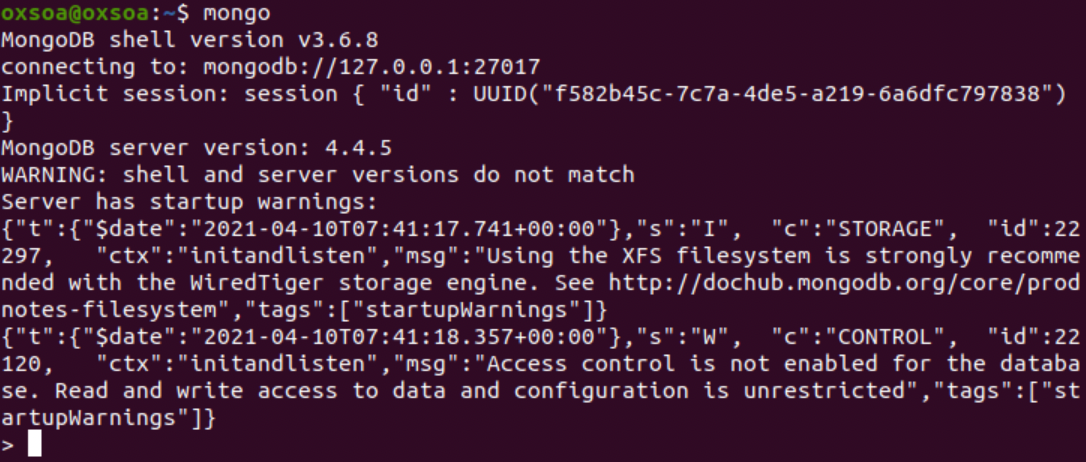
Visual Studio Code

*Thanks to this guide which this is heavily based on:*<https://freo.me/do-node-graphql>

**Steps**

1. First let’s install MongoDB client tools  
     
   sudo apt install mongodb-clients mongo-tools -y
2. Now let’s run mongodb in a container:  
     
   docker run -p 27017:27017 mongo
3. Check it works:

mongo



1. Please note that we haven’t set up any security for the database. This is not a good thing. Don’t do this in production :-)
2. Type   
   exit   
   to leave the mongo client command prompt.
3. Clone my simple sample repository:

cd ~  
git clone <https://github.com/pzfreo/graphql-example.git>

cd graphql-example

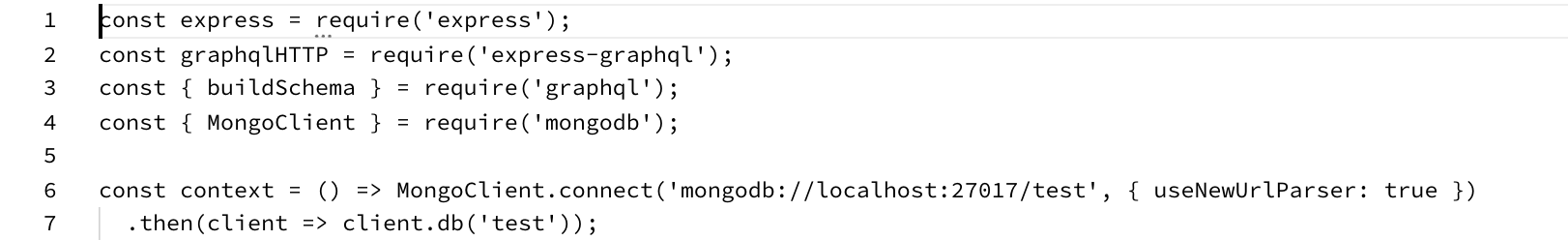
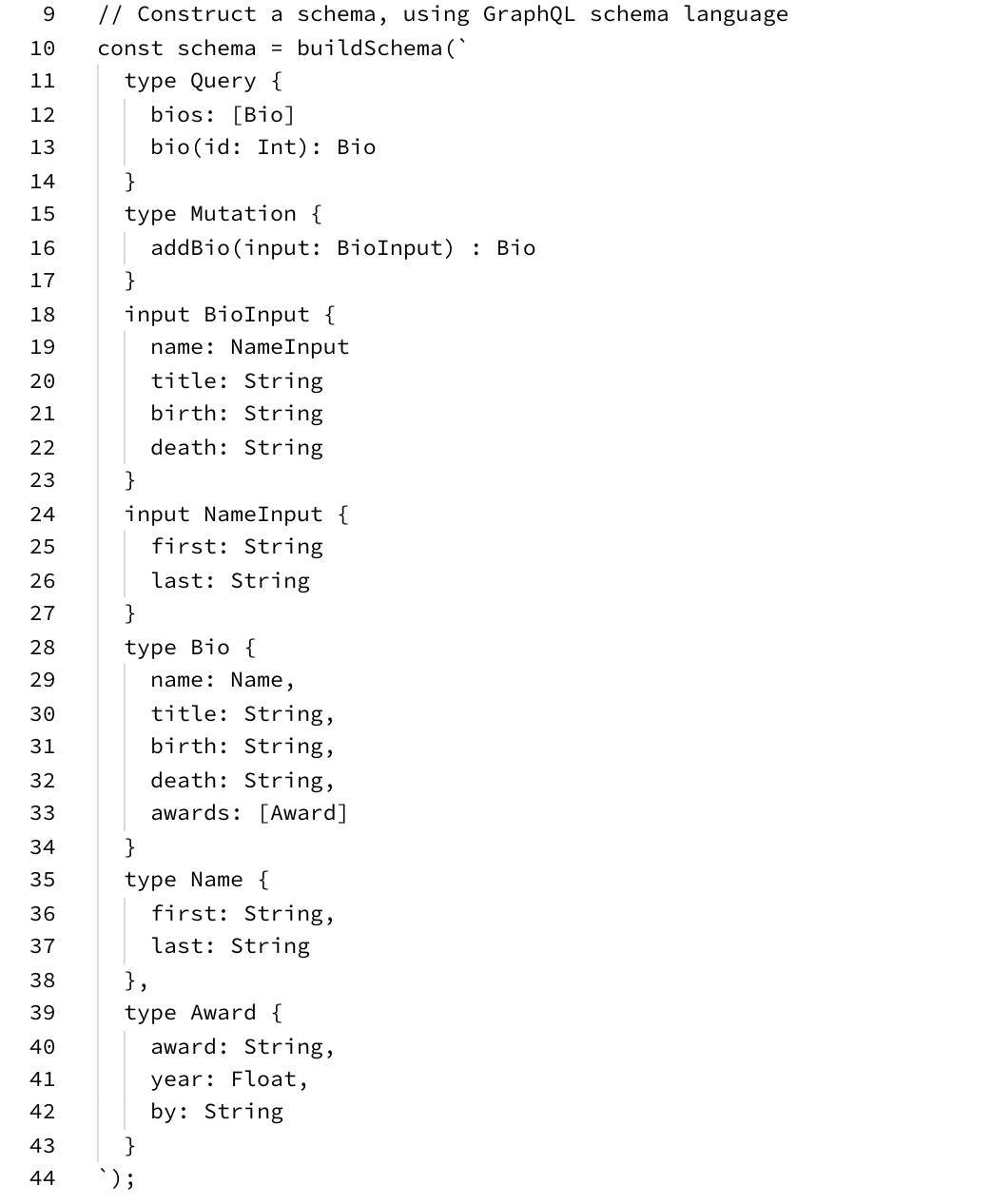
yarn install

1. Import some data into Mongo:  
     
   ﻿mongoimport -d test -c bios bios.json  
     
   This is this data:  
   <https://docs.mongodb.com/manual/reference/bios-example-collection/>

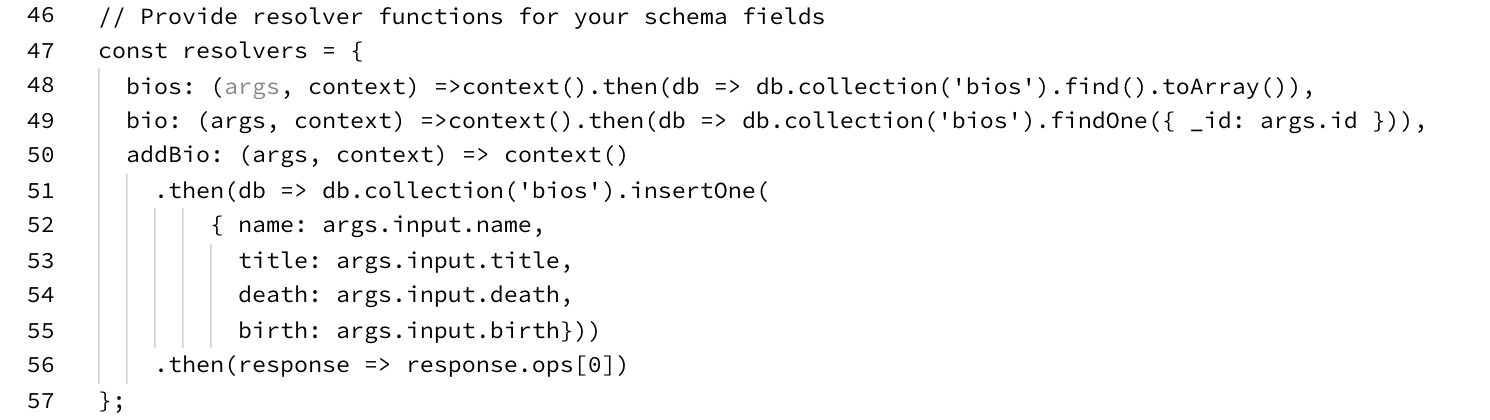
1. Have a look using the mongo client  
     
   mongo  
     
   > use test  
   ﻿switched to db test  
     
   > ﻿db.bios.find({})  
   You should see something like:  
     
   ﻿{ "\_id" : 4, "name" : { "first" : "Kristen", "last" : "Nygaard" }, "birth" : ISODate("1926-08-27T04:00:00Z"), "death" : ISODate("2002-08-10T04:00:00Z"), "contribs" : [ "OOP", "Simula" ], "awards" : [ { "award" : "Rosing Prize", "year" : 1999, "by" : "Norwegian Data Association" }, { "award" : "Turing Award", "year" : 2001, "by" : "ACM" }, { "award" : "IEEE John von Neumann Medal", "year" : 2001, "by" : "IEEE" } ] }
2. Exit the mongo client

1. Take a look at our app:

code index.js

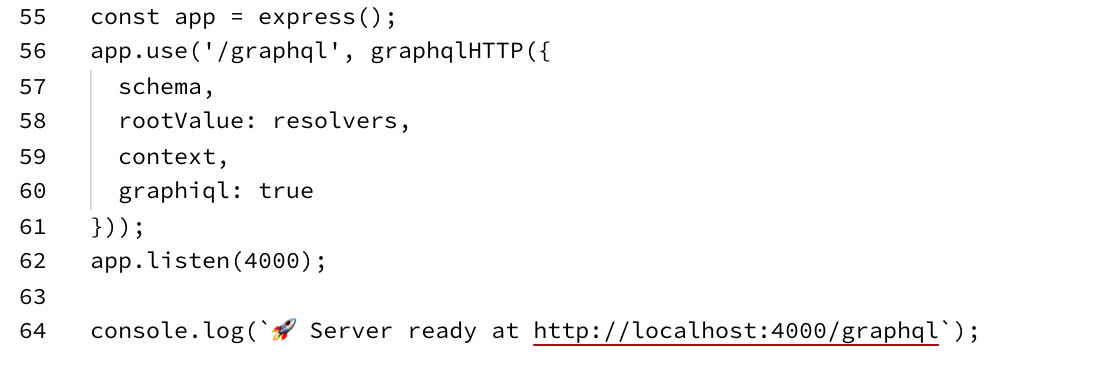
The first part imports and sets up the connection to the Mongo database.  
  
  
  
Next is the definition of the GraphQL schema.  
  


The next interesting part is:



This defines what queries do when called.   
  
For example,

* when you do a GraphQL query “bios”
* this will do a mongodb   
    
  db.collection(‘bios’).find().toArray().

The rest of the file is basically “boilerplate” and would be almost the same in any other example using express-graphql to implement graphql.  
  
  
  
One interesting thing to note is the enabling of *GraphiQL:*

graphiql: true

This is super cool and we’ll see it in a minute.

1. Start the server  
     
   $ node index.js  
   ﻿🚀 Server ready at <http://localhost:4000/graphql>
2. You may see a warning:  
   node:4087) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.

Ignore this!

1. In a new terminal window try:

﻿http localhost:4000/graphql query='{ bios { name { first }}}'  
  
You should see something like:  
﻿HTTP/1.1 200 OK

Connection: keep-alive

Content-Length: 298

Content-Type: application/json; charset=utf-8

Date: Wed, 27 Nov 2019 08:56:09 GMT

ETag: W/"12a-aMvPeBKQdQnnT/UJvxWxZ4tD9Pc"

X-Powered-By: Express

{

"data": {

"bios": [

{

"name": {

"first": "Kristen"

}

},

{

"name": {

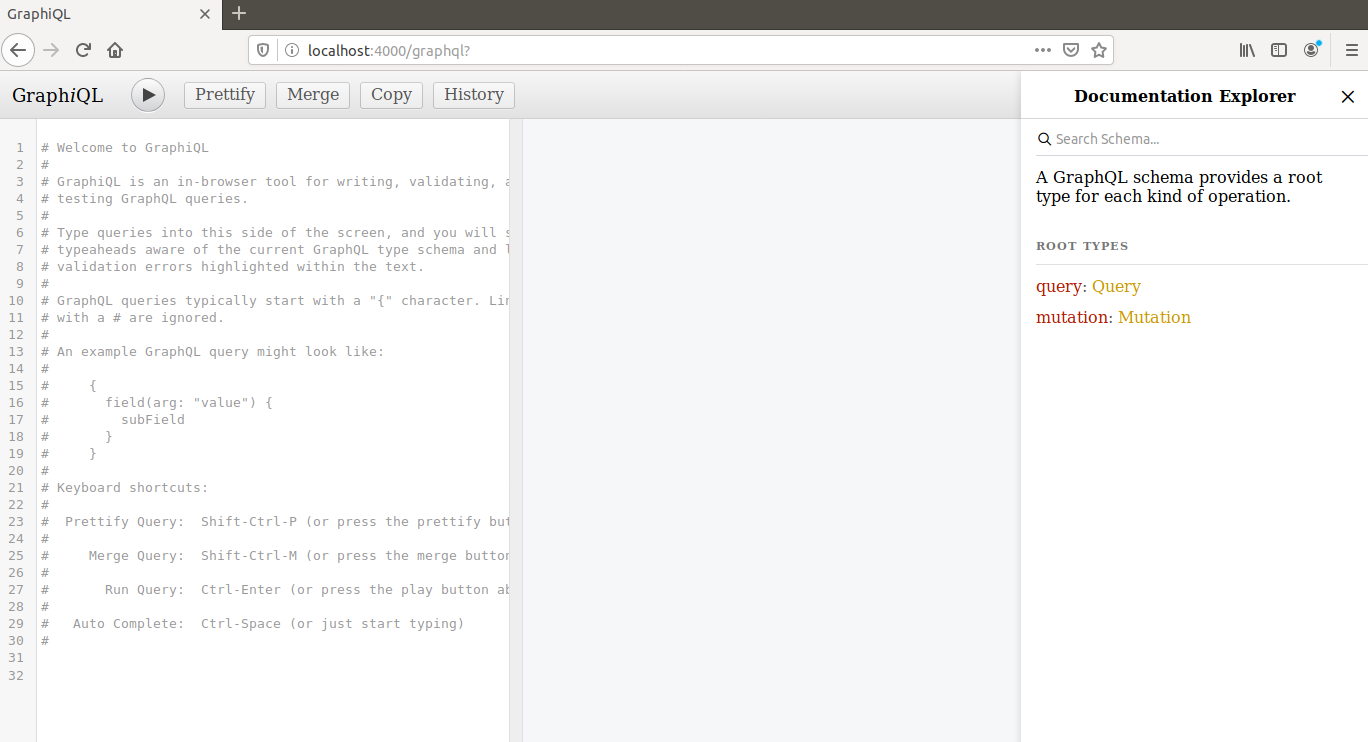
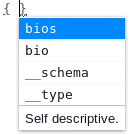
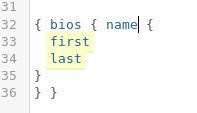
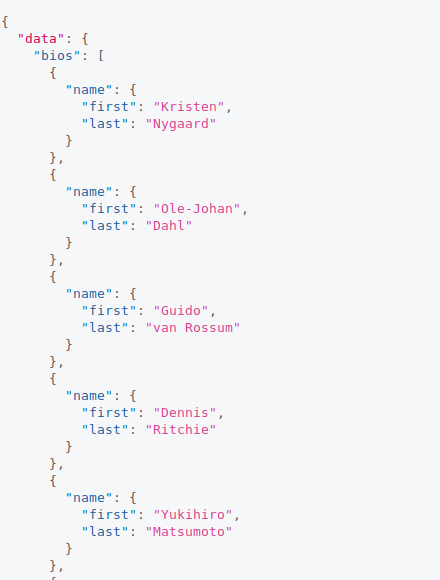
"first": "Ole-Johan"

}

},

…

}

1. Now browse to <http://localhost:4000/graphql>  
   This is the GraphiQL interface (pronounced “graphical”).  
     
   You should see something like:  
   
2. Have a read of the commented out help.
3. Below the comments start typing:   
   { bi  
   You will see the auto-completion kick in:  
     
   
4. Add name to the query:  
   
5. Hit the Play button  or Ctrl-Enter
6. You will see GraphiQL will add first / last into the query to make it into a valid query:  
   
7. You should see the query response like this:  
   
8. ﻿If we look at the schema (in index.js) again, you should see this part:  
   ﻿type Query {

bios: [Bio]

bio(id: Int): Bio

}

And this is the corresponding code:

﻿bios: (args, context) =>context().then(db =>

db.collection('bios').find().toArray()),

bio: (args, context) =>context().then(db =>

db.collection('bios').findOne({ \_id: args.id })),

What this means, is that the “bios” query has no parameters and pulls back all the records from the collection (find()), while the “bio” query has a single parameter (id) and queries the collection to findOne with that id.

1. Try out the find one method:  
     
     
   ﻿{ bio(id:1) {

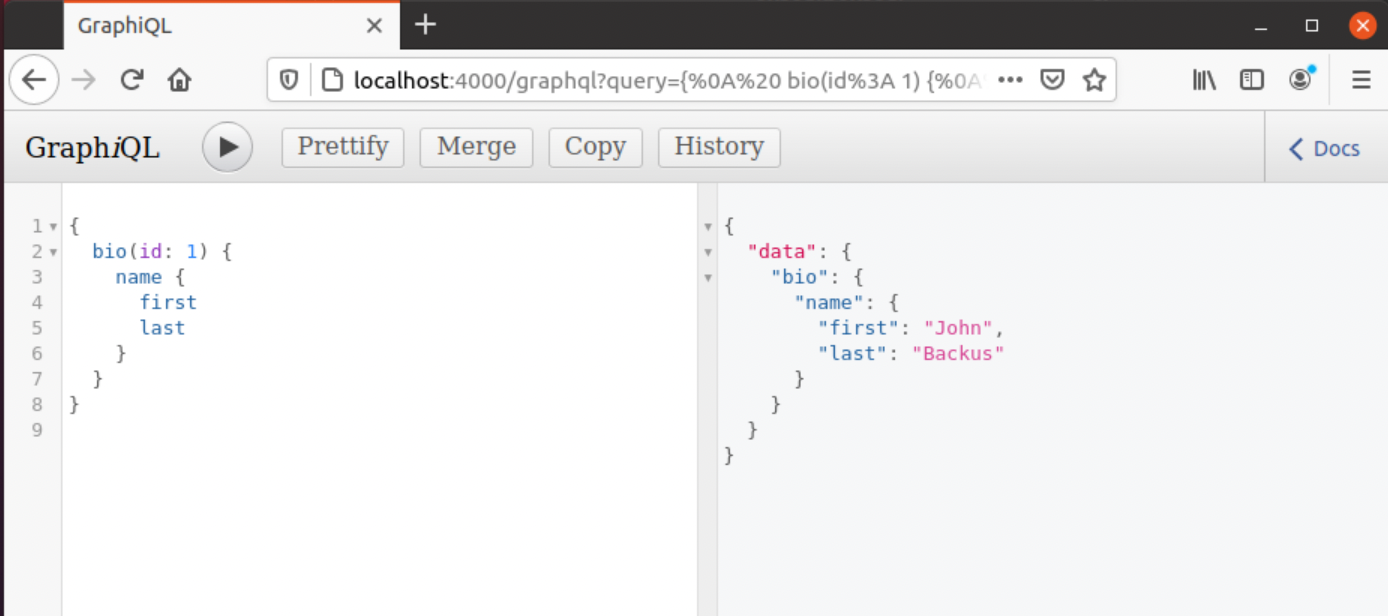
name {

first

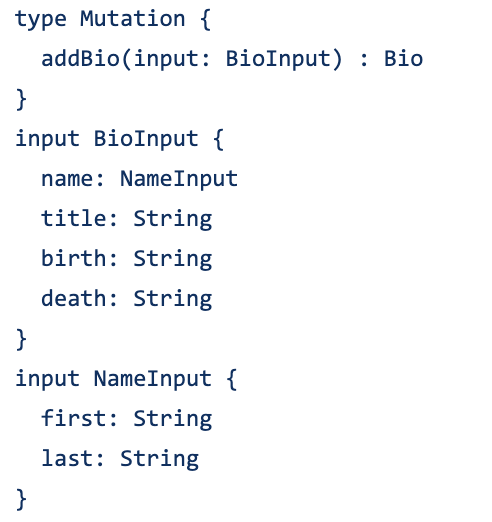
last

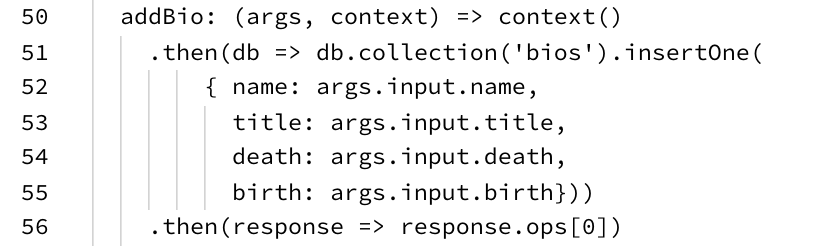
}

}}



1. Updates in GraphQL are called mutations.

Here is the part of the schema that lets us do an update:  


And here is the code that is called when you do a mutation:  


1. Try adding some data into the database:  
     
   mutation {

addBio(input: { name: { first: "John", last: "Smith" } })

{ name { first, last } }

}

1. Rerun the “bios” query and you will now see John Smith in the list
2. Re-run the update and new query from HTTPie (i.e. not using GraphiQL)
3. Is GraphQL “restful”? What reasons do you have for saying yes or no?

If you aren’t doing extensions you can remove the docker instance running Mongo and stop the node.js server.

That’s all for a basic intro to GraphQL

**Extensions:**

1. Add a query to search by first name and return all the records with that first name.
2. If you have done the API management lab, add your GraphQL API as a managed API  
   (you will need to create the schema as a separate file)  
     
   Docs are here:  
   <https://apim.docs.wso2.com/en/latest/learn/design-api/create-api/create-a-graphql-api/>
3. **(Harder!)**Create an Order service that has a similar schema to our RESTful service but uses GraphQL instead.