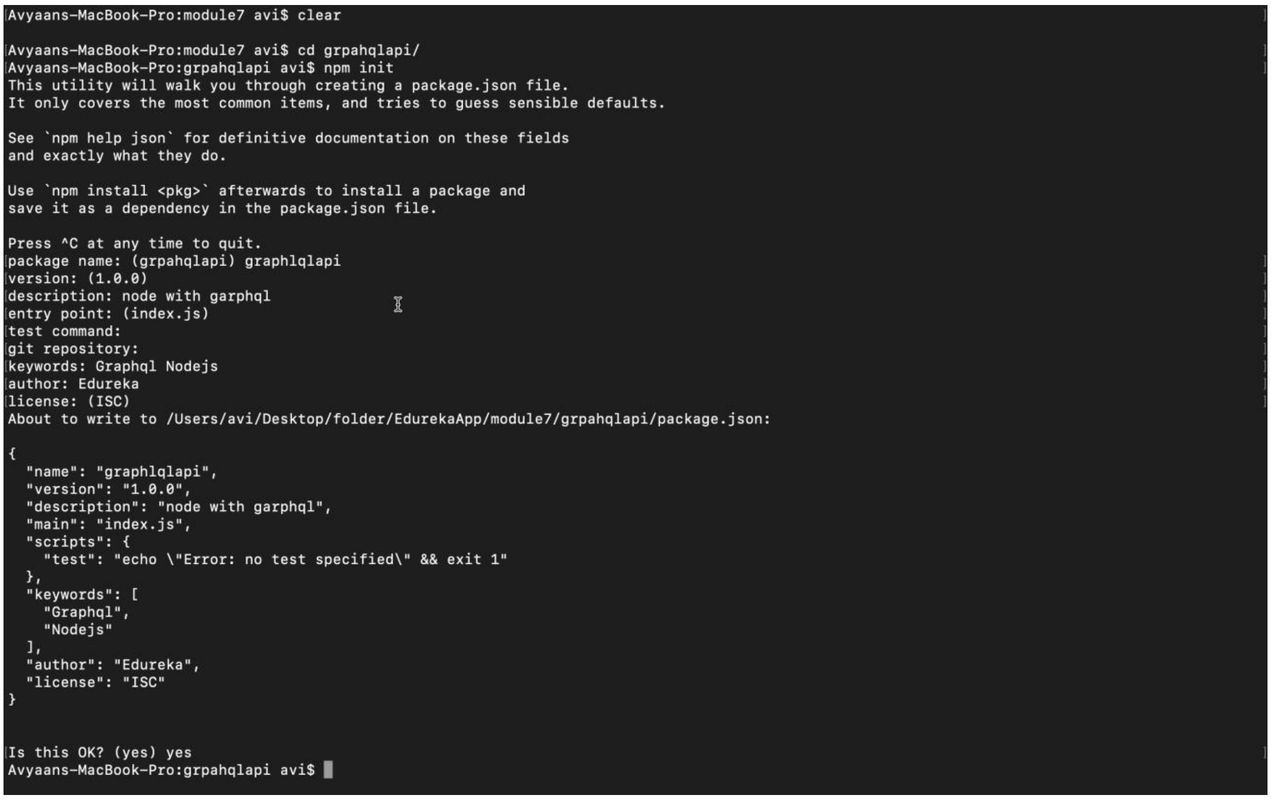
Module 5: REST APIs And GraphQL

Demo Document 3

edureka!

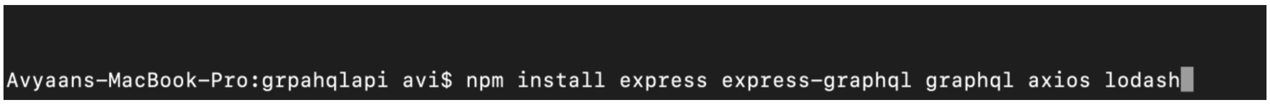
Working with GraphQL API

Step 1: We will be creating GraphQl API using Nodejs and express. For that first we must generate the package.json file



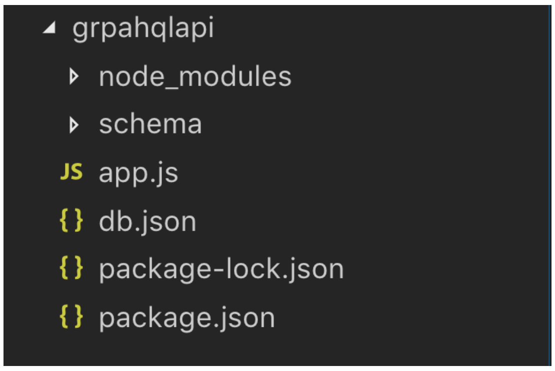
```
Avyaans-MacBook-Pro:module7 avi$ clear

Avyaans-MacBook-Pro:module7 avi$ cd grpahqlapi/
Avyaans-MacBook-Pro:grpahqlapi avi$ 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 json` 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: (grpahqlapi) graphlqlapi
version: (1.0.0)
description: node with garphql
entry point: (index.js)
test command:
git repository:
keywords: Graphql Nodejs
author: Edureka
license: (ISC)
About to write to /Users/avi/Desktop/folder/EdurekaApp/module7/grpahqlapi/package.json:

{
  "name": "graphlqlapi",
  "version": "1.0.0",
  "description": "node with garphql",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Graphql",
    "Nodejs"
  ],
  "author": "Edureka",
  "license": "ISC"
}


Is this OK? (yes) yes
Avyaans-MacBook-Pro:grpahqlapi avi$
```

Step 2: Then we need to install package listed below for GraphQL Api



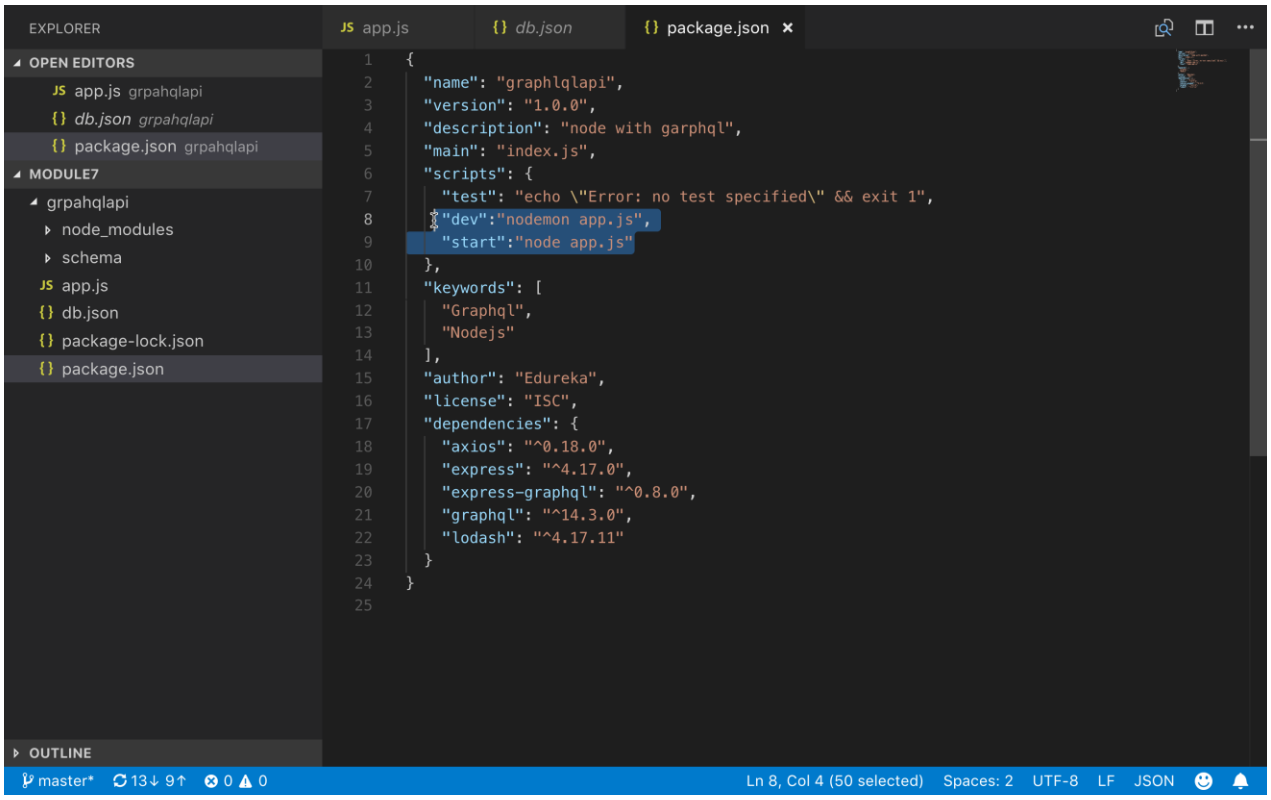
```
Avyaans-MacBook-Pro:grpahqlapi avi$ npm install express express-graphql graphql axios lodash
```

Step 3: In the graphQL api folder we need to one schema folder to define graphQL Schema



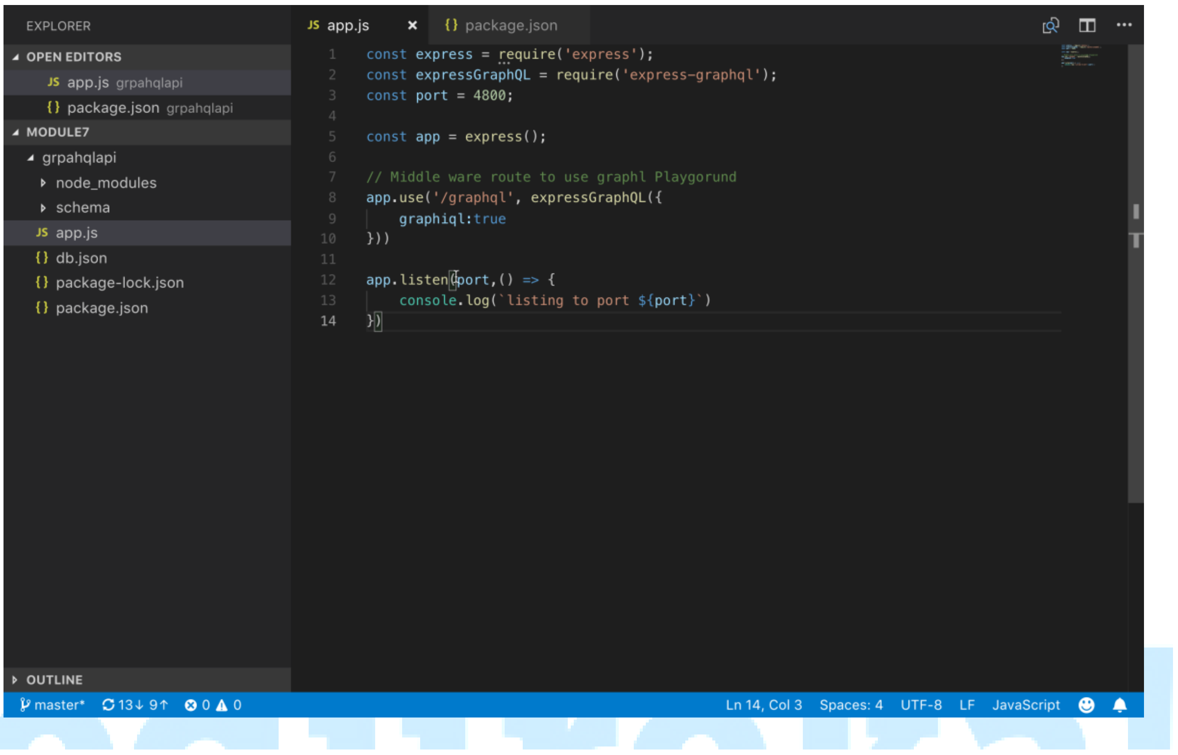
```
grpahqlapi
  node_modules
  schema
  app.js
  db.json
  package-lock.json
  package.json
```

Step 4: Then we need to start application in Production and Dev mode. For this we need to add commands in package.json



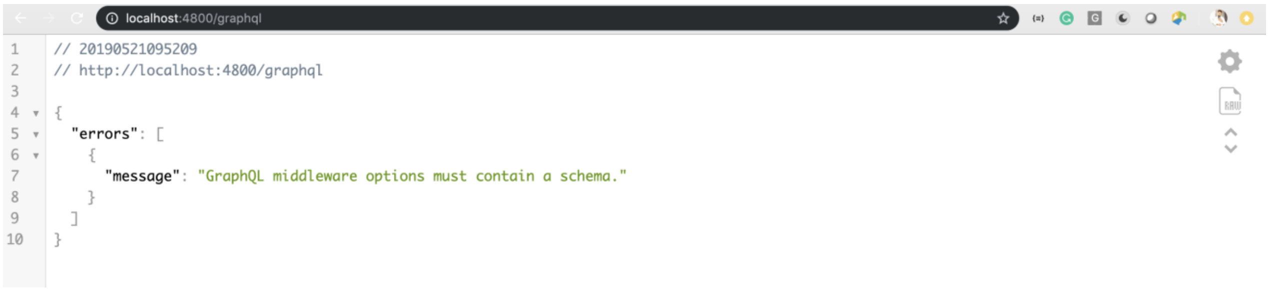
```
{
  "name": "graphlqlapi",
  "version": "1.0.0",
  "description": "node with garphql",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev":"nodemon app.js",
    "start":"node app.js"
  },
  "keywords": [
    "Graphql",
    "Nodejs"
  ],
  "author": "Edureka",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.18.0",
    "express": "^4.17.0",
    "express-graphql": "^0.8.0",
    "graphql": "^14.3.0",
    "lodash": "^4.17.11"
  }
}
```

Step 5: Add Grpahql as middleware and set graphQli as true to use GraphQL interface on the browser



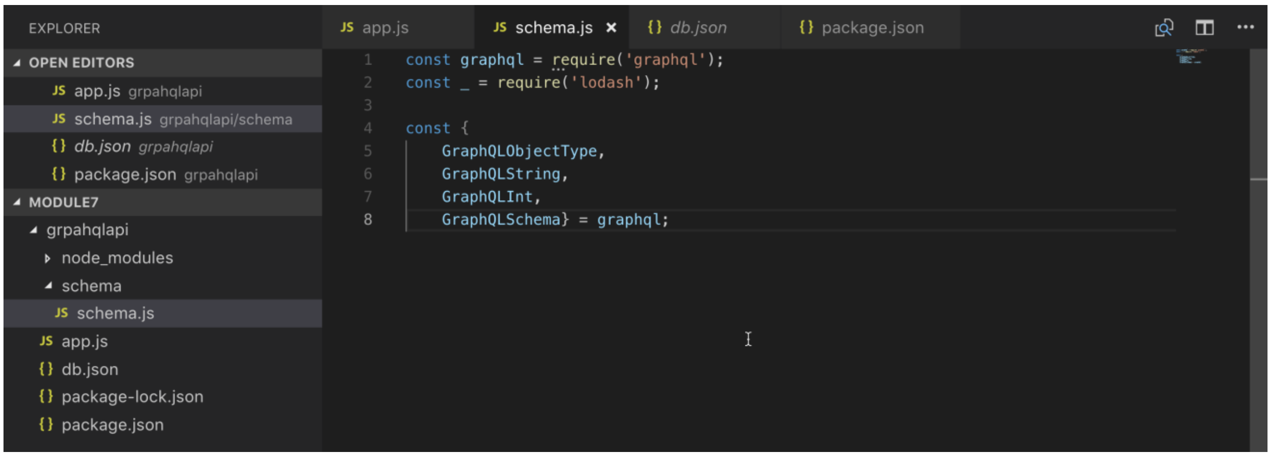
```
const express = require('express');
const expressGraphQL = require('express-graphql');
const port = 4800;

const app = express();

// Middle ware route to use graphl Playgorund
app.use('/graphql', expressGraphQL({
    graphiql:true
}))

app.listen(port,() => {
    console.log(`listing to port ${port}`)
})
```

Step 6: After that when we run app it will give you error as Graphql need schema to execute the app



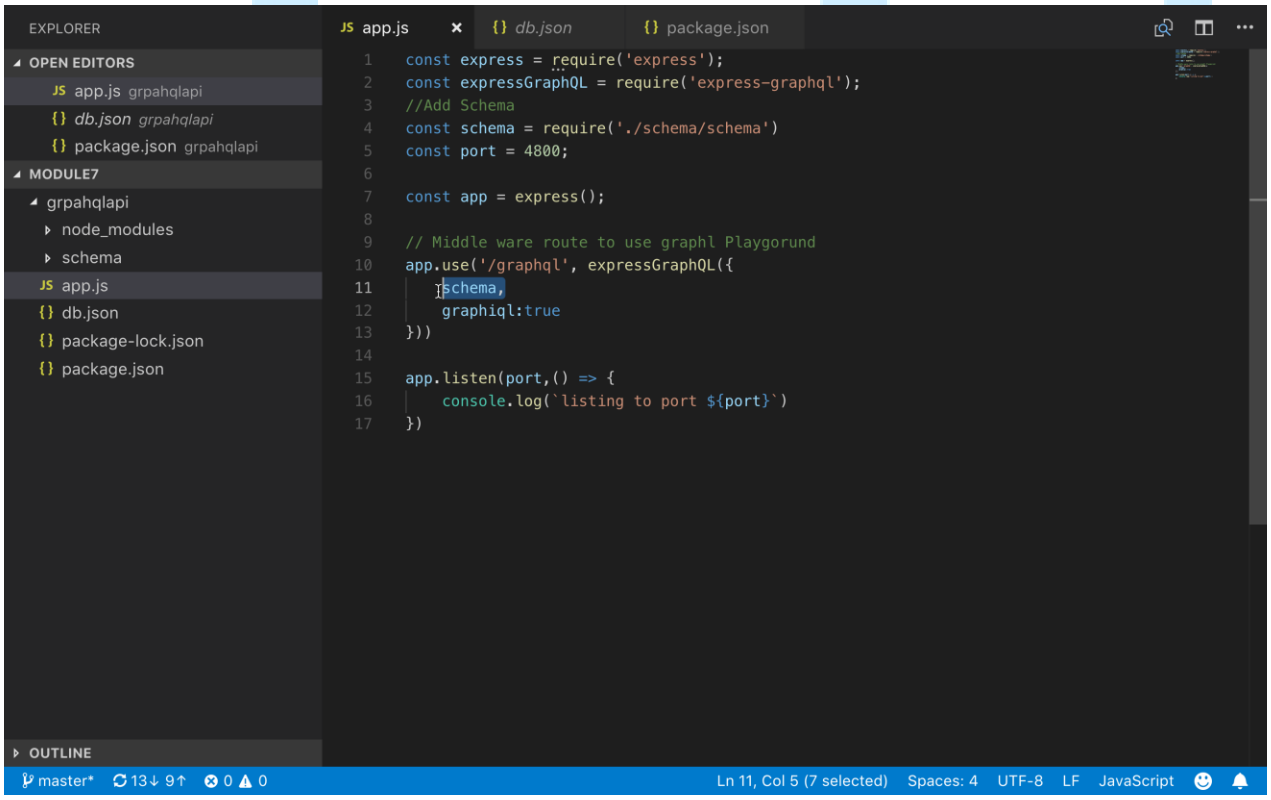
```
// 20190521095209
// http://localhost:4800/graphql

{
  "errors": [
    {
      "message": "GraphQL middleware options must contain a schema."
    }
  ]
}
```

Step 7: In schema first, we must declare object of graphQL



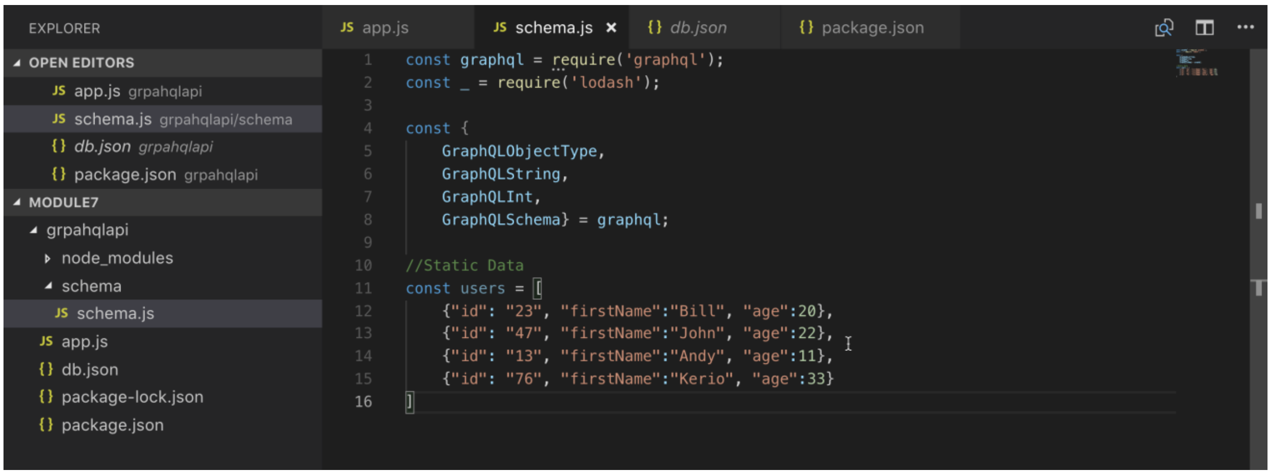
```
const graphql = require('graphql');
const _ = require('lodash');

const {
    GraphQLObjectType,
    GraphQLString,
    GraphQLInt,
    GraphQLSchema} = graphql;
```

Step 8: Defined the schema that is needed to be added in app file where we are using graphql as middleware



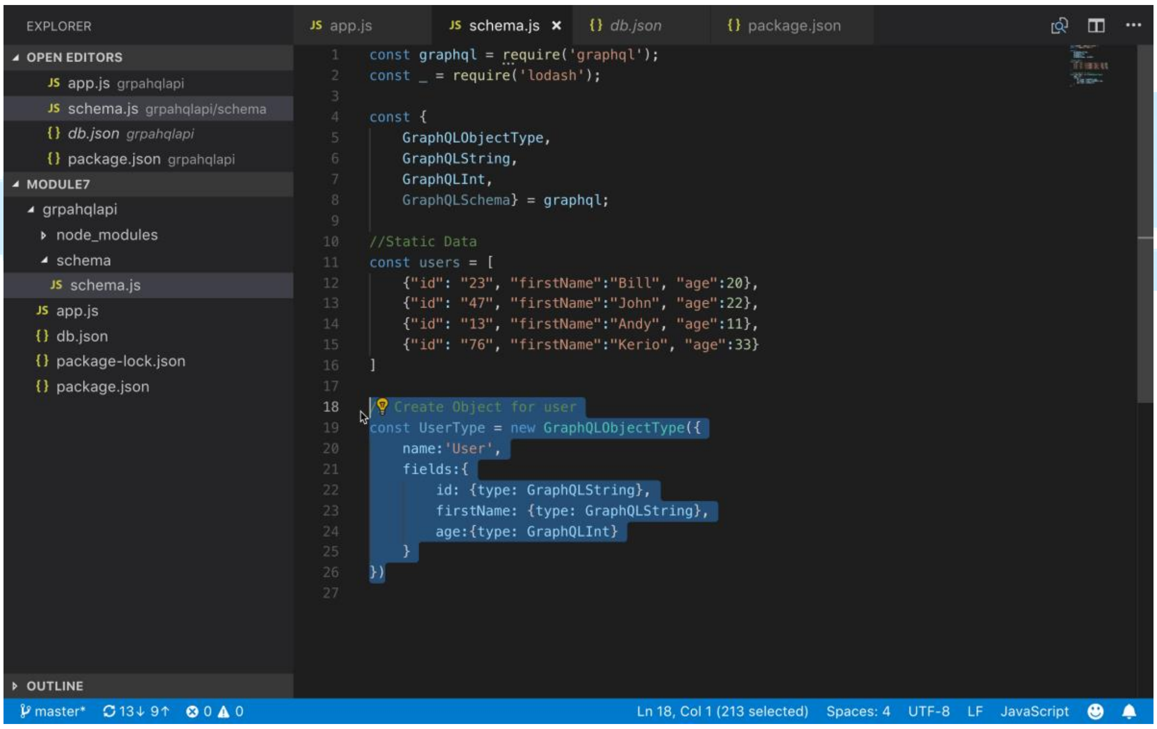
```
const express = require('express');
const expressGraphQL = require('express-graphql');
//Add Schema
const schema = require('./schema/schema')
const port = 4800;

const app = express();

// Middle ware route to use graphl Playgorund
app.use('/graphql', expressGraphQL({
    schema,
    graphiql:true
}))

app.listen(port,() => {
    console.log(`listing to port ${port}`)
})
```

Step 9: Now we need to define some static data that we will consume in the graphQL API



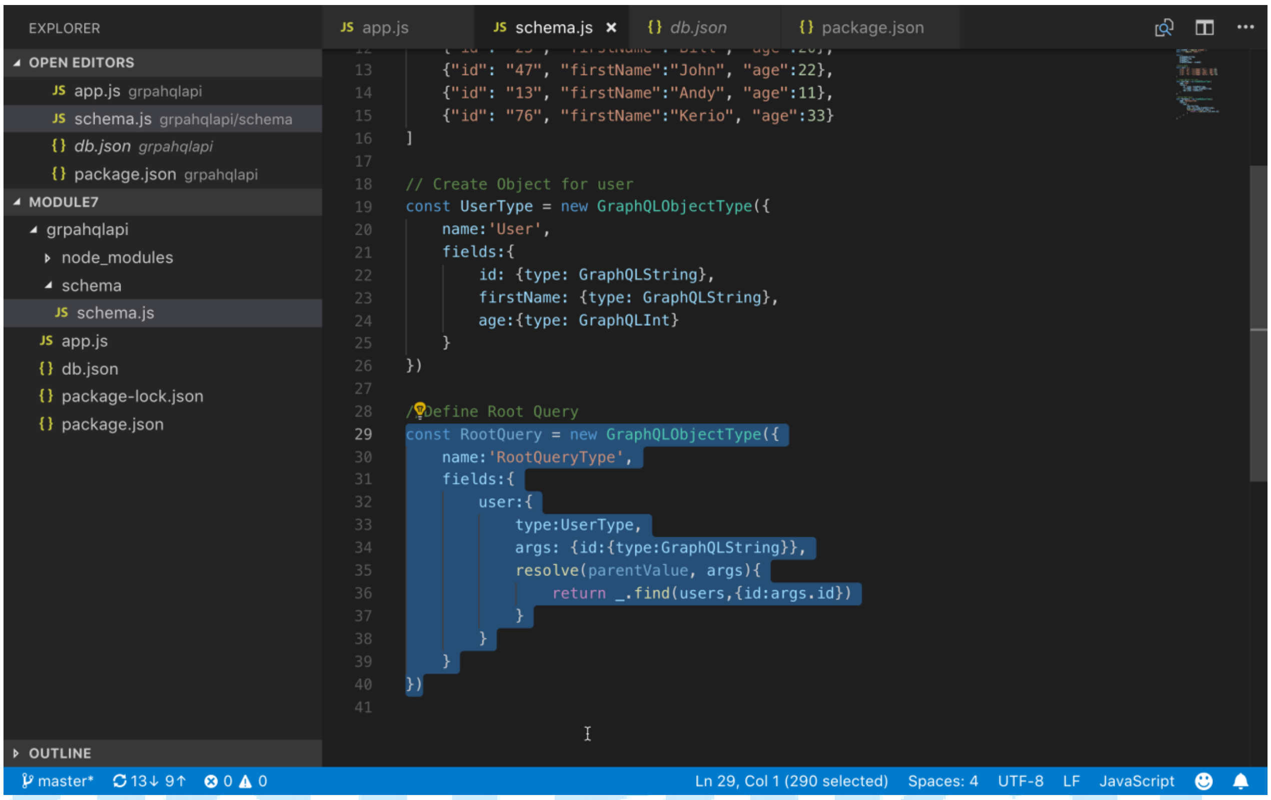
```
const graphql = require('graphql');
const _ = require('lodash');

const {
    GraphQLObjectType,
    GraphQLString,
    GraphQLInt,
    GraphQLSchema} = graphql;

//Static Data
const users = [
    {"id": "23", "firstName":"Bill", "age":20},
    {"id": "47", "firstName":"John", "age":22},
    {"id": "13", "firstName":"Andy", "age":11},
    {"id": "76", "firstName":"Kerio", "age":33}
]
```

Step 10: After data we create an object for the user, where we define the data type of keys using GrpahQl Object type.



```
const graphql = require('graphql');
const _ = require('lodash');

const {
    GraphQLObjectType,
    GraphQLString,
    GraphQLInt,
    GraphQLSchema} = graphql;

//Static Data
const users = [
    {"id": "23", "firstName":"Bill", "age":20},
    {"id": "47", "firstName":"John", "age":22},
    {"id": "13", "firstName":"Andy", "age":11},
    {"id": "76", "firstName":"Kerio", "age":33}
]

//Create Object for user
const UserType = new GraphQLObjectType({
    name:'User',
    fields:{
        id: {type: GraphQLString},
        firstName: {type: GraphQLString},
        age:{type: GraphQLInt}
    }
})
```

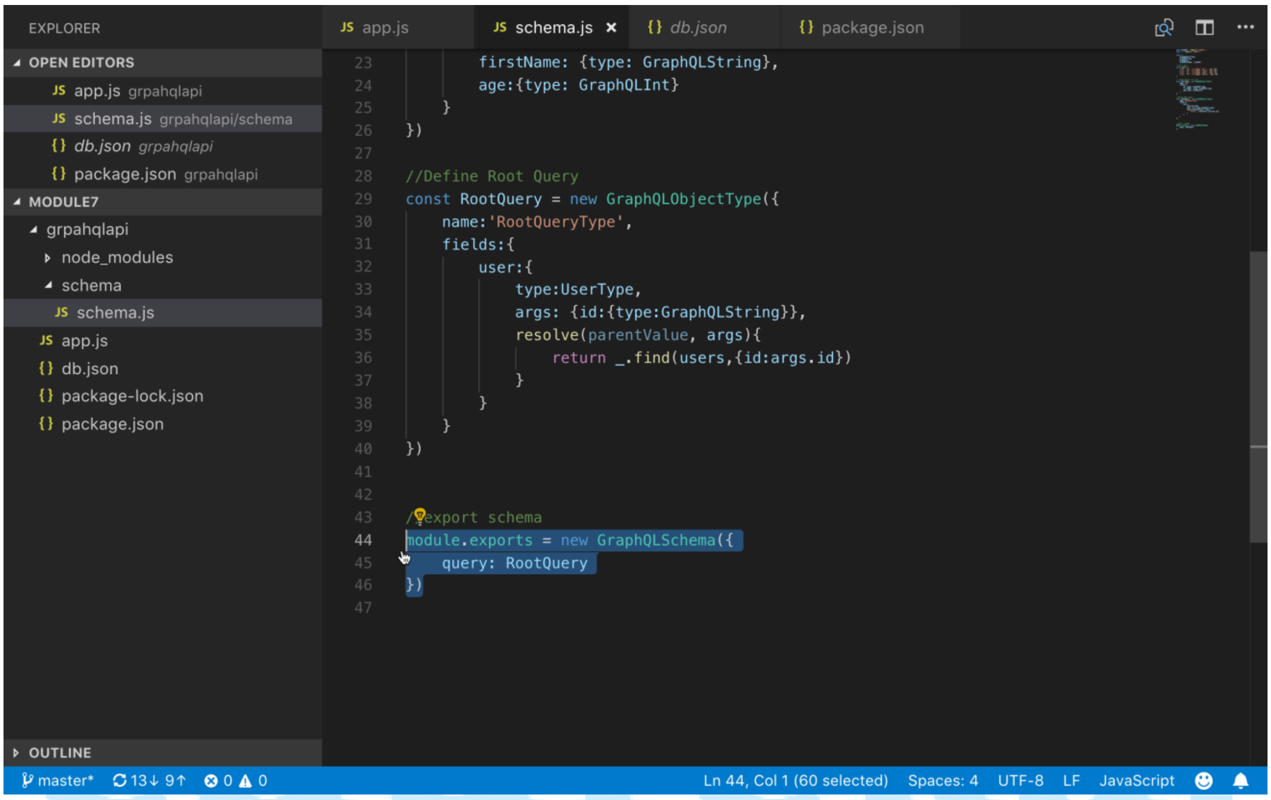
Step 11: Finally, we need root query to define the user data and create Query based on which we



```
    {"id": "47", "firstName":"John", "age":22},
    {"id": "13", "firstName":"Andy", "age":11},
    {"id": "76", "firstName":"Kerio", "age":33}
]

// Create Object for user
const UserType = new GraphQLObjectType({
    name:'User',
    fields:{
        id: {type: GraphQLString},
        firstName: {type: GraphQLString},
        age:{type: GraphQLInt}
    }
})

//Define Root Query
const RootQuery = new GraphQLObjectType({
    name:'RootQueryType',
    fields:{
        user:{
            type:UserType,
            args: {id:{type:GraphQLString}},
            resolve(parentValue, args){
                return _.find(users,{id:args.id})
            }
        }
    }
})
```

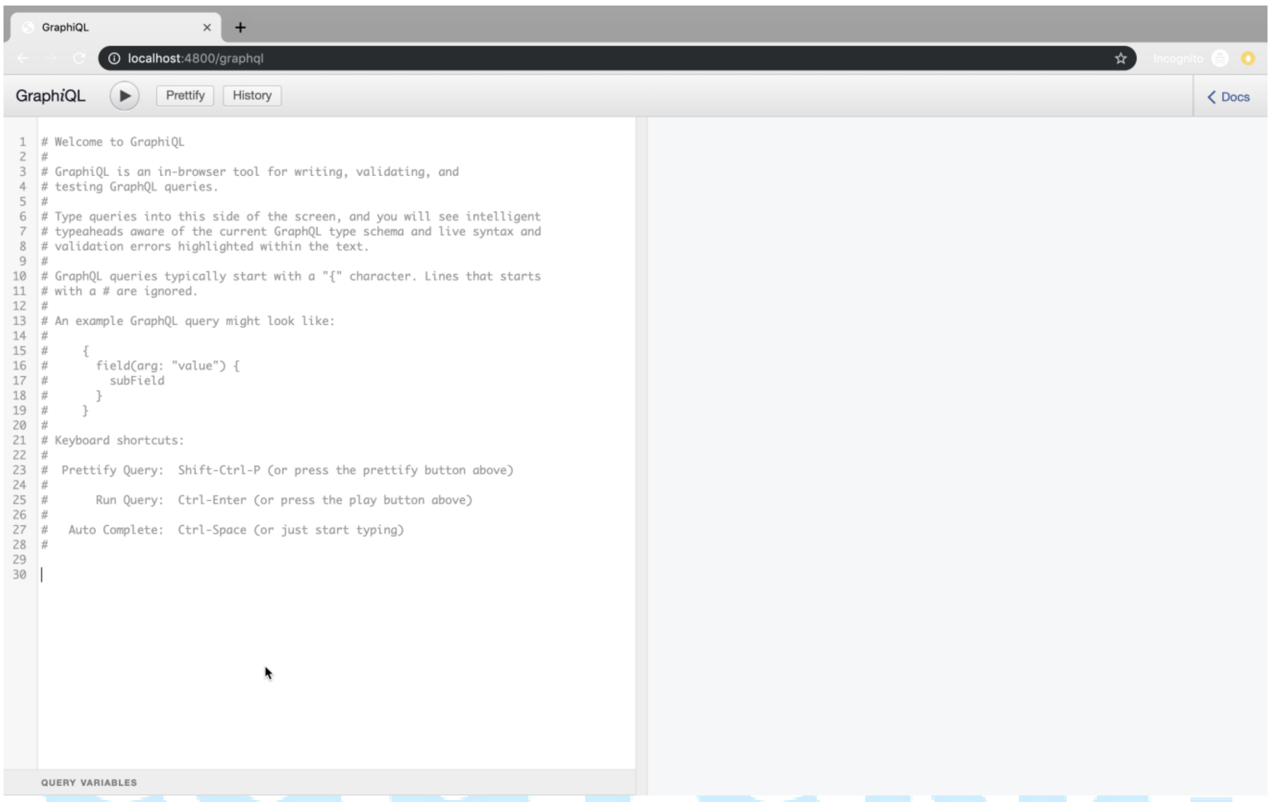
will process our search.

Step 12: As we are using schema in app.js we need to export schema file with root Query



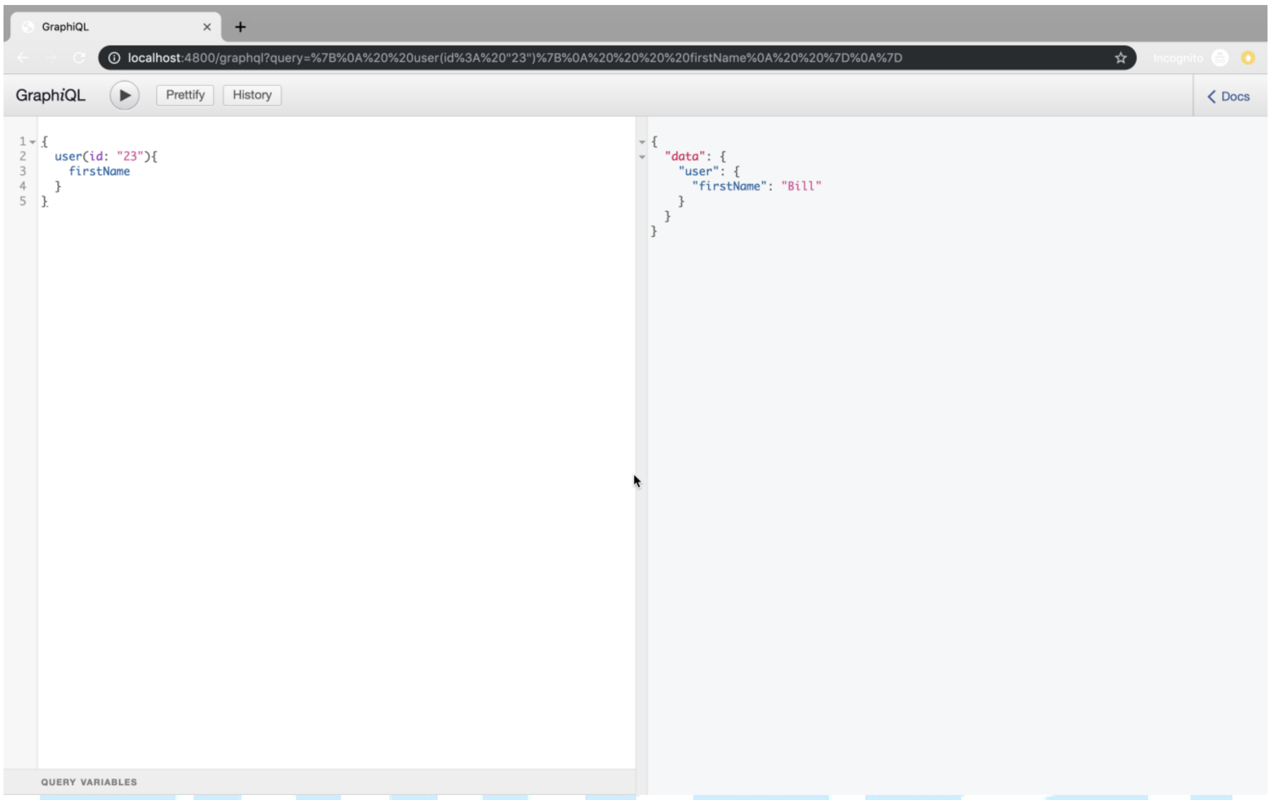
```
        firstName: {type: GraphQLString},
        age:{type: GraphQLInt}
    }
})

//Define Root Query
const RootQuery = new GraphQLObjectType({
    name:'RootQueryType',
    fields:{
        user:{
            type:UserType,
            args: {id:{type:GraphQLString}},
            resolve(parentValue, args){
                return _.find(users,{id:args.id})
            }
        }
    }
})


//export schema
module.exports = new GraphQLSchema({
    query: RootQuery
})
```

Step 13: As all things are in place now when we run our 'loaclhost:4800/graphql' it will open the playground of graphQL for us



```
# 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 starts
# 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)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#


```

Step 14: In root Query we have defined search based on the user id. Now we can define what all field we have based on this user ID.

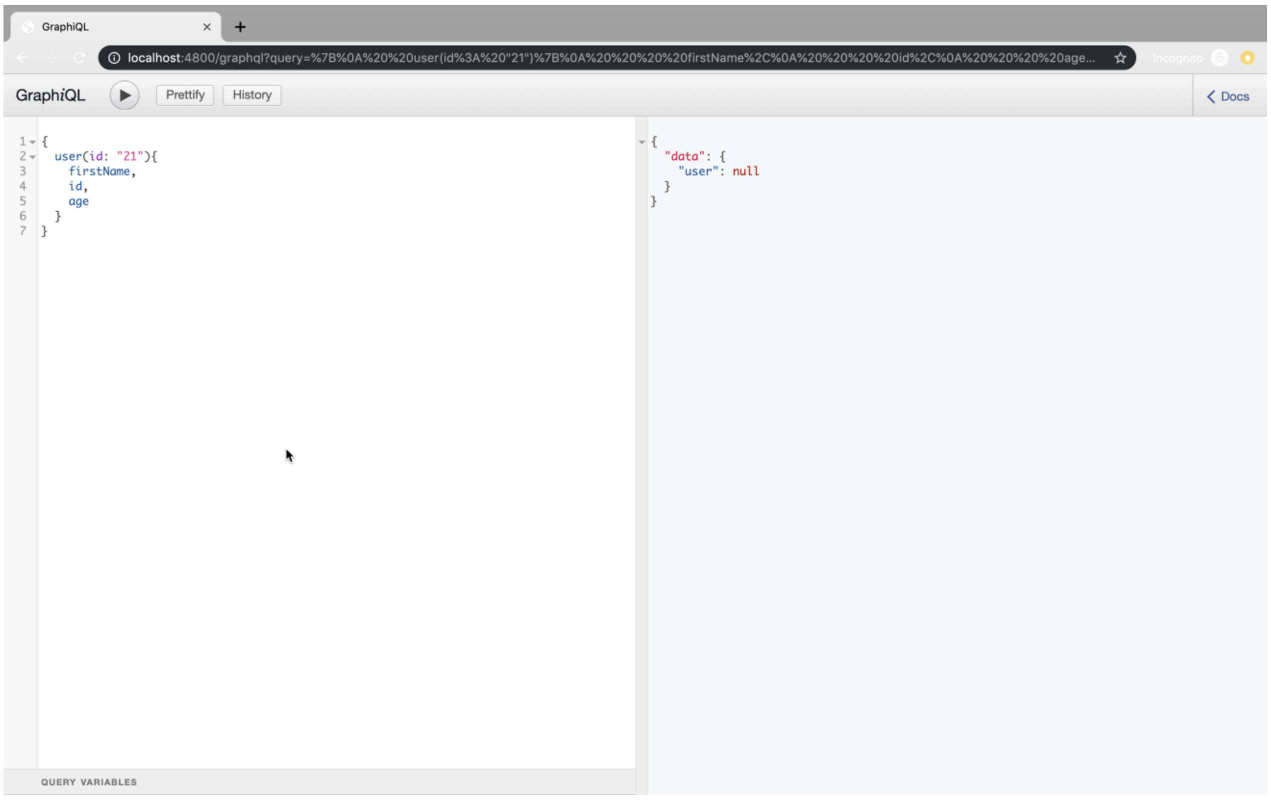


```
localhost:4800/graphql?query=%7B%0A%20%20user(id%3A%20"23")%7B%0A%20%20%20%20firstName%0A%20%20%7D%0A%7D
```

```
{
  user(id: "23"){
    firstName
  }
}
```

```
{
  "data": {
    "user": {
      "firstName": "Bill"
    }
  }
}
```

Step 15: If we enter wrong user id it will return null



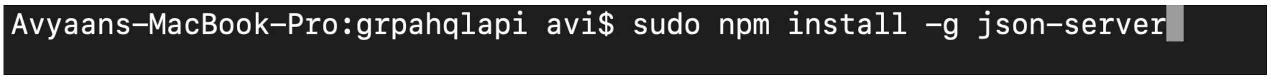
```
localhost:4800/graphql?query=%7B%0A%20%20user(id%3A%20"21")%7B%0A%20%20%20%20firstName%2C%0A%20%20%20%20id%2C%0A%20%20%20%20age...
```

```
{
  user(id: "21"){
    firstName,
    id,
    age
  }
}
```

```
{
  "data": {
    "user": null
  }
}
```

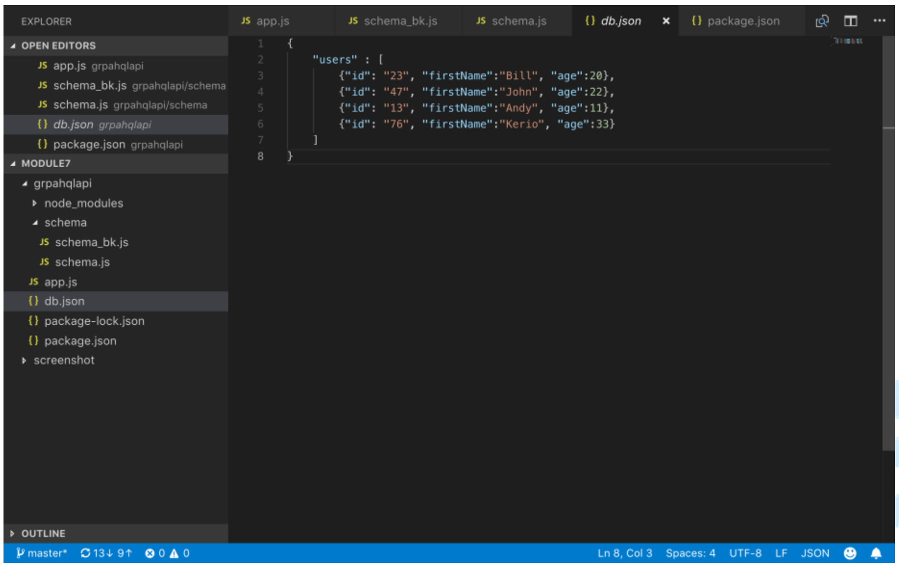
Step 16: Now let us see how to consume data through an API.

First we need to install json-server globally.



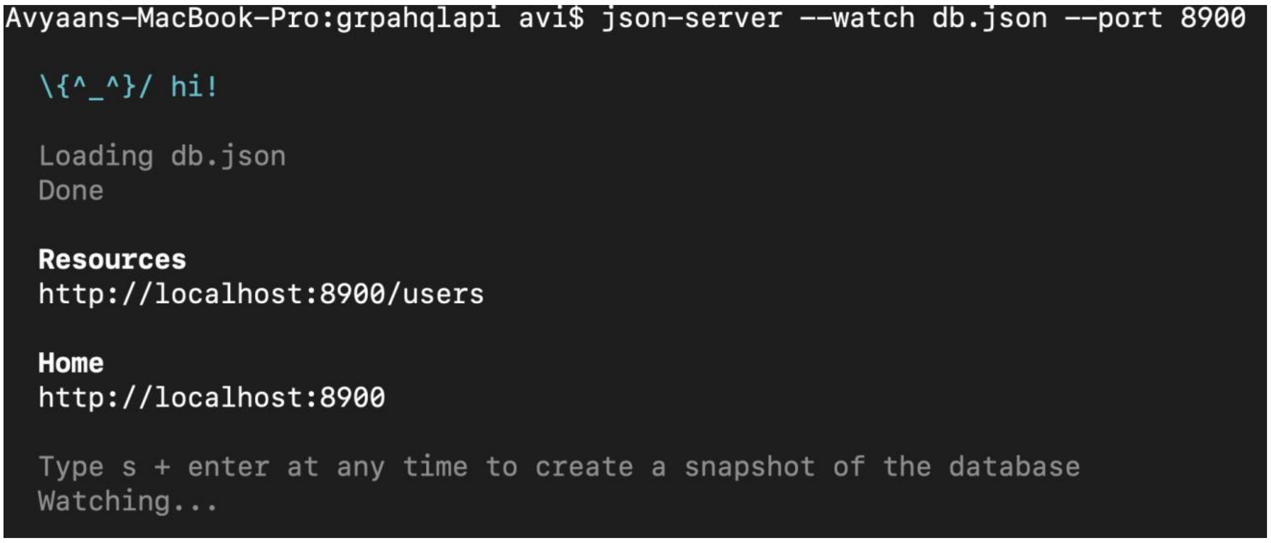
```
Avyaans-MacBook-Pro:grpahqlapi avi$ sudo npm install -g json-server
```

Step 17: Create Db.json file and move all the static data to this file



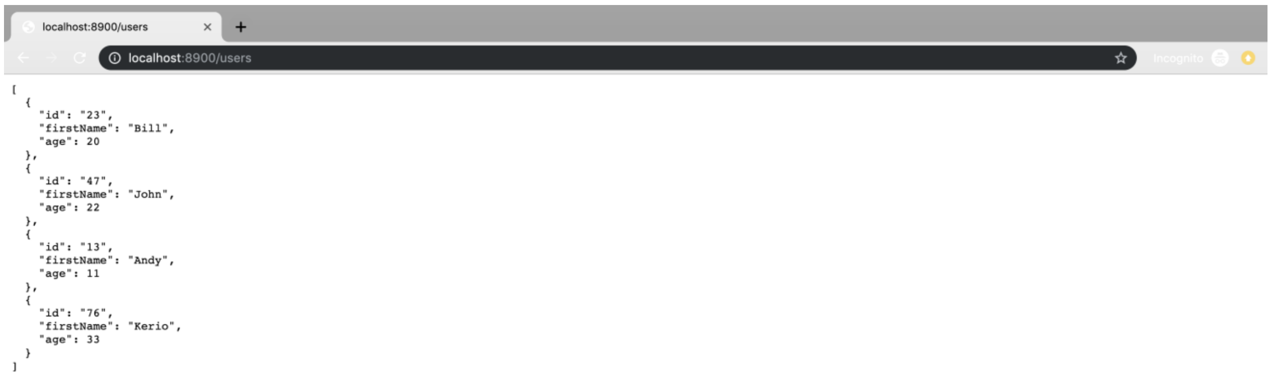
```
{
    "users" : [
        {"id": "23", "firstName":"Bill", "age":20},
        {"id": "47", "firstName":"John", "age":22},
        {"id": "13", "firstName":"Andy", "age":11},
        {"id": "76", "firstName":"Kerio", "age":33}
    ]
}
```

Step 18: By using 'json-server --watch db.json --port 8900' we have generated API with user route



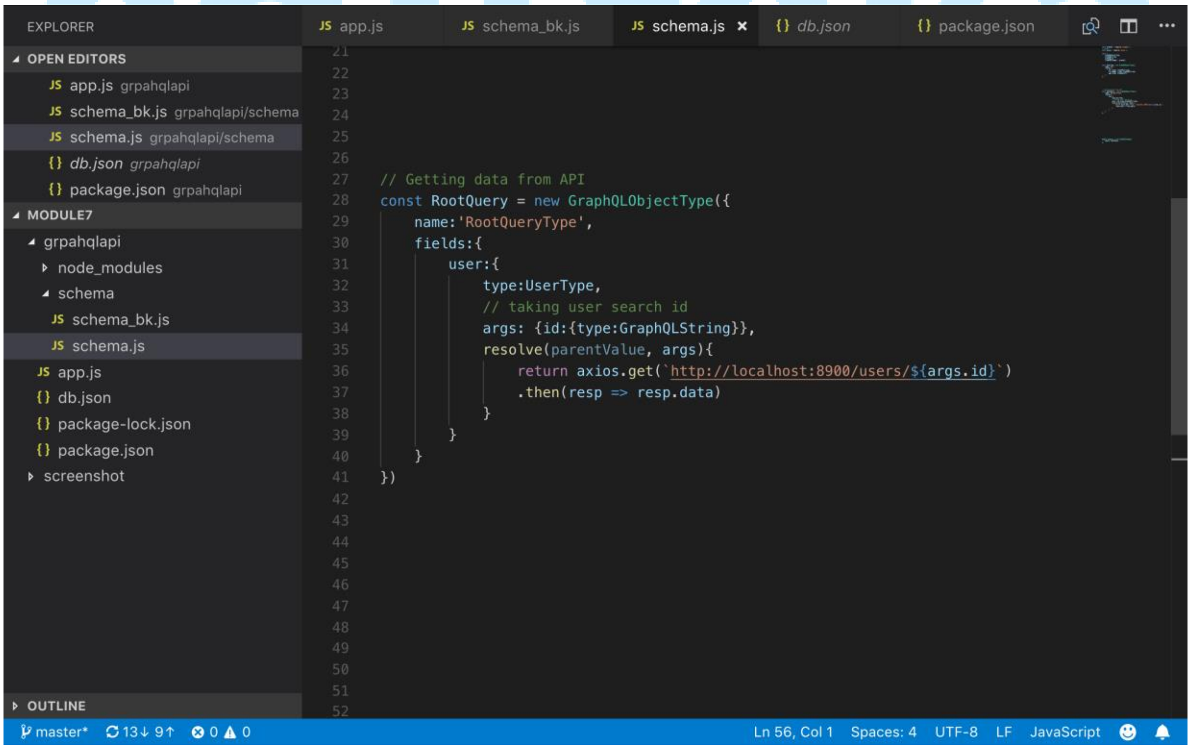
```
Avyaans-MacBook-Pro:grpahqlapi avi$ json-server --watch db.json --port 8900

  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:8900/users

  Home
  http://localhost:8900

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

Step 19: When we run api in browser we can see all results. In next step we will consume this data with graphQL



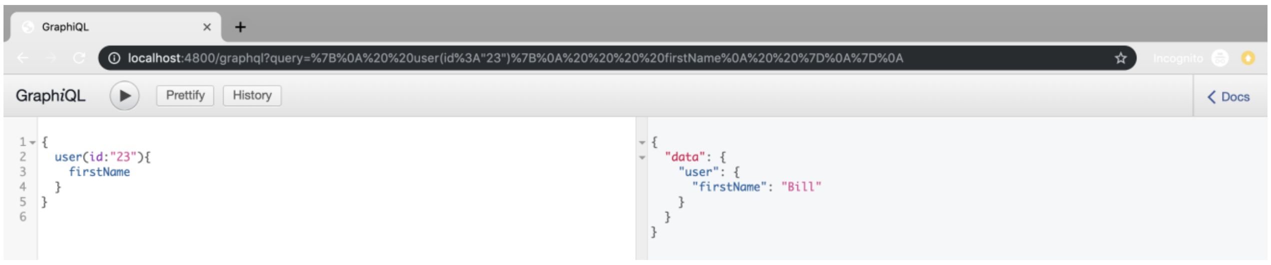
```
[
  {
    "id": "23",
    "firstName": "Bill",
    "age": 20
  },
  {
    "id": "47",
    "firstName": "John",
    "age": 22
  },
  {
    "id": "13",
    "firstName": "Andy",
    "age": 11
  },
  {
    "id": "76",
    "firstName": "Kerio",
    "age": 33
  }
]
```

Step 20: With the help of axios we will make an API call to API and in return we are getting user data



```
// Getting data from API
const RootQuery = new GraphQLObjectType({
    name:'RootQueryType',
    fields:{
        user:{
            type:UserType,
            // taking user search id
            args: {id:{type:GraphQLString}},
            resolve(parentValue, args){
                return axios.get(`http://localhost:8900/users/${args.id}`)
                .then(resp => resp.data)
            }
        }
    }
})
```

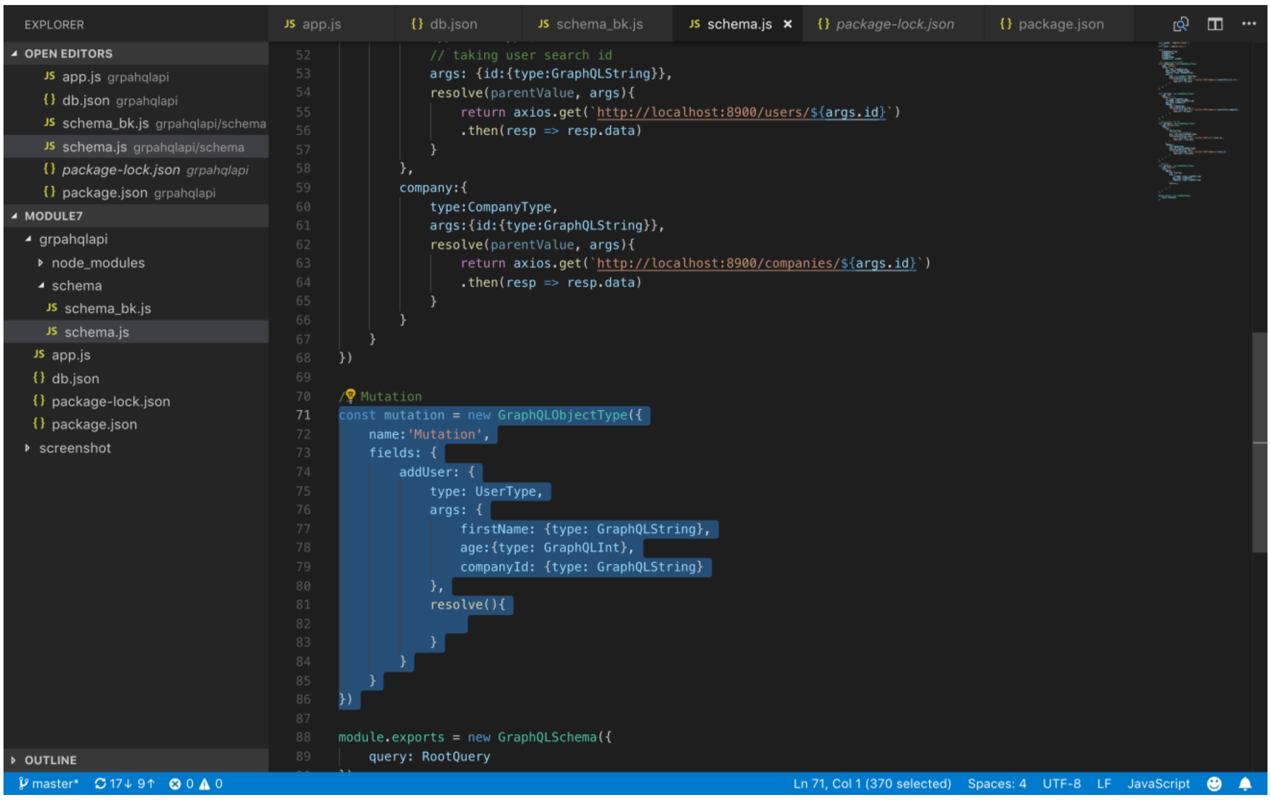
Step 21: When we try executing Graphql playground with user id we can get data and this data is coming from API


GraphiQL
localhost:4800/graphql?query=%7B%0A%20%20user(id%3A"23")%7B%0A%20%20%20%20firstName%0A%20%20%7D%0A%7D%0A
Prettify
History
Docs

```
{
  user(id:"23"){
    firstName
  }
}
```

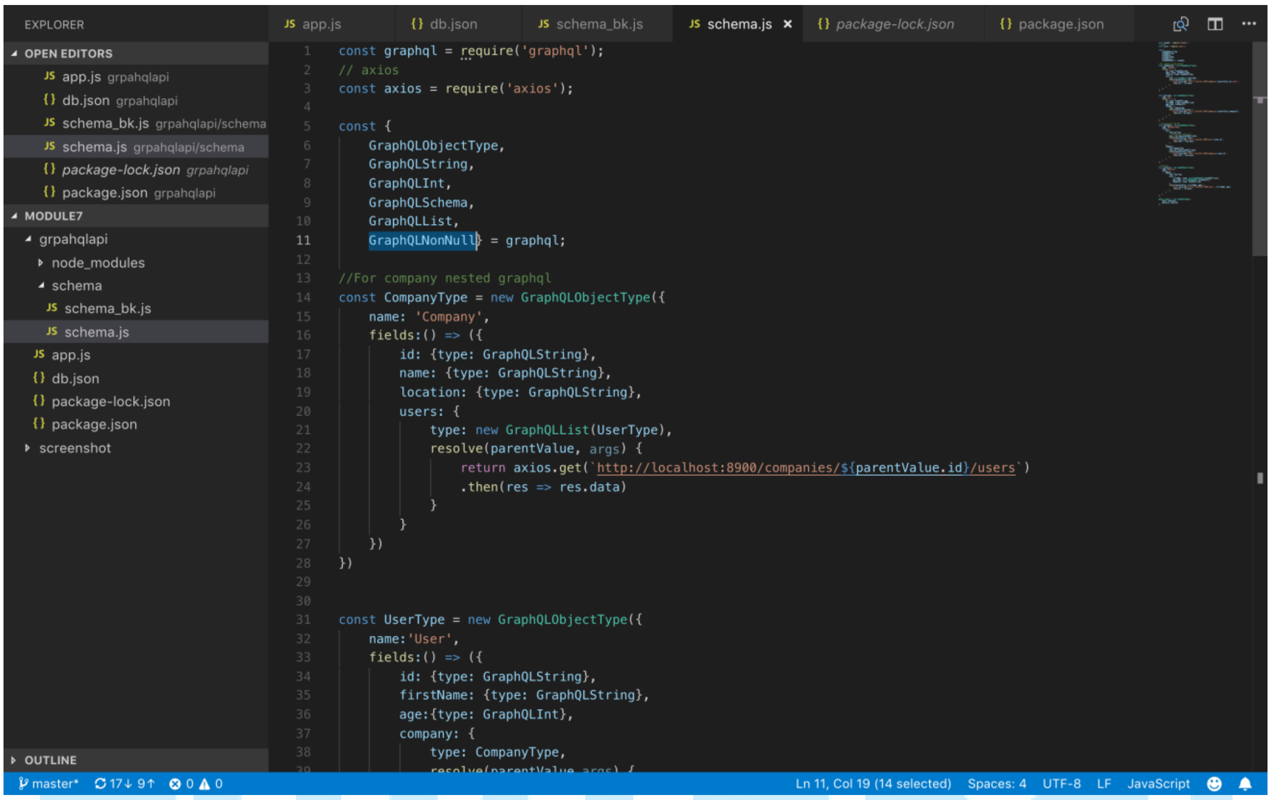
```
{
  "data": {
    "user": {
      "firstName": "Bill"
    }
  }
}
```

Step 22: As query is for the GET call, we need to add separate script known as mutation



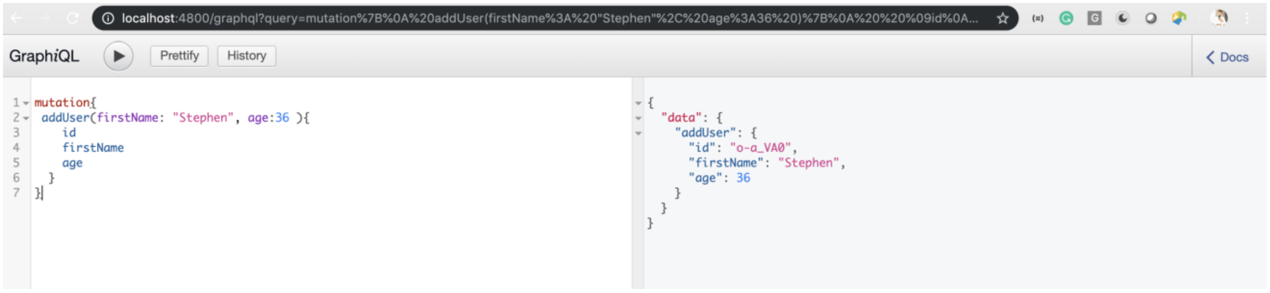
```
                // taking user search id
                args: {id:{type:GraphQLString}},
                resolve(parentValue, args){
                    return axios.get(`http://localhost:8900/users/${args.id}`)
                    .then(resp => resp.data)
                }
            },
            company:{
                type:CompanyType,
                args:{id:{type:GraphQLString}},
                resolve(parentValue, args){
                    return axios.get(`http://localhost:8900/companies/${args.id}`)
                    .then(resp => resp.data)
                }
            }
        }
})

//Mutation
const mutation = new GraphQLObjectType({
    name:'Mutation',
    fields: {
        addUser: {
            type: UserType,
            args: {
                firstName: {type: GraphQLString},
                age:{type: GraphQLInt},
                companyId: {type: GraphQLString}
            },
            resolve(){

            }
        }
    }
})

module.exports = new GraphQLSchema({
    query: RootQuery
```

Step 23: We have added mutation, now with axios only we can make post call to the API and send data



```
const graphql = require('graphql');
// axios
const axios = require('axios');

const {
    GraphQLObjectType,
    GraphQLString,
    GraphQLInt,
    GraphQLSchema,
    GraphQLList,
    GraphQLNonNull} = graphql;

//For company nested graphql
const CompanyType = new GraphQLObjectType({
    name: 'Company',
    fields:() => ({
        id: {type: GraphQLString},
        name: {type: GraphQLString},
        location: {type: GraphQLString},
        users: {
            type: new GraphQLList(UserType),
            resolve(parentValue, args) {
                return axios.get(`http://localhost:8900/companies/${parentValue.id}/users`)
                .then(res => res.data)
            }
        }
    })
})


const UserType = new GraphQLObjectType({
    name:'User',
    fields:() => ({
        id: {type: GraphQLString},
        firstName: {type: GraphQLString},
        age:{type: GraphQLInt},
        company: {
            type: CompanyType,
```

Step 24: Now we will try to post data using mutation and on return we will get auto generated Id

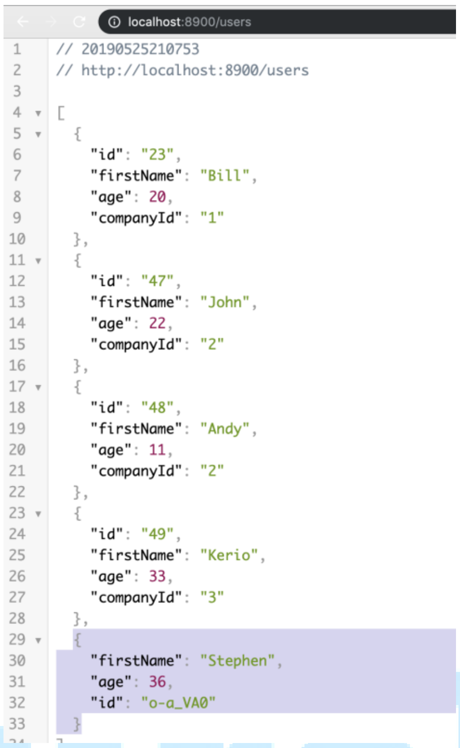


localhost:4800/graphql?query=mutation%7B%0A%20addUser(firstName%3A%20"Stephen"%2C%20age%3A36%20)%7B%0A%20%20%09id%0A...

```
mutation{
 addUser(firstName: "Stephen", age:36 ){
    id
    firstName
    age
  }
}
```

```
{
  "data": {
    "addUser": {
      "id": "o-a_VA0",
      "firstName": "Stephen",
      "age": 36
    }
  }
}
```

Step 25: Once done with mutation, if we check the real API then we will get the new data inserted using graphQL playground



localhost:8900/users

```
// 20190525210753
// http://localhost:8900/users

[
  {
    "id": "23",
    "firstName": "Bill",
    "age": 20,
    "companyId": "1"
  },
  {
    "id": "47",
    "firstName": "John",
    "age": 22,
    "companyId": "2"
  },
  {
    "id": "48",
    "firstName": "Andy",
    "age": 11,
    "companyId": "2"
  },
  {
    "id": "49",
    "firstName": "Kerio",
    "age": 33,
    "companyId": "3"
  },
  {
    "firstName": "Stephen",
    "age": 36,
    "id": "o-a_VA0"
  }
```

Thus, you have successfully learned how to use the GraphQL API