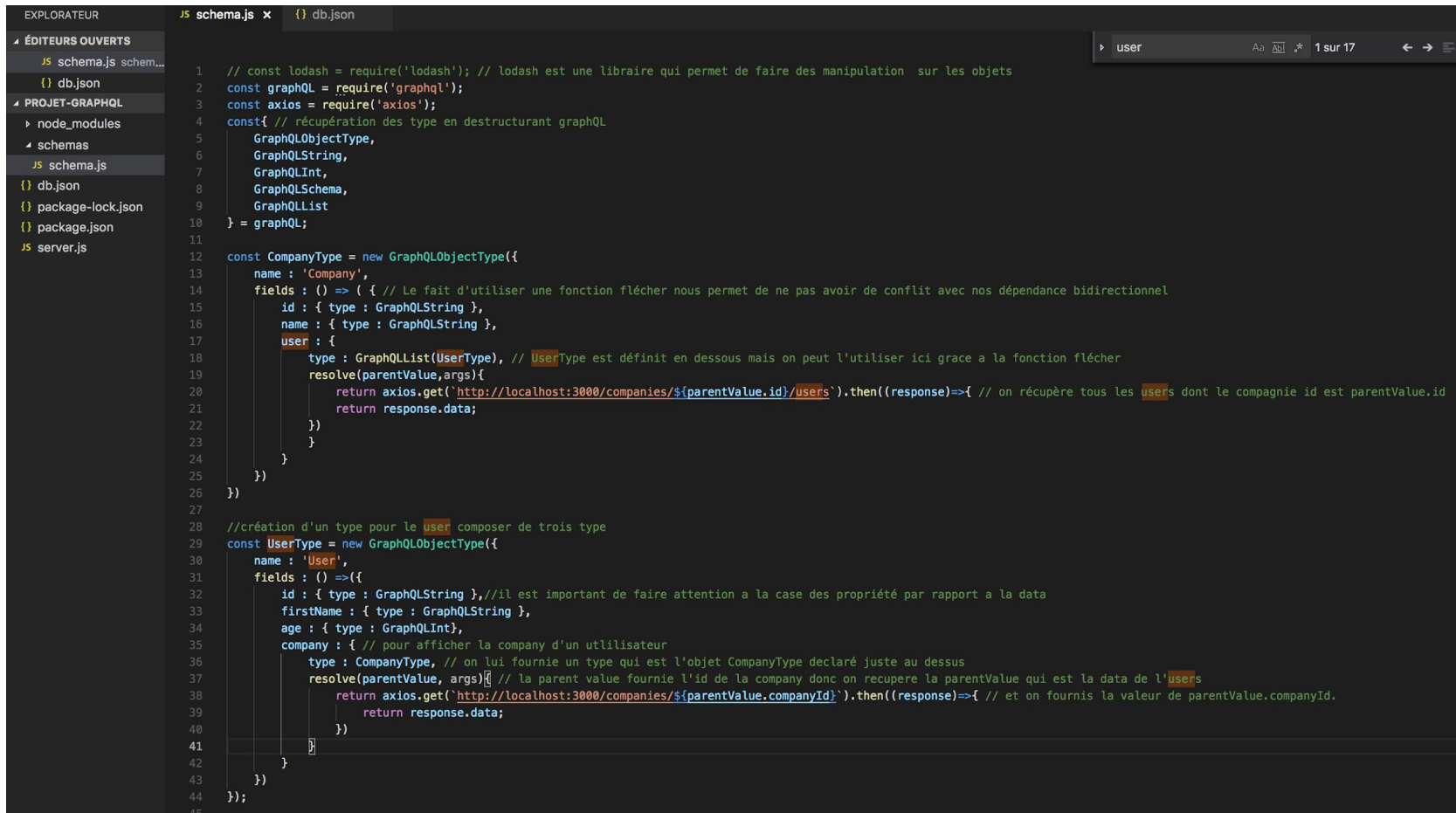


## Premiere partie du fichier schemas.js



```
EXPLORATEUR JS schema.js x {} db.json
ÉDITEURS OUVERTS
  JS schema.js schem...
  {} db.json
PROJET-GRAPHQL
  node_modules
  schemas
  JS schema.js
  {} db.json
  {} package-lock.json
  {} package.json
  JS server.js

1 // const lodash = require('lodash'); // lodash est une librairie qui permet de faire des manipulation sur les objets
2 const graphql = require('graphql');
3 const axios = require('axios');
4 const { // récupération des type en destructurant graphql
5   GraphQLObjectType,
6   GraphQLString,
7   GraphQLInt,
8   GraphQLSchema,
9   GraphQLList
10 } = graphql;
11
12 const CompanyType = new GraphQLObjectType({
13   name: 'Company',
14   fields: () => ( { // Le fait d'utiliser une fonction fléchier nous permet de ne pas avoir de conflit avec nos dépendance bidirectionnel
15     id: { type: GraphQLString },
16     name: { type: GraphQLString },
17     user: {
18       type: GraphQLList(UserType), // UserType est défini en dessous mais on peut l'utiliser ici grace a la fonction fléchier
19       resolve(parentValue, args) {
20         return axios.get(`http://localhost:3000/companies/${parentValue.id}/users`).then((response) => { // on récupère tous les users dont le compagnie id est parentValue.id
21           return response.data;
22         });
23       }
24     }
25   })
26 })
27
28 //création d'un type pour le user composer de trois type
29 const UserType = new GraphQLObjectType({
30   name: 'User',
31   fields: () => ({
32     id: { type: GraphQLString }, //il est important de faire attention a la case des propriété par rapport a la data
33     firstName: { type: GraphQLString },
34     age: { type: GraphQLInt },
35     company: { // pour afficher la company d'un utilisateur
36       type: CompanyType, // on lui fournis un type qui est l'objet CompanyType déclaré juste au dessus
37       resolve(parentValue, args) { // la parent value fournis l'id de la company donc on recupere la parentValue qui est la data de l'users
38         return axios.get(`http://localhost:3000/companies/${parentValue.companyId}`).then((response) => { // et on fournis la valeur de parentValue.companyId.
39           return response.data;
40         });
41       }
42     }
43   })
44 });
```

## Deuxième partie du fichier schemas.js

```
45
46 //création d'une Root Query qui est notre point d'entrée
47 const RootQuery = new GraphQLObjectType({
48   name : 'RootQuery',
49   fields : { // le champs
50     user : { // le nom du champs de notre Root Query USER EST EGALEMENT UNE FONCTION.
51       type : UserType, // on fournit le type de user qui est notre UserType
52       args : {id : { type : GraphQLString}}, // <- on fournis un argument qui précise quel information peut recevoir notre user.
53       resolve(parentValue,args){//<-et ensuite on lui fournie une promesse qui signifie qu'est ce qu'il doit faire quand il a reçu l'id.
54         return axios.get(`http://localhost:3000/users/${args.id}`).then((response)=>{ // la requête est une promesse, .then = quand tu a fini.
55           return response.data;
56         })
57       },
58     },
59     company : {
60       type : CompanyType,
61       args : {id : { type : GraphQLString }},
62       resolve(parentValue,args){
63         return axios.get(`http://localhost:3000/companies/${args.id}`).then((response)=>{
64           return response.data;
65         })
66       },
67     },
68   }
69 });
70 module.exports = new GraphQLSchema({ //permet d'exporter tout le schema
71   query : RootQuery // a partir de ce moment là Rootquery contient tout les informations de UserType
72 })
```

GraphiQL



Prettify

History

```
1 query{
2   company(id:"1") {
3     name,
4     user {
5       firstName
6     }
7   }
8 }
9
10
```

```
{
  "data": {
    "company": {
      "name": "Apple",
      "user": [
        {
          "firstName": "Mathieu"
        },
        {
          "firstName": "Mickey"
        }
      ]
    }
  }
}
```

Documentation Explorer



Search Schema...

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: RootQuery