✓ Terminé

Le code ci-dessous a été testé sur la VDI (bureau distant). Cela devrait passer crème sur les machines de TP. Mais il ne faut pas travailler dans le dossier `Documents` car les modules à installer créent des liens symboliques. Travailler dans votre home directory. Attention, sur la VDI, tout dossier qui n'est pas dans Documents est effacé lorsque vous quittez la session.

Pré-requis

Si besoin (ce qui n'est pas le cas sur la VDI ni les machines de TP), installer `node` et `npm` :

```
sudo apt-get install nodejs npm
```

On peut également installer un éditeur plus agréable que `nano` : `jed` :

```
sudo apt-get install jed
```

Mise en place du serveur GraphQL

Inspiré de https://www.apollographql.com/docs/apollo-server/getting-started/ mais avec un découpage en trois fichiers :

1. schéma des données
2. données et résolveur
3. instanciation du serveur

Pré-requis : `node > 10`.

On réalise une application `node` en suivant les étapes suivantes :

1. initialiser l'application Node

```
$ mkdir graphql-server
$ cd graphql-server
$ npm init --yes && npm pkg set type="module"
$ npm install @apollo/server graphql --save
```

2. créer le schéma `GraphQL` dans un fichier `schema.graphql` :

```
type Book {
  title: String
  author: String
}

type Query {
  books: [Book]
}
```

3. dans le fichier `resolvers.js`, insérer des données et concevoir un résolveur qui renvoie ces données :

?

```
// Resolvers define how to fetch the types defined in your schema.
// This resolver retrieves books from the "books" array above.
const books = [
  {
    title: 'The Awakening',
    author: 'Kate Chopin',
  },
  {
    title: 'City of Glass',
    author: 'Paul Auster',
  },
];

const resolvers = {
  Query: {
      books: () => books,
  },
};

export default resolvers;
```

4. créer le fichier `index.js` qui va instancier le serveur.

```
import { ApolloServer } from '@apollo/server';
import { startStandaloneServer } from '@apollo/server/standalone';

import { readFileSync } from 'fs';
const typeDefs = readFileSync('./schema.graphql', { encoding: 'utf-8' });

import resolvers from './resolvers.js';

// The ApolloServer constructor requires two parameters: your schema
// definition and your set of resolvers.
const server = new ApolloServer({
  typeDefs,
  resolvers,
  csrfPrevention: false
});

// Passing an ApolloServer instance to the `startStandaloneServer` function:
//  1. creates an Express app
//  2. installs your ApolloServer instance as middleware
//  3. prepares your app to handle incoming requests
const { url } = await startStandaloneServer(server, {
  listen: { port: 4000 },
});

console.log(`🚀  Server ready at: ${url}`);
```

5. lancer l'application

```
node index.js
```

6. sur la machine hôte, en se rendant avec le navigateur à http://localhost:4000 on dispose d'une interface de test qui fournit de nombreuses informations. Dans la colonne de gauche, on dispose d'icônes pour accéder au schéma et à l'exploreur.

7. dans l'exploreur, saisir la requête suivante et l'exécuter (on peut utiliser l'auto-complétion en appuyant sur Ctrl + espace :

```
{books {title author}}
```

8. en cliquant sur les trois petits points en face de la requête, le menu contextuel propose de copier l'opération vers un commande `curl`, que l'on peut lancer dans un terminal\ :

```
$ curl --request POST --header 'content-type: application/json' --url 'http://localhost:4000' \
  --data '{"query":"query books {\n  books {\n    author title\n  }\n}"}'
```

9. on peut également effectuer une requête HTTP GET pour obtenir le résultat\ :

```
$ curl --request GET http://localhost:4000/?query=%7Bbooks%7Btitle%20author%7D%7D
```

(les valeurs %xx sont la traduction des caractères spéciaux `{ ' ' }`. On peut utiliser ce site pour encoder les URL.)

On peut également obtenir un résultat formaté par `jq`\ :

?

```
$ curl --request GET http://localhost:4000/?query=%7Bbooks%7Btitle%20author%7D%7D | jq .
```

10. Noter que la documentation d'Apollo précise que le serveur GraphQL, en cas de requête GET, est prémuni contre les attaques CSRF. La requête ci-dessus ne devrait pas fonctionner en standard, mais ce mécanisme de préventation a été désactivé grâce à l'option `csrfPrevention: false` lors de la création du serveur. Sinon, il faudrait par exemple indiquer dans la requête `curl` que le format attendu est JSON\ :

```
$ curl -H 'Content-Type: application/json' --request GET http://localhost:4000/?query=%7Bbooks%7Btitle%20author%7D%7D
```

11. Attention, si vous avez paramétré un proxy par l'intermédiaire de la variable `http_proxy`, pensez à le désactiver pour éviter l'erreur `Connection refused. The remote host or network may be down. Please try the request again.`\ :

```
$ http_proxy='' curl --request GET http://localhost:4000/?query=%7Bbooks%7Btitle%20author%7D%7D
```

Modifié le: jeudi 21 septembre 2023, 16:03

◄ Cours GraphQL

Choisir un élément

Aller à...

TP 2.2 - Connecteur Node.js pour PostGres ►

?