Nom : Andrianjarasoa Abel Jakinà

N° : 30187

Question Théorique :

1. **GraphQL - Architecture et Principes** (2 points)  
    la différence entre GraphQL et REST se trouve dans la manier où l’on interagit avec les données.

Avec REST, on peut travailler avec plusieurs Endpoint définis à l’avance comme /user, /post et on doit souvent faire plusieurs appels pour obtenir toutes les données utiles.

GraphQL, lui fonctionne avec un seul Endpoint et permet de formuler des requêtes précises pour obtenir exactement les données dont on a besoin, rien de plus.

Les principales raisons pour lesquelles on choisirait GraphQL pour une API ?

Comme on l’a vu GraphQL utilise un seul Endpoint. De plus on demande uniquement les champs qu’on souhaite, ce qui évite les réponses trop lourdes.

1. **GraphQL - Types et Résolvers (2 points)**

Un resolver en GraphQL, c’est une fonction comment aller chercher les données pour un champ données

Quand on fait une requête, ce n’est pas le schéma qui contient les données, c’est le resolver qui va les récupérer, que se soit depuis une base de sonnées, une API externe ou même un tableau en mémoire.

Son rôle, c’est de faire le lien entre la requête du client et les données réelles. Chaque champ qu’on demande peut avoir son propre resolver, ce qui permet un contrôle très précis sur ce qui est retourné

Exemple :

Voici un schéma GraphQL pout gérer des utilisateurs

Type User {

id : ID

name : String

}

Type Query {

getUser(id: ID!): User

}

Dans ce cas on a un resolver qui va chercher un utilisateur dans un tableau comme si c’était une base de données :

Const users = [

{id : ‘1’, name : ‘Alice’},

{id : ‘1’, name : ‘Alice’},

] ;

Const users = {

Query: {

getUser : (\_, { id }) => {

return users.find(user => user.id === id);

},

},

} ;

Et quand on fait on fait une requete comme

{

getUser(id: "2") {

name

}

}

Le resolver getUser est appelé, il cherche dans le tableau l’utilisateur avec l’id "2", et retourne "Bob" comme réponse.

1. **GraphQL - Requêtes et Mutations (2 points)**

En GraphQL, la différence entre une requête (query) et une mutation, c’est le type d’opération qu’on effectue sur les données.

* Une requête (query) est utilisée quand on veut lire des données, sans rien modifier.
* Une mutation (mutation), c’est ce qu’on utilise quand on veut changer quelque chose, comme créer, modifier ou supprimer une ressource.

Autrement dit, on utilise les requêtes pour récupérer des infos, et les mutations pour agir sur les données.

Voici un exemple :

Schéma GraphQL :

Type User {

id: ID!

name: String!

}

Type Mutation {

createUser(name: String!): User

}

Resolver associé :

let users = [ ];

let idCounter = 1;

const resolvers = {

Mutation: {

createUser: (\_, { name }) => {

const newUser = { id: String(idCounter++), name };

users.push(newUser);

return newUser;

},

},

};

Requête GraphQL côté client :

mutation {

createUser(name: "Jakinà") {

id

name

}

}

Cette mutation crée un nouvel utilisateur avec le nom "Jakinà" et renvoie les informations du nouvel utilisateur.

1. WebSockets - Fonctionnement (2 points)

WebSocket, c’est un protocole qui permet d’établir une connexion bidirectionnelle persistante entre le client (comme un navigateur) et le serveur.

Contrairement à HTTP classique, où le client doit envoyer une requête à chaque fois qu’il veut des données, avec WebSocket, la connexion reste ouverte après l’établissement, et les deux côtés peuvent s’envoyer des messages à tout moment.

Et don par rapport à HTTP on a un canal ouvert en continu sur WebSocket.

**Avantages de WebSocket pour les applications en temps réel :**

* **Réactivité** : on peut recevoir instantanément les nouvelles données dès qu’elles sont disponibles (ex : messages dans un chat, scores en direct, notifications…).
* **Moins de surcharge réseau** : pas besoin de faire du "polling" (demander toutes les 2 secondes si quelque chose a changé).
* **Connexion continue** : une fois ouverte, la communication est plus rapide et fluide, car il n’y a pas besoin de rouvrir une connexion à chaque échange.

Trois scénarios où WebSocket est plus approprié que REST :

* 1. Application de messagerie instantanée (chat)
  2. Jeux en ligne multijoueur
  3. Mises à jour en temps réel (ex : tableau de bord de données, cours de bourse)