UE AD FIL A1 - GraphQL

2023-2024

Hélène Coullon

Table of Contents

1. Pourquoi GraphQL ?

2. Principes et fonctionnement de GraphQL

Pourquoi GraphQL ?

Avantages et inconvénients de REST

Avantages

- API orientée ressources
- Clarté de l'API : un point d'entrée, une ressource
- Simplicité et portabilité des requêtes HTTP

Inconvénients

- Toutes les données envoyées pour une requête donnée : over-fetching
- Besoin de combiner plusieurs requêtes pour obtenir les données : under-fetching
- Si on souhaite sous diviser les ressources il faut beaucoup de points d'entrée et l'API devient complexe
- Si on souhaite ajouter une ressource il faut un/des nouveaux points d'entrée

Avantages et inconvénients de GraphQL

Avantages

- Combinaison des principes de REST avec un langage de requête
  - demander et récupérer uniquement les données nécessaires
  - pas de multiplication du nombre de points d'entrée
  - Flexibilité d'ajouter des ressources sans ajouter de point d'entrée

Inconvénients

- Comme chaque requête est différente la mise en cache est plus difficile

Principes et fonctionnement de GraphQL

Principe général

Client

- HTTP/1.1 méthode POST
- Requête HTTP sur un point d'entrée unique
- body = requête GraphQL

Serveur

- Réception de la requête
- Résolution de la requête
- Envoi de la réponse

API GraphQL - le Schéma

Le “Schéma” GraphQL est la définition de l’API (Détails ici)

- équivalent des points d’entrée en REST
- équivalent de fichier protocol buffers en gRPC

Contenu du schéma GraphQL

- *Object types* :
  - types existants racines : *Query*, *Mutation*
  - ajout de nouveaux types
- *Fields* : Le contenu des Object types
  - `flied_name(argument:type):type`
  - les arguments sont optionnels
- *Type scalaires* : e.g., String, Int, Float, lists

Schéma GraphQL - Exemple

```
type Query {
    hero(name:String) : Character
}

type Character {
    name : String
    friends : [Character]
    homeworld : Planet
    species : Species
}

type Planet {
    name : String
    climate : String
}

type Species {
    name : String
    lifespan : Int
    origin : Planet
}
```

- 1 *field* dans l'*Object type Query*
- 3 *object types* : Character, Planet et Species
- 5 *fields* avec un type de retour *Object*
  - Query hero, Character friends, Character homeworld, Character species, Species origin
- 5 *fields* avec un type de retour scalaire
  - Character name, Planet name, Planet climate, Species name, Species lifespan

Le langage de requête

Le langage de requête GraphQL consiste à construire une structure des objets et fields souhaités dans la réponse

- Le format de la requête est proche d'un format JSON
- Les arguments sont donnés aux fields qui en demandent

Le langage de requête - Exemple

```
{
    hero(name : "Luke Skywalker") {
        name
        friends {
            name
            homeworld {
                name
                climate
            }
            species {
                name
                lifespan
                origin {
                    name
                }
            }
        }
    }
}
```

Par exemple on ne demande pas ici à recevoir

- ni l'espèce de Luke Skywalker, mais seulement son nom et sa liste d'amis
- ni le climat de la planète de ses amis, mais seulement le nom de cette planète

La résolution

1. Chaque *field* qui retourne un *Object type* (non scalaire) est associé à un *Resolver*

Un *resolver* est une fonction contenant le code métier

- entrées : paramètres du *field* et informations complémentaires (voir tuto)
- sortie : une donnée correspondant à l'*Object type* de sortie du *field*
    - e.g., un dictionnaire, une liste

2. Si le type d'un *field* est un scalaire la résolution remonte dans l'arbre au field suivant

- Parcours en profondeur de l'arbre : DFS - Depth First Search

Des détails ici : `https://graphql.org/learn/execution/`

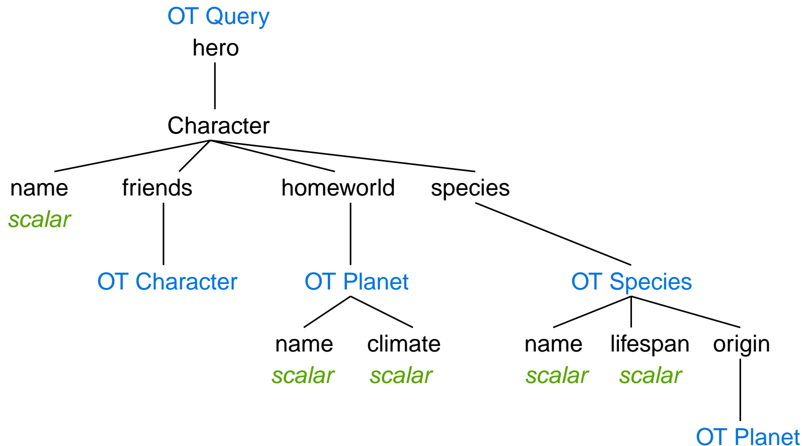
La résolution - Exemple

```
type Query {
    hero(name:String) : Character
}

type Character {
    name : String
    friends : [Character]
    homeworld : Planet
    species : Species
}

type Planet {
    name : String
    climate : String
}

type Species {
    name : String
    lifespan : Int
    origin : Planet
}
```

- 5 *fields* nécessitant un *resolver* car leur type de retour est un *Object*
  - Query hero
  - Character friends
  - Character homeworld
  - Character species
  - Species origin

La résolution - Exemple

Voici le graphe de résolution dont la racine est le field Query `hero`



La résolution - Exemple de réponse

```
{
    "data": {
        "hero": {
            "name" : "Luke Skywalker",
            "friends" : [
                {
                    "name" : "R2-D2",
                    "homeworld" : {
                        "name" : "X",
                        "climate" : "good"
                    },
                    "species" : {
                        "name" : "robot",
                        "lifespan" : "eternal",
                        "origin" : {
                            "name" : "Y"
                        }
                    }
                },
                {
                    "name" : "C3PO",
                    "homeworld" : {
                        ...
                    },
                    "species" : {
                        ...
                    }
                }
            ]
        }
    }
}
```

La résolution - Pas de boucle

Dans notre exemple la liste des amis peut elle même contenir une liste d’amis etc.

Est-ce qu’on créée une boucle infinie ?

La résolution - Pas de boucle

Non car la résolution suit la requête de l'utilisateur qui est elle nécessairement finie !

```
{
    hero(name : "Luke Skywalker") {
        name
        friends {
            name
            friends {
                name
                friends {
                    name
                }
            }
            homeworld {
                name
                climate
            }
            species {
                name
                lifespan
                origin {
```

Comparaison REST et GraphQL

	Performances	Debug	Découverte	Flexibilité	Nb fonctions
REST/OpenAPI					
GraphQL					

Tutoriel

Tutoriel GraphQL