

UE AD FIL A1

UE Services FISE A3 LOGIN

API GraphQL

2025-2026

Hélène Coullon



IMT Atlantique

Bretagne-Pays de la Loire

École Mines-Télécom

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

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

Le “Schéma” GraphQL est la définition de l'API ([Détails ici](#))

- équivalent des points d'entrée en REST
- sert aussi en quelque sorte de documentation
- (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
 - `field_name(argument:type):type`
 - les arguments sont optionnels
- *Type scalaires* : e.g., String, Int, Float, lists

Object types

```
type Query{  
    # list of fields  
}
```

```
type Mutation{  
    # list of fields  
}
```

```
type Character{  
    # list of fields  
}
```

```
type Planet{  
  
}
```

Fields

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

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

```
type Planet{  
    name : String  
    climate : String  
}
```

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  
      }  
      friends {  
        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*. Il correspond (à peu près) au code d'un point d'entrée en REST.

- 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 *Object type* la résolution continue en descendant dans l'arbre, sinon (type scalaire) la résolution remonte dans l'arbre

- 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  
}
```

```
type Planet{  
    name : String  
    climate : String  
}
```

3 **fields** nécessitant un **resolver** car leur type de retour est un **Object**

- Query hero (type de retour Character)
- Character friends (type de retour [Character])
- Character homeworld (type de retour Planet)

La résolution - Exemple

La résolution est toujours **basée sur un requête** ! Une requête étant finie il n'y a **pas de boucles** !

```
{  
  hero(name : 'Luke Skywalker') {  
    name  
    friends {  
      name  
      homeworld {  
        name  
      }  
      friends {  
        name  
      }  
    }  
  }  
}
```

1. appel du resolver du field
hero de Query
2. appel du resolver du field
friends de Character
3. appel du resolver du field
homeworld de Character
4. appel du resolver du field
friends de Character

Comparaison REST et GraphQL

	Performances	Debug	Flexibilité	Nb fonctions
REST/OpenAPI				
GraphQL				

Tutoriel GraphQL