graphQl APIs

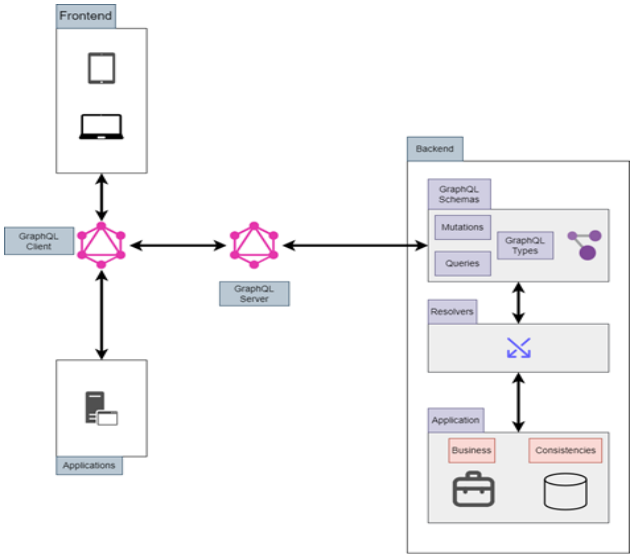
Vue d'ensemble

Graph QL est un langage open source de manipulation et de requêtage de données. il a pour but de permettre au client de choisir spécifiquement la structure et la liste des attributs des données qu'il souhaite récupérer (données dont le stockage peut éventuellement être distribué). Le serveur suit ensuite cette structure afin de construire et retourner la réponse. Cette méthode fortement typée a pour but d'éviter les problèmes de retour de données insuffisantes ou surnuméraires que les Api REST peuvent connaître.

Ce langage d'API se base sur les concepts clés suivants :

- Le client définit lui-même ce dont il a besoin et le serveur répond dans la même structure (*Ask for what you need, get exactly that*)
- Le typage fort de ces APIs permet d'éviter les problématiques de retour de données insuffisant (*under-fetching*) ou à l'inverse superflu (*over-fetching*)
- Requêter sur de nombreuses ressources grâce à une seule requête.

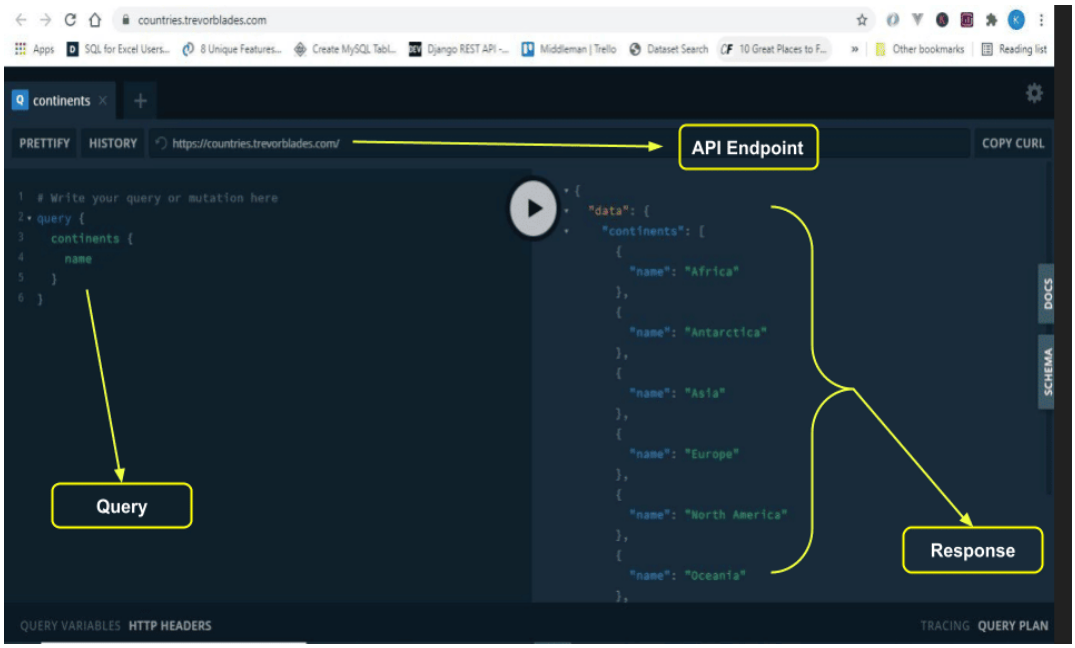
La typologie d'API GraphQL a été créée pour répondre aux problématiques des API REST. Elle s'attaque principalement au problème de flexibilité de ces dernières. En effet, son modèle en graphe d'entités lui permet de répondre à différents besoins sans multiplier les requêtes.



La caractéristique principale de GraphQL est sa capacité à demander et à recevoir uniquement les données spécifiques demandées – rien de plus. Il est donc beaucoup plus simple de faire évoluer vos API en même temps que votre application.

La partie la plus excitante de GraphQL est sa capacité à vous fournir toutes les données en un seul point de terminaison.

Les clients effectuent des requêtes à partir de différents appareils, et GraphQL traite leurs requêtes et renvoie uniquement les données demandées. Cela résout parfaitement le problème de l'over-fetching et du under-fetching dans les API RESTful.



En haut se trouve le point de terminaison de l'API,

à gauche se trouve la requête qui demande les noms des continents, et enfin,

à droite, nous répondons à la requête

Les endpoints / URI

Sur une API GraphQL il n'y a qu'un seul endpoint : « myApi/graphql ». C'est à partir de ce endpoint (côté serveur) que l'on va pouvoir requêter les entités du Graph.

Les requêtes en GraphQL

En GraphQL il existe deux types de requêtes :

- Les mutations: des requêtes dont le but est un changement d'état de la donnée. En d'autres termes, la mutation a pour but de modifier la donnée.

- Les queries: des requêtes qui ont pour objet de récupérer de la donnée. Aucune modification n'est réalisée via cette dernière.

Les requêtes HTTP GET et POST avec GraphQl

En GraphQL, il est possible de faire des requêtes GET et POST :

- Si la requête comporte le paramètre query alors il faut la traiter comme une requête GET

  myApi/graphql?query=myquery

- Si le Content-Type de la requête est « application/GraphQL » alors il s'agit d'une requête en POST

Les réponses de l'API GraphQL sont toujours formatées en JSON sous le format suivant

- Une entrée data contenant l'ensemble des données requêtées
- Une entrée errors contenant l'ensemble des erreurs retournées lors de la requête

Pour créer un service GraphQL, on commence par définir des types de schéma et créer des champs à l'aide de ces types.

Ensuite, on fournis un résolveur de fonctions à exécuter sur chaque champ et chaque type chaque fois que des données sont demandées par le côté client.

Le système de types GraphQL est utilisé pour décrire les données qui peuvent être interrogées et celles que vous pouvez manipuler. Il s'agit du cœur de GraphQL. Discutons des différentes façons dont nous pouvons décrire et manipuler les données dans GraphQl

Types d'objets

Les types d'objets GraphQL sont des modèles de données contenant des champs fortement typés. Il devrait y avoir une correspondance 1 à 1 entre vos modèles et les types GraphQL. un exemple de type GraphQl

```
type User {
  id: ID! # The "!" means required
  firstname: String
```

```
  lastname: String
  email: String
  username: String
  todos: [Todo] # Todo is another GraphQL type
}
```

Requêtes

GraphQL Query définit toutes les requêtes qu'un client peut exécuter sur l'API GraphQL. Par convention, vous devez définir un RootQuery qui contiendra toutes les requêtes existantes.

Ci-dessous, nous définissons et mappons les requêtes à l'API RESTful correspondante :

```
type RootQuery {
  user(id: ID): User          # Corresponds to GET /api/users/:id
  users: [User]               # Corresponds to GET /api/users
  todo(id: ID!): Todo    # Corresponds to GET /api/todos/:id
  todos: [Todo]         # Corresponds to GET /api/todos
}
```

Mutations

Si les requêtes GraphQL sont des demandes GET, les mutations sont des requêtes POST, PUT, PATCH, et DELETE qui manipulent l'API GraphQL.

Nous allons mettre toutes les mutations dans un seul RootMutation pour faire la démonstration :

```
type RootMutation {
  createUser(input: UserInput!): User            # Corresponds to POST /api/users
  updateUser(id: ID!, input: UserInput!): User    # Corresponds to PATCH /api/users
  removeUser(id: ID!): User                    # Corresponds to DELETE /api/users
  createTodo(input: TodoInput!): Todo
  updateTodo(id: ID!, input: TodoInput!): Todo
```

```
  removeTodo(id: ID!): Todo
}
```

Résolveurs

Les résolveurs indiquent à GraphQL ce qu'il doit faire lorsque chaque requête ou mutation est demandée. Il s'agit d'une fonction de base qui fait le travail difficile de frapper la couche de base de données pour effectuer les opérations CRUD (création, lecture, mise à jour, suppression), de frapper un point de terminaison interne RESTful API, ou d'appeler un microservice pour répondre à la demande du client.

on crée un nouveau fichier resolvers.js

Schéma

Le schéma GraphQL est ce que GraphQL expose au monde. Par conséquent, les types, les requêtes et les mutations seront inclus dans le schéma pour être exposés

```
schema {
  query: RootQuery
  mutation: RootMutation
}
```

Exemple GraphQL :

Requête GraphQL pour récupérer les informations d'un utilisateur :

```
graphql
Copy code
query {
  user(id: 123) {
    id
    name
    email
    posts {
      title
      body
```

```
    }
  }
}
```