BACK-END ESSENTIALS

GRAPHQL

By Mohamed OUHAMI, Khalil ZAATARI

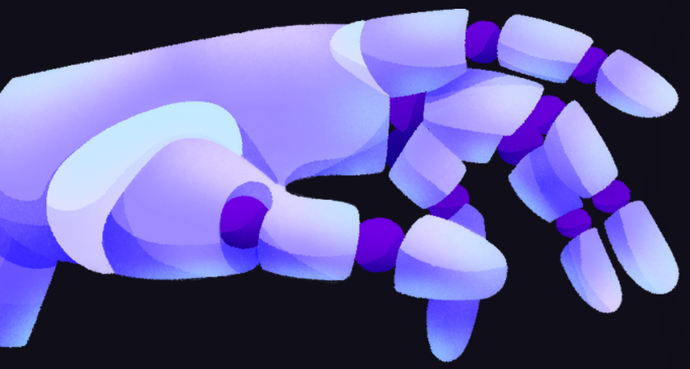


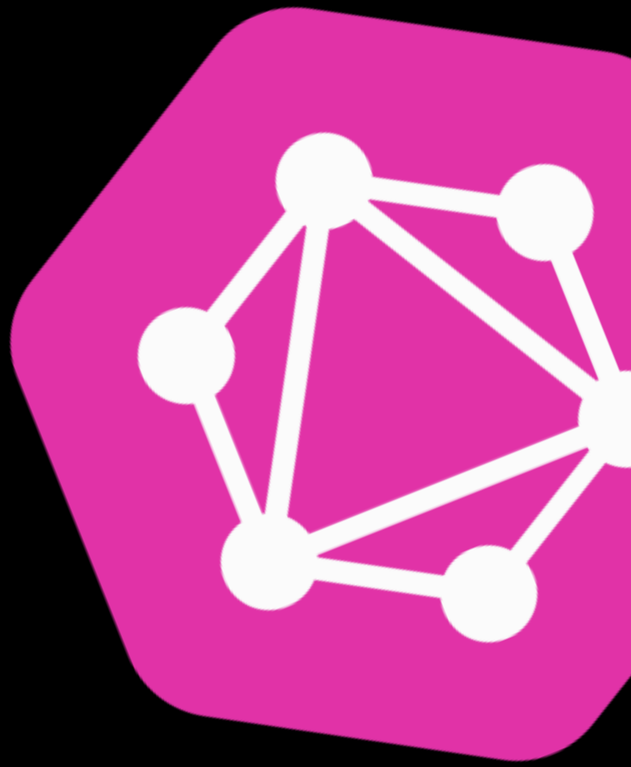
TABLE DE CONTENU

- C’est quoi GraphQL ?
- REST API vs GraphQL API
- Queries / Mutations
- GraphQL for Spring
- Exemple pratique
- Conclusion

C'EST QUOI GRAPHQL ?

C'EST QUOI GRAPHQL ?

GraphQL est un langage de requête pour la lecture et la mutation des données via une API. En tant que développeur backend, GraphQL offre l'opportunité d'établir un système de typage en décrivant un schéma de données. En retour, cela permet aux utilisateurs frontend de consommer l'API et de récupérer précisément les données dont ils ont besoin.




CLIENT
API

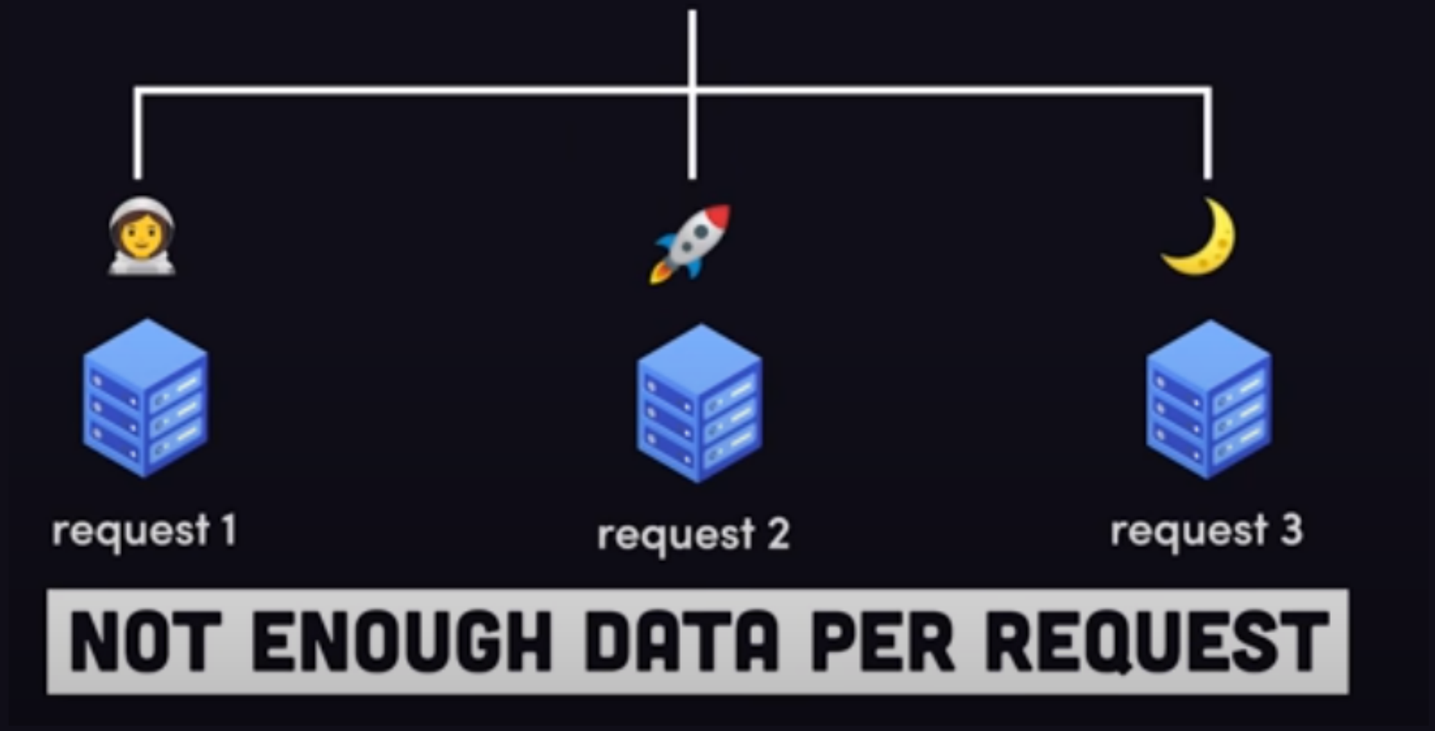
```
16
17    # Define the Student type
18    type Student {
19        id: ID!
20        name: String!
21        age: Int!
22        filiere: Filiere!
23    }
24
25    # Define the Filiere type
26    type Filiere {
27        id: ID!
28        code: String!
29        libelle: String!
30        students: [Student]
31    }
32
```

AVEC L'UTILISATION DE TYPES, ON PEUT DÉFINIR LA MANIÈRE DONT ON PEUT À LA FOIS RECEVOIR LES DONNÉES OU LES MANIPULER (MUTATIONS).

REST VS GRAPHQL

DANS LES APIS CLASSIQUES, ON TRAVAILLE AVEC PLUSIEURS POINTS D'ACCÈS (ENDPOINTS), CE QUI PEUT ÊTRE UN DÉFI EN TERMES DE SCALABILITÉ DES PROJETS.

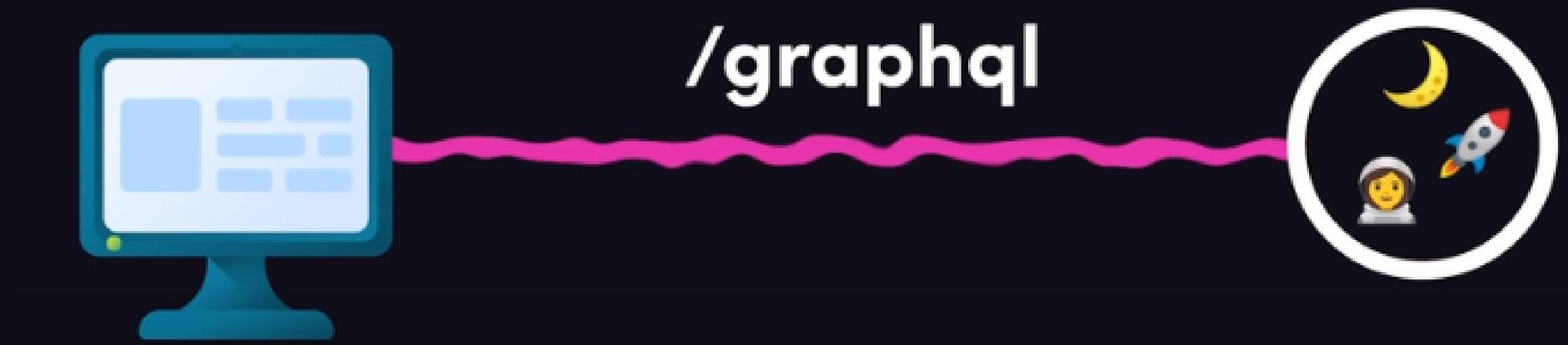
UNDERFECTHING



"UNDER-FETCHING : NE PAS RÉCUPÉRER TOUTES LES DONNÉES EN UNE SEULE FOIS."

OVERFETCHING

OVER-FETCHING : RÉCUPÉRER PLUS DE DONNÉES QUE NÉCESSAIRE, CE QUI PEUT ENTRAÎNER UNE SURCONSOMMATION DE RESSOURCES.



ICI, NOUS TRAVAILLONS AVEC UN SEUL POINT D'ACCÈS, QUI EST "/GRAPHQL". C'EST L'UNIQUE POINT D'ENTRÉE PAR LEQUEL TOUS LES VERBES HTTP TRANSITENT.

```
query Query {
  Students {
    age
    id
    name
    filiere {
      code
    }
  }
}

```

```
  "data": {
    "Students": [
      {
        "age": 28,
        "id": "4",
        "name": "Bassma Ouhami",
        "filiere": {
          "code": "Genie Informatique"
        }
      },
```

CLIENT

REQUEST

query.gql query.gql

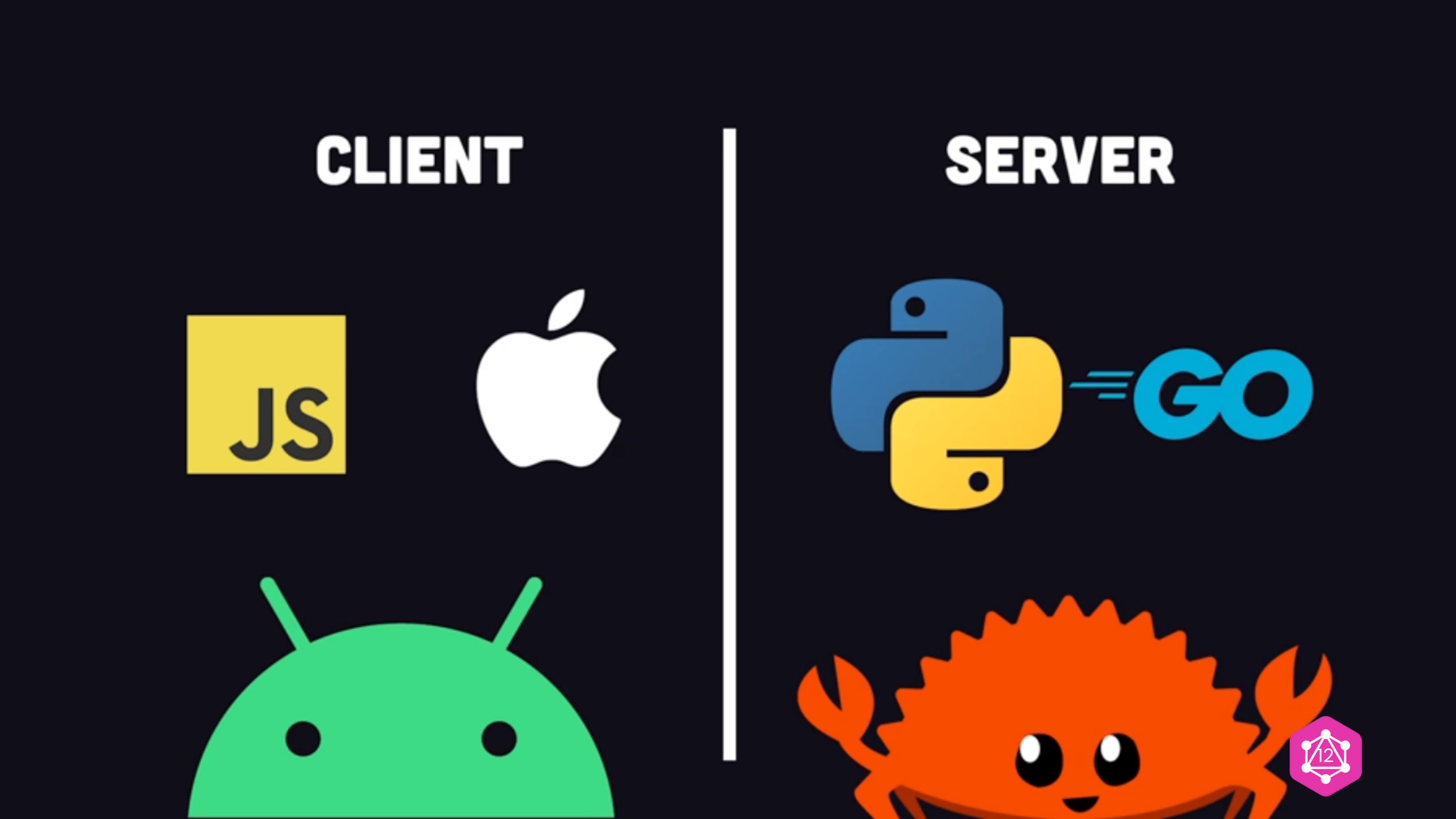
```
query getRockets {
    Execute Query
    rocket {
        name
        thrust
        captain {
            name
            callsign
        }
    }
}
```

SERVER

RESOLVE

{ } response.json response.json\...

```
{
    "name": "Saturn V",
    "thrust": 7891000,
    "captain": {
        "name": "Neil Armstrong",
        "callsign": "Neil"
    }
}
```


CLIENT
SERVER
JS
GO

```
26    type Filiere {
27        id: ID!
28        code: String!
29        libelle: String!
30        students: [Student]
31    }
```

FIELD TYPE
FIELD TYPE
FIELD TYPE

!
required

```
17    # Define the Student type
18    type Student {
19        id: ID!
20        name: String!
21        age: Int!
22        filiere: Filiere!
23    }
24
```

QUERIES / MUTATIONS

```
33  # Define the Query type for fetching data
34  type Query {
35      Students: [Student]
36      filieres: [Filiere]
37      filiereById(id: ID!): Filiere
38      studentById(id: ID!): Student
39      findByFiliere(filiereId: FiliereIdentifier!): [Student]
40  }
41
```

```
42  # Define the Mutation type for modifying data
43  type Mutation {
44      # Filiere Mutations
45      addFiliere(filiereInput: FiliereInput!): Filiere
46      deleteFiliere(id: ID!) : Boolean
47      updateFiliere(id:ID!, filiereInput: FiliereInput!): Boolean
48
49      # Student Mutations
50      addStudent(studentInput: StudentInput!): Student
51      deleteStudent(id: ID!) : Boolean
52      updateStudent(id:ID!, studentInput: StudentInput!): Boolean
53  }
```

JS resolver.js resolver.js\...

```
export const resolvers = {
  Query: {
    creator(obj, args, context, info) {
      return context
                .db
                .getCreator(args.id);
    },
  },
};
```

```java
// Deleting Student
@MutationMapping
public Boolean deleteStudent(@Argument int id) {
    try {
        studentService.deleteById(id);
        return true;
    } catch (Exception err) {
        System.out.println("The id to delete is " + id);
        return false;
    }
}
```

```
26      @QueryMapping
27      List<Student> Students() {
28          return studentService.findAll();
29      }
30
```

EXEMPLE PRATIQUE

CONCLUSION

Malgré l'utilisation des APIs REST, les problèmes d'over-fetching et d'under-fetching persistent. GraphQL est devenu très populaire pour offrir aux développeurs la flexibilité nécessaire pour interagir avec leurs données de manière efficace et sur mesur

THANK YOU!