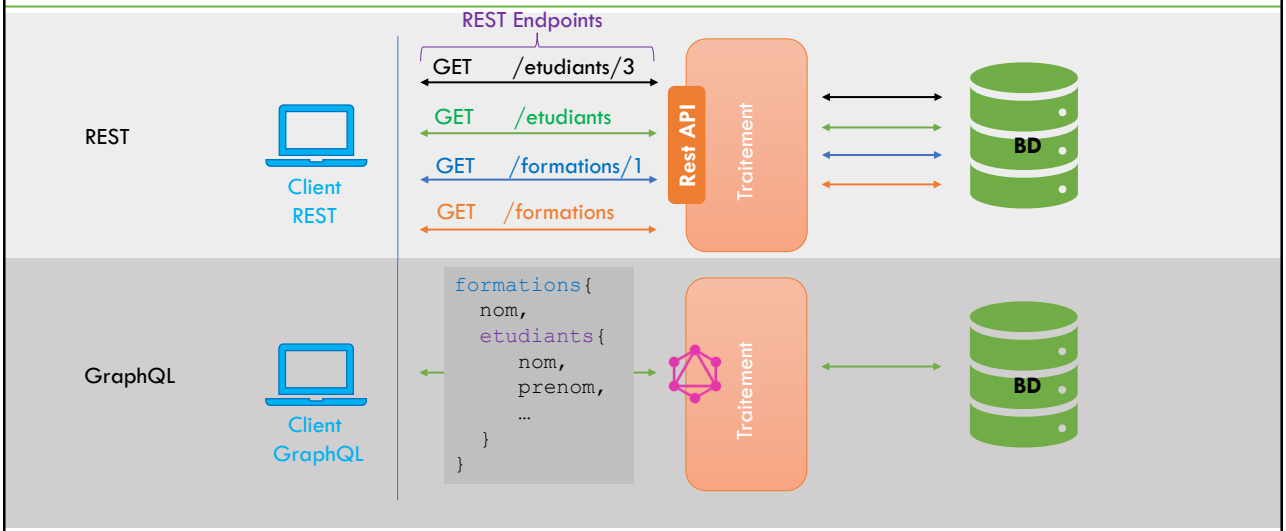
III- Web Services GraphQL



GRAPHQL

- GraphQL est **un langage de requêtes** de données pour API.
- QL signifie Query Language.
- Il permet de manipuler de la donnée de façon simple, flexible et très précise.
- Première version créée en 2012 par Facebook, ensuite il passe en open-source en 2015.

GRAPHQL

Avantages

- **Réduction du nombre de requêtes** : Dans les API REST, il est fréquent d'effectuer plusieurs requêtes pour obtenir différentes données liées. GraphQL permet de regrouper ces demandes en une seule requête, simplifiant les appels et améliorant la performance, notamment pour les applications mobiles où la latence réseau est un facteur critique.
- **Flexibilité pour l'évolution des API** : GraphQL facilite les mises à jour des API sans affecter les clients existants. Les nouveaux champs peuvent être ajoutés sans casser les requêtes des clients actuels, car ces derniers demandent uniquement les champs dont ils ont besoin.
- **Meilleure gestion des erreurs** : Dans une réponse GraphQL, chaque partie de la requête retourne ses propres erreurs, permettant de mieux identifier les problèmes spécifiques à chaque donnée demandée

GraphQL est particulièrement utile dans les applications modernes, notamment les applications mobiles et les front-ends interactifs, où la flexibilité des données et la réduction des appels au serveur sont cruciales.

GRAPHQL

Requêtes

En GraphQL il existe deux types de requêtes :

i. Les mutations: des requêtes dont le but est un changement d'état de la donnée

→ La mutation a pour but de modifier la donnée.

ii. Les queries: des requêtes qui ont pour objet de récupérer de la donnée.

→ Aucune modification n'est réalisée via cette dernière.

GRAPHQL

Shema Definition Langage (SDL)

- Utilisé pour définir un schéma GraphQL;
- Utilisé pour exposer les fonctionnalités disponibles aux utilisateurs d'une application donnée.
- Il contient :
  - Types: similaires aux classes java
  - Operations: similaires aux méthodes java

GRAPHQL

Exemple

```
type Query {
    getBook(id : Int ): Book
    getBooks: [Book]
}

type Mutation {
    createBook(name : String, pages : Int ): Int
    deleteBook(id:Int): String
}

type Book {
    id: Int
    name : String
    pages : Int
}
```


GRAPHQL

Les requêtes HTTP GET et 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

```
{
"data": { ... },
"errors": [ ... ]
}
```

GRAPHQL

Structure des requêtes & réponses GraphQL

Requête

```
query{
getEtudiant(id:3){
     nom
     genre
     }
}
```

Réponse

```
"data": {
    "getEtudiant": {
        "nom": "Fadli",
        "genre": "Homme"
    }
}
```


GRAPHQL

Opérations

Lire

```
query {
     search(q: "name" ){
          title
          author
        }
}
```

Ecrire

```
mutation {
     create(title: "book" ){
          id
        }
}
```

Ecouter (Listen)

```
subscription {
     onCreate {
          id
          title
        }
}
```

Subscription

- Les *subscriptions*(abonnements) dans GraphQL permettent d'établir une connexion en temps réel entre un client et un serveur.
- Les *subscriptions* GraphQL sont principalement conçus pour écouter quand des données sont créées sur le serveur, quand une donnée est mise à jour , quand une donnée est supprimée et quand une donnée est lue via une requête.
- L'événement à émettre dépend de ce que le développeur souhaite. Les événements sont transmis du serveur aux clients abonnés, sans que le client ait à demander ces données à chaque fois.
- Utile pour des applications nécessitant des mises à jour en temps réel, comme les notifications, les flux de chat, ou le suivi de données en direct.

Subscription – Fonctionnement

Contrairement aux *queries* et *mutations* (les requêtes classiques de GraphQL), les *subscriptions* fonctionnent généralement sur le protocole WebSocket pour maintenir une connexion ouverte et permettre la transmission d'événements continus.

Subscription – Fonctionnement

- Les clients s'abonnent à un événement côté serveur en utilisant la requête d'abonnemen.
- La requête d'abonnement ci-dessus envoie une requête via WebSocket au serveur GraphQL, la requête définit un événement sur le serveur avec une fonction de rappel de résolution.
- Chaque fois que l'événement est émis sur le serveur GraphQL, la fonction de résolution est appelée, la valeur de retour de l'appel est envoyée à la requête d'abonnement.

```
subscription{
  etudiantAdded{
    id
    nom
    prenom
    genre
   }
}
```


Subscription – Avantages

- Les abonnements GraphQL sont très utiles pour créer des applications en temps réel dans GraphQL.
- La plupart des entreprises utilisant GraphQL dans leur API utilisent l'abonnement pour diffuser du contenu en temps réel.

Subscription – Projet Reactor

1- Flux

Représente une **séquence réactive de 0 à N éléments.**

Utilisé pour modéliser des **flux asynchrones** de données

i. Création d'un Flux

❖ Flux offre plusieurs méthodes statiques pour créer des instances :

Flux.just() : Émet une séquence de valeurs spécifiques

Flux.fromIterable() : Crée un Flux à partir d'une collection

Flux.range() : Génère une séquence d'entiers dans une plage donnée.

Subscription – Projet Reactor

ii. Opérations courantes sur Flux

- **map()** : Transforme chaque élément émis: `flux.map(value -> value.toLowerCase());`
- **filter()** : Filtre les éléments selon une condition : `flux.filter(value -> value.startsWith("A"));`
- ...

GRAPHQL

Subscription – Projet Reactor

2- sink

Une abstraction puissante permettant de produire des données de manière programmatique dans un flux réactif (**Publisher**)

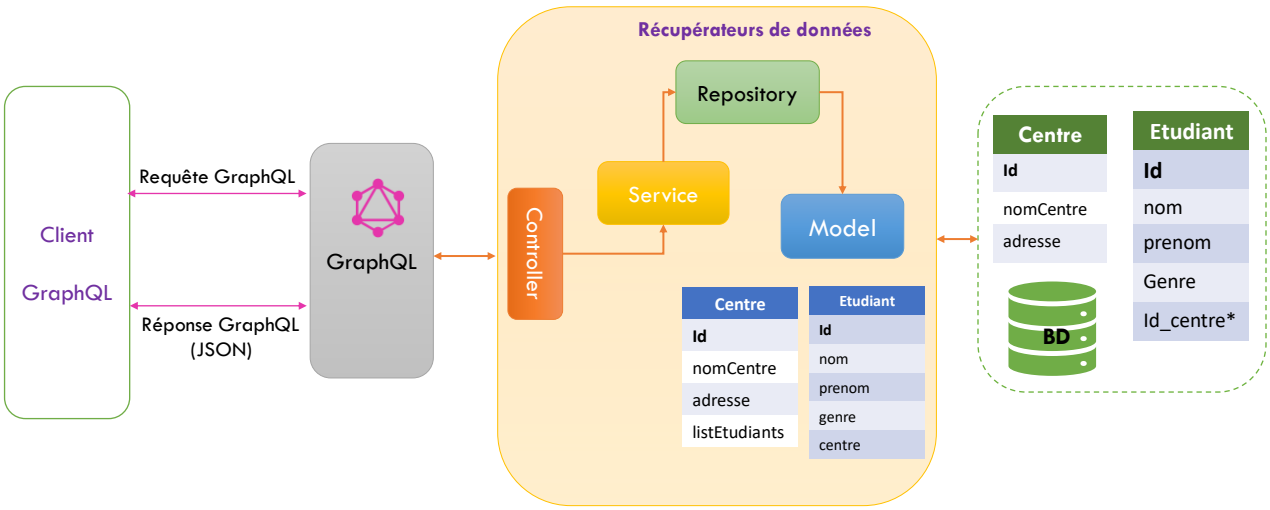
Types de Sinks dans Reactor

i. **Sink.one()** : Utilisé pour émettre une seule valeur

ii. Sink**.many()** : Utilisé pour émettre plusieurs valeurs ou gérer des flux dynamiques.
   - **Sink.many().unicast()** :Flux émis vers **un seul abonné**.
   - **Sink.many().multicast()** :Flux émis vers **plusieurs abonnés** en diffusion simultanée.
   - **Sink.many().replay()** : Stocke les éléments émis pour les rejouer aux nouveaux abonnés.

iii. **Sink.empty()** : Émet simplement une complétion ou une erreur.

APPLICATION - SPRING DATA REST- GRAPHQL



APPLICATION - SPRING DATA REST- GRAPHQL

ÉTAPES À SUIVRE

i. Création d'un projet Spring Boot y compris les dépendances
ii. Ajout des Entités JPA
iii. Création des Repository JPA
iv. Ajout d'une classe Controller
v. Création des objets de transfert de données (DTO)
vi. Définition du schéma GraphQL
vii. Définition de la source de données
viii. Teste

- **Spring Web**
- **H2 Database**
- **Lombok**
- **Spring Data JPA**
- **Rest Repositories**
- **Spring for GraphQL**
- **WebSocket**

APPLICATION - SPRING DATA REST- GRAPHQL

Entités JPA

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Centre {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    Long id;
    String nom;
    String adresse;
    @OneToMany(mappedBy = "centre", cascade = CascadeType.ALL)
    List<Etudiant> listEtudiants;
}
```

```
@Entity @Data
@AllArgsConstructor @NoArgsConstructor
@Builder @Table(name="etudiants")
public class Etudiant {
    @Id
    @GeneratedValue(strategy = GenerationType.AU
    Long id;
    @Column(name="nom_etudiant", nullable=false)
    String nom;
    @Column(name="prenom_etudiant")
    String prenom;
    @Enumerated(EnumType.STRING)
    Genre genre;
    @ManyToOne
    @NotNull
    @JoinColumn(name="centre_id")
    Centre centre;
}
```

APPLICATION - SPRING DATA REST- GRAPHQL

Création des objets de transfert de données (DTO)

```
public record EtudiantDTO (
        String nom,
        String prenom,
        Genre genre,
        Long centreId
){   }
```

NB : Ajouter la classe CentreDTO

APPLICATION - SPRING DATA REST- GRAPHQL

Ajouter la classe de mapping : EtudiantDTO->Etudiant

```
@Component
public class DtoToEtudiant {
@Autowired
CentreRepository centreRepository;
public  void toEtudiant(Etudiant et, EtudiantDTO dto) {
    Centre centre= centreRepository.findById(dto.centreId()).orElse(null);
    if (dto != null) {
        BeanUtils.copyProperties(etudiantDTO,etudiant);
        et.setCentre(centre);
    }
}

}
```

APPLICATION - SPRING DATA REST- GRAPHQL

3- Repository JPA

```
public interface CentreRepository extends JpaRepository<Centre,Long> {
}

public interface EtudiantRepository extends JpaRepository<Etudiant,Long>
{
}
```


APPLICATION - SPRING DATA REST- GRAPHQL

Services

```
@Service
public class EtudiantService {
    @Autowired
    DtoToEtudiant dtoToEtudiant;
    @Autowired
    EtudiantRepository etudiantRepository;

    public List<Etudiant> getStudents() {
        return etudiantRepository.findAll();
    }
public Etudiant getEtudiant(Long id){
        return etudiantRepository.findById(id).orElse(null);
   }
public Etudiant addEtudiant(EtudiantDTO etudiantDTO) {
        Etudiant etudiant=new Etudiant();
        dtoToEtudiant.toEtudiant(etudiant, etudiantDTO);
        etudiantRepository.save(etudiant);
        return etudiant;
    }
```

SERVICE (SUITE)

```
    public Etudiant updateEtudiant(Long id, EtudiantDTO etudiantDTO){
         if(etudiantRepository.findById(id).isPresent()){
             Etudiant etudiant=etudiantRepository.findById(id).get();
             dtoToEtudiant.toEtudiant(etudiant,etudiantDTO);
             return etudiantRepository.save(etudiant);
         }
             return null;
     }

     public String deleteEtudiant(Long id){
         if(etudiantRepository.findById(id).isPresent()){
           Etudiant et=etudiantRepository.findById(id).get();
           etudiantRepository.deleteById(id);

         return String.format("l'étudiant %s est bien supprimé !",id);
         }
         return String.format("l'étudiant %s n'existe pas !",id);
     }
}
```


CONTROLLER

```
@Controller
public class EtudiantCentreController {
     @Autowired
    EtudiantService etudiantService;
     @Autowired
     CentreService centreService;
    @QueryMapping
    public List<Centre> getAllCentres(){
        return centreService.centres();
    }
    @QueryMapping
    public List<Etudiant>getAllEtudiants(){

        return etudiantService.getStudents();
    }
    @QueryMapping
    public Centre getCentre(@Argument int id){
        return centreService.getCentre(id);
    }
```

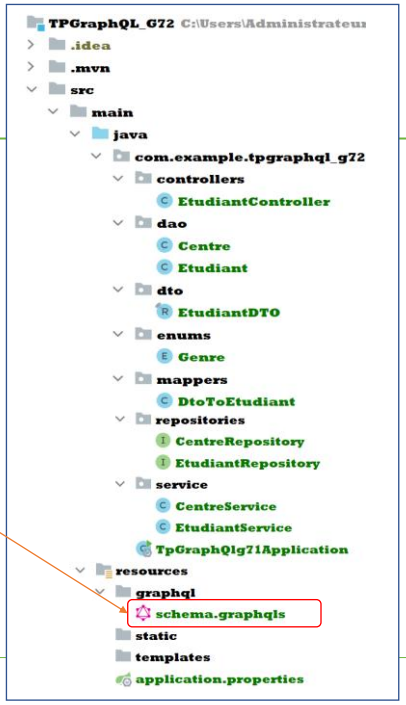
CONTROLLER(SUITE)

```
    @QueryMapping
    public Etudiant getEtudiant(@Argument Long id){
        return etudiantService.getEtudiant(id);
    }
    @MutationMapping
    public Etudiant addEtudiant(@Argument EtudiantDTO etudiantDTO) {
        return etudiantService.addEtudiant(etudiantDTO);
    }
    @MutationMapping
    public String suppEtudiant(@Argument Long id){
        return etudiantService.deleteEtudiant(id);
    }
    @MutationMapping
    public Etudiant updateEtudiant(@Argument Long id,@Argument EtudiantDTO etudiantDTO){
        return etudiantService.updateEtudiant(id,etudiantDTO);
    }

}
```


APPLICATION - SPRING DATA REST- GRAPHQL

Ajout d'un fichier graphqls : schema.graphqls



APPLICATION - SPRING DATA REST- GRAPHQL

5- Schéma GraphQL

```
type Query{
    listEtudiants : [Etudiant]
    getEtudiantById(id:Float):Etudiant
    centres:[Centre]
    getCentreById(id:Float):Centre
}
type Mutation{
    addEtudiant(etudiantDTO : EtudiantDTO):Etudiant
    updateEtudiant(id:Float,etudiantDTO : EtudiantDTO):Etudiant
    deleteEtudiant(id:Float):String
}

enum Genre {
    Homme,
    Femme
}
```

```
type Etudiant{
    id:Float
    nom:String
    prenom:String
    genre:Genre
    centre:Centre
}
type Centre{
    id:Int
    nom: String
    adresse:String
    listEtudiants:[Etudiant]
}
input EtudiantDTO{
    nom:String
    prenom:String
    genre:String
    centreId:Float
}
```


APPLICATION - SPRING DATA REST- GRAPHQL

Configuration & dependencies

Application.properties

```
spring.h2.console.enabled=true
spring.datasource.username=12
spring.datasource.password=
spring.datasource.url=jdbc:h2:mem:centredb
spring.graphql.graphiql.enabled=true
spring.graphql.websocket.path=/graphql
```

APPLICATION - SPRING DATA REST- GRAPHQL

Ajout d'un jeu d'enregistrements

```
public class TpGraphQlApplication implements CommandLineRunner{
    @Autowired
    EtudiantRepository etudiantRepository;
    @Autowired
    CentreRepository centreRepository;
    public static void main(String[] args) {
        SpringApplication.run(TpGraphQlApplication.class, args);
    }
    @Override
    public void run(String... args) throws Exception {
        Centre centre1=Centre.builder()
                .nom("Maarif").adresse("Biranzarane").build();
        centreRepository.save(centre1);
        Centre centre2=Centre.builder()
                .nom("Oranges").adresse("Oulfa").build();
        centreRepository.save(centre2);
        Etudiant et1=Etudiant.builder()
                .nom("Adnani").prenom("Brahim").genre(Genre.Homme)
                .centre(centre1).build();
        etudiantRepository.save(et1);
        …
```


TEST-QUERY

http://localhost:8080/graphiql?path=/graphql

```
query
{
  getEtudiantById(id: 1) {
    nom
    prenom
    centre {
      nom
      adresse
    }
  }
}
```

```
{
  "data": {
    "getEtudiantById": {
      "nom": "Adnani",
      "prenom": "Brahim",
      "centre": {
        "nom": "Maarif",
        "adresse": "Biranzarane"
      }
    }
  }
}
```

```
query{
  listEtudiants{
    id
    nom
    prenom
    centre{
      nom
      adresse
    }
  }
}
```

```
{
  "data": {
    "listEtudiants": [
      {
        "id": 1,
        "nom": "Adnani",
        "prenom": "Brahim",
        "centre": {
          "nom": "Maarif",
          "adresse": "Biranzarane"
        }
      },
      {
        "id": 2,
        "nom": "Alami",
        "prenom": "Ilham",
        "centre": {
          "nom": "Maarif",
          "adresse": "Biranzarane"
        }
      },
      {
        "id": 3,
        "nom": "Fadli",
        "prenom": "Moad",
        "centre": {
          "nom": "Oranges",
          "adresse": "Oulfa"
        }
      },
      {
        "id": 4,
        "nom": "Hamdani",
        "prenom": "Adil",
        "centre": {
          "nom": "Oranges",
          "adresse": "Oulfa"
        }
      }
    ]
  }
}
```

```
query{
  getCentreById(id:1){
    nom
    listEtudiants{
      nom
      prenom
    }
  }
}
```

```
{
  "data": {
    "getCentreById": {
      "nom": "Maarif",
      "listEtudiants": [
        {
          "nom": "Adnani",
          "prenom": "Brahim"
        },
        {
          "nom": "Alami",
          "prenom": "Ilham"
        }
      ]
    }
  }
}
```

```
fragment champsEtudiant on Etudiant{
  nom
  prenom
  genre
}
query{
  listEtudiants{
    id
    ...champsEtudiant
    centre{
      id
      nom
    }
  }
}
```

INSOMNIA POR GRAPHQL