

- TP3 et TP4 - Directus REST API et GraphQL

- Setup Directus
  - 1. Lancer Directus
  - 2. Ce qui est automatique
  - 3. Configuration manuelle requise

- TP3 : Directus REST API

- Format de l'API REST
- Requêtes REST
  - 1. Liste des praticiens
  - 2. La spécialité d'ID 2
  - 3. La spécialité d'ID 2, uniquement son libellé
  - 4. Un praticien avec sa spécialité (libellé)
  - 5. Une structure (nom, ville) et la liste des praticiens rattachés (nom, prénom)
  - 6. Idem en ajoutant le libellé de la spécialité des praticiens
  - 7. Structures dont la ville contient "sur" avec praticiens (nom, prénom, spécialité)

- TP4 : GraphQL

- Requêtes Query
  - 1. Liste des praticiens (id, nom, prénom, téléphone, ville)
  - 2. Idem avec le libellé de la spécialité
  - 3. Avec filtre ville = "Paris"
  - 4. Avec nom et ville de la structure
  - 5. Avec filtre emails contenant ".fr"
  - 6. Praticiens rattachés à une structure dont la ville est "Paris"
  - 7. Deux listes avec alias (Paris et Bourdon-les-Bains)
  - 8. Avec fragment
  - 9. Avec variable
  - 10. Structures avec praticiens
- Autorisations dans Directus
  - Configuration à faire :
  - Requêtes à tester avec authentification
    - Auth 1 : Lister les moyens de paiement
    - Auth 2 : Spécialités avec motifs de visite
- Mutations GraphQL
  - 1. Créer la spécialité "cardiologie"
  - 2. Créer un praticien (sans spécialité au départ)

- 3. Modifier le praticien pour le "rattacher" à la spécialité "cardiologie"
- 4. Créer un praticien directement rattaché à "cardiologie"
- 5. Créer un praticien et créer sa spécialité "chirurgie" en même temps (GraphQL)
- 6. Ajouter un autre praticien à la spécialité "chirurgie"
- 7.Modifier le premier praticien créé pour le rattacher à une structure existante
- 8. Supprimer les deux derniers praticiens créés (GraphQL)

## TP3 et TP4 - Directus REST API et GraphQL

---



---

### Setup Directus

#### 1. Lancer Directus

```
cd directus
docker-compose up -d
```

Accès : <http://localhost:8055>

- Email : [admin@example.com](mailto:admin@example.com)
- Mot de passe : [d1r3ctu5](#)

#### 2. Ce qui est automatique

Les tables sont créées automatiquement au premier lancement grâce aux scripts SQL dans [init/](#) :

- [01-schema.sql](#) : Crée les tables (specialites, structures, praticiens, motifs\_visite, moyens\_paiement, tables de liaison)
- [02-data.sql](#) : Insère les données de test

Directus détecte automatiquement ces tables et génère l'API.

### 3. Configuration manuelle requise

Configurer les relations O2M (One-to-Many) dans Directus :

Les clés étrangères SQL créent les relations M2O (Many-to-One) automatiquement, mais les relations inverses (O2M) doivent être configurées manuellement :

1. Aller dans **Settings > Data Model > structures**

2. Cliquer sur **Create Field > Alias - O2M**

3. Configurer :

- Field Key : **praticiens**
- Related Collection : **praticiens**
- Foreign Key : **structure\_id**

Répéter pour :

- **specialites** → **praticiens** (Foreign Key: **specialite\_id**)
- **specialites** → **motifs\_visite** (Foreign Key: **specialite\_id**)---

## TP3 : Directus REST API

### Format de l'API REST

```
GET http://localhost:8055/items/{collection}
GET http://localhost:8055/items/{collection}/{id}
GET http://localhost:8055/items/{collection}?fields=champ1,champ2
GET http://localhost:8055/items/{collection}?filter[champ][_operateur]=valeur
```

Opérateurs de filtre : **\_eq, \_contains, \_gt, \_gte, \_lt, \_lte**

### Requêtes REST

#### 1. Liste des praticiens

```
GET http://localhost:8055/items/praticiens
```

## 2. La spécialité d'ID 2

```
GET http://localhost:8055/items/specialites/2
```

## 3. La spécialité d'ID 2, uniquement son libellé

```
GET http://localhost:8055/items/specialites/2?fields=libelle
```

## 4. Un praticien avec sa spécialité (libellé)

```
GET http://localhost:8055/items/praticiens?fields=*,specialite_id.libelle&limit=1
```

## 5. Une structure (nom, ville) et la liste des praticiens rattachés (nom, prénom)

```
GET http://localhost:8055/items/structures?  
fields=nom,ville,praticiens.nom,praticiens.prenom&limit=1
```

## 6. Idem en ajoutant le libellé de la spécialité des praticiens

```
GET http://localhost:8055/items/structures?  
fields=nom,ville,praticiens.nom,praticiens.prenom,praticiens.specialite_id.libelle&  
limit=1
```

## 7. Structures dont la ville contient "sur" avec praticiens (nom, prénom, spécialité)

```
GET http://localhost:8055/items/structures?filter[ville]  
[_contains]=sur&fields=nom,ville,praticiens.nom,praticiens.prenom,praticiens.specialite_id.libelle
```

## TP4 : GraphQL

Endpoint unique : POST <http://localhost:8055/graphql>

## Requêtes Query

### 1. Liste des praticiens (id, nom, prénom, téléphone, ville)

```
query {  
  praticiens {  
    id  
    nom  
    prenom  
    telephone  
    ville  
  }  
}
```

### 2. Idem avec le libellé de la spécialité

```
query {  
  praticiens {  
    id  
    nom  
    prenom  
    telephone  
  }  
}
```

```
ville
specialite_id {
    libelle
}
}
```

### 3. Avec filtre ville = "Paris"

```
query {
  praticiens(filter: { ville: { _eq: "Paris" } }) {
    id
    nom
    prenom
    telephone
    ville
    specialite_id {
      libelle
    }
  }
}
```

### 4. Avec nom et ville de la structure

```
query {
  praticiens(filter: { ville: { _eq: "Paris" } }) {
    id
    nom
    prenom
    telephone
    ville
    specialite_id {
      libelle
    }
    structure_id {
      nom
      ville
    }
  }
}
```

### 5. Avec filtre emails contenant ".fr"

```
query {
  praticiens(filter: { email: { _contains: ".fr" } }) {
    id
    nom
    prenom
    telephone
    ville
    email
    specialite_id {
      libelle
    }
    structure_id {
      nom
      ville
    }
  }
}
```

## 6. Praticiens rattachés à une structure dont la ville est "Paris"

```
query {
  praticiens(filter: { structure_id: { ville: { _eq: "Paris" } } }) {
    id
    nom
    prenom
    telephone
    ville
    specialite_id {
      libelle
    }
    structure_id {
      nom
      ville
    }
  }
}
```

## 7. Deux listes avec alias (Paris et Bourdon-les-Bains)

```
query {
  praticiensParis: praticiens(filter: { ville: { _eq: "Paris" } }) {
    id
```

```
nom
prenom
telephone
ville
specialite_id {
    libelle
}
}
praticiensBourdon: praticiens(filter: { ville: { _eq: "Bourdon-les-Bains" } }) {
    id
    nom
    prenom
    telephone
    ville
    specialite_id {
        libelle
    }
}
}
```

## 8. Avec fragment

```
fragment PraticienFields on praticiens {
    id
    nom
    prenom
    telephone
    ville
    specialite_id {
        libelle
    }
}

query {
    praticiensParis: praticiens(filter: { ville: { _eq: "Paris" } }) {
        ...PraticienFields
    }
    praticiensBourdon: praticiens(filter: { ville: { _eq: "Bourdon-les-Bains" } }) {
        ...PraticienFields
    }
}
```

## 9. Avec variable

```
query GetPraticiensByVille($ville: String!) {
    praticiens(filter: { ville: { _eq: $ville } }) {
        id
    }
}
```

```
    nom
    prenom
    telephone
    ville
    specialite_id {
        libelle
    }
}
```

## Variables (JSON) :

```
{
    "ville": "Paris"
}
```

## 10. Structures avec praticiens

```
query {
  structures {
    nom
    ville
    praticiens {
      nom
      prenom
      email
      specialite_id {
        libelle
      }
    }
  }
}
```

## Autorisations dans Directus

### Configuration à faire :

#### 1. Créer un rôle "API Reader" :

- Settings > Roles & Permissions > Create Role

- Donner les droits de lecture (Read) sur toutes les collections

## 2. Créer deux utilisateurs :

- **Utilisateur avec token statique** : dans son profil, générer un "Static Token"
- **Utilisateur avec JWT** : se connecte via [/auth/login](#) pour obtenir un token temporaire

## 3. Retirer les droits Public :

- Settings > Roles & Permissions > Public
- Retirer Read sur [motifs\\_visite](#) et [moyens\\_paiement](#)

# Requêtes à tester avec authentification

## Auth 1 : Lister les moyens de paiement

```
query {
  moyens_paiement {
    id
    libelle
  }
}
```

## Auth 2 : Spécialités avec motifs de visite

```
query {
  specialites {
    id
    libelle
    motifs_visite {
      id
      libelle
    }
  }
}
```

## Header d'authentification :

```
Authorization: Bearer VOTRE_TOKEN
```

# Mutations GraphQL

Note : Pour les relations (`specialite_id`, `structure_id`), il faut passer un objet `{ id: ... }` (comme vu en cours) et pas juste l'ID.

## 1. Créer la spécialité "cardiologie"

```
mutation {
  create_specialites_item(data: {
    libelle: "cardiologie"
    description: "Maladies du cœur et du système cardiovasculaire"
  }) {
    id
    libelle
  }
}
```

## 2. Créer un praticien (sans spécialité au départ)

```
mutation {
  create_praticiens_item(data: {
    nom: "Martin"
    prenom: "Pierre"
    ville: "Lyon"
    email: "pierre.martin@example.com"
    telephone: "04 72 00 00 00"
  }) {
    id
    nom
    prenom
  }
}
```

Je note l'id du praticien (`ID_PRATICIEN_MARTIN`) et celui de cardiologie (`ID_CARDIO`) pour la suite.

## 3. Modifier le praticien pour le "rattacher" à la spécialité "cardiologie"

```

mutation {
  update_praticiens_item(
    id: "ID_PRATICIEN_MARTIN"
    data: {
      specialite_id: { id: ID_CARDIO }
    }
  ) {
    id
    nom
    specialite_id {
      id
      libelle
    }
  }
}

```

## 4. Créer un praticien directement rattaché à "cardiologie"

Le rattachement direct dans le `create` ne passe pas sur cette version. Je le fais en deux étapes : création puis update.

```

mutation {
  create_praticiens_item(data: {
    nom: "Bernard"
    prenom: "Sophie"
    ville: "Marseille"
    email: "sophie.bernard@example.com"
    telephone: "04 91 00 00 00"
  }) {
    id
    nom
  }
}

```

Puis rattachement avec l'id récupéré (`ID_SOPHIE`) :

```

mutation {
  update_praticiens_item(
    id: "ID_SOPHIE"
    data: { specialite_id: { id: ID_CARDIO } }
  ) {
    id
    nom
    specialite_id { id libelle }
  }
}

```

```
}
```

## 5. Créer un praticien et créer sa spécialité "chirurgie" en même temps (GraphQL)

```
mutation {
  create_praticiens_item(
    data: {
      nom: "Durand",
      prenom: "Michel",
      ville: "Bordeaux",
      email: "michel.durand@example.com",
      telephone: "05 56 00 00 00",
      specialite_id: {
        create: {
          libelle: "chirurgie"
          description: "Interventions chirurgicales"
        }
      }
    }
  ) {
    id
    nom
    specialite_id {
      id
      libelle
    }
  }
}
```

Je note **ID\_CHIRURGIE** pour la suite.

## 6. Ajouter un autre praticien à la spécialité "chirurgie"

Même chose que pour la 4 : création d'abord, puis rattachement.

```
mutation {
  create_praticiens_item(data: {
    nom: "Leroy"
    prenom: "Anne"
    ville: "Toulouse"
    email: "anne.leroy@example.com"
  })
```

```
    telephone: "05 61 00 00 00"
}) {
  id
  nom
}
}
```

Avec l'id **ID\_ANNE** :

```
mutation {
  update_praticiens_item(
    id: "ID_ANNE"
    data: { specialite_id: { id: ID_CHIRURGIE } }
  ) {
    id
    nom
    specialite_id { id libelle }
  }
}
```

## 7. Modifier le premier praticien créé pour le rattacher à une structure existante

```
mutation {
  update_praticiens_item(
    id: "ID_PRATICIEN_MARTIN"
    data: {
      structure_id: { id: "3444bdd2-8783-3aed-9a5e-4d298d2a2d7c" }
    }
  ) {
    id
    nom
    structure_id {
      nom
      ville
    }
  }
}
```

## 8. Supprimer les deux derniers praticiens créés (GraphQL)

```
mutation {
  delete_praticiens_item(id: "ID_AVANT_DERNIER") {
    id
  }
}
```

```
mutation {
  delete_praticiens_item(id: "ID_DERNIER") {
    id
  }
}
```

(**ID\_AVANT\_DERNIER** et **ID\_DERNIER** correspondent aux ids des mutations 4, 5 ou 6).