

# TD3 - CMS headless Directus - Compte rendu

Auteur : Carette Robin

Depot : <https://github.com/CaretteRobin/nouveaux-paradigmes-TD3-4.git>

## Introduction

Ce compte rendu presente la mise en place d'un CMS headless avec Directus pour un domaine "praticiens de sante". L'objectif est de construire un backoffice permettant de gerer les donnees, puis de les exposer via l'API REST de Directus. Les captures ecran documentent chaque etape cle et servent de preuves de realisation.

## Objectifs du TD

- Installer Directus en Docker.
- Construire le modele de donnees (collections + relations).
- Importer un jeu de donnees (CSV) en respectant l'ordre des dependances.
- Verifier les donnees via le backoffice.
- Consommer l'API REST pour obtenir des donnees simples et imbriques.

## Environnement et demarrage

Le service est deploye avec Docker Compose. Le conteneur Directus est expose sur le port 8055.

```
c:\Users\robin\OneDrive\Bureau\exercices\exercice-2\td-database>docker compose up -d
unable to get image 'directus/directus:10.10.5': Cannot connect to the docker daemon at unix:///how/robin/.docker/desktop/docker.sock. Is the docker daemon running?
c:\Users\robin\OneDrive\Bureau\exercices\exercice-2\td-database>docker compose up -d
Running on host
✓ Container cms-1 started
Container cms-1 Started
c:\Users\robin\OneDrive\Bureau\exercices\exercice-2\td-database>ps
CONTAINER ID  IMAGE          COMMAND           CREATED          STATUS          PORTS          NAMES
0519f2289064  directus/directus:10.10.5   "docker-entrypoint.s..."  13 hours ago   Restarting (1)  1 second ago   0.0.0.0:8055->8055/tcp   automation_exercice-2-td-database_1
07ef9289064  directus/directus:10.10.5   "docker-entrypoint.s..."  7 days ago     Up 5 seconds   5432/tcp          cms-directus-1
5b749baec53  postgres:16                "docker-entrypoint.s..."  7 days ago     Up 5 seconds   cms-db-1
c:\Users\robin\OneDrive\Bureau\exercices\exercice-2\td-database>
```

Ce terminal montre le lancement via `docker compose up -d` puis la verification avec `docker ps`. On constate que Directus est actif (port 8055) ainsi que la base de donnees. Une tentative initiale a signale un daemon Docker non disponible, puis le demarrage a reussi.

## Modele de donnees

Le modele suit le schema du domaine "praticiens de sante". Les collections principales sont : - praticien - specialite - structure - motif\_visite - moyen\_paiement - tables de jonction `praticien2motif` et `praticien2moyen`



Cette vue liste toutes les collections définies dans Directus, confirmant l'existence du modèle complet et la préparation des relations.

Les champs principaux du praticien sont visibles, ainsi que les relations vers **specialite** et **structure**. Cela valide la modélisation M2O attendue.

## Chargement des données (CSV)

Les données sont importées depuis des fichiers CSV avec séparateur ; afin de reprendre un jeu existant.

1;Martin;Claire;Nancy;claire.martin@example.org;0610101010;RPPS1001:false,true;Dr;1;1	
2;Durand;Luc;Villers-sur-Mer;luc.durand@example.org;0620202020;RPPS1002:false,false;Dr;2;2	
3;Petit;Sarah;Lunéville;sarah.petit@example.org;0630303030;RPPS1003:true,true;Dr;2;3	
4;Bernard;Alex;Nancy;alex.bernard@example.org;0640404040;RPPS1004:false,true;Dr;3;1	

L'extrait montre les colonnes attendues (nom, prenom, ville, email, tele-

phone, etc.) et les clés d'association (specialité, structure). Cela garantit la compatibilité avec les relations du modèle.

La liste des praticiens confirme que l'import a été effectué correctement et que les enregistrements sont consultables dans l'interface.

## Verification dans le backoffice

Après import, les collections sont consultables et navigables directement via l'interface.

Cet écran illustre l'accès aux collections et aux items, confirmant la disponibilité des données dans le backoffice.

## API REST (exposition des données)

Base URL (exemple local) : <http://localhost:8055>

Si un token est utilisé :

**Authorization: Bearer <TOKEN>**

Requêtes REST réalisées (conformes aux exigences du TD) :

- 1) Liste des praticiens

```
curl -H "Authorization: Bearer <TOKEN>" \
"http://localhost:8055/items/praticien?fields=id,nom,prenom,ville"
```

- 2) Spécialité d'ID 2

```
curl -H "Authorization: Bearer <TOKEN>" \
"http://localhost:8055/items/specialite/2"
```

- 3) Spécialité d'ID 2 avec uniquement le libellé

```
curl -H "Authorization: Bearer <TOKEN>" \
"http://localhost:8055/items/specialite/2?fields=libelle"
```

4) Un praticien avec sa specialite (libelle)

```
curl -H "Authorization: Bearer <TOKEN>" \
  "http://localhost:8055/items/praticien/<ID>?fields=nom,prenom,specialite.libelle"
```

5) Une structure (nom, ville) avec la liste de ses praticiens

```
curl -H "Authorization: Bearer <TOKEN>" \
  "http://localhost:8055/items/structure/<ID>?fields=nom,ville,praticiens.nom,praticiens.prac-
```

6) Idem avec le libelle de la specialite des praticiens

```
curl -H "Authorization: Bearer <TOKEN>" \
  "http://localhost:8055/items/structure/<ID>?fields=nom,ville,praticiens.nom,praticiens.prac-
```

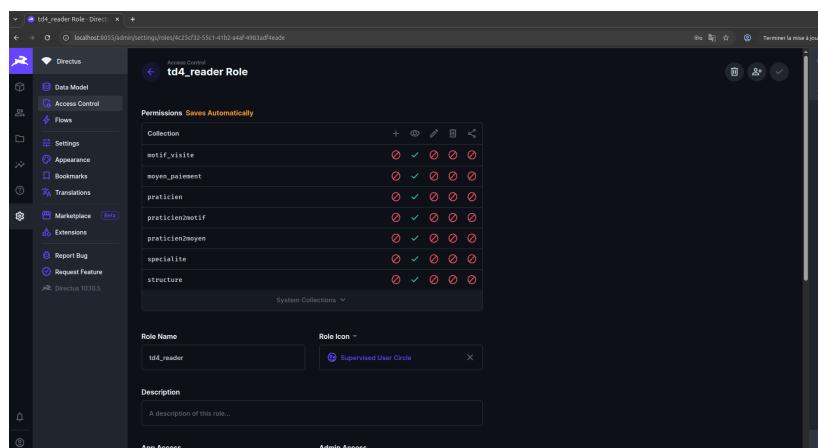
7) Structures dont la ville contient “sur”, avec la liste des praticiens

```
curl -H "Authorization: Bearer <TOKEN>" \
  "http://localhost:8055/items/structure?filter[ville][_contains]=sur&fields=nom,ville,prati-
```

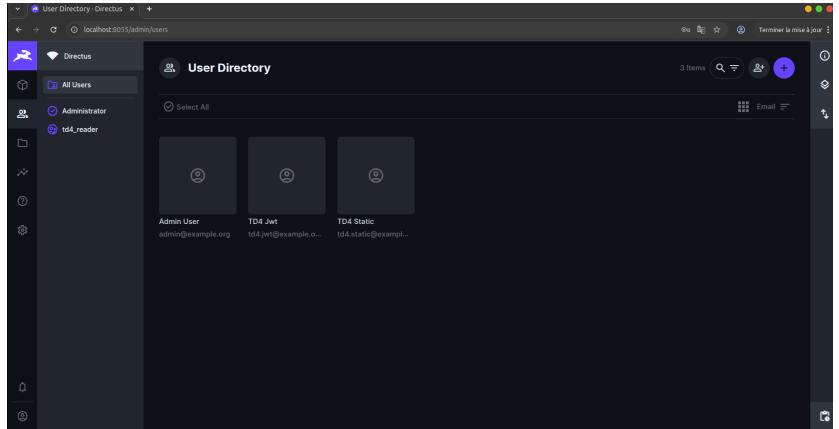
Ces requetes valident l'exposition REST attendue, la selection de champs (fields) pour limiter les donnees retournees et la navigation dans les relations.

## Annexes - Securite et GraphQL (preparation TD4)

Les captures suivantes illustrent des actions realisees en parallele pour preparer l'API GraphQL et les droits d'accès (hors perimetre strict du TD3, mais utiles pour la suite).



Creation d'un role dedie avec les droits de lecture sur les collections necessaires.  
Cela sert a controler l'accès à l'API.



Deux comptes sont associés au rôle : un utilisateur avec token statique et un utilisateur pour JWT, afin de tester l'authentification.

```
1: {
2:   "errors": [
3:     {
4:       "message": "GraphQL validation error.",
5:       "extensions": {
6:           "code": "GRAPHQL_VALIDATION_FAILED"
7:       },
8:       "errors": [
9:           {
10:               "message": "Cannot query field \"structure\" on type \"Query\".",
11:               "locations": [
12:                   {
13:                       "line": 2,
14:                       "column": 3
15:                   }
16:               ]
17:           }
18:       ]
19:   }
20: }
```

Test d'accès sans authentification : la requête échoue, ce qui confirme que les permissions publiques sont bien restreintes.

The screenshot shows a browser window titled "dev web server" with the URL "POST API" and "GraphQL" selected. The address bar shows "POST http://localhost:8055/graphiql". The main content area displays a GraphQL query and its resulting JSON response.

**Query:**

```
query {  
  structure { nom ville praticiens { nom prenom email specialite { libelle } } }  
}
```

**Response:**

```
[  
  {  
    "ville": "Paris",  
    "praticiens": [  
      {  
        "nom": "Bernard",  
        "prenom": "Alex",  
        "email": "alex.bernard@example.org",  
        "specialite": [  
          {  
            "libelle": "Pediatrics"  
          }  
        ]  
      },  
      {  
        "nom": "Centre Medical du Centre",  
        "ville": "Villefranche-sur-Mer",  
        "praticiens": [  
          {  
            "nom": "Durand",  
            "prenom": "Luc",  
            "email": "luc.durand@example.org",  
            "specialite": [  
              {  
                "libelle": "Oncologie"  
              }  
            ]  
          },  
          {  
            "nom": "Maison de Santé",  
            "ville": "Lyonville",  
            "praticiens": [  
              {  
                "nom": "Petit",  
                "prenom": "Sarah",  
                "email": "sarah.petit@example.org",  
                "specialite": [  
                  {  
                    "libelle": "Dermatologie"  
                  }  
                ]  
              },  
              {  
                "nom": "Généraliste",  
                "prenom": "Marie",  
                "email": "marie.generale@example.org",  
                "specialite": [  
                  {  
                    "libelle": "Gastroenterologie"  
                  }  
                ]  
              }  
            ]  
          }  
        ]  
      }  
    ]  
  }  
]
```

The JSON response consists of a single array containing three objects. Each object represents a city and its associated practitioners. The practitioners are represented as objects with fields: nom, prenom, email, and specialite (which is an array of speciality objects). The speciality objects have a libelle field representing the specialty name.

cation d'une requête GraphQL renvoyant des structures et leurs praticiens avec spécialité. Cela valide la navigation dans les relations côté API.

The screenshot shows the Postman application interface. At the top, there's a header bar with 'My Workspace' and a profile picture of 'Bruno'. Below it, a navigation bar includes 'Collections', 'Variables', 'Auth', 'Vars', 'Script', 'Assert', 'Tests', and tabs for 'Docs' and 'Schema'. The main area has a title 'dev web server' and a 'POST API()' button. A dropdown menu for 'GQL' is open. The URL 'POST http://localhost:8085/graphql' is entered. On the left, a 'Query' tab is selected, showing the following GraphQL query:

```
query {  
  create_specialite_item(data: { libelle: "cardiologie" }) {  
    id  
    libelle  
  }  
}
```

To the right, a 'Response' tab is open, showing the JSON response:

```
1 | {  
2 |   "data": {  
3 |     "createSpecialiteItem": {  
4 |       "id": "123",  
5 |       "libelle": "cardiologie"  
6 |     }  
7 |   }  
8 | }
```

At the bottom right, there are buttons for 'JSON', '200 OK', '59ms', and '7B'.

Exemple de mutation (creation d'une specialite) en tant qu'admin, pour verifier les operations d'ecriture.

## Conclusion

Le CMS headless Directus a été déployé avec succès, le modèle de données a été construit et alimenté via CSV, puis vérifié dans le backoffice. L'API REST permet de consommer les données avec sélection de champs et navigation relationnelle. Les captures annexes montrent en plus la mise en place des rôles et l'utilisation de GraphQL pour préparer la suite du projet. L'ensemble constitue un rendu clair, exploitable et conforme aux objectifs du TD.