

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.

```
root@robin-ordinateur:~/dev/nouveaux_paradigmes/Ch$ docker compose up -d
Error: failed to get image "directus:19.10.5": Cannot connect to the Docker daemon at unix:///home/robin/.docker/desktop/docker.sock. Is the docker daemon running?
root@robin-ordinateur:~/dev/nouveaux_paradigmes/Ch$ docker compose up -d
[+] Container cns-db-1 started
[+] Container cns-directus-1 started
root@robin-ordinateur:~/dev/nouveaux_paradigmes/Ch$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
8ffed2898d4        directus/directus:19.10.5   "docker-entrypoint.s..."   23 hours ago       Restarting (1) 1 second ago   0.0.0.0:8055->8055/tcp   automatisation-exercice-2-td-database-1
8b74e9b0ee5c3       postgres/postgres:16          "docker-entrypoint.s..."   7 days ago        Up 5 seconds          5432/tcp            cns-db-1
```

Figure 1: Demarrage des conteneurs Docker

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 : - `practicien` - `specialite` - `structure` - `motif_visite` - `moyen_paiement` - tables de jonction `practicien2motif` et `practicien2moyen`

Cette vue liste toutes les collections definies dans Directus, confirmant l'existence du modele complet et la preparation des relations.

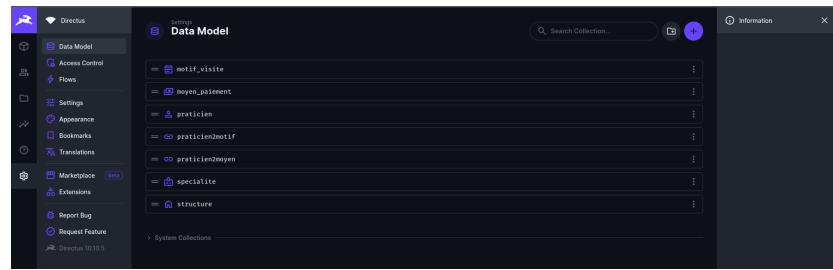


Figure 2: Vue globale du data model

A screenshot of the Directus Praticien collection settings interface. The sidebar on the left is identical to Figure 2. The main area is titled "Praticien" and shows the "Fields & Layout" section with the message "Saves Automatically". It lists fields: id, nom, prenom, ville, email, telephone, rpps_id, organisation, nouveau_patient, titre, specialite, and structure. Each field has a small icon and edit buttons. At the bottom is a blue "Create Field" button.

Figure 3: Champs de la collection praticien

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.

```
id;nom;prenom;ville;email;telephone;rpos_id;organisation;nouveau_patient;titre;specialite;structure
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;Luneville;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
```

Figure 4: Extrait du fichier praticien.csv

L'extrait montre les colonnes attendues (nom, prenom, ville, email, telephone, etc.) et les clés d'association (specialite, structure). Cela garantit la compatibilité avec les relations du modèle.

ID	Nom	Prenom	Ville
1	Martin	Claire	Nancy
2	Durand	Luc	Villers-sur-Mer
3	Petit	Sarah	Luneville
4	Bernard	Alex	Nancy
5	Dupont	Alice	Paris

 A sidebar on the right contains 'Information', 'Layout Options', 'Auto Refresh', and 'Import / Export' buttons."/>

Figure 5: Données importées dans Directus

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

Vérification dans le backoffice

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

ID	Libelle	Specialite
1	Contrôle cardia...	1
2	Examen de la peau...	2
3	Consultation pied...	3

 A sidebar on the right contains 'Information', 'Layout Options', 'Auto Refresh', and 'Import / Export' buttons."/>

Figure 6: Vue backoffice Directus

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 spécialité (libellé)

```
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.pre"
```

- 6) Idem avec le libellé de la spécialité des praticiens

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

- 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 requêtes valident l'exposition REST attendue, la sélection de champs (`fields`) pour limiter les données renvoyées 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).

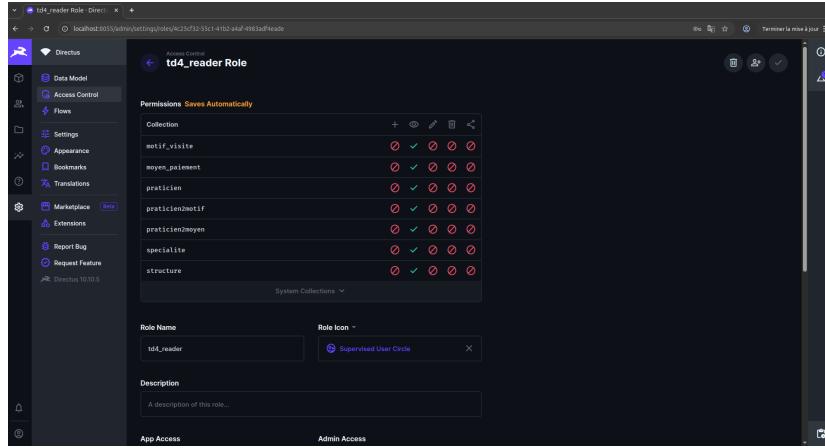


Figure 7: Role lecteur et permissions

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

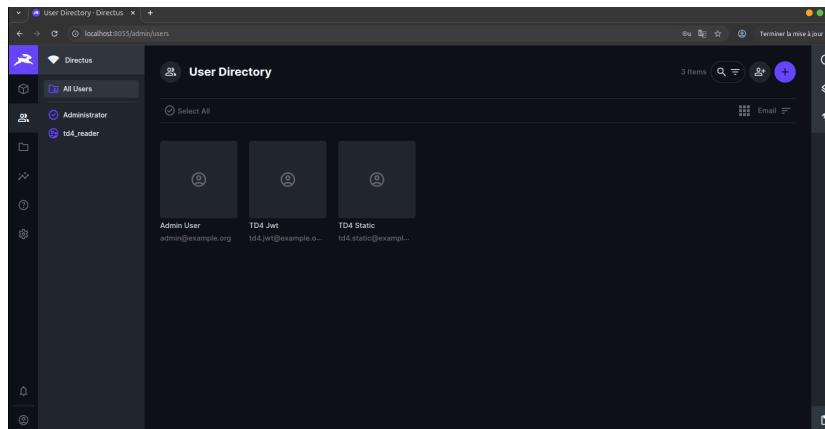


Figure 8: Utilisateurs lies au role

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

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

The screenshot shows a GraphQL playground interface with the following details:

- URL:** POST http://localhost:8055/graphiql
- Headers:** Authorization (empty)
- Query:**

```

1 query {
2   structure { nom ville praticiens { nom prenom email specialite { libelle } } }
3 }

```

- Response:** (Error) 400 Bad Request 21ms 2088
- JSON Response:**

```

1 {
2   "errors": [
3     {
4       "message": "GraphQL validation error.",
5       "extensions": {
6         "code": "GRAPHQL_VALIDATION_FAILED",
7         "errorType": "Validation"
8       }
9     }
10    {
11      "message": "Cannot query field \"structure\" on type \"Query\".",
12      "locations": [
13        {
14          "line": 2,
15          "column": 3
16        }
17      ]
18    }
19  ]
20 }
21

```

Figure 9: Requete GraphQL sans token

The screenshot shows a GraphQL playground interface with the following details:

- URL:** POST http://localhost:8055/graphiql
- Headers:** Authorization (empty)
- Query:**

```

1 query {
2   structure { nom ville praticiens { nom prenom email specialite { libelle } } }
3 }

```

- Response:** (Success) 200 OK 43ms 8088
- JSON Response:**

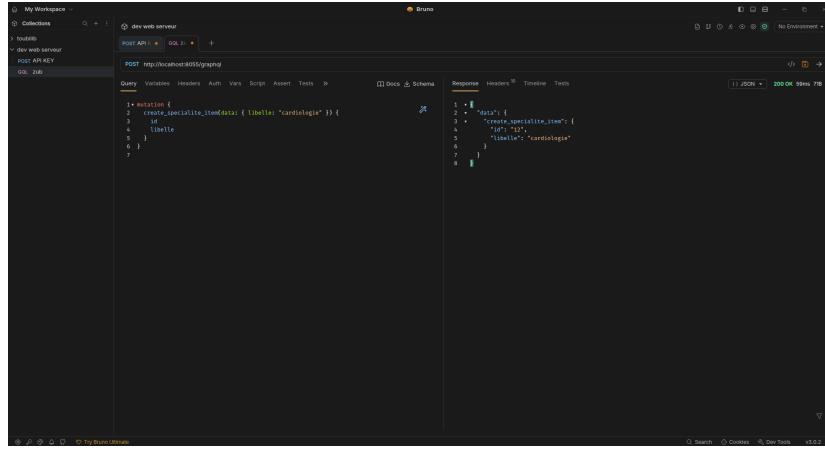
```

1 {
2   "data": {
3     "structure": [
4       {
5         "nom": "Bernard",
6         "prenom": "Alex",
7         "email": "alex.bernard@example.org",
8         "specialites": [
9           {
10             "libelle": "Pediatrie"
11           }
12         ],
13         "ville": "Paris"
14       },
15       {
16         "nom": "Cabinet Medical du Centre",
17         "ville": "Villiers-sur-Marne",
18         "praticiens": [
19           {
20             "nom": "Durand",
21             "prenom": "Luc",
22             "email": "luc.durand@example.org",
23             "specialites": [
24               {
25                 "libelle": "Dermatologie"
26               }
27             ],
28           }
29         ],
30         "ville": "Maison de Sante",
31         "praticiens": [
32           {
33             "nom": "Petit",
34             "prenom": "Sarah",
35             "email": "sarah.petit@example.org",
36             "specialites": [
37               {
38                 "libelle": "Dermatologie"
39               }
40             ],
41           }
42         ],
43         "ville": "Uneville",
44         "praticiens": [
45           {
46             "nom": "Petit",
47             "prenom": "Sarah",
48             "email": "sarah.petit@example.org",
49             "specialites": [
50               {
51                 "libelle": "Dermatologie"
52               }
53             ],
54           }
55         ],
56       }
57     ],
58   }
59 }

```

Figure 10: Requete GraphQL avec donnees imbriquées

Verification d'une requete GraphQL renvoyant des structures et leurs praticiens avec specialite. Cela valide la navigation dans les relations cote API.



The screenshot shows a GraphQL playground interface. On the left, there's a sidebar with 'My Workspace' containing 'Collections' (empty), 'YouTube' (empty), 'dev web server' (selected), 'POST API KEY', 'GQL', and 'Zulu'. The main area has tabs for 'Query', 'Variables', 'Headers', 'Auth', 'Vars', 'Script', 'Assert', 'Tests', 'Docs', and 'Schema'. A 'Query' tab is active, showing the following GraphQL code:

```
1 mutation {  
2   create_specialite(item: { libelle: "cardiologie" }) {  
3     id  
4     libelle  
5   }  
6 }  
7
```

Below the code, the 'Response' tab shows the JSON response:

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

The status bar at the bottom indicates 'JSON', '200 OK', '9ms', and '7B'.

Figure 11: Mutation GraphQL admin

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

Conclusion

Le CMS headless Directus a ete deploie avec succes, le modele de donnees a ete construit et alimente via CSV, puis verifie dans le backoffice. L'API REST permet de consommer les donnees avec selection de champs et navigation relationnelle. Les captures annexes montrent en plus la mise en place des roles et l'utilisation de GraphQL pour preparer la suite du projet. L'ensemble constitue un rendu clair, exploitable et conforme aux objectifs du TD.