

COMPTE RENDU

TP4



CHAIMAE EL MATTARI

304

ÉTAPE 1 – C'EST LA BASE

• Installation de Mongo:

- Dans un premier temps, j'ai créé un nouveau projet vide, et lancé la commande : `npm init -y`
- Puis, j'ai exécuté la commande suivante, pour installer le pilote dans le répertoire de mon projet : `npm install mongodb@6.3`

• Documentation MongoDB :

- J'ai créé un déploiement MongoDB gratuit sur Atlas en suivant la documentation du site officiel :
 - J'ai créé et configuré un cluster avec succès, avec les informations suivantes :

<div>Nom d'utilisateur <input type="text" value="user_tps"/> Mot de passe <input type="password" value="owQDjTwwpgPlma7D"/> </div>	<table><thead><tr><th>Liste d'accès IP</th><th>Description</th></tr></thead><tbody><tr><td>77.136.66.37/32</td><td>Mon adresse IP</td></tr></tbody></table>	Liste d'accès IP	Description	77.136.66.37/32	Mon adresse IP
Liste d'accès IP	Description				
77.136.66.37/32	Mon adresse IP				

• Installation de Mongosh :

- Pour ce faire, j'ai utilisé les commandes suivantes :
 - `brew tap mongodb/brew`
 - `brew install mongodb-community`
- Malheureusement, j'ai rencontré des difficultés lors de l'installation, cependant, j'ai opté pour une solution alternative en utilisant Docker :
- `docker run --name mongodb -d -p 27017:27017 mongodb/mongodb-community-server:$MONGODB_VERSION`

• Connexion entre Node.js et MongoDB :

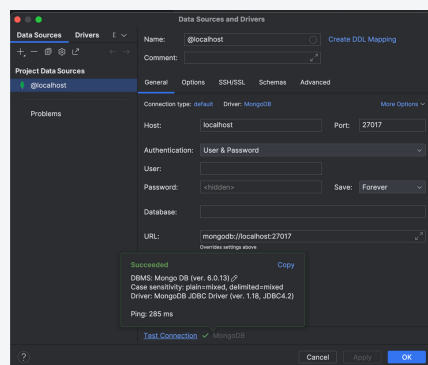
- Pour ce faire, j'ai utilisé un script Node.js trouvé dans la documentation du site de MongoDB, permettant de se connecter au cluster Atlas. J'ai bien évidemment personnalisé en remplaçant le username, le mot de passe et le nom de mon cluster.

• Vérification de la connexion :

- J'ai ensuite vérifié la connexion en utilisant Mongosh pour se connecter au cluster.
- J'ai exécuté la commande 'show dbs' et j'ai pu vérifier la présence des bases de données par défaut.
- J'ai également vérifié la connexion avec l'outil intégré sur PHPSTORM.

```
test> show dbs
admin    40.00 KiB
config   12.00 KiB
local    40.00 KiB
test>
```

Test avec la commande : show dbs



Test avec l'outil intégré de base de données

ÉTAPE 2 – C'EST LA BASE

- **Installation des dépendances Fastify et Mongoose :**

- Pour ce faire, j'ai utilisé la commande suivante : `npm install fastify mongoose`

- **Génération de clés de chiffrement :**

- Pour activer HTTPS avec Fastify, j'ai généré des clés de chiffrement en utilisant OpenSSL avec la commande suivante : `openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365`

- **Configuration de la connexion à MongoDB :**

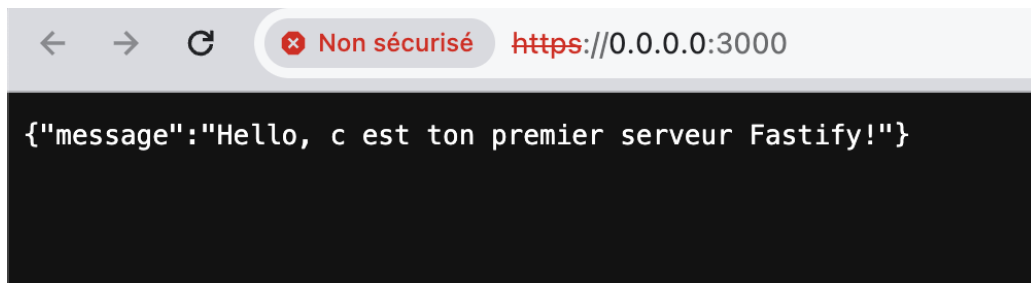
- J'ai dans un premier temps créé un fichier 'db.js' dans un dossier 'database', ce fichier de configuration permet de gérer la connexion à MongoDB.

- **Fichier décrivant le format de données :**

- J'ai créé un fichier 'livre.js' dans un dossier 'modele' qui décrit le format de données pour enregistrer un livre dans la base de données en utilisant 'mongoose'.
- Le schéma d'un livre comprend donc un titre, un auteur, une description et un format. Le format par défaut 'poche' a également été configuré dans ce schéma.

- **Configuration du serveur Fastify :**

- Le fichier principal 'app.js' permet la configuration du serveur Fastify avec la prise en charge de la connexion avec la base de données, de CORS et l'activation de HTTPS.

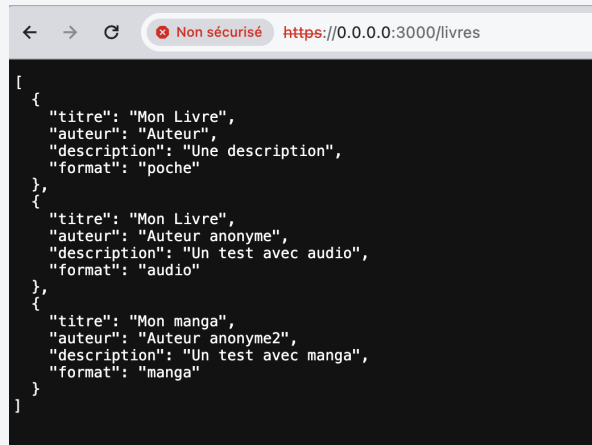


Test du serveur Fastify

ÉTAPE 3– C’EST LA BASE

• Contrôleur :

- J’ai créé un fichier ‘LivresControllers.js’ dans le dossier ‘controllers’. Ce fichier définit avec des fonctions les différentes opérations CRUD sur les livres :
 - Une fonction getAllLivres qui récupère tous les livres de la base de données en utilisant le modèle Livre. Si aucun livre n’est trouvé, elle renvoie un message approprié, sinon elle formate les livres en JSON avant de les renvoyer.



```
[
  {
    "titre": "Mon Livre",
    "auteur": "Auteur",
    "description": "Une description",
    "format": "poche"
  },
  {
    "titre": "Mon Livre",
    "auteur": "Auteur anonyme",
    "description": "Un test avec audio",
    "format": "audio"
  },
  {
    "titre": "Mon manga",
    "auteur": "Auteur anonyme2",
    "description": "Un test avec manga",
    "format": "manga"
  }
]
```

https://0.0.0.0:3000/livres, Fonction : getAllLivres

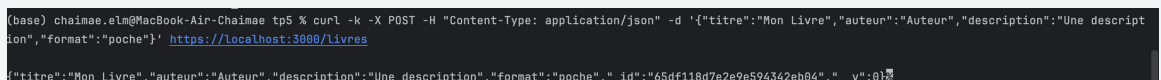
- Une fonction getLivresById, elle prend l’ID d’un livre en paramètre et récupère ce livre dans la base de données. Si aucun livre n’est trouvé elle renvoie également un message approprié, sinon elle formate les livres en JSON et les renvoie.



```
{ "titre": "Mon Livre", "auteur": "Auteur anonyme", "description": "Un test avec audio", "format": "audio" }
```

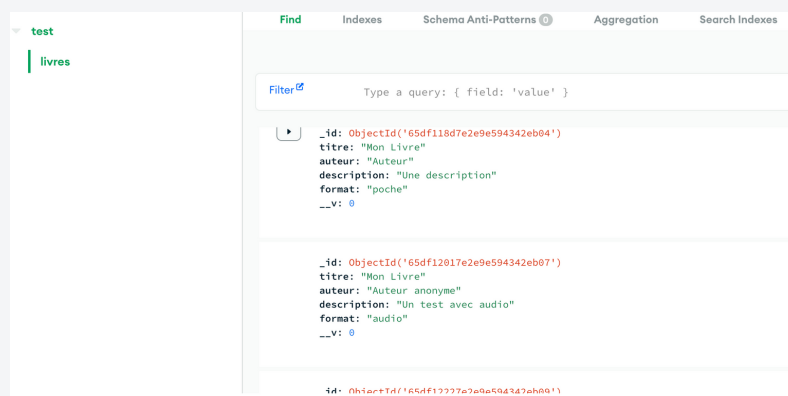
https://localhost:3000/livres/65df12017e2e9e594342eb07, Fonction : getLivresById

- Une fonction addLivres, elle permet de créer un nouveau livre en utilisant les données fournies dans le corps de la requête, à savoir les différents champs du modèle Livre prédéfini. Le nouveau livre est enregistré dans la base de données.



```
(base) chaimae.e@MacBook-Air-Chaimae:tp5 % curl -X POST -H "Content-Type: application/json" -d '{"titre": "Mon Livre", "auteur": "Auteur", "description": "Une description", "format": "poche"}' https://localhost:3000/livres
{"titre": "Mon Livre", "auteur": "Auteur", "description": "Une description", "format": "poche", "_id": "65df118d7e2e9e594342eb04", "__v": 0}
```

Création d’un nouveau livre en utilisant curl



Ajout dans la base de données du nouveau livre

- Une fonction `updateLivre`, elle met à jour un livre existant en utilisant son ID et les données fournies dans le corps de la requête. Si le livre n'est pas trouvé, elle renvoie une réponse 404, sinon elle renvoie le livre mis à jour.

```
(base) chaimae.elm@MacBook-Air-Chaimae tp5 % curl -k -X PUT -H "Content-Type: application/json" -d '{"titre":"Nouveau Titre","auteur":"Auteur anonyme","description":"Nouvelle Description","format":"audio"}' https://localhost:3000/livres/65df12017e2e9e594342eb07
{"_id":"65df12017e2e9e594342eb07","titre":"Nouveau Titre","auteur":"Auteur anonyme","description":"Nouvelle Description","format":"audio","__v":0}
```

Fonction `updateLivre`



Fonction `updateLivre`

```
_id: ObjectId('65df12017e2e9e594342eb07')
titre: "Nouveau Titre"
auteur: "Auteur anonyme"
description: "Nouvelle Description"
format: "audio"
__v: 0
```

Mise à jour dans la base de données

- Une fonction `deleteLivre`, elle supprime un livre en utilisant son ID. Si le livre n'est pas trouvé elle renvoie une erreur 404, sinon elle renvoie un message indiquant que le livre a été supprimé avec succès.

```
(base) chaimae.elm@MacBook-Air-Chaimae tp5 % curl -k -X DELETE https://localhost:3000/livres/65df12017e2e9e594342eb07
{"message":"Livre supprimé avec succès"}
```

Fonction `deleteLivre`

```
[
  {
    "titre": "Mon Livre",
    "auteur": "Auteur",
    "description": "Une description",
    "format": "poche"
  },
  {
    "titre": "Mon manga",
    "auteur": "Auteur anonyme2",
    "description": "Un test avec manga",
    "format": "manga"
  }
]
```

Le livre a été supprimé avec succès

• Routes :

- J'ai créé un fichier 'routes.js' dans un dossier 'routes' qui associe chaque route à sa fonction de contrôleur.
 - Route ('/') : route d'accueil qui renvoie vers le message de test du serveur
 - Route ('/livres') : Associe cette route à la fonction 'getAllLivres', et récupère donc tout les livres. Il s'agit du Read de CRUD.
 - Route ('livres/:id') : Associe cette route à la fonction 'getLivresById', qui récupère donc un livre par son Id, il s'agit également du Read.
 - Route ('/livres') de type Post, associe cette route à la fonction 'addLivres', et permet de créer un nouveau livre. Il s'agit du Create de CRUD
 - Route ('/livres/:id') de type Put, associe cette route à la fonction 'updateLivres', met à jour un livre existant. Il s'agit du Update de CRUD.
 - Route ('/livres/:id') de type delete, associe cette route à la fonction 'deleteLivres', permet la suppression d'un livre. Il s'agit du Delete de CRUD.

• Mise à jour de l'app.js :

- Ajout de l'enregistrement des routes définies dans le fichier 'route.js'.