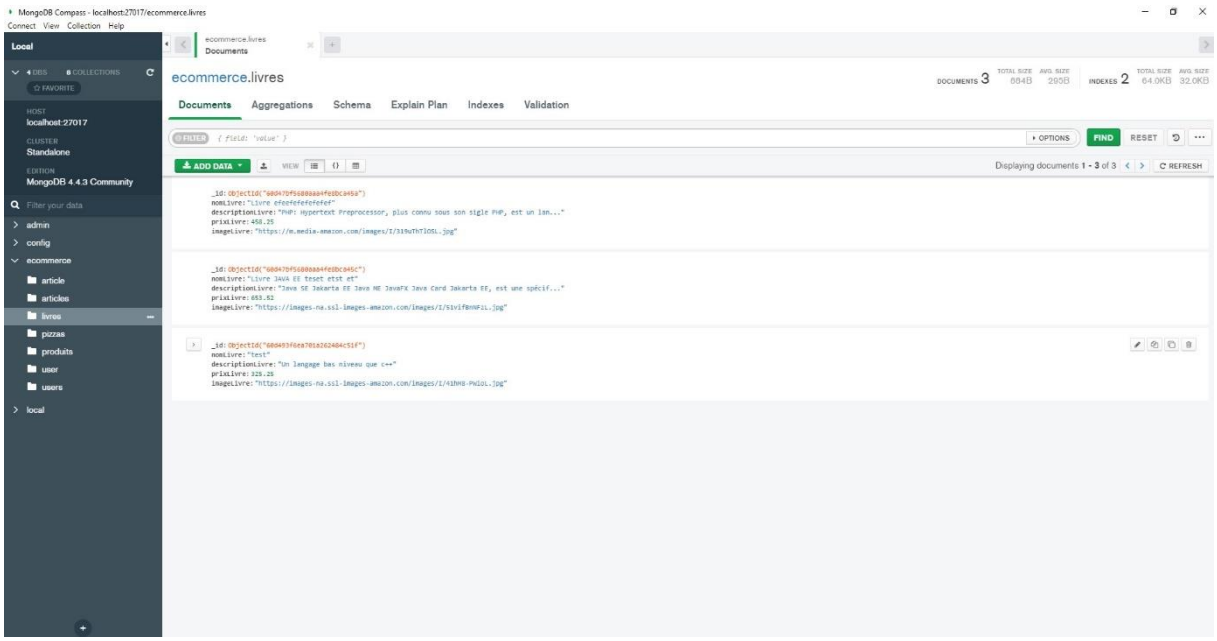
Projet 9 – C : Backend ExpressJs – Node Js et MongoDb :

Objectif :

Créer une application web robuste capable de communiquer avec MongoDB et afficher les données de votre collections livres via un routing simple.

Sources : https://github.com/michelonlineformapro/Projet-9-C-ExpressJs-NodeJs-et-MongoDB

1. Installer MongoDB Compass et MongoDB Windows disponible dans votre espace apprenant => dossier partagé => OUTILS
2. Lancer MongoDB Compass et se connecter en local (localhost : 27017)
3. Créer une base de données (pour exemple : ecommerce)
4. Créer une collection livres au format json



5. Installer ExpressJs : https://expressjs.com/fr/starter/installing.html
6. Lancer la commande npm init pour générer votre package.json

server.js | package.json

```
{
  "name": "nodecrud",
  "version": "1.0.0",
  "description": "CRUD Express MongoDb",
  "main": "server.js",
  "scripts": {
    "test": "dev",
    "prod": "node server",
    "watch": "nodemon server",
    "start": "npm run watch"
  },
  "author": "Michel",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^10.0.0",
    "express": "^4.17.1",
    "mongoose": "^5.12.14",
    "nodemon": "^2.0.7"
  }
}
```

7. Ajouter le point d'entrée server.js comme ''main'' : ''server.js'
8. Créer le fichier server.js
9. Installer Mongoose via npm : https://github.com/Automattic/mongoose
10. Appel du middleware Mongoose et dotenv (module de variable d'environnement)
11. https://www.npmjs.com/package/dotenv
12. Paramétré le middleware dotenv qui pointe vers un fichier .env
13. Dans .env : configurer votre connexion à MongoDB : DATABASE=mongodb://localhost:27017/ecommerce
14. Créer une connexion à MongoDB via Mongoose et ajouter les options de connexion

15. Créer une promesse qui renvoie une erreur en cas d'échec de connexion
16. Importer le fichier app.js en tant que module
17. Démarrer votre serveur avec app.listen sur le port 3000
18. Créer votre fichier app.js
19. Importer ExpressJs et créer une instance
20. Importer Mongoose
21. Créer un Schéma de votre collection Livres avec mongoose.Schema
22. Stocker le model de votre collection dans une variable à l'aide de mongoose.model
23. Créer une route de base pour votre serveur a l'aide de la méthode GET et request et response

```
app.get('/', (request, response) => {
    console.log('tets')
    response.send('Le serveur est demarrer !');
});

//Notre routes pour afficher tous les livres http://localhost:3000/livres

//ici app (ligne 4) utilise la methode GET + une requète http similaire a fetch() requete + reponse

app.get('/livres', async (request, response) => {
    //Mongoose utilise la methode find() pour parcourir les element comme un forEach()
    const livres = await Livres.find();
    //La reponse envoyée est au format Json() + notre find en paramètres
    response.json(livres)
});

//Exporter app pour l'importer dans le fichier server.js
module.exports = app;
```

24. Créer une seconde route ‘/livres’ et une requête http asynchrone capable d’appeler votre model et retournée une réponse au format json
25. Exporter votre module app.js
26. Tester votre url : http://localhost:3000/livres
27. Les données affichées proviennent bien de votre base de données ecommerce et votre collections livres de MongoDB.