Bouallegue Lina **Date :** 14-10-2025

# Compte Rendu – TP MERN Semaine 3

# Persistance des Données : MongoDB, Mongoose et la Programmation Asynchrone

## 1. Objectifs

L'objectif principal de cette séance est d'apprendre à **intégrer MongoDB** à une application **Node.js** à l'aide de la bibliothèque **Mongoose**, et de comprendre la **programmation asynchrone** à travers les mots-clés `async` et `await`.

À la fin du travail, nous devons être capables de :

-Comprendre le principe du **NoSQL** et l'organisation d'une base MongoDB (base → collections → documents).

-Expliquer et utiliser la **programmation asynchrone** (`callback`, `Promise`, `async/await`).

-Créer un **cluster MongoDB Atlas** et s'y connecter depuis une application Node.js.

-Créer et utiliser un **modèle Mongoose** (schéma + validation).

-Tester des routes API avec **Postman**.

-Réaliser un **modèle User** complet et son **contrôleur** pour la gestion des utilisateurs.

# 2. Partie 1 : Concepts Techniques Approfondis

## 2.1. Le Monde Asynchrone de Node.js

Lorsqu'on communique avec une base de données, on fait des opérations dites d'**Entrée/Sortie (I/O)**.
Node.js les exécute de façon **asynchrone**, c'est-à-dire **sans bloquer** le reste du programme.

### a) Les Callbacks (ancienne méthode)

On passe une fonction en paramètre pour qu'elle s'exécute après la fin d'une opération :

```
database.query("SELECT * FROM users", function(err, results) {
  if (err) console.error(err);
  console.log(results);
});
```

➔ Inconvénient : si on enchaîne plusieurs callbacks, le code devient illisible = **"Callback Hell"**.

### b) Les Promises

Elles représentent une **valeur future** (succès ou échec).
On les enchaîne avec `.then()` et `.catch()` :

```
database.query("SELECT * FROM users")
  .then(results => console.log(results))
  .catch(err => console.error(err));
```

### c) async/await — La méthode moderne

Elle rend le code asynchrone plus **simple et lisible** :

```
async function getUsers() {
  try {
    const results = await database.query("SELECT * FROM users");
    console.log(results);
  } catch (err) {
    console.error(err);
  }
}
```

Remarque :

-await ne peut être utilisé **que dans une fonction async**.

-await met la fonction en pause **sans bloquer Node.js**.

2.2 MongoDB et Mongoose :

**MongoDB**

- Base de données **NoSQL** (non relationnelle).
- Stocke les données sous forme de **documents** (objets JSON).
- Les documents sont regroupés dans des **collections** (comme des tables dans SQL).

**Mongoose**

C'est une **couche intermédiaire** (ODM : *Object Data Modeling*) entre Node.js et MongoDB.
Elle permet de :

- Créer des **schémas** (structure des données).
- Définir des **modèles** (objets utilisables dans le code).
- Ajouter des **validations** (ex. champs obligatoires).

Exemple :

```
const articleSchema = new mongoose.Schema({
  title: String,
  content: String
});
module.exports = mongoose.model('Article', articleSchema);
```

Schéma = plan du document

Modèle = usine qui crée les documents à partir du plan

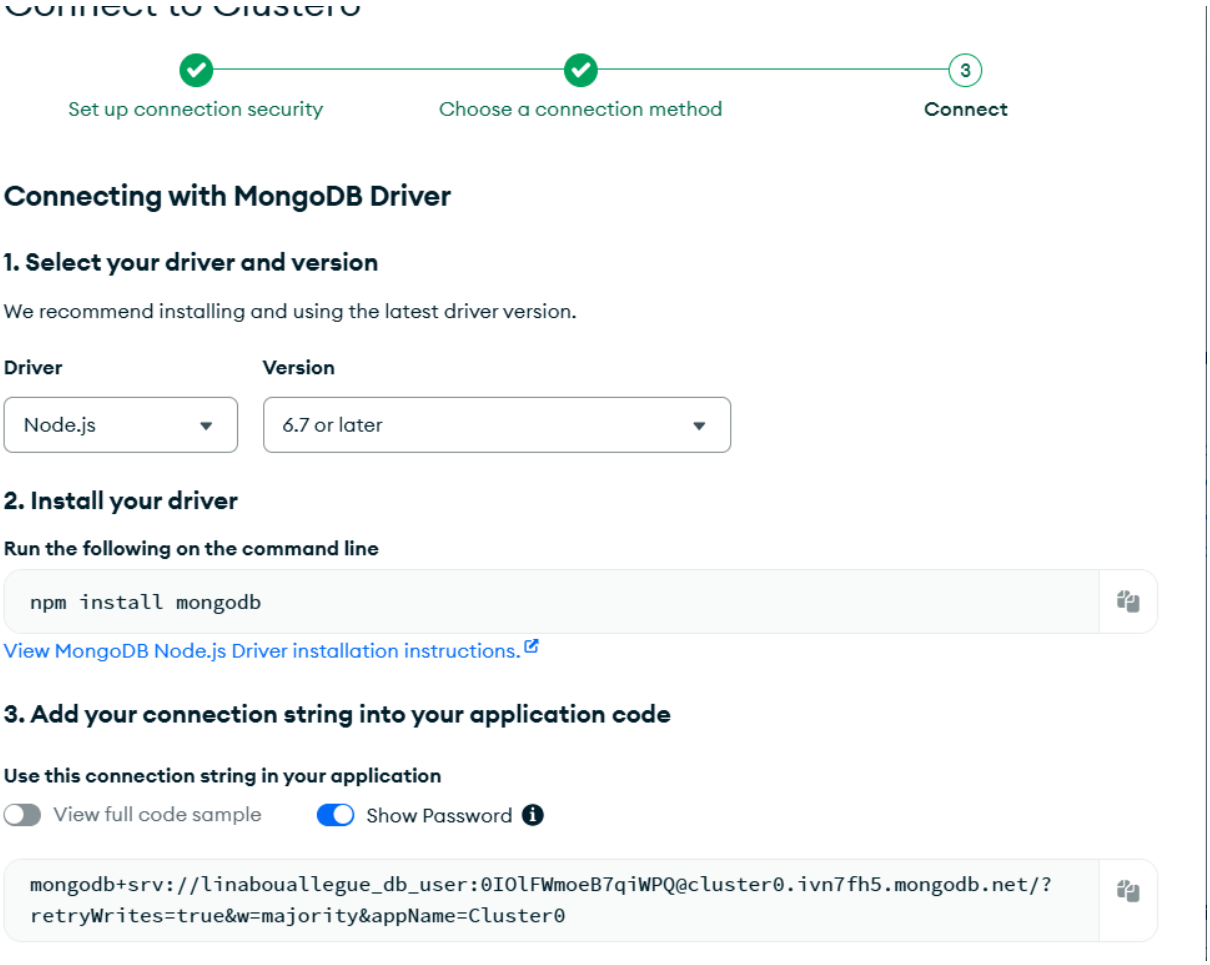
3. **Mise en pratique**

Étape 1 : Création du cluster MongoDB Atlas

1. Connexion sur https://www.mongodb.com/atlas/database.
2. Création d'un **cluster gratuit (M0)**.
3. Configuration de la sécurité :

- Création d'un utilisateur avec mot de passe.
- Autorisation de l'accès réseau 0.0.0.0/0.

4. Copie de la **chaîne de connexion** :

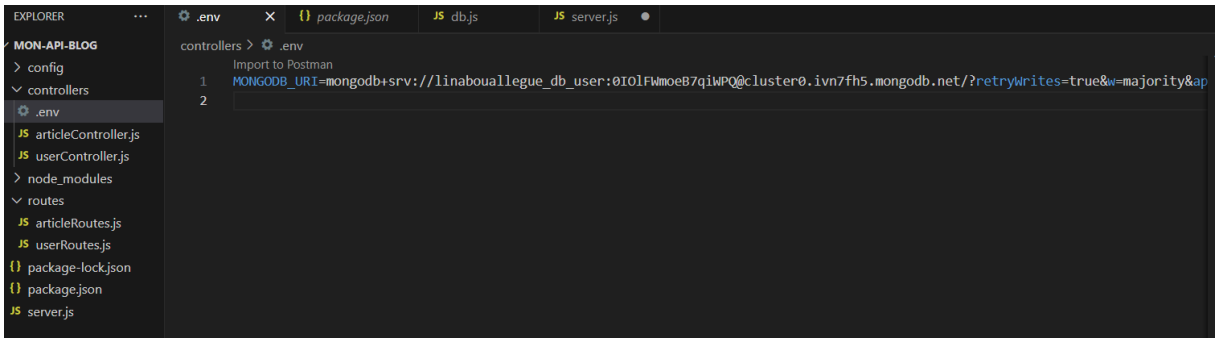


Étape 2 : Installation et configuration

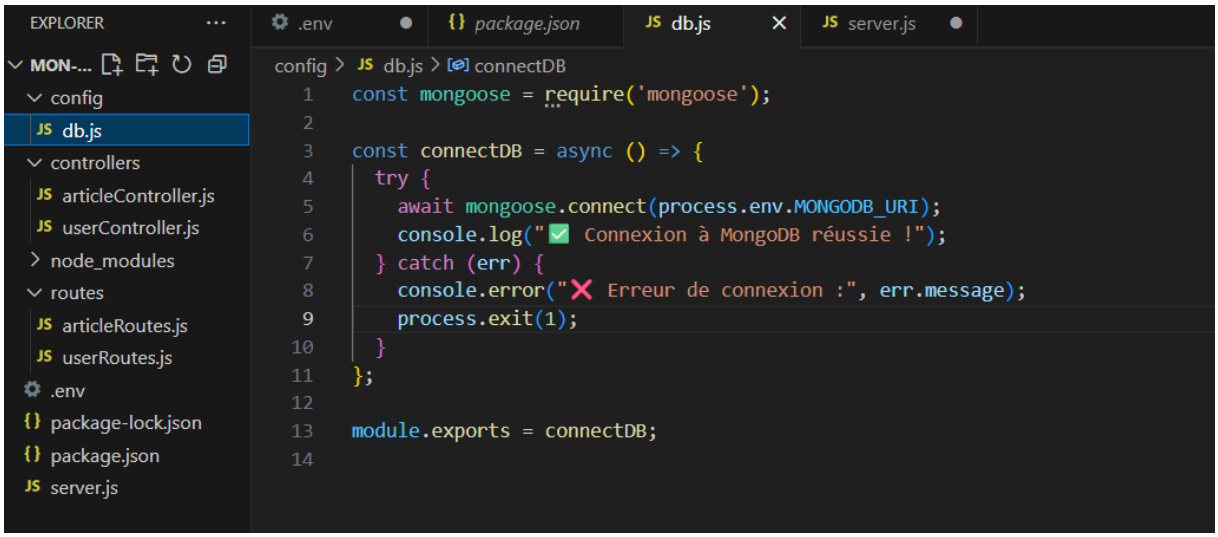
Installation des dépendances :

```
C:\Users\Lina\Desktop\cr_mern\mon-api-blog>npm install mongoose dotenv expre
ss

changed 1 package, and audited 114 packages in 3s

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 11.1.0 -> 11.6.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.6.2
npm notice To update run: npm install -g npm@11.6.2
npm notice

C:\Users\Lina\Desktop\cr_mern\mon-api-blog>
```

Création du fichier.env :




Création du fichier config/db.js :



Fichier server.js :

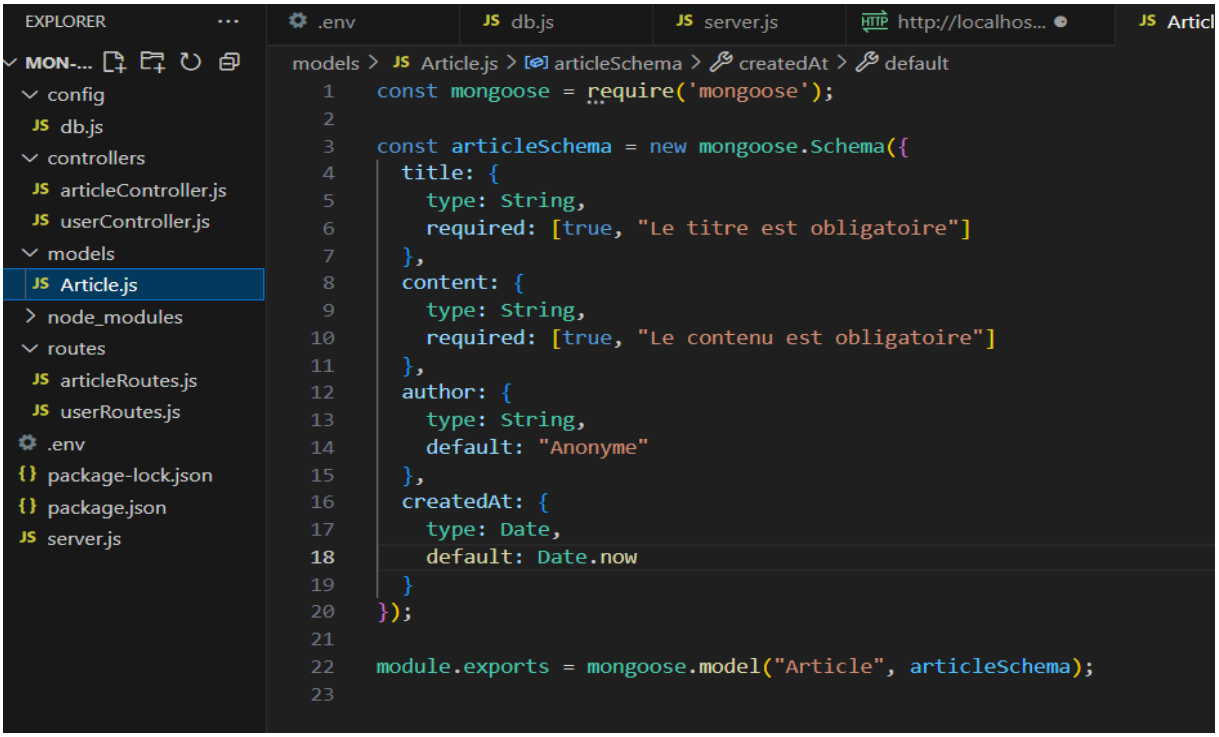
```
server.js > ...
1   require('dotenv').config();
2   const express = require('express');
3   const connectDB = require('./config/db');
4
5   const app = express();
6   app.use(express.json());
7
8
9   connectDB();
10
11
12  app.get('/', (req, res) => {
13    res.send('Serveur et MongoDB fonctionnent parfaitement 🚀');
14  });
15
16
17  app.listen(3000, () => console.log('🚀 Serveur démarré sur le port 3000'));
18
```

Puis exécute ton serveur :

```
PS C:\Users\Lina\Desktop\cr_mern\mon-api-blog> node server.js
[dotenv@17.2.3] injecting env (1) from .env -- tip: 🛠️ run anywhere with `dotenvx run -- yourcommand`
🚀 Serveur démarré sur le port 3000
✅ Connexion à MongoDB réussie !
```

Étape 3 : Création du modèle Article :

Un modèle définit la structure des données que tu veux stocker dans ta base MongoDB.



Étape 4 : Création du contrôleur Article

```
controllers > JS articleController.js > ...
1   const Article = require('../models/Article');
2
3   // Test API
4   const testApi = (req, res) => {
5     res.status(200).json({
6       message: 'Le test depuis le contrôleur a fonctionné 🎉',
7       success: true
8     });
9   };
10
11  // Créer un article
12  const createArticle = async (req, res) => {
13    try {
14      const newArticle = new Article(req.body);
15      const savedArticle = await newArticle.save();
16      res.status(201).json({
17        message: '✅ Article créé avec succès !',
18        article: savedArticle
19      });
20    } catch (err) {
21      res.status(400).json({ message: 'Erreur lors de la création', error: err.message });
22    }
23  };
24
25  // Lire tous les articles
26  const getAllArticles = async (req, res) => {
27    try {
28      const articles = await Article.find();
29      res.status(200).json(articles);
30    } catch (err) {
31      res.status(500).json({ message: 'Erreur lors de la lecture', error: err.message });
32    }
33  };
34
35  module.exports = { testApi, createArticle, getAllArticles };
36
```

Étape 5 : Mise à jour du routeur

```
routes > JS articleRoutes.js > ...
1   const express = require('express');
2   const router = express.Router();
3   const { testApi, createArticle, getAllArticles } = require('../controllers/articleController');
4
5   // 🧪 Route de test
6   router.get('/test', testApi);
7
8   // 📄 Récupérer tous les articles
9   router.get('/', getAllArticles);
10
11  // ➕ Créer un article
12  router.post('/', createArticle);
13
ktop\cr_mern\mon-api-blog\routes\userRoutes.js
```

Étape 6 : Test final avec Postman