MASTER 1

Création d'une Application CRUD avec MEAN

Nous allons créer une application **CRUD** sous **MEAN** qui permet de gérer une liste d'articles avec les actions : **Créer, Lire, Mettre à jour et Supprimer**.

Étape 1 : Initialisation du projet

1.1 Installer Node.js et MongoDB

Assure-toi que **Node.js** et **MongoDB** sont installés sur ton système :

- Télécharger Node.js
- Installer MongoDB

Étape 2 : Mise en place du Backend (API avec Express et MongoDB)

2.1 Initialiser le backend

Crée un dossier `backend` et initialise le projet Node.js :

```
mkdir backend && cd backend
npm init -y
```

2.2 Installer les dépendances

Ajoute Express, Mongoose (ORM pour MongoDB) et CORS :

```
npm install express mongoose cors dotenv body-parser
```

> 📝 **ORM :** (Object-Relational Mapping) est un outil qui permet de manipuler une base de données en utilisant des objets et du code au lieu d'écrire directement des requêtes SQL.
> Au lieu d'écrire

```
SELECT * FROM utilisateurs WHERE id = 1;
```

On écrit :

```
User.findById(1);
```

Ajoute aussi Nodemon pour le développement :

```
npm install --save-dev nodemon
```

📝 **Nodemon :** Nodemon est un outil utilisé en développement Node.js pour surveiller les changements dans les fichiers de votre projet. Lorsque vous modifiez un fichier, Nodemon redémarre automatiquement votre application Node.js, ce qui vous évite de devoir le faire manuellement à chaque fois

Modifie ***package.json*** pour ajouter un script :

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

2.3 Configurer Express et MongoDB

Crée un fichier server.js :

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

// Connexion à MongoDB
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log("MongoDB connecté"))
  .catch(err => console.log(err));
```

```
// Routes
app.use('/articles', require('./routes/articleRoutes'));

app.listen(PORT, () => console.log(`Serveur démarré sur le port
${PORT}`));
```

2.4 Définir le modèle MongoDB

Crée un dossier models et un fichier Article.js :

```
const mongoose = require('mongoose');

const ArticleSchema = new mongoose.Schema({
  title: { type: String, required: true },
  content: { type: String, required: true }
}, { timestamps: true });

module.exports = mongoose.model('Article', ArticleSchema);
```

2.5 Créer les routes Express

Crée un dossier routes et un fichier articleRoutes.js :

```
const express = require('express');
const router = express.Router();
const Article = require('../models/Article');

// Créer un article
router.post('/', async (req, res) => {
  try {
    const article = new Article(req.body);
    await article.save();
    res.status(201).json(article);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});

// Lire tous les articles
router.get('/', async (req, res) => {
  try {
    const articles = await Article.find();
    res.json(articles);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
```

```
// Mettre à jour un article
router.put('/:id', async (req, res) => {
  try {
    const article = await Article.findByIdAndUpdate(req.params.id,
req.body, { new: true });
    res.json(article);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});

// Supprimer un article
router.delete('/:id', async (req, res) => {
  try {
    await Article.findByIdAndDelete(req.params.id);
    res.json({ message: "Article supprimé" });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

module.exports = router;
```

2.6 Lancer le backend

Ajoute un fichier .env avec ta connexion MongoDB :

```
MONGO_URI=mongodb://localhost:27017/mean_db
```

> 📝 **Fichier .env :** Le fichier .env est utilisé pour stocker des informations sensibles ou des configurations d'un projet, comme :
> • Les clés d'API
> • Les mots de passe
> • Les URLs de bases de données
> • D'autres paramètres d'environnement (mode développement ou production)
> Pourquoi l'utiliser ?
> • Sécurité : Évite d'exposer des données sensibles dans le code.
> • Flexibilité : Change facilement les configurations sans modifier le code.
> • Facile à gérer : Chaque environnement (dev, test, prod) peut avoir son propre fichier .env.
> 👉 Astuce : Ne jamais pousser le fichier .env sur Git (utiliser .gitignore) !

Puis lance le serveur :

```
npm run dev
```

2.7 Tester sur Postman

1. Télécharger et installer Postman sur https://www.postman.com

2. Ouvrir Postman

3. Créer une nouvelle requête POST

- Méthode : POST
- URL : http://localhost:5000/articles

4. Configurer le corps de la requête

- Dans l'onglet Body, sélectionne raw.
- Change le type en JSON.
- Ajoute un corps JSON comme ci-dessous

```
{
  "title": "Mon premier article",
  "content": "Ceci est le contenu de l'article"
}
```

5. Envoyer la requête

- Clique sur Send.
- Si tout fonctionne, tu devrais recevoir une réponse avec l'article ajouté et son ID généré par MongoDB.

6. Vérifier l'ajout

- Pour voir tous les articles, fais une requête GET sur :

```
http://localhost:5000/articles
```

7. Tester la modification et l'ajout.