TP API REST Node.js + Express + MongoDB

1. Objectif du TP

Réaliser une API REST permettant de gérer des produits en utilisant Node.js, Express, MongoDB et Mongoose.

L'objectif est de comprendre les bases d'une architecture backend moderne : modèle, contrôleur, routes et serveur.

2. Résultat attendu

À la fin du TP, vous devez avoir :

- Une API opérationnelle accessible via `http://localhost:3000/products`
- Un CRUD complet :
  - POST /products : créer un produit
  - GET /products : lister tous les produits
  - GET /products/:id : récupérer un produit par ID
  - PUT /products/:id : modifier un produit
  - DELETE /products/:id : supprimer un produit
- Une base de données MongoDB fonctionnelle
- Un projet structuré selon une architecture propre (MVC)

3. Technologies utilisées

- Node.js
- Express
- MongoDB
- Mongoose

- dotenv
- CORS
- Nodemon (développement)

4. Structure du projet à respecter

```
/src
  /controllers
    product.controller.js
  /models
    product.js
  /routes
    product.routes.js
  server.js
.env
```

Cette structure est obligatoire.

Aucun code métier ne doit être placé dans `server.js`.

5. Modèle attendu : Product

Chaque produit doit contenir :

Champ	Type	Obligatoire	Valeur par défaut
name	String	Oui	-
description	String	Non	-
price	Number	Oui	-
quantity	Number	Oui	0
createdAt	Date	Non	Date actuelle

6. Fonctions à implémenter dans le contrôleur

Vous devez écrire un contrôleur contenant les 5 fonctionnalités suivantes :

1. **createProduct(req, res)**
    a. Crée un produit à partir des données du body
    b. Retourne une réponse 201 ou 400 en cas d'erreur
2. **getAllProducts(req, res)**
    a. Retourne tous les produits stockés en base
3. **getProductById(req, res)**
    a. Recherche un produit par son ID
    b. Retourne 404 si non trouvé
4. **updateProduct(req, res)**
    a. Met à jour un produit existant
    b. Retourne le produit mis à jour
    c. Retourne 404 si l'ID n'existe pas
5. **deleteProduct(req, res)**
    a. Supprime un produit
    b. Retourne 200 si supprimé
    c. Retourne 404 si non trouvé

Les fonctions doivent être écrites avec `async/await` et un bloc `try/catch`.

7. Routes à créer

Dans `product.routes.js`, vous devez déclarer :

Méthode	URI	Contrôleur appelé
POST	/products	createProduct
GET	/products	getAllProducts
GET	/products/:id	getProductById
PUT	/products/:id	updateProduct

DELETE	/products/:id	deleteProduct

8. Configuration du serveur (server.js)

Votre serveur doit :

1. Importer express, mongoose, dotenv, cors
2. Charger la configuration `.env`
3. Configurer les middlewares :
    a. `express.json()`
    b. `cors()`
4. Connecter mongoose à MongoDB
5. Monter les routes `/products`
6. Démarrer le serveur avec le port défini dans `.env`

Exemple minimal :

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

9. Exigences minimales

Pour valider le TP, votre projet doit :

- Être entièrement fonctionnel
- Envoyer uniquement des réponses JSON
- Respecter les codes HTTP (200, 201, 400, 404)
- Avoir une structure claire et propre
- Ne contenir aucune logique métier dans les routes ou dans server.js

10. Bonus possibles (non obligatoires)

- Validation de données dans le body
- Gestion des catégories de produits
- Pagination
- Système de logs
- Middlewares personnalisés