Énoncé TP - Création d'une API REST pour la gestion de tâches (Todo)

Objectif

Développer une API REST complète pour la gestion de tâches (todos) en utilisant Node.js, Express, TypeScript et MongoDB avec une architecture MVC.

Prérequis

- Node.js et npm installés
- MongoDB installé localement ou accès à MongoDB Atlas
- Un éditeur de code (VS Code recommandé)
- Postman, Insomnia ou Thunder Client pour tester l'API

Instructions

Étape 1 : Initialisation du projet

1. Créez un nouveau dossier pour votre projet et initialisez un projet npm
2. Initialisez un dépôt Git
3. Installez les dépendances nécessaires :
    - Express, Mongoose, dotenv, cors, helmet (principales)
    - TypeScript, @types/node, @types/express, @types/mongoose, @types/cors, ts-node, nodemon (développement)
4. Configurez TypeScript (tsconfig.json)
5. Créez un fichier .gitignore et .env
6. Établissez la structure de dossiers suivante :

```
todo-api/
├── src/
│   ├── config/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── repositories/
│   ├── routes/
│   ├── services/
│   ├── types/
│   ├── utils/
│   ├── app.ts
│   └── index.ts
└── tests/
```

Étape 2 : Configuration d'ESLint

1. Installez ESLint et les plugins nécessaires pour TypeScript
2. Configurez ESLint pour respecter les bonnes pratiques TypeScript
3. Ajoutez Prettier pour le formatage du code
4. Ajoutez des scripts npm pour le lint et le formatage

Étape 3 : Mise en place de la base du serveur

1. Créez un fichier de configuration de base de données (connexion MongoDB)
2. Implémentez votre fichier app.ts avec les middlewares de base (json, cors, helmet)
3. Créez le point d'entrée index.ts pour démarrer le serveur
4. Testez que le serveur démarre correctement

Étape 4 : Définition des modèles et types

1. Créez l'interface ITodo avec les propriétés suivantes :
    - id (optionnel)
    - title (obligatoire)
    - description (optionnel)
    - completed (boolean, par défaut à false)
    - dueDate (optionnel, Date)
    - priority (enum: LOW, MEDIUM, HIGH)
    - createdAt, updatedAt (timestamps)
2. Définissez des types pour les DTO (Data Transfer Objects) :
    - CreateTodoDto
    - UpdateTodoDto
3. Implémentez le schéma Mongoose pour Todo

Étape 5 : Implémentation du Pattern Repository

1. Créez une interface Repository générique avec les méthodes CRUD de base
2. Implémentez une classe abstraite MongooseRepository utilisant cette interface
3. Créez un TodoRepository spécifique avec des méthodes additionnelles :
    - findByStatus() - pour filtrer par statut (completed)
    - findByPriority() - pour filtrer par priorité

Étape 6 : Création des Services

1. Créez une classe d'erreur personnalisée pour gérer proprement les erreurs
2. Implémentez un TodoService avec :
    - Méthodes CRUD (findAll, findById, create, update, delete)
    - Une méthode toggleComplete pour basculer l'état de complétion
    - Logique de filtrage par statut et priorité
3. Gérez correctement les erreurs (ex: NotFoundError quand un todo n'existe pas)

Étape 7 : Middlewares

1. Créez un middleware de gestion d'erreurs centralisé
2. Installez Zod pour la validation
3. Implémentez un middleware de validation utilisant Zod

4. Définissez des schémas de validation pour la création et mise à jour de todos

Étape 8 : Contrôleurs

1. Créez un utilitaire asyncHandler pour gérer les fonctions asynchrones
2. Implémentez un TodoController avec les méthodes :
    - getAllTodos (avec filtrage)
    - getTodo
    - createTodo
    - updateTodo
    - deleteTodo
    - toggleTodoComplete

Étape 9 : Routes

1. Créez les routes pour les todos avec les patterns REST :
    - GET /todos - liste de todos
    - POST /todos - création
    - GET /todos/:id - détail d'un todo
    - PATCH /todos/:id - mise à jour
    - DELETE /todos/:id - suppression
    - PATCH /todos/:id/toggle - basculer l'état de complétion
2. Appliquez les middlewares de validation appropriés
3. Intégrez les routes à l'application Express

Étape 10 : Tests manuels

1. Utilisez Postman/Insomnia/Thunder Client pour tester chaque endpoint
2. Testez les cas de validation et d'erreur
3. Assurez-vous que tous les endpoints renvoient les formats de réponse attendus

Étape 11 : Tests automatisés (optionnel)

1. Configurez Jest et la base de données en mémoire MongoDB
2. Écrivez des tests unitaires pour le repository
3. Écrivez des tests d'intégration pour l'API

Étape 12 : Documentation Swagger (bonus)

1. Intégrez Swagger pour documenter votre API
2. Annotez chaque route avec des commentaires JSDoc pour Swagger

Exigences techniques

- Suivre les principes REST
- Utiliser une architecture MVC claire
- Implémenter une validation robuste avec Zod
- Gérer correctement les erreurs
- Ajouter des commentaires pertinents

- Suivre les règles de linting
- Effectuer des commits réguliers

Livrables

- Code source sur un dépôt GitHub
- Documentation des endpoints (manuel ou Swagger)
- Instructions pour démarrer l'application

Notes

- Commencez simple et ajoutez des fonctionnalités progressivement
- Testez régulièrement avec Postman à chaque nouvelle fonctionnalité
- Committez après chaque étape fonctionnelle
- N'hésitez pas à consulter la documentation officielle des bibliothèques

Bon développement !