ECE Paris – École d'ingénieurs

Mastere Data Engineer

TP MongoDB

Conception et implémentation d'une API REST complète

Étudiante : Hélène CAKPOSSE
Encadrant : Mme BOUCHLAGHEM LYDIA
Année universitaire : 2025–2026

Paris, January 10, 2026

Contents

1 Introduction **3**

2 Architecture du projet **4**
2.1 Structure des dossiers 4
2.2 Description des composants 4

3 Diagramme UML **5**
3.1 Diagramme de classes 5
3.2 Analyse du besoin fonctionnel 6
3.3 Analyse de la modélisation des données 6

4 Schéma de base de données **7**
4.1 Users 7
4.2 Categories 7
4.3 Posts 7
4.4 Comments 8

5 Routes API **9**
5.1 Utilisateurs 9
5.2 Posts 9
5.3 Commentaires 10
5.4 Comments 10
5.5 Catégories 10

6 Transactions MongoDB **11**

7 Seed Script **12**

8 Monitoring **13**

9 Difficultés rencontrées **14**

10 Améliorations possibles **15**

11 Conclusion **16**

1. Introduction

Dans un contexte où les applications web modernes reposent sur des architectures distribuées et orientées services, les API REST constituent un composant fondamental des systèmes d'information.

Ce travail pratique s'inscrit dans le cadre du module dédié aux bases de données NoSQL et a pour objectif la conception et l'implémentation d'une API REST complète reposant sur les technologies **Node.js**, **Express**, **MongoDB** et **Mongoose**.

L'API développée permet la gestion d'un système de publication de contenu structuré autour de quatre entités principales : *Users*, *Posts*, *Comments* et *Categories*. Elle intègre les opérations CRUD, des mécanismes de validation avancés, la gestion des relations entre collections ainsi que des fonctionnalités plus complexes telles que les transactions MongoDB et le monitoring des performances.

L'objectif pédagogique de ce projet est double : comprendre la modélisation des données dans un environnement NoSQL et mettre en œuvre une architecture backend robuste, maintenable et évolutive.

2. Architecture du projet

L'architecture du projet repose sur une organisation modulaire visant à séparer clairement les responsabilités de chaque composant, conformément aux bonnes pratiques du développement backend.

2.1 Structure des dossiers

```
.
 config/
 middleware/
 models/
 routes/
 services/
 seed/
 server.js
```

2.2 Description des composants

Le dossier **config** contient la configuration de connexion à MongoDB.

Le dossier **models** regroupe les schémas Mongoose définissant la structure des données, les contraintes de validation et les relations entre collections.

Le dossier **routes** expose les points d'entrée REST de l'API, chaque route correspondant à une ressource métier.

Le dossier **middleware** contient les middlewares personnalisés, notamment le module de monitoring des performances.

Le dossier **services** isole la logique métier complexe, en particulier la gestion des transactions MongoDB.

Enfin, le dossier **seed** permet l'initialisation de la base de données avec des données de test.

3. Diagramme UML

3.1 Diagramme de classes

Le diagramme de classes UML ci-dessous représente la modélisation conceptuelle du système. Il décrit les entités principales, leurs attributs et les relations existantes. Chaque utilisateur peut créer plusieurs posts. Les posts sont associés à une catégorie unique et peuvent recevoir plusieurs commentaires. Les commentaires supportent une structure hiérarchique via une auto-référence.

email : String
password : String
firstName : String
lastName : String
role : String
isActive : Boolean ;
[class, below=of User] (Post) **Post**
title : String
content : String
excerpt : String
status : String
viewCount : Number
featured : Boolean ;
[class, right=of Post] (Category) **Category**
name : String
slug : String
color : String ;
[class, below=of Post] (Comment) **Comment**
content : String
isEdited : Boolean
isDeleted : Boolean ;
[relation] (User) – node[right]1..* (Post); [relation] (Category) – node[above]1..* (Post);
[relation] (Post) – node[right]1..* (Comment); [relation] (Comment) edge[loop below]
node0..* (Comment);

3.2 Analyse du besoin fonctionnel

L'objectif fonctionnel principal de l'API est de fournir un backend capable de gérer un système de publication de contenu similaire à une plateforme de blog ou de gestion éditoriale.

Le système doit permettre :

- la gestion des utilisateurs avec différents niveaux de droits,
- la création et la publication de contenus structurés,
- l'interaction entre utilisateurs via des commentaires,
- l'organisation des contenus à l'aide de catégories.

Ces besoins impliquent la gestion de relations complexes entre entités, tout en conservant une certaine flexibilité dans la structure des données, ce qui justifie l'utilisation d'une base de données NoSQL orientée documents.

3.3 Analyse de la modélisation des données

La modélisation repose sur quatre entités principales correspondant aux collections MongoDB. Bien que MongoDB soit une base NoSQL, une modélisation conceptuelle de type UML a été réalisée afin de structurer la conception.

Le diagramme de classes permet d'identifier :

- les attributs fondamentaux de chaque entité,
- les relations logiques entre les collections,
- les cardinalités associées.

Les relations entre entités sont implémentées à l'aide de références (*ObjectId*) plutôt que par de l'imbrication systématique, afin de limiter la redondance des données et de faciliter les opérations de mise à jour.

4. Schéma de base de données

4.1 Users

La collection *Users* stocke les informations relatives aux utilisateurs et permet la gestion des rôles.

Champ	Type	Contraintes
_id	ObjectId	PK
username	String	unique, required
email	String	unique, required
password	String	required
firstName	String	required
lastName	String	required
role	String	enum(user, admin)
isActive	Boolean	default: true
createdAt	Date	auto
updatedAt	Date	auto

4.2 Categories

Les catégories permettent de structurer et classer les contenus publiés.

_id	ObjectId	PK
name	String	unique, required
slug	String	unique, required
color	String	default: #000000
createdAt	Date	auto
updatedAt	Date	auto

4.3 Posts

Les posts constituent le cœur fonctionnel de l'application.

_id	ObjectId	PK
title	String	required, regex
content	String	required
excerpt	String	auto
author	ObjectId	FK → Users
category	ObjectId	FK → Categories
tags	[String]	max 10
status	String	enum(draft, published, archived)
viewCount	Number	default: 0
likes	[ObjectId]	FK → Users
publishedAt	Date	auto if published
featured	Boolean	default: false
createdAt	Date	auto
updatedAt	Date	auto

4.4 Comments

Les commentaires permettent l'interaction entre utilisateurs.

_id	ObjectId	PK
content	String	required, anti-spam
author	ObjectId	FK → Users
post	ObjectId	FK → Posts
parentComment	ObjectId	FK → Comments
likes	[ObjectId]	FK → Users
isEdited	Boolean	default: false
isDeleted	Boolean	default: false
createdAt	Date	auto
updatedAt	Date	auto

5. Routes API

Les routes exposées respectent les conventions REST.

5.1 Utilisateurs

Création, consultation, mise à jour, désactivation et suppression transactionnelle.

- POST /api/users
- GET /api/users
- GET /api/users/:id
- PUT /api/users/:id
- DELETE /api/users/:id
- PATCH /api/users/:id/toggle-active
- GET /api/users/:id/stats
- DELETE /api/users/:id/transaction

5.2 Posts

Gestion complète des publications avec contrôle du statut.

- POST /api/posts
- GET /api/posts
- GET /api/posts/:id
- PUT /api/posts/:id
- DELETE /api/posts/:id

5.3 Commentaires

Ajout, consultation et suppression logique des commentaires.

5.4 Comments

- POST /api/comments
- GET /api/comments
- DELETE /api/comments/:id

5.5 Catégories

CRUD complet pour la classification des contenus.

6. Transactions MongoDB

La suppression d'un utilisateur repose sur une transaction MongoDB afin de garantir la cohérence des données. Les posts associés sont réassignés à un utilisateur technique *deleted*.

7. Seed Script

Un script d'initialisation permet de peupler automatiquement la base de données avec des données de test cohérentes.

- vide la base,
- crée des catégories,
- crée des utilisateurs,
- crée des posts,
- crée des commentaires.

8. Monitoring

Un middleware personnalisé mesure le temps d'exécution des requêtes Mongoose, facilitant l'analyse des performances et la détection de goulots d'étranglement.

9. Difficultés rencontrées

Les principales difficultés ont concerné l'utilisation des middlewares Mongoose, la validation stricte des schémas et la gestion des transactions.

10. Améliorations possibles

Plusieurs axes d'amélioration peuvent être envisagés :

- mise en place d'une authentification JWT,
- ajout de tests automatisés,
- documentation Swagger,
- intégration d'un cache Redis.

11. Conclusion

Ce projet a permis de concevoir une API REST complète reposant sur une architecture moderne et professionnelle. Il a consolidé les compétences en modélisation NoSQL, en développement backend et en conception logicielle.

Les choix techniques effectués garantissent la robustesse et l'évolutivité du système, tout en constituant une base solide pour une exploitation dans un contexte professionnel réel.