

Rapport Technique — Application MERN

BlogCollaboratif

1. Contexte & Objectif

1.1 Problématique

Les plateformes de publication de contenu nécessitent :

- ✓ un espace sécurisé permettant la publication d'articles
- ✓ une gestion des utilisateurs et rôles
- ✓ un système de catégorisation
- ✓ un système de commentaires avec modération
- ✓ un backend structuré avec API REST
- ✓ un frontend moderne

De nombreuses solutions existantes sont soit trop complexes, soit non adaptées à un usage éducatif et scalable.

1.2 Solution proposée

Ce projet MERN implémente :

- ✓ gestion de comptes utilisateurs
- ✓ rôles user, author, admin
- ✓ création/publication de posts
- ✓ catégories + tags
- ✓ profil utilisateur (1–1)
- ✓ commentaires + modération
- ✓ sécurité JWT + middleware
- ✓ admin dashboard

1.3 Intérêt pédagogique

Ce projet démontre l'usage complet de la stack MERN, incluant :

- architecture MVC backend
- API REST avec rôles
- relations 1–1, 1–N, N–N
- modération + workflows
- Auth en JWT
- SPA React moderne
- CRUD complet

2. Architecture Générale

Le projet suit une architecture **client–serveur** :

Couche	Technologie
Frontend	React + React Router + Axios
Backend	Node.js + Express
Base de données	MongoDB + Mongoose
Authentification	JWT
Stockage session	localStorage

3. Architecture Backend (MVC)

Le backend utilise **Express + Mongoose** avec séparation :

Routes → Controllers → Services (optionnel) → Models → MongoDB

3.1 Modèles (Mongoose)

5 modèles principaux :

Modèle	Relation	Description
User	1–1 avec Profile / 1–N Posts / 1–N Comments	Auth, Role, Email
Profile	1–1 User	Bio, Avatar
Post	N–N Tags / 1–N Comments / 1–1 Category	Titre, contenu, statut
Category	1–N Posts	Classification principale
Tag	N–N Posts	Mots-clés
Comment	1–1 Post / 1–1 User	Texte + modération

3.2 Résumé des relations

Type	Entités
------	---------

- 1–1 User ↔ Profile
- 1–N User → Posts
- 1–N Post → Comments
- 1–N Category → Posts
- N–N Tags ↔ Posts

3.3 Rôles & Permissions

Trois rôles :

Rôle	Permissions
user	lire + commenter
author	écrire/éditer ses posts
admin	gestion totale + modération

3.4 Authentification

Basée sur JWT :

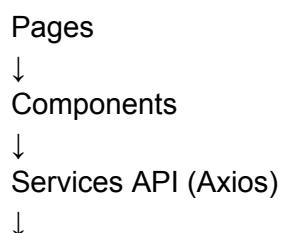
- ✓ bcryptjs → hash mot de passe
- ✓ jsonwebtoken → token généré lors du login
- ✓ middleware protect
- ✓ middleware authorize(...roles)

Flux :

Login → JWT via header Authorization: Bearer token → decode → req.user

4. Architecture Frontend

4.1 Organisation des couches



Context Auth

4.2 Routing (React Router v6)

Routes :

- ✓ Publiques : accueil, login, register, posts list
- ✓ User : commenter
- ✓ Author : /author/posts/*
- ✓ Admin : /users, /categories, /tags, /posts

4.3 Gestion Auth

AuthContext gère :

- ✓ informations utilisateur
 - ✓ JWT
 - ✓ auto-persistence via localStorage
 - ✓ restriction routes
-

5. Fonctionnalités

5.1 Users & Profile

- ✓ Profil utilisateur (avatar, bio)
- ✓ editable avec UI “Lecture seule → Modifier → Enregistrer”

5.2 Gestion Posts

- ✓ CRUD auteur
- ✓ workflow DRAFT → PUBLISHED
- ✓ affichage public seulement des PUBLISHED

5.3 Catégories & Tags

- ✓ Category: 1–N
- ✓ Tag: N–N
- ✓ admin CRUD complet

5.4 Commentaires & Modération

- ✓ user peut commenter
- ✓ admin peut hide commentaire
- ✓ affichage :

visible → affiché

hidden → « ce commentaire a été masqué »

5.5 Dashboard Admin

- ✓ gestion utilisateurs
 - ✓ gestion contenu
 - ✓ modération
-

6. Sécurité

Implémenté :

- ✓ JWT + expiration
 - ✓ hash bcrypt
 - ✓ filtrage par rôle
 - ✓ filtrage accès sur relations (ex: auteur ne modifie pas post d'un autre)
 - ✓ CORS
-

7. Choix technologiques

Composant	Technologie
DB	MongoDB
API	Node + Express
Auth	JWT
UI	React + Bootstrap
HTTP	Axios
Hash	bryptjs

8. Conclusion

Ce projet démontre :

- ✓ une architecture scalable

- ✓ une gestion complexe de rôles et relations
 - ✓ une modération fonctionnelle
 - ✓ une interface propre et moderne
-

Auteur : Mohamed Aziz Chebil

Date : Janvier 2025