

Rapport de projet

Conception et développement d'une application blog MERN

Authentification JWT, gestion des articles, commentaires, tags
et profils

Réalisé par : Jassem Boughattas

Année Universitaire : 2025–2026

Remerciements

Nous adressons nos remerciements à toutes les personnes ayant contribué au bon déroulement de ce projet. Nous remercions en particulier nos enseignants pour leurs conseils, ainsi que l'ensemble des personnes ayant facilité l'accès aux ressources techniques nécessaires. Enfin, nous remercions nos proches pour leur soutien durant cette période.

Table des matières

Remerciements	1
Introduction générale	6
1 Cadre général du projet	7
1.1 Contexte	7
1.2 Problématique	7
1.3 Objectifs	8
1.3.1 Objectifs fonctionnels	8
1.3.2 Objectifs techniques	8
1.4 Méthodologie de travail	8
1.5 Périmètre retenu	9
2 Analyse et spécification des besoins	10
2.1 Acteurs du système	10
2.1.1 Utilisateur (User)	10
2.1.2 Administrateur (Admin)	10
2.2 Besoins fonctionnels	10
2.2.1 Authentification	10
2.2.2 Profil utilisateur	11
2.2.3 Articles (Posts)	11
2.2.4 Commentaires	11
2.2.5 Tags	11
2.2.6 Recherche	11
2.3 Besoins non fonctionnels	11
2.4 Diagramme de cas d'utilisation (UML) — vue générale	12
2.5 Cas d'utilisation (description textuelle)	12
2.5.1 Connexion	12
2.5.2 Mise à jour du profil	13
2.5.3 Recherche d'articles	13

3	Conception	15
3.1	Architecture générale	15
3.2	Organisation du backend	15
3.3	Modèle de données	15
3.3.1	Entités	15
3.3.2	Relations	16
3.4	Diagramme de classes (UML)	16
3.5	Sécurité	17
3.5.1	JWT	17
3.5.2	Bcrypt	17
3.5.3	Vérification des permissions (RBAC)	17
3.6	Conception de l'API (exemples d'endpoints)	17
4	Réalisation et tests	18
4.1	Technologies et outils	18
4.1.1	Backend	18
4.1.2	Frontend	18
4.1.3	Outils de développement	18
4.2	Réalisation backend (approche)	18
4.2.1	Organisation	18
4.2.2	Exemple d'en-tête Authorization	19
4.3	Réalisation frontend (approche)	19
4.3.1	Navigation et routes protégées	19
4.3.2	Appels API	19
4.4	Plan de tests (manuel)	20
4.5	Résultats	20
4.6	Difficultés et solutions	20
4.7	Perspectives	21
4.8	Présentation des interfaces	21
4.8.1	Interface d'authentification	21
4.8.2	Liste des articles	22
4.8.3	Détail d'un article et commentaires	23
4.8.4	Profil utilisateur (nouveau)	24
	Conclusion générale	26

Table des figures

2.1	Diagramme de cas d'utilisation général (User/Admin)	12
3.1	Diagramme de classes UML	16
4.1	Interface de connexion	22
4.2	Liste des articles	23
4.3	Détail d'un article et commentaires	24
4.4	Page profil utilisateur	25

Liste des tableaux

4.1 Scénarios de tests manuels	20
--	----

Introduction générale

Les applications de publication de contenu (blogs, sites d'actualités, forums) constituent un cas d'usage important du web moderne. Elles permettent de mettre en pratique plusieurs concepts essentiels du génie logiciel : conception d'API REST, gestion d'utilisateurs, sécurité, communication client/serveur et persistance des données.

Ce projet consiste à développer une application blog full-stack basée sur la stack MERN (MongoDB, Express.js, React et Node.js). L'objectif est de livrer une solution fonctionnelle et structurée mettant en avant : (i) une authentification sécurisée (JWT) avec gestion des rôles et (ii) des fonctionnalités de gestion de contenu (articles, commentaires, tags), ainsi que (iii) la gestion des profils utilisateurs et une fonction de recherche.

Le présent rapport est organisé comme suit : le chapitre 1 décrit le cadre général du projet (contexte, problématique, objectifs, méthodologie) ; le chapitre 2 détaille l'analyse des besoins, les acteurs et un diagramme de cas d'utilisation UML ; le chapitre 3 présente la conception (architecture, modèle de données, sécurité, diagramme de classes, conception API) ; le chapitre 4 décrit la réalisation et les tests, puis illustre l'interface par quelques captures d'écran ; enfin, une conclusion générale synthétise les résultats et les perspectives.

Chapitre 1

Cadre général du projet

1.1 Contexte

Le développement d'applications web modernes nécessite de concilier rapidité de livraison, qualité, sécurité et évolutivité. Les stacks JavaScript full-stack, notamment MERN (MongoDB, Express.js, React, Node.js), se sont imposées comme une solution populaire pour construire des applications complètes grâce à une forte modularité et une séparation claire entre l'interface utilisateur et l'API.

Dans ce projet, nous développons une application blog permettant à des utilisateurs authentifiés de publier des articles, d'interagir via des commentaires et d'organiser le contenu grâce à des tags. Une gestion de profil utilisateur est également intégrée afin de centraliser des informations liées au compte et améliorer l'expérience utilisateur.

1.2 Problématique

Une application de blog, bien que simple en apparence, soulève plusieurs problématiques :

- **Sécurité** : protection des routes sensibles, gestion du token côté client, hachage des mots de passe, contrôle d'accès par rôles.
- **Cohérence des données** : relations entre utilisateurs, profils, articles, commentaires et tags.
- **Expérience utilisateur** : navigation fluide, pages protégées, gestion des erreurs, recherche de contenu.
- **Maintenabilité** : structuration du code et séparation des responsabilités.

1.3 Objectifs

1.3.1 Objectifs fonctionnels

1. Permettre l'inscription et la connexion des utilisateurs.
2. Gérer le profil utilisateur (consultation et mise à jour).
3. Gérer les articles : création, consultation, modification et suppression.
4. Ajouter un système de commentaires sur les articles.
5. Organiser les articles via des tags.
6. Rechercher des articles selon des critères (ex. titre, tags ou contenu, selon l'implémentation).
7. Différencier les accès selon le rôle (User/Admin).

1.3.2 Objectifs techniques

1. Développer une **API REST** avec Node.js + Express.
2. Utiliser **MongoDB** comme base de données (Mongoose).
3. Implémenter une authentification **JWT** et un hachage sécurisé avec **bcrypt**.
4. Mettre en place un contrôle d'accès par rôles (**RBAC** : User/Admin) et la protection des routes.
5. Développer le frontend avec **React (Vite)**, React Router et Axios.
6. Mettre en place une organisation claire : routes, contrôleurs, modèles, middlewares.
7. Ajouter une fonctionnalité de **recherche** via un endpoint dédié côté backend.
8. Assurer la cohérence entre le frontend et l'API via une convention de réponses (succès/erreurs).

1.4 Méthodologie de travail

Le projet a été conduit de manière itérative :

- découpage en tâches courtes et priorisées ;
- implémentation progressive des fonctionnalités principales (auth, posts, comments, tags, profiles, search) ;
- validation via tests manuels et correction des anomalies ;
- utilisation de Git/GitHub pour le contrôle de version.

1.5 Périmètre retenu

Le projet se concentre sur :

- authentification (register/login) avec JWT ;
- gestion des rôles (User/Admin) et routes protégées ;
- gestion des profils ;
- gestion des articles (CRUD) ;
- commentaires liés aux articles ;
- tags associés aux articles ;
- recherche (endpoint de recherche) ;
- conteneurisation (support Docker) pour faciliter l'exécution locale.

Chapitre 2

Analyse et spécification des besoins

2.1 Acteurs du système

Dans le cadre de l'analyse UML, nous identifions deux acteurs principaux.

2.1.1 Utilisateur (User)

L'utilisateur représente un membre de la plateforme capable de :

- s'inscrire et se connecter ;
- consulter et rechercher des articles ;
- créer, modifier et supprimer ses propres articles ;
- commenter des articles ;
- consulter les tags ;
- consulter et mettre à jour son profil.

2.1.2 Administrateur (Admin)

L'administrateur dispose de droits de gestion et de modération :

- accéder à des sections réservées ;
- gérer les utilisateurs (selon l'implémentation) ;
- supprimer/modérer des articles et/ou commentaires si nécessaire ;
- consulter certaines informations de gestion (selon le périmètre retenu).

2.2 Besoins fonctionnels

2.2.1 Authentification

- Inscription (username, email, mot de passe).

- Connexion et génération d'un JWT.
- Déconnexion (suppression/invalidations côté client selon l'approche).
- Protection des routes nécessitant une authentification.

2.2.2 Profil utilisateur

- Consultation du profil.
- Mise à jour des informations du profil.

2.2.3 Articles (Posts)

- Créer un article (titre, contenu, tags).
- Consulter la liste des articles.
- Consulter le détail d'un article.
- Modifier un article (réservé à l'auteur, ou Admin).
- Supprimer un article (réservé à l'auteur, ou Admin).

2.2.4 Commentaires

- Ajouter un commentaire sur un article.
- Consulter la liste des commentaires d'un article.
- Supprimer un commentaire (auteur, ou Admin si modération).

2.2.5 Tags

- Lister les tags.
- Associer des tags à un article.

2.2.6 Recherche

- Rechercher des articles à partir d'un mot-clé (par exemple dans le titre ou le contenu, selon l'implémentation).

2.3 Besoins non fonctionnels

- **Sécurité** : mots de passe hachés, JWT, routes protégées, autorisation par rôles.
- **Maintenabilité** : code structuré, validation des entrées, gestion centralisée des erreurs.

- **Ergonomie** : interface simple, feedback utilisateur, messages d'erreur clairs.
- **Performance** : pagination/limitation côté API (si implémentée), requêtes optimisées.

2.4 Diagramme de cas d'utilisation (UML) — vue générale

La figure 2.1 présente le diagramme de cas d'utilisation général de l'application (acteurs : User et Admin).

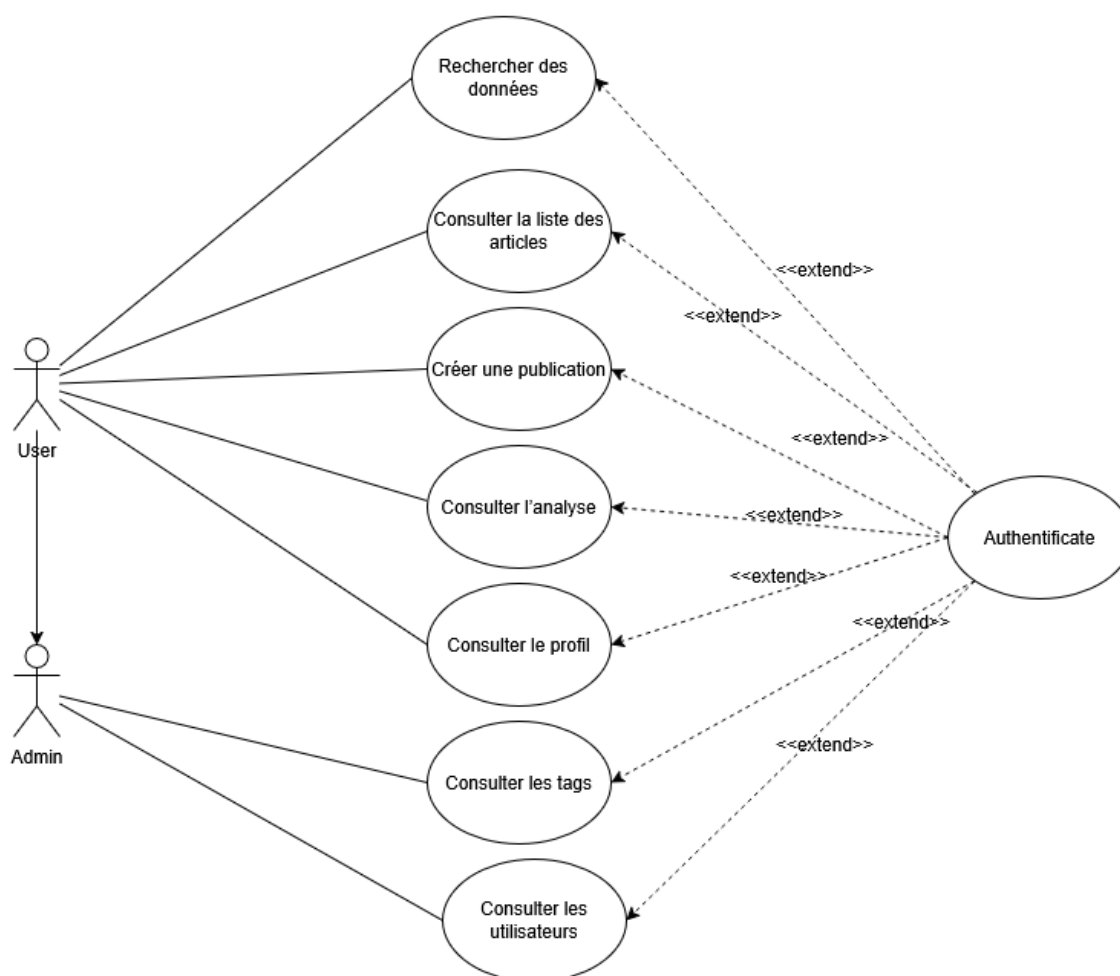


FIGURE 2.1 – Diagramme de cas d'utilisation général (User/Admin)

2.5 Cas d'utilisation (description textuelle)

2.5.1 Connexion

Élément	Description
Nom	Connexion utilisateur
Acteur	User / Admin
Préconditions	Compte existant
Scénario nominal	<ol style="list-style-type: none">1. Accès à la page de connexion.2. Saisie email et mot de passe.3. Vérification des identifiants.4. Génération et retour d'un JWT.5. Accès aux pages protégées.
Postconditions	Utilisateur authentifié

2.5.2 Mise à jour du profil

Élément	Description
Nom	Mettre à jour le profil
Acteur	User
Préconditions	JWT valide
Scénario nominal	<ol style="list-style-type: none">1. Accès à la page profil.2. Modification des informations autorisées.3. Envoi d'une requête PUT/PATCH à l'API avec le token.4. Validation côté backend puis mise à jour en base.5. Retour du profil mis à jour.
Postconditions	Profil mis à jour

2.5.3 Recherche d'articles

Élément	Description
Nom	Rechercher des articles
Acteur	User
Préconditions	Aucune (ou JWT, selon la configuration)
Scénario nominal	<ol style="list-style-type: none">1. Saisie d'un mot-clé dans la barre de recherche.2. Envoi de la requête de recherche à l'API.3. L'API retourne les articles correspondants.4. Affichage des résultats dans l'interface.

Postconditions	Résultats affichés
----------------	--------------------

Chapitre 3

Conception

3.1 Architecture générale

L'application suit une architecture **client/serveur** :

- **Frontend (React)** : interface, pages, routage, gestion d'état côté client et appels API.
- **Backend (Express)** : API REST, sécurité (JWT/RBAC), validation, logique métier.
- **Base de données (MongoDB)** : stockage des utilisateurs, profils, articles, commentaires et tags.

3.2 Organisation du backend

Le backend est organisé autour de contrôleurs dédiés (ex. authentification, utilisateurs, profils, articles, commentaires, tags, recherche). Cette structuration facilite la maintenabilité, la lisibilité et les tests.

3.3 Modèle de données

3.3.1 Entités

- **User** : username, email, mot de passe haché, rôle (user/admin).
- **Profile** : informations complémentaires liées à un utilisateur (relation 1 :1).
- **Post** : titre, contenu, auteur, tags, dates.
- **Comment** : contenu, auteur, post, date.
- **Tag** : nom/libellé, relation avec les posts.

3.3.2 Relations

- User (1) — (1) Profile
- User (1) — (N) Post
- Post (1) — (N) Comment
- User (1) — (N) Comment
- Post (N) — (N) Tag

3.4 Diagramme de classes (UML)

La figure 3.1 illustre le diagramme de classes représentant les principales entités de l'application et leurs relations.

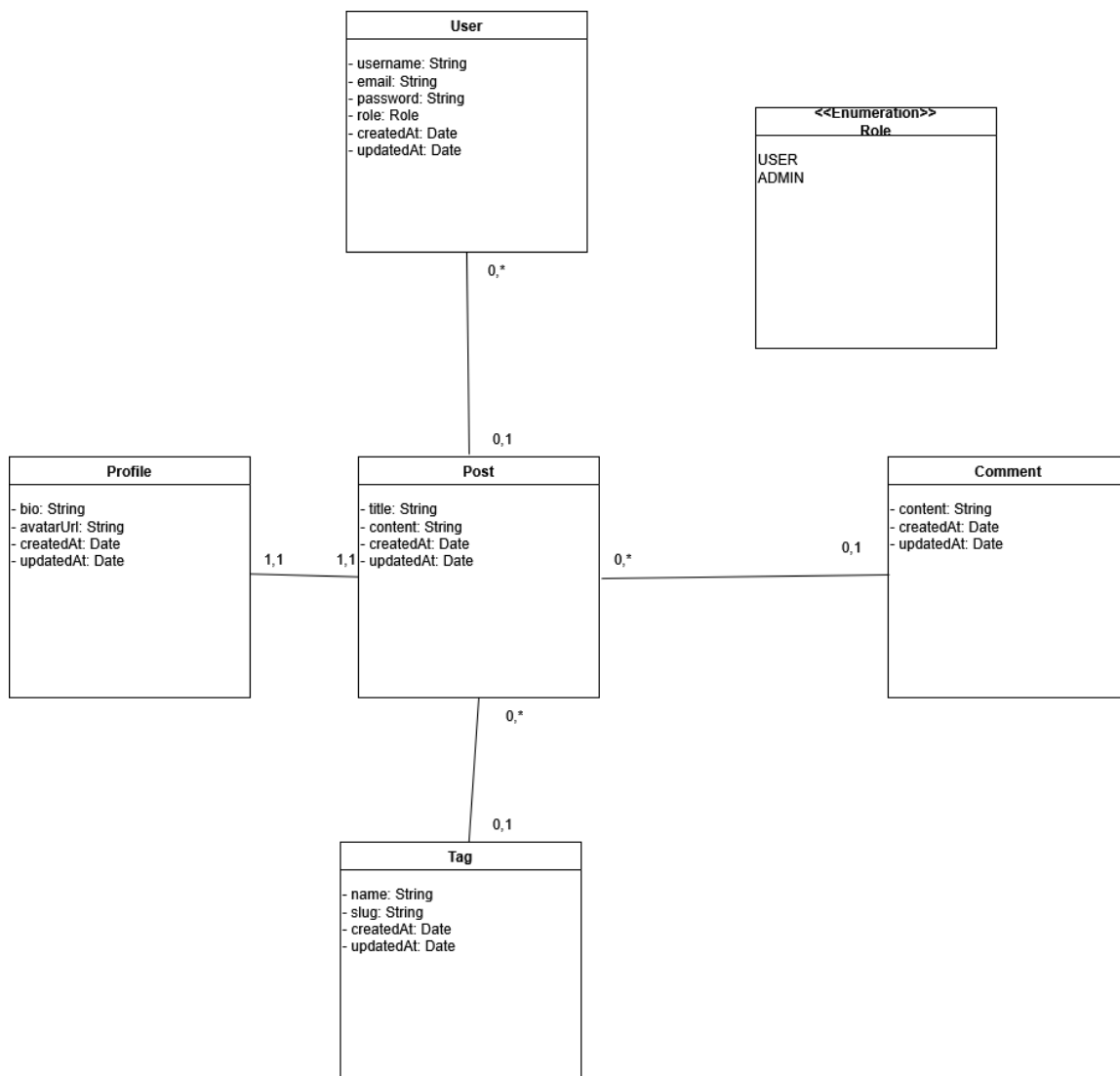


FIGURE 3.1 – Diagramme de classes UML

3.5 Sécurité

3.5.1 JWT

Le backend génère un token JWT après connexion. Le frontend envoie ce token sur les routes protégées via l'en-tête Authorization.

3.5.2 Bcrypt

Les mots de passe sont hachés avec bcrypt afin d'éviter le stockage en clair.

3.5.3 Vérification des permissions (RBAC)

- Les routes de création nécessitent un utilisateur authentifié.
- La modification/suppression d'un article est réservée à son auteur (ou Admin).
- Les actions de modération sont réservées à l'Admin.

3.6 Conception de l'API (exemples d'endpoints)

- `/api/auth/*` : register, login
- `/api/users/*` : gestion utilisateur (selon besoin)
- `/api/profile/*` : profil
- `/api/posts/*` : CRUD articles
- `/api/posts/:postId/comments/*` : commentaires
- `/api/tags/*` : tags
- `/api/search/*` : recherche

Chapitre 4

Réalisation et tests

4.1 Technologies et outils

4.1.1 Backend

- Node.js, Express.js
- MongoDB (Mongoose)
- JWT, bcrypt
- Middlewares (auth, autorisation, gestion d'erreurs)

4.1.2 Frontend

- React 18, Vite
- React Router
- Axios
- Tailwind CSS

4.1.3 Outils de développement

- Visual Studio Code
- Git/GitHub
- Docker (support conteneurisation)

4.2 Réalisation backend (approche)

4.2.1 Organisation

Le backend est structuré en couches :

- **Routes** : définition des endpoints.
- **Contrôleurs** : traitement des requêtes et logique métier.
- **Modèles** : schémas Mongoose.
- **Middlewares** : authentification JWT, autorisation RBAC, gestion d'erreurs.

4.2.2 Exemple d'en-tête Authorization

```
Authorization: Bearer <JWT_TOKEN>
```

4.3 Réalisation frontend (approche)

4.3.1 Navigation et routes protégées

Le frontend utilise React Router pour organiser les pages. Les routes sensibles (création/édition) sont protégées via un mécanisme vérifiant la présence d'un token. Certaines sections sont accessibles uniquement à l'administrateur.

4.3.2 Appels API

Axios est utilisé pour communiquer avec l'API REST. Une approche classique consiste à configurer une instance Axios afin d'ajouter automatiquement le token aux requêtes authentifiées.

4.4 Plan de tests (manuel)

ID	Scénario	Résultat attendu
T1	Inscription (données valides)	Compte créé
T2	Connexion (valide)	JWT reçu
T3	Accès page profil sans token	Accès refusé / redirection
T4	Mise à jour du profil	Profil mis à jour
T5	Création d'un article	Article visible
T6	Modification article par auteur	Article mis à jour
T7	Suppression article par non-auteur	Refus (403)
T8	Ajout commentaire sur article	Commentaire affiché
T9	Recherche d'articles	Résultats affichés

TABLE 4.1 – Scénarios de tests manuels

4.5 Résultats

Les fonctionnalités principales ont été implémentées :

- Authentification JWT.
- Gestion des rôles et des accès (User/Admin).
- CRUD des articles avec restrictions.
- Commentaires associés à un article.
- Tags et association aux articles.
- Gestion de profil utilisateur.
- Recherche d'articles via un endpoint dédié.

4.6 Difficultés et solutions

- **Gestion du token côté client** : mise en place d'une instance Axios ou d'un interceptor pour inclure le token.
- **Permissions auteur/Admin** : vérification côté backend (auteur vs rôle) avant mise à jour/suppression.

- **Recherche** : définition de critères simples (mot-clé) et normalisation des paramètres.

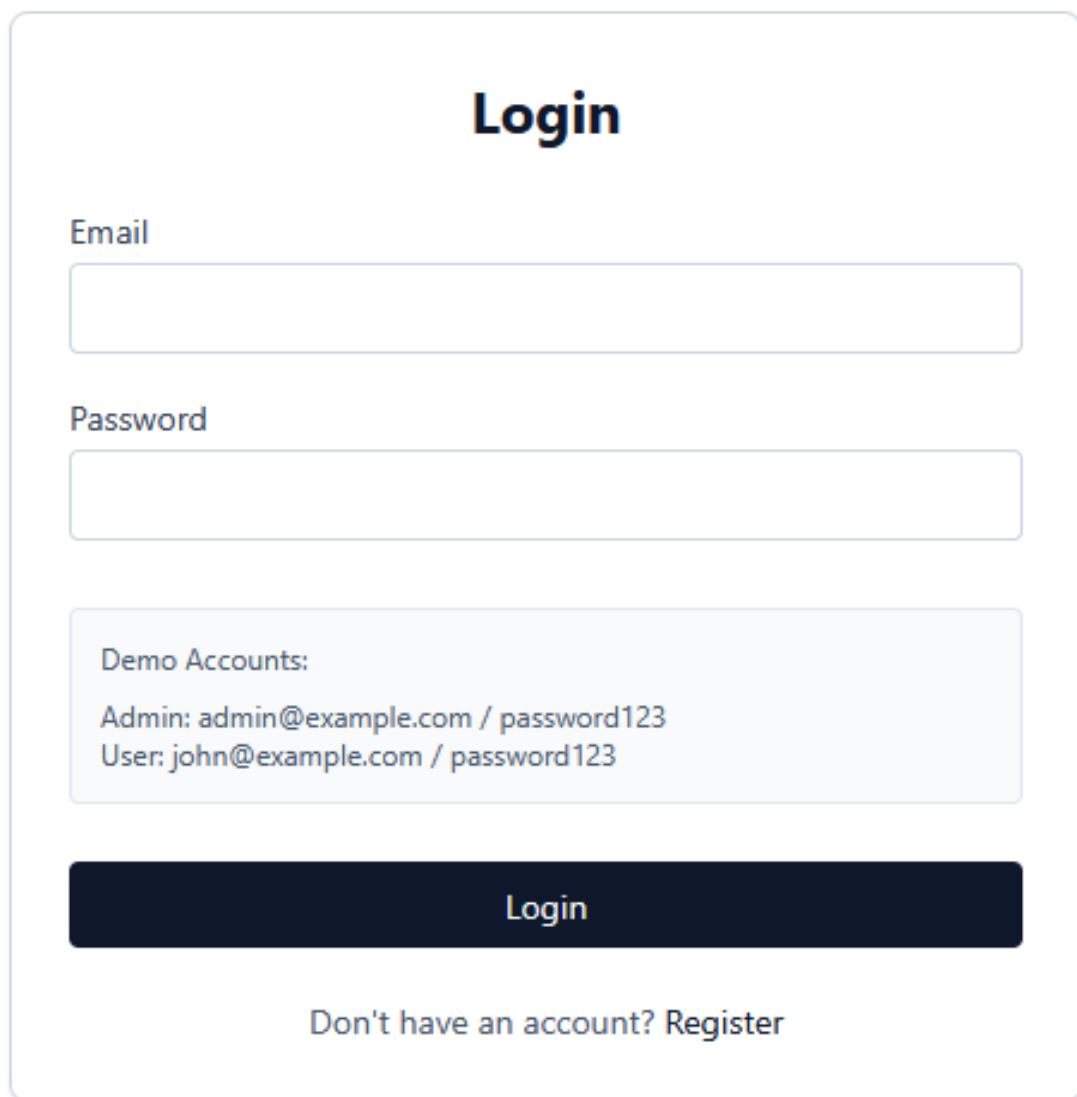
4.7 Perspectives

- Ajouter pagination, filtres avancés et tri.
- Ajouter tests automatisés (backend et frontend).
- Améliorer l'UX (éditeur riche, upload images, messages plus détaillés).
- Renforcer la sécurité (limitation de débit, règles CORS affinées, logs).

4.8 Présentation des interfaces

4.8.1 Interface d'authentification

La figure [4.1](#) présente l'interface de connexion.

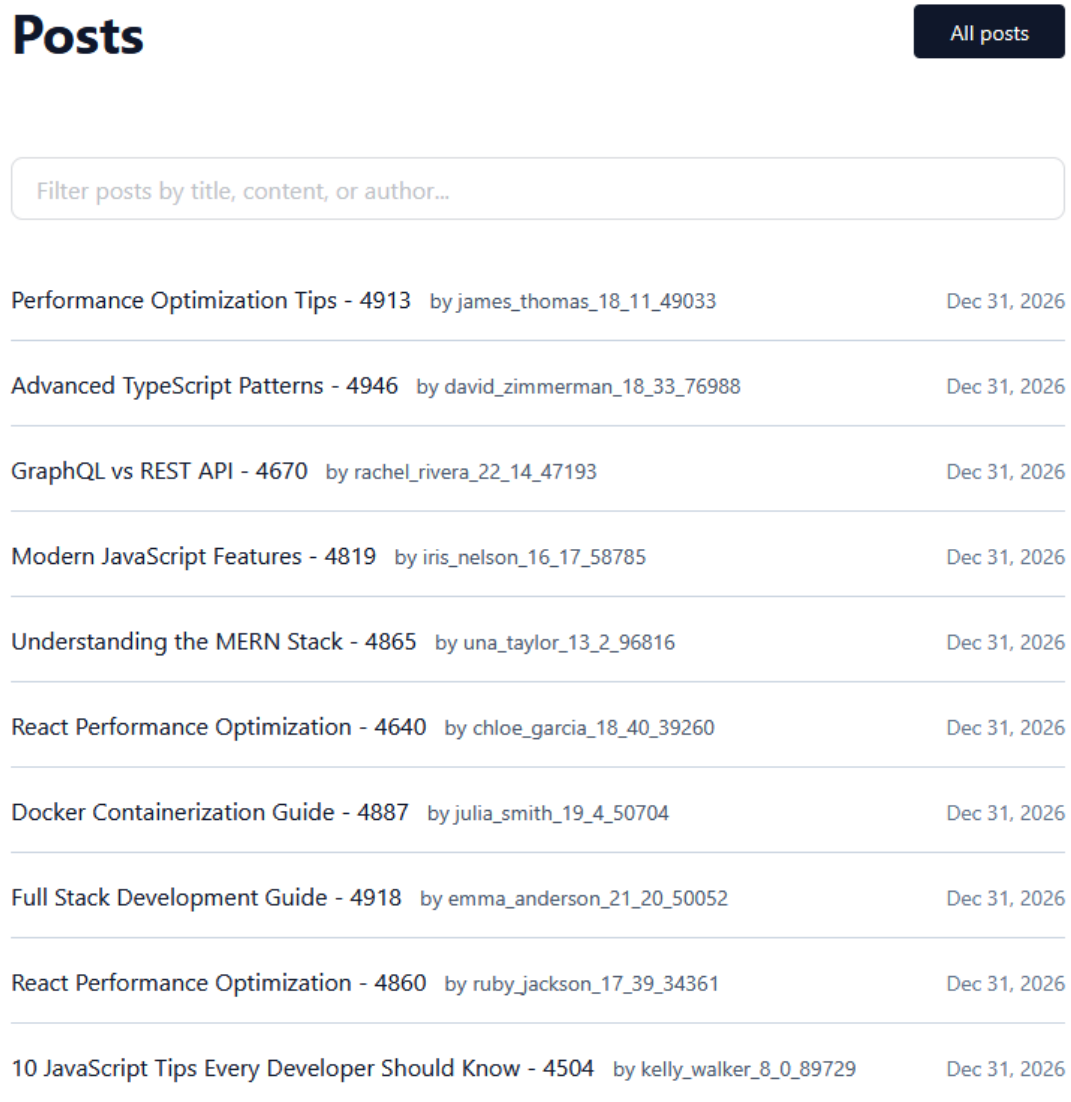


The image shows a login interface within a light blue rounded rectangle. At the top, the word "Login" is centered in a large, bold, black font. Below it, the label "Email" is positioned to the left of a white rectangular input field. Further down, the label "Password" is to the left of another white rectangular input field. Below these fields is a light blue shaded box containing the text "Demo Accounts:" followed by two lines of text: "Admin: admin@example.com / password123" and "User: john@example.com / password123". At the bottom of the interface is a dark blue rectangular button with the word "Login" in white. Below the button, the text "Don't have an account? Register" is displayed, with "Register" being a blue hyperlink.

FIGURE 4.1 – Interface de connexion

4.8.2 Liste des articles

La figure 4.2 présente la page listant les articles.



Posts			All posts
Filter posts by title, content, or author...			
Performance Optimization Tips - 4913	by james_thomas_18_11_49033		Dec 31, 2026
Advanced TypeScript Patterns - 4946	by david_zimmerman_18_33_76988		Dec 31, 2026
GraphQL vs REST API - 4670	by rachel_rivera_22_14_47193		Dec 31, 2026
Modern JavaScript Features - 4819	by iris_nelson_16_17_58785		Dec 31, 2026
Understanding the MERN Stack - 4865	by una_taylor_13_2_96816		Dec 31, 2026
React Performance Optimization - 4640	by chloe_garcia_18_40_39260		Dec 31, 2026
Docker Containerization Guide - 4887	by julia_smith_19_4_50704		Dec 31, 2026
Full Stack Development Guide - 4918	by emma_anderson_21_20_50052		Dec 31, 2026
React Performance Optimization - 4860	by ruby_jackson_17_39_34361		Dec 31, 2026
10 JavaScript Tips Every Developer Should Know - 4504	by kelly_walker_8_0_89729		Dec 31, 2026

FIGURE 4.2 – Liste des articles

4.8.3 Détail d'un article et commentaires

La figure 4.3 présente le détail d'un article et l'espace commentaires.

10 JavaScript Tips Every Developer Should Know - 1084

In-depth detailed guide: Understanding the MERN Stack. This article provides extensive examples, best practices, and real-world use cases. We'll cover multiple approaches, common pitfalls, and advanced techniques. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt. Ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat.

By liam_anderson_12_10_48391

Tags: Frontend, Node.js, Testing

Comments

Please [log in](#) to add a comment.

Could you expand on section Y? I'd love to see more examples. — una_taylor_13_2_96816

This saved me hours of research. The community here is amazing! — frank_jackson_17_18_46114

FIGURE 4.3 – Détail d'un article et commentaires

4.8.4 Profil utilisateur (nouveau)

La figure 4.4 présente la page profil.

My Profile

Username: admin

Email: admin@example.com

Role: admin



Bio

Early adopter exploring new platforms. admin here.

Avatar URL

<https://i.pravatar.cc/150?img=1>

Edit Profile

FIGURE 4.4 – Page profil utilisateur

Conclusion générale

Ce projet a permis de concevoir et développer une application blog MERN mettant en œuvre une API REST sécurisée et une interface web moderne. La mise en place de JWT, du hachage des mots de passe et du contrôle d'accès par rôles contribue à la sécurité de l'application. L'ajout de la gestion de profil et d'un module de recherche améliore l'expérience utilisateur.

Le résultat obtenu constitue une base fonctionnelle et évolutive, pouvant être enrichie par des fonctionnalités supplémentaires telles que la pagination, l'amélioration de l'interface et l'automatisation des tests.