

FORMATION FULLSTACK DEVELOPER

Projet de Fin de Formation

Plateforme de Réseau Social Web

Documentation Technique Complète

Réalisé par :
BOUZZITE Mohammed
ISFI Abderrahmane
MAKEA Lamya
TABCHI Achraf

Encadré par :
SELMOUNI Zineb

Date de soutenance : 10 Février 2025

REMERCIEMENTS

Nous souhaitons exprimer notre sincère gratitude à toutes les personnes et institutions ayant contribué à la réussite de ce projet et de notre formation.

Nous remercions vivement **Mme SELMOUNI Zineb**, notre formatrice, pour son encadrement, sa disponibilité et la qualité de ses conseils.

Nos remerciements s'adressent également à l'ensemble des **professeurs et intervenants** pour leur expertise et leur accompagnement tout au long du parcours.

Nous remercions tout particulièrement **Ynov Campus Maroc** et **Jobintech** pour cette formation de qualité et l'opportunité de mener ce projet.

Nous exprimons enfin notre reconnaissance aux **membres du jury** pour l'évaluation de notre travail, ainsi qu'à nos collègues et à nos familles pour leur soutien.

Table des matières

1 Introduction	5
1.1 Contexte du projet	5
1.2 Problématique	5
1.3 Objectifs du projet	5
1.3.1 Objectifs fonctionnels	5
1.3.2 Objectifs techniques	6
1.4 Utilisateurs cibles	6
1.5 Choix technologiques	6
2 Analyse des besoins et cahier des charges	7
2.1 Besoins fonctionnels	7
2.1.1 Gestion des utilisateurs	7
2.1.2 Création et gestion de contenu	7
2.1.3 Interactions sociales	8
2.1.4 Fil d'actualité, recherche et notifications	8
2.2 Priorisation des exigences (Méthode MoSCoW)	9
2.3 Exigences non fonctionnelles	9
2.3.1 Performance	9
2.3.2 Sécurité	9
2.3.3 Scalabilité	9
3 Gestion de projet	10
3.1 Backlog et User Stories	10
3.2 Suivi de Projet avec Jira	11
3.3 Planification et Suivi	12
3.4 Livrables	14
4 Architecture globale du système	15
4.1 Vue d'ensemble	15
4.2 Couche fonctionnelles de l'application	15
4.3 Arborescence globale du projet	15
4.4 Organisation du backend	16
4.5 Organisation du frontend	17

4.6	Synthèse architecturale	17
5	Diagrammes UML	18
5.1	Diagramme de Cas d'Utilisation	18
5.2	Diagrammes de Séquence	19
5.3	Diagramme de Classes	21
6	Modélisation des Données	23
6.1	Organisation des collections MongoDB	23
6.1.1	Collection Users	23
6.1.2	Collection Threads	24
6.1.3	Collection Comments	24
6.1.4	Collection Likes	25
6.1.5	Collection Follows	25
6.1.6	Collection Notifications	25
6.2	Relations entre les entités	26
7	API REST	27
7.1	Endpoints principaux	27
8	Sécurité	29
8.1	Authentification basée sur les JSON Web Tokens	29
8.2	Gestion et sécurisation des mots de passe	29
8.3	Protection contre les attaques courantes	30
8.3.1	Prévention des injections NoSQL	30
8.3.2	Protection contre les attaques XSS	30
8.3.3	Limitation du nombre de requêtes	30
8.4	Mesures de sécurité complémentaires	30
9	Déploiement	31
9.1	Architecture de conteneurisation	31
9.2	Orchestration avec Docker Compose	31
9.3	Exécution locale	32
9.4	Déploiement sur Railway	32
9.5	Aperçu du Processus de Déploiement	32
10	Démonstration de l'Application	38

10.1	Authentification	38
10.2	Fil d'Actualité et Modes d'Affichage	38
10.3	Création et Publication de Contenu	39
10.4	Gestion du Profil Utilisateur	40
10.5	Personnalisation et Système de Recommandation	40
10.6	Notifications et Gestion des Interactions	41
10.7	Configuration et Paramètres de Sécurité	42
11	Difficultés Rencontrées et Solutions	43
11.1	Synthèse des défis rencontrés	43
11.2	Défis Techniques	43
11.2.1	Gestion des notifications en temps réel	43
11.2.2	Stockage scalable des médias	44
11.2.3	Contrôle de la confidentialité des contenus	44
11.2.4	Prévention des doublons d'engagement	44
11.2.5	Optimisation des performances de recherche	44
11.3	Défis Organisationnels	45
11.3.1	Synchronisation de l'état entre composants	45
11.3.2	Persistance de l'authentification	45
12	Conclusion	46
12.1	Réalisations et apports techniques	46
12.2	Compétences consolidées	46
12.3	Perspectives d'évolution	46
Annexes		48

1 Introduction

1.1 Contexte du projet

À l'ère du numérique, les réseaux sociaux jouent un rôle essentiel dans la communication et l'interaction entre les utilisateurs, en facilitant le partage de contenu et la création de communautés.

Dans ce cadre, le projet **Social** consiste en la conception d'une plateforme de réseau social web moderne, réalisée dans le cadre de la formation *Full Stack Developer* dispensée par **Jobintech** en partenariat avec **Ynov Campus Maroc**. Ce projet de fin de formation vise à mettre en application les compétences techniques et méthodologiques acquises.

Inspirée des plateformes de micro-publication telles que X (anciennement Twitter), la plateforme **Social** permet aux utilisateurs de publier du contenu, d'interagir entre eux et de participer à des échanges autour de centres d'intérêt communs.

1.2 Problématique

L'évolution des usages numériques a renforcé le besoin de plateformes sociales à la fois accessibles, performantes et sécurisées, capables de supporter de nombreuses interactions tout en protégeant les données des utilisateurs.

Dans ce contexte, le projet **Social** répond à ces enjeux en proposant une plateforme équilibrée, combinant robustesse technique, évolutivité et simplicité d'utilisation.

1.3 Objectifs du projet

L'objectif principal du projet est de concevoir et de développer une plateforme de réseau social web complète, répondant aux usages essentiels d'un produit moderne, tout en s'inscrivant dans une démarche de développement professionnelle.

1.3.1 Objectifs fonctionnels

1. Mettre en place un parcours utilisateur complet incluant l'inscription, la connexion et la gestion de session.
2. Permettre la gestion du profil utilisateur (informations, avatar, bannière et confidentialité).
3. Offrir la création et la gestion de contenu sous forme de threads avec prise en charge des médias.
4. Implémenter les interactions sociales (likes, commentaires, suivi) et la gestion des comptes privés.
5. Proposer des mécanismes de découverte de contenu via le fil d'actualité, la recherche et les recommandations.

6. Informer l'utilisateur des événements importants à travers un système de notifications, incluant le temps réel.

1.3.2 Objectifs techniques

1. Concevoir une API REST robuste et maintenable, structurée par domaines fonctionnels.
2. Assurer la sécurité des échanges et des données à l'aide de mécanismes éprouvés (JWT, hachage, validation).
3. Mettre en place une architecture évolutive reposant sur un backend stateless et une gestion optimisée des données.
4. Garantir des performances maîtrisées grâce à l'indexation, la pagination et le chargement progressif.
5. Livrer une application déployable via une approche DevOps incluant Docker et un déploiement sur Railway.

1.4 Utilisateurs cibles

La plateforme **Social** s'adresse à un public varié, incluant aussi bien des utilisateurs souhaitant partager des contenus personnels que des créateurs de contenu ou des communautés thématiques. Elle vise également les groupes professionnels désireux de disposer d'un espace d'échange structuré favorisant la discussion et la mise en relation.

1.5 Choix technologiques

Afin de répondre aux objectifs du projet, la plateforme **Social** repose sur la stack **MERN**, un ensemble cohérent de technologies JavaScript largement utilisées dans le développement web moderne.

Choix de la stack MERN

La stack MERN a été retenue pour les raisons suivantes :

- utilisation d'un langage unique sur l'ensemble de la chaîne applicative ;
- communication fluide entre le client et le serveur via des API REST ;
- bonnes performances et capacités de montée en charge ;
- forte adoption dans le monde professionnel.

2 Analyse des besoins et cahier des charges

Cette section présente l'analyse des besoins de la plateforme **Social**, réalisée en amont du développement afin d'établir un cadre fonctionnel et technique clair. Cette étape a permis de traduire les attentes exprimées dans le cadre de la formation en exigences concrètes, tout en tenant compte des contraintes réelles liées au développement d'une application web moderne.

L'analyse distingue les besoins fonctionnels, qui définissent les usages et services offerts aux utilisateurs, des exigences non fonctionnelles, qui encadrent la qualité, la performance et la sécurité du système. Afin de maîtriser le périmètre du projet et de prioriser les fonctionnalités, la méthode **MoSCoW** a été adoptée.

2.1 Besoins fonctionnels

Les besoins fonctionnels décrivent l'ensemble des services que la plateforme doit fournir pour répondre aux usages attendus d'un réseau social contemporain.

2.1.1 Gestion des utilisateurs

La gestion des utilisateurs constitue un pilier fondamental de la plateforme. Elle couvre l'ensemble du cycle de vie du compte, depuis l'inscription jusqu'à la gestion avancée du profil, tout en garantissant la sécurité et la confidentialité des données personnelles.

Fonctionnalité	Description
Inscription et authentification	Création de compte et connexion sécurisées à l'aide de JSON Web Tokens (JWT), avec gestion de sessions via access tokens et refresh tokens.
Gestion du profil	Personnalisation du profil utilisateur incluant avatar, biographie et centres d'intérêt.
Confidentialité	Possibilité de définir un compte public ou privé, influençant l'accès aux contenus publiés.
Statistiques sociales	Affichage du nombre de followers et de comptes suivis.

Table 1: Fonctionnalités liées à la gestion des utilisateurs

2.1.2 Création et gestion de contenu

Le cœur fonctionnel de la plateforme repose sur un système de publications appelées *threads*. Ces contenus constituent le point d'entrée principal des interactions entre utilisateurs.

Fonctionnalité	Description
Création de threads	Publication de contenus textuels enrichis par des médias (images ou vidéos).
Gestion des contenus	Possibilité d'éditer, supprimer ou archiver ses propres publications.
Hashtags et mentions	Extraction automatique des hashtags et mentions pour faciliter la recherche et la catégorisation.
Types de publications	Prise en charge des posts originaux, reposts et quotes.

Table 2: Fonctionnalités de création et de gestion de contenu

2.1.3 Interactions sociales

Les interactions sociales visent à encourager l'engagement des utilisateurs et à structurer les échanges au sein de la plateforme.

Interaction	Description
Likes et commentaires	Interactions sur les publications et commentaires, avec mécanisme de prévention des doublons.
Commentaires imbriqués	Réponses hiérarchisées permettant des discussions structurées.
Système de suivi	Fonctionnalités de follow/unfollow avec validation des demandes pour les comptes privés.
Favoris	Mise en favoris (bookmarks) des publications pour consultation ultérieure.

Table 3: Fonctionnalités d'interaction sociale

2.1.4 Fil d'actualité, recherche et notifications

La consultation et la découverte du contenu s'appuient sur plusieurs mécanismes complémentaires.

La plateforme propose différents fils d'actualité : un fil public, un fil basé sur les abonnements de l'utilisateur et un fil recommandé construit à partir de ses centres d'intérêt. Les règles de confidentialité sont systématiquement appliquées afin de garantir le respect des comptes privés.

Un moteur de recherche permet de retrouver utilisateurs et publications grâce à des index textuels, tandis qu'un système de notifications en temps réel informe les utilisateurs des événements importants (likes, commentaires, suivis) via WebSocket.

2.2 Priorisation des exigences (Méthode MoSCoW)

La méthode **MoSCoW** a été utilisée afin de prioriser les fonctionnalités en fonction de leur importance, tout en tenant compte des contraintes de temps et de complexité propres au cadre de la formation.

Catégorie	Fonctionnalités associées
Must Have	Authentification et gestion des profils, création et interaction avec les threads, système de suivi, fil d'actualité, API REST sécurisée, stockage des médias via MinIO.
Should Have	Notifications en temps réel, moteur de recherche, fil recommandé basé sur les intérêts, chargement progressif des contenus (scroll infini).
Could Have	Reposts et quotes avancés, statistiques d'engagement, personnalisation de l'interface utilisateur.
Won't Have	Messagerie privée, modération automatisée par intelligence artificielle, monétisation et application mobile native.

Table 4: Priorisation des fonctionnalités selon la méthode MoSCoW

2.3 Exigences non fonctionnelles

Les exigences non fonctionnelles définissent les critères de qualité que doit respecter la plateforme afin d'assurer une expérience utilisateur fiable et performante.

2.3.1 Performance

L'application doit garantir des temps de réponse API maîtrisés, un chargement rapide de l'interface et la capacité à supporter un nombre élevé de connexions simultanées, notamment pour les fonctionnalités temps réel.

2.3.2 Sécurité

La sécurité repose sur une authentification JWT via cookies HTTP-only, le hachage des mots de passe avec bcrypt, la validation des entrées utilisateur et la protection contre les attaques courantes telles que les injections NoSQL et les attaques XSS.

2.3.3 Scalabilité

L'architecture backend stateless, associée à la séparation du stockage des médias et à la pagination des contenus, permet une montée en charge progressive et maîtrisée du système.

Cette analyse a servi de fondation aux choix techniques et architecturaux présentés dans les sections suivantes, assurant une cohérence globale entre les besoins identifiés et la solution mise en œuvre.

3 Gestion de projet

Le développement de la plateforme a suivi une approche **Scrum**, organisée en sprints successifs. Cette méthode a permis de structurer le travail autour d'un backlog priorisé, de livrer progressivement les fonctionnalités essentielles et d'ajuster le périmètre en fonction de l'avancement du projet.

La collaboration au sein de l'équipe s'est appuyée sur des outils professionnels. Le code source a été versionné avec **Git** et hébergé sur **GitHub**, tandis que la gestion des tâches, des user stories et des sprints a été assurée à l'aide de **Jira**. Ces outils ont permis un suivi clair de l'avancement et une coordination efficace entre les membres de l'équipe.

3.1 Backlog et User Stories

Le périmètre fonctionnel a été traduit en **user stories**, regroupées dans un **product backlog** priorisé. Chaque user story décrit un besoin utilisateur et des critères d'acceptation, permettant de valider objectivement la fonctionnalité livrée.

Exemples de User Stories

US-01 : Authentification

En tant qu'utilisateur, je veux créer un compte et me connecter **afin** d'accéder aux fonctionnalités de la plateforme.

US-02 : Publication de threads

En tant qu'utilisateur, je veux publier un thread (texte et/ou média) **afin** de partager du contenu avec la communauté.

US-03 : Interactions sociales

En tant qu'utilisateur, je veux liker et commenter un thread **afin** d'interagir avec les publications.

US-04 : Système de suivi

En tant qu'utilisateur, je veux suivre un autre utilisateur **afin** de retrouver ses publications dans mon fil.

US-05 : Notifications temps réel

En tant qu'utilisateur, je veux recevoir des notifications **afin** d'être informé des interactions importantes en temps réel.

3.2 Suivi de Projet avec Jira

La gestion du projet s'est appuyée sur **Jira** pour structurer le backlog, suivre l'avancement des sprints et coordonner les contributions de l'équipe. L'organisation des tâches en user stories, sous-tâches et épics a permis de maintenir une visibilité claire sur l'état du projet tout au long de son développement.

Backlog du Projet

Le backlog Jira centralise l'ensemble des user stories priorisées, réparties par état d'avancement. Chaque story est associée à un sprint spécifique, facilitant la planification et le suivi des livrables.

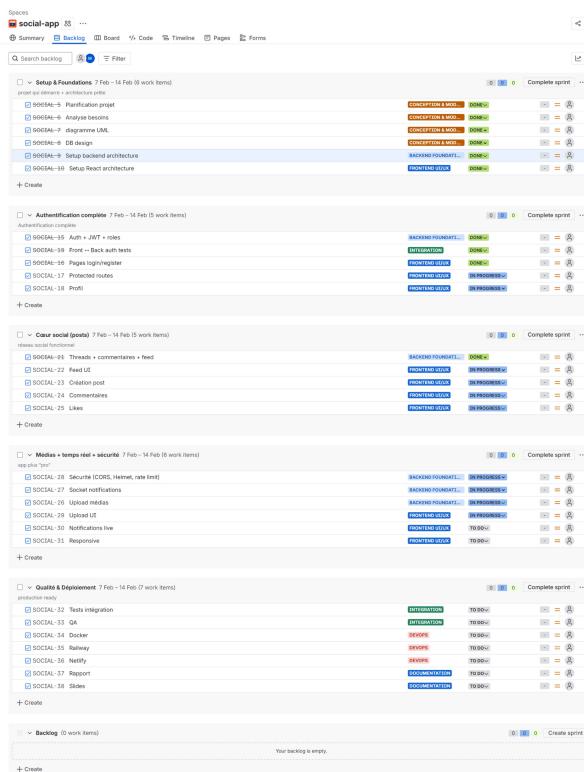


Figure 1: Backlog du projet dans Jira – Visualisation des user stories et de leur priorisation

La figure illustre la répartition des user stories selon les sprints planifiés, avec indication des statuts (To Do, In Progress, Done) et des assignations aux membres de l'équipe.

Sprints et Suivi d'Avancement

Le découpage en sprints a permis d'organiser le développement par itérations successives, chacune délivrant un ensemble cohérent de fonctionnalités. Jira assure le suivi de l'avancement de chaque sprint via des tableaux de bord dédiés.

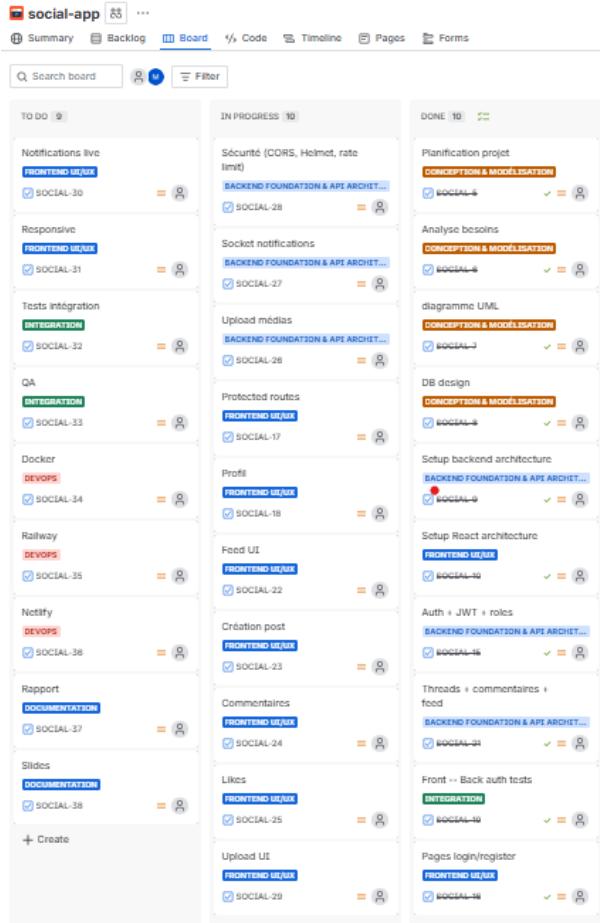


Figure 2: Vue des sprints dans Jira – Suivi détaillé de l'avancement par itération

Cette vue détaille l'état d'avancement des différents sprints, incluant les story points complétés, les tâches en cours et les éventuels blocages rencontrés. L'utilisation de cette interface a favorisé une gestion agile et réactive du projet, permettant des ajustements réguliers en fonction des priorités et des contraintes identifiées.

3.3 Planification et Suivi

La planification du projet a été structurée à l'aide de deux supports complémentaires, visant à représenter la chronologie des tâches et leurs dépendances :

- un **diagramme de Gantt**, permettant de visualiser la répartition temporelle des tâches et des livrables ;
- un **diagramme PERT**, mettant en évidence les dépendances entre tâches et l'enchaînement logique menant aux livrables principaux.

Diagramme de Gantt

La figure ci-dessous illustre la planification globale du projet sur la période définie.

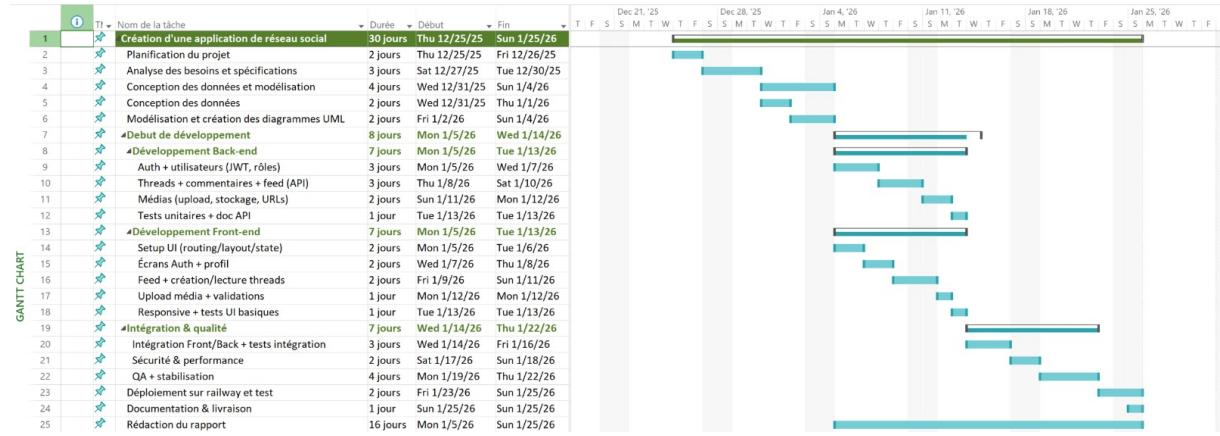


Figure 3: Diagramme de Gantt du projet **Social**

Diagramme PERT

La figure suivante illustre les dépendances entre les principales tâches du projet.

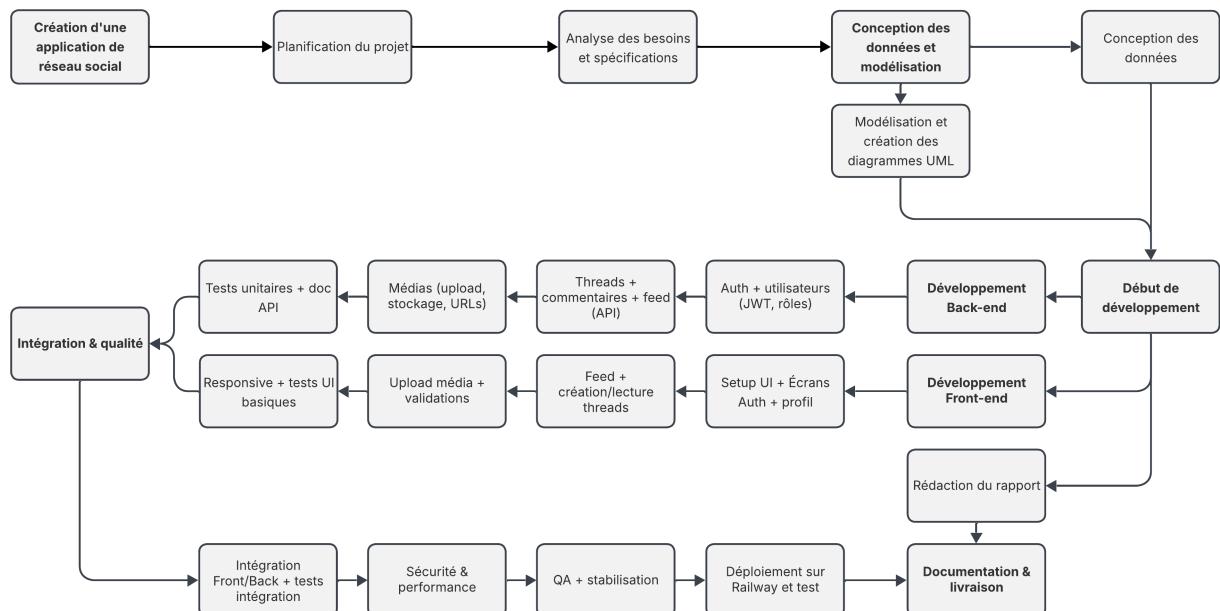


Figure 4: Diagramme PERT et dépendances principales des tâches

3.4 Livrables

À l'issue du projet, plusieurs livrables ont été produits afin de matérialiser le travail réalisé et de garantir l'exploitabilité de la solution développée.

- **Application web fonctionnelle** reposant sur une architecture complète frontend/backend, avec une interface développée en React et une API REST construite avec Express.js.
- **Backend structuré et sécurisé**, incluant une base de données MongoDB optimisée (indexation, relations logiques) et une API documentée respectant les bonnes pratiques.
- **Gestion des médias externalisée** via MinIO, permettant le stockage efficace des images et vidéos tout en allégeant la base de données.
- **Déploiement et portabilité**, assurés par la conteneurisation de l'application à l'aide de Docker (Dockerfile et docker-compose).
- **Documentation technique consolidée**, regroupant les choix d'architecture, les diagrammes de conception et les éléments nécessaires à la compréhension du système.

4 Architecture globale du système

4.1 Vue d'ensemble

La plateforme **Social** repose sur une architecture client–serveur moderne, conçue selon les principes de séparation des responsabilités, de modularité et de maintenabilité. Elle s'appuie sur la stack **MERN** (MongoDB, Express, React, Node.js), largement utilisée dans les environnements professionnels pour le développement d'applications web interactives et scalables.

L'architecture globale est organisée autour de trois couches principales : la couche de présentation, la couche applicative et la couche de données. Cette structuration permet d'assurer une évolution indépendante de chaque composant du système, tout en facilitant la compréhension globale du projet.

4.2 Couches fonctionnelles de l'application

Couche Présentation (Frontend) La couche frontend correspond à l'interface utilisateur de l'application. Développée sous forme d'application monopage (SPA) avec **React.js**, elle est responsable de l'affichage des données, de la gestion des interactions utilisateur et de la communication avec le serveur via des appels HTTP et WebSocket. La gestion globale de l'état est assurée par **Redux Toolkit**, tandis que la navigation entre les vues est prise en charge par **React Router**.

Couche Applicative (Backend) La couche backend constitue le cœur logique de l'application. Elle est développée avec **Node.js** et **Express.js** et expose une API REST sécurisée. Elle centralise la logique métier, la gestion des utilisateurs, des publications, des relations sociales ainsi que le traitement des notifications en temps réel via **Socket.io**. Cette couche intègre également des middlewares dédiés à la sécurité, à la validation des données et à la gestion des erreurs.

Couche Données La couche données est assurée par **MongoDB**, utilisée pour le stockage persistant des informations structurées (utilisateurs, threads, commentaires, relations, notifications). Les fichiers multimédias (images, vidéos) sont externalisés et stockés via **MinIO**, ce qui permet de préserver les performances de la base de données et de faciliter la montée en charge du système.

4.3 Arborescence globale du projet

L'organisation générale du projet reflète la séparation claire entre les différentes responsabilités techniques (frontend, backend, déploiement, ressources).

```

1  web-social-platform-main/
2  |-- backend/
3  |-- frontend/
4  |-- docker-compose.yml
5  |-- Ressources           #Rapport et présentation
6  |-- package.json
7  |-- README.md
8  '-- start.sh

```

Listing 1: Arborescence globale du projet

4.4 Organisation du backend

Le backend adopte une architecture en couches, facilitant la lisibilité du code et la maintenance. Chaque dossier correspond à un rôle bien défini dans le cycle de traitement des requêtes.

```

1  backend/
2  |-- node_modules/
3  |-- src/
4  |   |-- config/          # Configuration (DB, JWT, CORS, ...)
5  |   |-- controllers/    # Logique métier des endpoints
6  |   |-- middlewares/    # Authentification, validation,
7  |   |   securite
8  |   |   |-- models/       # Schemas MongoDB (Mongoose)
9  |   |   |-- routes/      # Définition des routes API
10 |   |   |-- utils/        # Fonctions utilitaires partagées
11 |   |   |-- validators/   # Schemas Joi de validation
12 |   |   |   |-- threadSchema.js
13 |   |   |   '-- userSchema.js
14 |   |   |-- app.js        # Configuration Express
15 |   |   |-- server.js     # Point d'entrée du serveur
16 |   |   '-- socket.js     # Gestion des WebSockets (Socket.io)
17 |   |-- Dockerfile
18 |   |-- .env
19 |   |-- .dockerignore
20 |   |-- package.json
21 '-- README.md

```

Listing 2: Arborescence du backend

Cette organisation permet :

- une séparation claire entre logique métier, accès aux données et sécurité ;
- une meilleure testabilité des composants ;
- une évolutivité facilitée en cas d'ajout de nouvelles fonctionnalités.

4.5 Organisation du frontend

Le frontend est structuré selon une approche modulaire basée sur les composants React. Cette organisation favorise la réutilisabilité du code et une meilleure lisibilité de l'interface.

```
1  frontend/
2  |-- node_modules/
3  |-- public/
4  |-- src/
5  |   |-- components/      # Composants UI reutilisables
6  |   |-- constants/      # Constantes globales
7  |   |-- hooks/          # Hooks personnalisés (scroll infini)
8  |   |-- lib/             # Fonctions utilitaires frontend
9  |   |-- pages/          # Pages principales de l'application
10 |   |-- services/       # Appels API (Axios)
11 |   |-- store/           # Redux Toolkit (slices, store)
12 |   |-- App.jsx          # Composant racine
13 |   |-- main.jsx         # Point d'entrée React
14 |   '-- index.css
15 |-- Dockerfile
16 |-- .env
17 |-- .dockerignore
18 |-- .gitignore
19 |-- nginx.conf
20 |-- eslint.config.js
21 |-- package.json
22 |-- package-lock.json
23 |-- vite.config.js
24 |-- postcss.config.js
25 |-- tailwind.config.js
26 '-- README.md
```

Listing 3: Arborescence du frontend

Cette structuration permet :

- une séparation claire entre logique d'affichage et logique métier ;
- une gestion cohérente de l'état global de l'application ;
- une intégration fluide avec le backend via des services dédiés.

4.6 Synthèse architecturale

L'architecture globale de la plateforme Social s'inscrit dans une démarche professionnelle inspirée des pratiques industrielles actuelles. Elle garantit une bonne maintenabilité, une évolution progressive du projet et une capacité de déploiement efficace, notamment grâce à la conteneurisation avec Docker et à l'hébergement prévu sur Railway.

5 Diagrammes UML

La compréhension globale du fonctionnement de la plateforme **Social** repose sur une modélisation claire des interactions, des processus internes et de la structure du système. Les diagrammes UML constituent un support essentiel pour formaliser les comportements attendus, les flux d'exécution et les relations entre les différentes composantes de l'application.

Les diagrammes présentés offrent une vision progressive du système : depuis les fonctionnalités accessibles à l'utilisateur (cas d'utilisation), jusqu'aux échanges dynamiques entre services (séquences), en passant par l'organisation logique des entités (classes). Ils servent de référence conceptuelle pour l'implémentation technique et la structuration des données.

5.1 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation synthétise les principales fonctionnalités offertes par la plateforme du point de vue de l'utilisateur. Il met en évidence les actions essentielles : authentification, gestion du profil, création de threads, interactions sociales (suivi, réactions, réponses) et consultation des notifications. Les relations «*include*» et «*extend*» représentent les dépendances fonctionnelles entre actions, notamment celles conditionnées par l'état d'authentification.

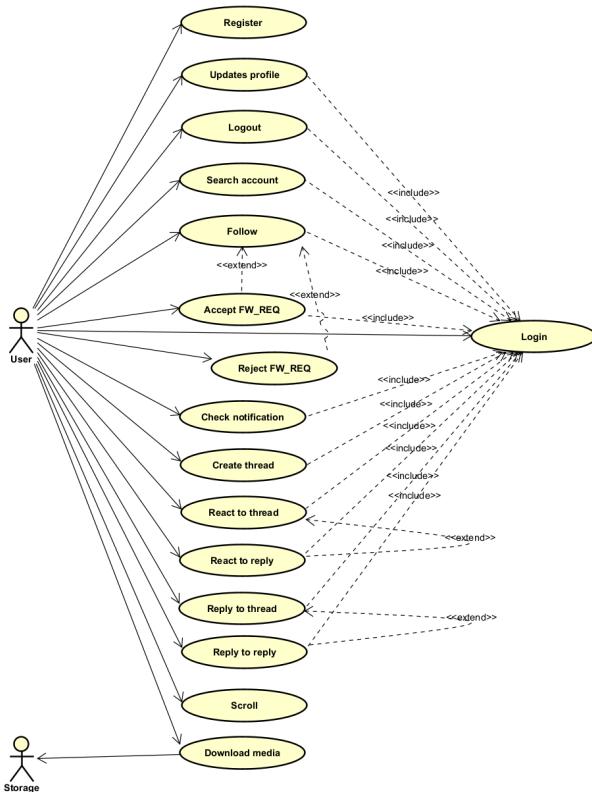


Figure 5: Diagramme de cas d'utilisation de la plateforme Social

5.2 Diagrammes de Séquence

Les diagrammes de séquence décrivent le comportement dynamique du système à travers plusieurs scénarios représentatifs. Ils permettent de visualiser les échanges successifs entre le frontend, l'API, les services métier, la base de données et les services externes, tout en mettant en évidence l'ordre logique des traitements.

Authentification et Gestion de Session

Ce diagramme illustre le processus d'inscription et de connexion d'un utilisateur. Il décrit la transmission des informations d'authentification, leur validation côté serveur, le hachage sécurisé du mot de passe, ainsi que la génération des tokens JWT (access et refresh) nécessaires à l'ouverture et au maintien de la session utilisateur.

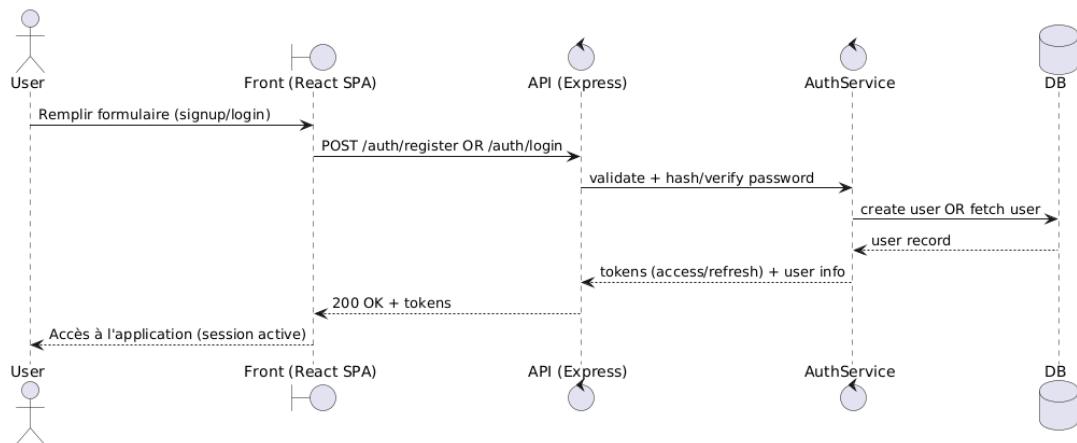


Figure 6: Diagramme de séquence – Authentification utilisateur

Création d'un Thread avec Média

Ce diagramme présente le scénario de publication d'un thread contenant un média (image ou vidéo). Il met en évidence la validation des données côté serveur, l'upload du fichier vers MinIO, l'enregistrement du contenu avec ses métadonnées en base MongoDB, puis son rendu dans le fil d'actualité.

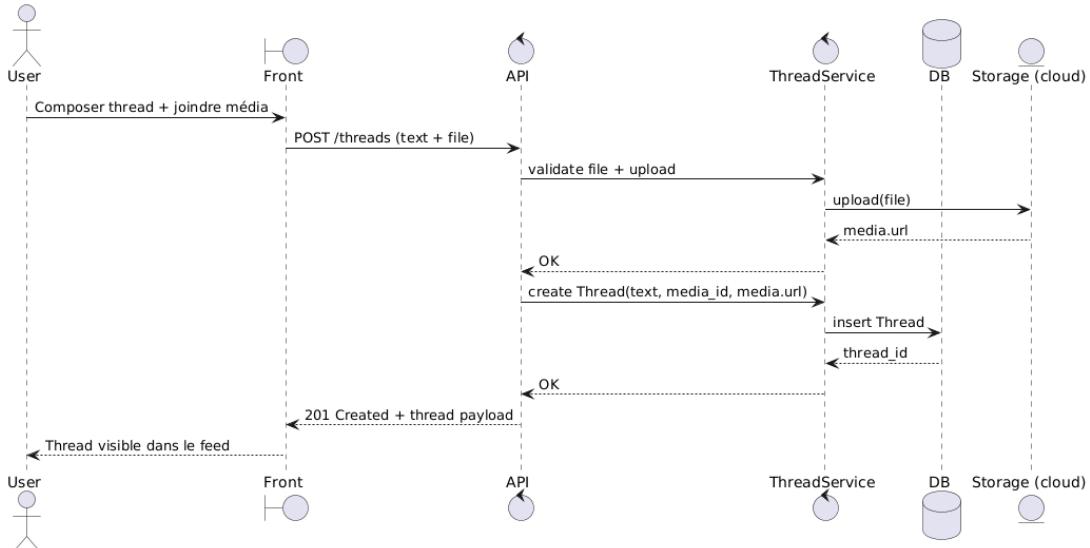


Figure 7: Diagramme de séquence – Création d'un thread avec média

Gestion du Suivi des Comptes

Les diagrammes suivants illustrent la gestion du suivi des utilisateurs, en distinguant les comportements selon le type de compte. Pour un compte public, le suivi est immédiat. Pour un compte privé, une demande est générée et transmise via notification, nécessitant validation ou refus par l'utilisateur concerné.

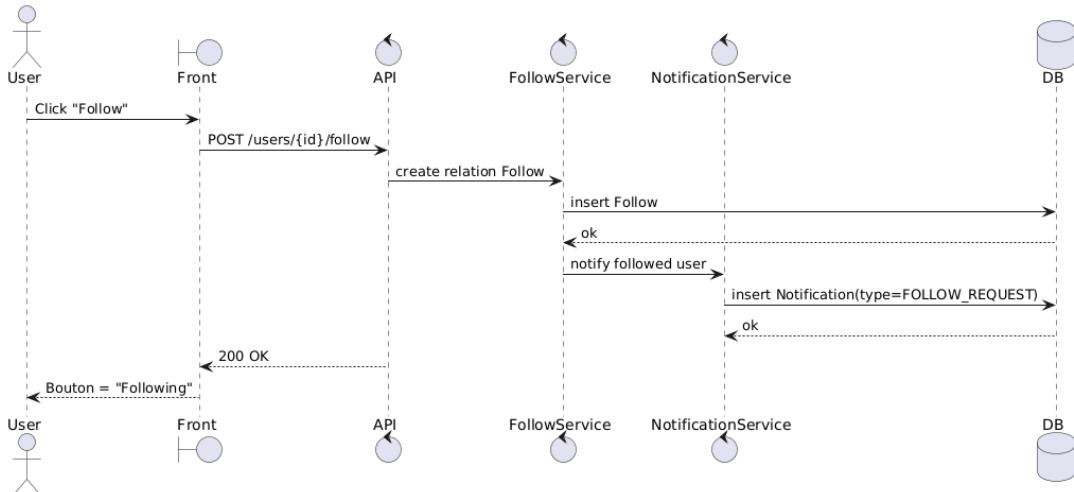


Figure 8: Suivi d'un compte public

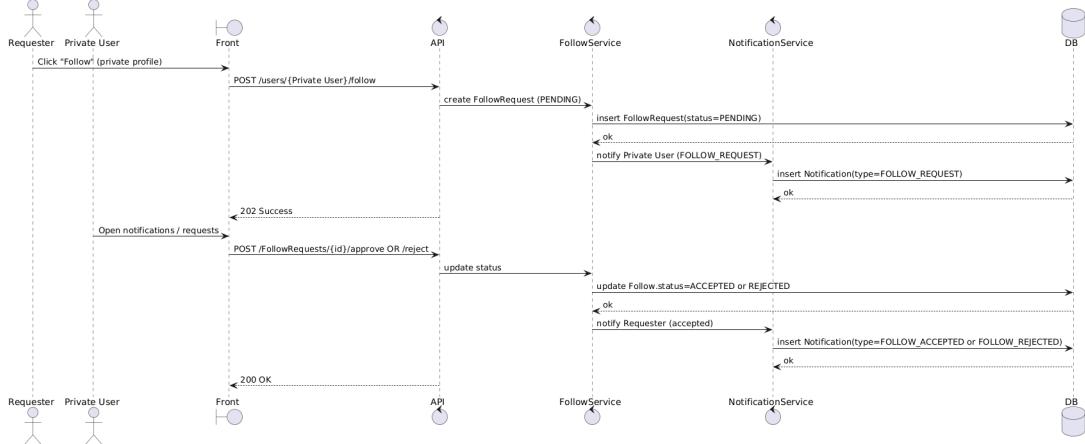


Figure 9: Suivi d'un compte privé

Réactions et Réponses aux Contenus

Ce diagramme décrit les mécanismes d'interaction autour des threads : ajout de likes et publication de commentaires. Chaque action peut entraîner la génération d'une notification temps réel via WebSocket, assurant une communication réactive entre les utilisateurs.

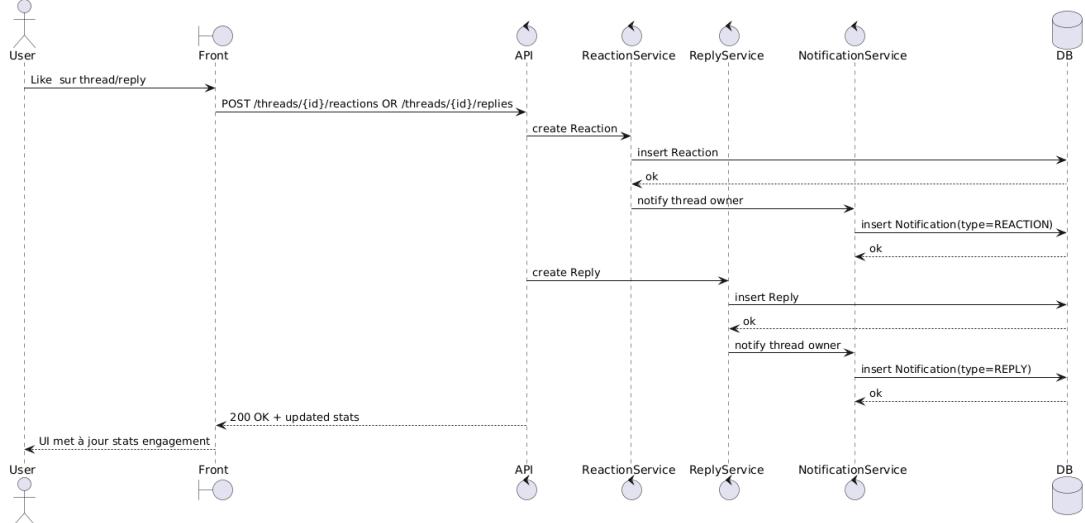


Figure 10: Diagramme de séquence – Réactions et réponses

5.3 Diagramme de Classes

Le diagramme de classes fournit une représentation statique de la structure du système. Il met en évidence les principales entités manipulées par la plateforme User, Thread, Comment, Like, Follow, Notification ainsi que les relations qui les lient (associations, compositions, cardinalités).

Ce diagramme constitue la base conceptuelle permettant de formaliser l'organisation logique des données avant leur implémentation dans la base MongoDB, où ces entités sont traduites en collections avec leurs schémas respectifs.

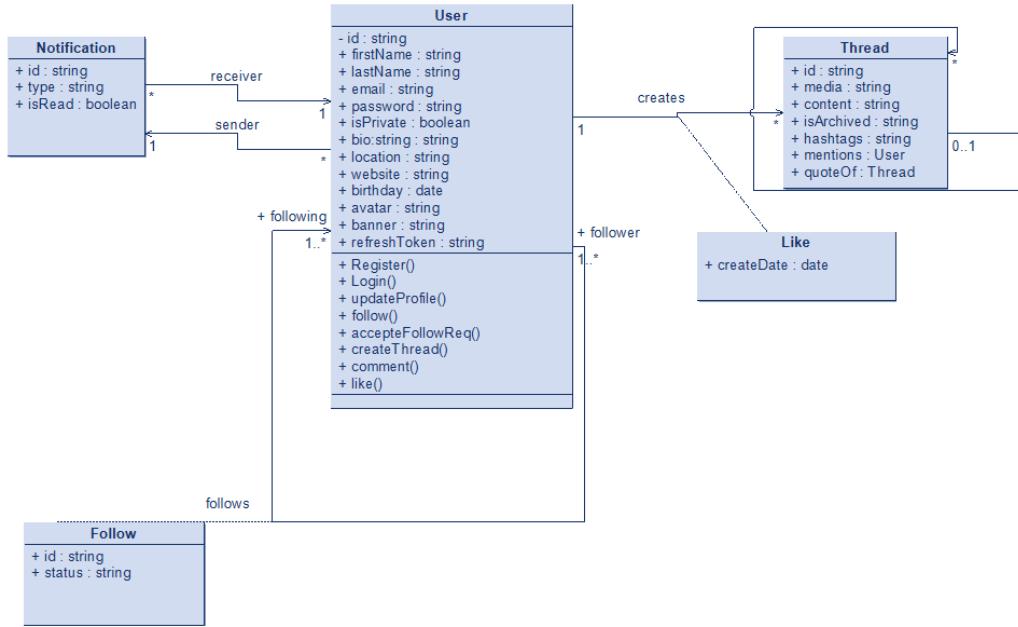


Figure 11: Diagramme de classes de la plateforme Social

L'ensemble de ces diagrammes serve de référence pour la modélisation des données présentée dans la section suivante, où les entités et relations décrites sont traduites en collections MongoDB adaptées aux contraintes de performance et d'évolutivité.

6 Modélisation des Données

La plateforme **Social** repose sur une base de données NoSQL MongoDB, choisie pour sa flexibilité et son adéquation avec les besoins d'un réseau social. La modélisation s'articule autour des entités centrales du système : utilisateurs, publications et interactions.

6.1 Organisation des collections MongoDB

La base de données est structurée autour de plusieurs collections distinctes, chacune correspondant à une entité métier clairement identifiée.

6.1.1 Collection Users

La collection `Users` centralise l'ensemble des informations relatives aux comptes utilisateurs, incluant les données d'identification, les paramètres de confidentialité ainsi que les éléments de profil public.

```
1  {
2      "_id": "651a2b3c4d5e6f7a8b9c0d11",
3      "firstName": "Lamya",
4      "lastName": "Makea",
5      "email": "lamya.makea@gmail.com",
6      "password": "$2b$10$hashed_password_here...",
7      "handle": "@lamyamakea0d11",
8      "fullName": "Lamya Makea",
9      "bio": "Full-stack developer | Love coding and coffee",
10     "location": "Rabat, Morocco",
11     "interests": ["Technology", "Coding", "Gaming"],
12     "isPrivate": false,
13     "avatar": {
14         "url": "http://minio:9000/avatars/user-123.jpg",
15         "key": "avatars/user-123.jpg"
16     },
17     "banner": {
18         "url": "https://minio.../banner.jpg",
19         "key": "banners/user123.jpg"
20     },
21     "followersCount": 120,
22     "followingCount": 85,
23     "bookmarks": ["651a2b3c4d5e6f7a8b9c0d22"],
24     "createdAt": "2025-01-19T10:00:00.000Z",
25     "updatedAt": "2025-01-19T12:30:00.000Z"
26 }
```

Listing 4: Schéma de la collection Users

Des index textuels sont appliqués sur les champs `firstName`, `lastName` et `email` afin d'optimiser la recherche, tandis qu'un index unique sur `email` garantit l'unicité des comptes utilisateurs.

6.1.2 Collection Threads

La collection **Threads** représente les publications effectuées par les utilisateurs. Chaque document est associé à un auteur et peut contenir du texte, des médias ainsi que des informations relatives aux reposts ou citations.

```
1  {
2    "_id": "651a2b3c4d5e6f7a8b9c0d22",
3    "author": "651a2b3c4d5e6f7a8b9c0d11",
4    "content": "Just launched my new project! Check it out #",
5      "coding #webdev",
6    "hashtags": ["coding", "webdev"],
7    "media": {
8      "mediaType": "image",
9      "url": "http://minio:9000/threads/post-99.png",
10     "key": "threads/post-99.png",
11     "contentType": "image/png"
12   },
13   "repostOf": null,
14   "quoteOf": null,
15   "isArchived": false,
16   "createdAt": "2025-01-19T11:00:00.000Z"
17 }
```

Listing 5: Schéma de la collection Threads

Un index texte est défini sur les champs **content** et **hashtags**, et un index composé sur (**author**, **createdAt**) permet d'optimiser l'affichage chronologique des publications par utilisateur.

6.1.3 Collection Comments

La collection **Comments** gère les réponses associées aux publications. Elle permet également la prise en charge de commentaires imbriqués via le champ **parentComment**.

```
1  {
2    "_id": "651a2b3c4d5e6f7a8b9c0d33",
3    "author": "651a2b3c4d5e6f7a8b9c0def",
4    "thread": "651a2b3c4d5e6f7a8b9c0d22",
5    "content": "Wow, this looks amazing! How long did it take?",
6    "parentComment": null,
7    "likesCount": 5,
8    "createdAt": "2025-01-19T11:15:00.000Z"
9  }
```

Listing 6: Schéma de la collection Comments

6.1.4 Collection Likes

La collection **Likes** enregistre les interactions de type engagement sur les publications ou commentaires.

```
1  {
2    "_id": "651a2b3c4d5e6f7a8b9c0d44",
3    "user": "651a2b3c4d5e6f7a8b9c0d11",
4    "thread": "651a2b3c4d5e6f7a8b9c0d22",
5    "comment": null,
6    "createdAt": "2025-01-19T11:20:00.000Z"
7 }
```

Listing 7: Schéma de la collection Likes

Un index unique composé sur (**user**, **thread**, **comment**) est utilisé afin d'empêcher tout doublon d'engagement.

6.1.5 Collection Follows

Cette collection gère les relations de suivi entre utilisateurs, en tenant compte du statut de validation pour les comptes privés.

```
1  {
2    "_id": "651a2b3c4d5e6f7a8b9c0d55",
3    "follower": "651a2b3c4d5e6f7a8b9c0def",
4    "following": "651a2b3c4d5e6f7a8b9c0d11",
5    "status": "ACCEPTED",
6    "createdAt": "2025-01-19T09:00:00.000Z"
7 }
```

Listing 8: Schéma de la collection Follows

Les statuts possibles sont PENDING, ACCEPTED et REFUSED, avec un index unique sur (**follower**, **following**).

6.1.6 Collection Notifications

La collection **Notifications** permet de notifier les utilisateurs lors des interactions importantes.

```

1  {
2    "_id": "651a2b3c4d5e6f7a8b9c0d66",
3    "type": "LIKE",
4    "receiver": "651a2b3c4d5e6f7a8b9c0d11",
5    "sender": "651a2b3c4d5e6f7a8b9c0def",
6    "thread": "651a2b3c4d5e6f7a8b9c0d22",
7    "comment": null,
8    "isRead": false,
9    "createdAt": "2025-01-19T11:20:05.000Z"
10 }

```

Listing 9: Schéma de la collection Notifications

Les types de notifications incluent notamment FOLLOW_REQUEST, FOLLOW_ACCEPTED, NEW_FOLLOWER, LIKE et COMMENT.

6.2 Relations entre les entités

Bien que MongoDB ne repose pas sur un schéma relationnel strict, les relations entre les différentes collections sont assurées par des références explicites via les identifiants `_id`.

Relation	Implémentation
User → Threads	Champ <code>author</code> référence <code>_id</code> utilisateur
Thread → Comments	Champ <code>thread</code> référence <code>_id</code> publication
Comment → Comment	Champ <code>parentComment</code> (auto-référence pour commentaires imbriqués)
User ↔ Thread (Likes)	Collection <code>Likes</code> avec champs <code>user</code> et <code>thread</code>
User ↔ Comment (Likes)	Collection <code>Likes</code> avec champs <code>user</code> et <code>comment</code>
User ↔ User (Follows)	Collection <code>Follows</code> avec champs <code>follower</code> et <code>following</code>
User ← Notifications	Champs <code>receiver</code> et <code>sender</code> référencent utilisateurs

Table 5: Matrice des relations entre collections

7 API REST

Les principaux endpoints de l'API REST de l'application sont présentés ci-dessous, organisés par domaine fonctionnel afin de faciliter la lecture et la compréhension des fonctionnalités exposées côté serveur.

Les mécanismes d'authentification et de sécurisation associés à ces endpoints sont détaillés ultérieurement.

7.1 Endpoints principaux

Table 6: Endpoints de l'API Social

Méthode	Endpoint	Description
Authentification		
POST	/api/auth/register	Créer un nouveau compte
POST	/api/auth/login	Se connecter
POST	/api/auth/logout	Se déconnecter
POST	/api/auth/refresh-token	Obtenir nouveau access token
GET	/api/auth/me	Récupérer profil actuel
Utilisateurs		
GET	/api/user/profile/:id	Consulter profil utilisateur
PATCH	/api/user/profile	Modifier son profil
GET	/api/user/suggestions	Obtenir suggestions de suivi
GET	/api/user/search?q=...	Rechercher utilisateurs
Threads (Posts)		
POST	/api/thread	Créer un nouveau post
GET	/api/thread	Récupérer fil public
GET	/api/thread/me	Récupérer ses posts
GET	/api/thread/recommended	Fil recommandé
GET	/api/thread/me/:id	Détail d'un post
PATCH	/api/thread/me/:id	Modifier son post
DELETE	/api/thread/me/:id	Supprimer son post
POST	/api/thread/:id/bookmark	Bookmarker un post
Commentaires		
GET	/api/thread/:id/comments	Récupérer commentaires
POST	/api/thread/:id/comments	Commenter un post
PATCH	/api/thread/comments/:id	Modifier commentaire
DELETE	/api/thread/comments/:id	Supprimer commentaire
Likes		
POST	/api/like/thread/:id	Liker un post
DELETE	/api/like/thread/:id	Unliker un post
POST	/api/like/comment/:id	Liker un commentaire
DELETE	/api/like/comment/:id	Unliker un commentaire

(Suite du tableau)

Méthode	Endpoint	Description
Suivis		
POST	/api/follower/:id	Suivre un utilisateur
DELETE	/api/follower/:id	Ne plus suivre
PATCH	/api/follower/:id/accept	Accepter demande
PATCH	/api/follower/:id/reject	Refuser demande
GET	/api/follower/followers	Liste des followers
GET	/api/follower/following	Liste des following
Notifications		
GET	/api/notification	Récupérer notifications
PATCH	/api/notification/:id/read	Marquer comme lu
PATCH	/api/notification/read-all	Tout marquer comme lu

L'organisation de ces endpoints permet une compréhension synthétique des fonctionnalités offertes par l'API et des interactions établies entre le client et le serveur.

8 Sécurité

La sécurité de l'application **Social** a été intégrée dès les premières étapes du développement comme un axe fondamental du système. Elle vise à garantir la confidentialité des données utilisateurs, l'intégrité des échanges entre le client et le serveur, ainsi que la protection contre les vulnérabilités les plus courantes rencontrées dans les applications web modernes.

8.1 Authentification basée sur les JSON Web Tokens

Le mécanisme d'authentification repose sur l'utilisation des **JSON Web Tokens (JWT)**, un standard largement adopté pour la sécurisation des échanges dans les architectures web modernes.

Lorsqu'un utilisateur accède à la plateforme, il s'authentifie à l'aide de son adresse e-mail et de son mot de passe. Ces informations sont transmises de manière sécurisée au serveur, qui procède à leur vérification en comparant le mot de passe fourni avec sa version hachée stockée en base de données à l'aide de l'algorithme **bcrypt**.

Une fois l'authentification validée, deux types de jetons sont générés :

- un **Access Token**, de courte durée (15 minutes), utilisé pour authentifier les requêtes vers les ressources protégées de l'application ;
- un **Refresh Token**, de durée plus longue (7 jours), permettant de renouveler la session sans imposer une reconnexion fréquente à l'utilisateur.

L'Access Token est transmis au client via un cookie de type *HTTP-only*, ce qui empêche son accès par des scripts côté client et limite ainsi les risques liés aux attaques de type XSS. Le Refresh Token est conservé côté serveur afin de mieux contrôler le cycle de vie des sessions.

À chaque requête, un middleware d'authentification vérifie la validité du jeton d'accès, garantissant que seules les requêtes autorisées peuvent accéder aux fonctionnalités protégées.

8.2 Gestion et sécurisation des mots de passe

La gestion des mots de passe repose sur des principes stricts de sécurité afin de limiter les risques de compromission des comptes utilisateurs.

Les mots de passe ne sont jamais stockés en clair. Lors de l'inscription ou de la modification d'un mot de passe, celui-ci est systématiquement haché à l'aide de **bcrypt**, avec un facteur de complexité fixé à 10 *salt rounds*.

Des règles de validation sont également appliquées afin d'imposer un niveau minimal de robustesse, notamment une longueur minimale. Lors de l'authentification, la vérification s'effectue exclusivement par comparaison de hachage, sans jamais exposer le mot de passe original.

8.3 Protection contre les attaques courantes

L’application intègre plusieurs mécanismes destinés à prévenir les attaques les plus fréquemment rencontrées dans les environnements web.

8.3.1 Prévention des injections NoSQL

Afin de se prémunir contre les attaques par injection NoSQL, le middleware **express-mongo-sanitize** est utilisé pour neutraliser les opérateurs malveillants présents dans les entrées utilisateur.

En complément, une validation stricte des données est assurée à l’aide de **Joi**. Cette approche garantit que seules des données conformes aux formats attendus sont traitées par l’application, renforçant ainsi la cohérence et la fiabilité des opérations sur la base de données.

8.3.2 Protection contre les attaques XSS

La protection contre les attaques de type Cross-Site Scripting (XSS) repose sur le nettoyage systématique des entrées utilisateur et sur la configuration d’en-têtes de sécurité HTTP à l’aide du middleware **Helmet**. Ces mesures réduisent significativement les risques d’exécution de code malveillant côté client.

8.3.3 Limitation du nombre de requêtes

Un mécanisme de **rate limiting** est mis en place afin de limiter le nombre de requêtes pouvant être émises par une même adresse IP sur une période donnée. Cette mesure permet de prévenir les attaques par force brute, notamment sur les routes d’authentification, et de réduire l’impact potentiel des attaques par déni de service.

8.4 Mesures de sécurité complémentaires

En complément des mécanismes précédents, une configuration stricte du **CORS** est appliquée afin de restreindre les origines autorisées à communiquer avec l’API. En environnement de production, l’utilisation de **HTTPS** garantit le chiffrement des échanges entre le client et le serveur.

Enfin, la journalisation des requêtes via **Morgan** permet d’assurer un suivi des accès et de faciliter l’analyse et le diagnostic en cas d’incident.

9 Déploiement

Le déploiement de l'application **Social** repose sur une stratégie de conteneurisation visant à garantir la portabilité, la reproductibilité et la cohérence des environnements d'exécution. Cette approche permet d'assurer une continuité entre les phases de développement, de test et de mise en production, tout en simplifiant l'exploitation de l'application.

Dans ce projet, l'orchestration des différents composants est assurée par **Docker Compose**, qui permet de structurer l'application en services indépendants, chacun encapsulé dans un conteneur dédié, avec une gestion explicite des dépendances et de la persistance des données.

9.1 Architecture de conteneurisation

L'architecture de déploiement est organisée autour de plusieurs services distincts, chacun remplissant un rôle précis au sein du système :

- **Backend** : API développée avec Node.js et Express, exposée sur le port **3000**, responsable de la logique métier, de la gestion des utilisateurs, des interactions sociales et des notifications.
- **Frontend** : application React fournissant l'interface utilisateur, exposée sur le port **8080**.
- **MongoDB** : base de données NoSQL utilisée pour le stockage des données applicatives (utilisateurs, threads, relations, notifications), avec persistance via un volume Docker.
- **MinIO** : service de stockage d'objets compatible S3, dédié à la gestion des médias (images, bannières, avatars), exposé sur les ports **9002** (API) et **9001** (console).
- **Mongo Express** : interface d'administration de MongoDB sur le port **8081**, utilisée principalement durant le développement pour l'inspection et le suivi des données.

Cette séparation des responsabilités permet d'améliorer la maintenabilité du système, de limiter les dépendances croisées et de faciliter une éventuelle montée en charge.

9.2 Orchestration avec Docker Compose

La configuration Docker Compose définit l'ensemble des services, leurs ports d'exposition, leurs dépendances et les volumes nécessaires à la persistance des données. Le backend utilise un fichier `.env` afin de centraliser la configuration sensible (clés JWT, accès base de données, paramètres MinIO), tandis qu'un montage de volume est activé pour faciliter le développement local.

9.3 Exécution locale

L'environnement Docker permet de lancer l'intégralité de la plateforme sans installation manuelle des dépendances système. Les commandes principales utilisées sont :

```
1 # Démarrer l'ensemble des services
2 docker compose up -d
3
4 # Consulter les logs
5 docker compose logs -f
6
7 # Arrêter les services
8 docker compose down
9
10 # Reconstruire les images après modification
11 docker compose up -d --build
```

Listing 10: Commandes Docker Compose

Une fois les services démarrés, les points d'accès principaux sont les suivants :

Service	URL d'accès
Frontend	http://localhost:8080
Backend API	http://localhost:3000
MinIO Console	http://localhost:9001
Mongo Express	http://localhost:8081

Table 7: Accès aux services en environnement local

9.4 Déploiement sur Railway

Le déploiement de l'application a été réalisé sur la plateforme cloud **Railway**, assurant son accessibilité publique et simplifiant la gestion infrastructurelle. L'intégration native avec GitHub facilite le déploiement continu depuis le dépôt du projet, tandis que l'interface Railway centralise la gestion des variables d'environnement sensibles.

Les services backend et frontend sont déployés depuis le même dépôt, chacun disposant d'une configuration spécifique.

Lien de l'application déployée :

<https://threads-app.up.railway.app>

9.5 Aperçu du Processus de Déploiement

Les figures suivantes illustrent les étapes successives du déploiement, depuis l'initialisation du projet jusqu'à la mise en ligne effective de l'application.

Initialisation du Projet

La création d'un nouveau projet sur Railway débute par la sélection de l'option « **Empty Project** », déclenchant la génération automatique de l'espace de travail.

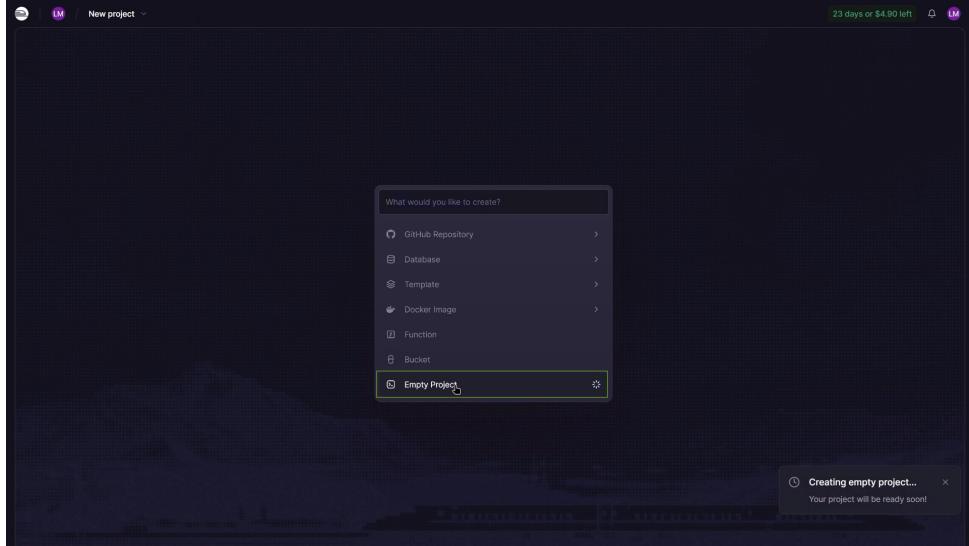


Figure 12: Crédit à la création du projet sur Railway

Déploiement du Service de Stockage (MinIO)

Le premier composant déployé est le service **MinIO**, responsable du stockage des fichiers multimédias. L'utilisation d'une template Railway accélère la configuration initiale.

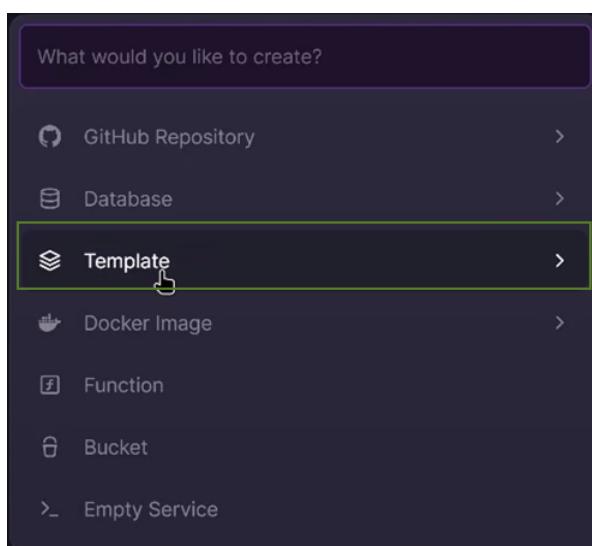


Figure 13: Ajout d'un service via template

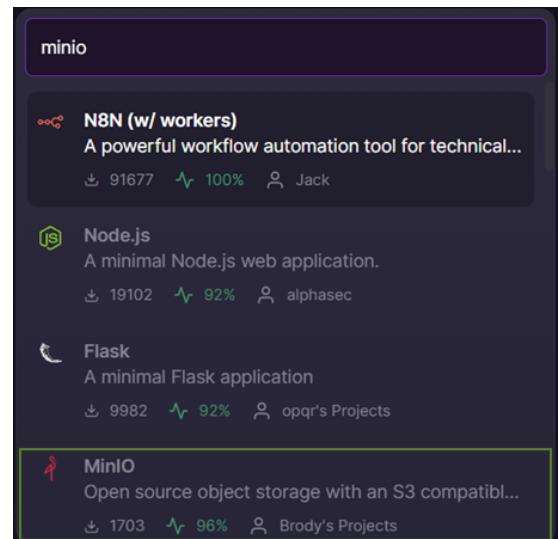


Figure 14: Initialisation du service MinIO

Le déploiement s'effectue après configuration des variables d'environnement essentielles (**MINIO_ROOT_USER**, ports, domaines). Le bouton « **Deploy Template** » déclenche l'initialisation du service.

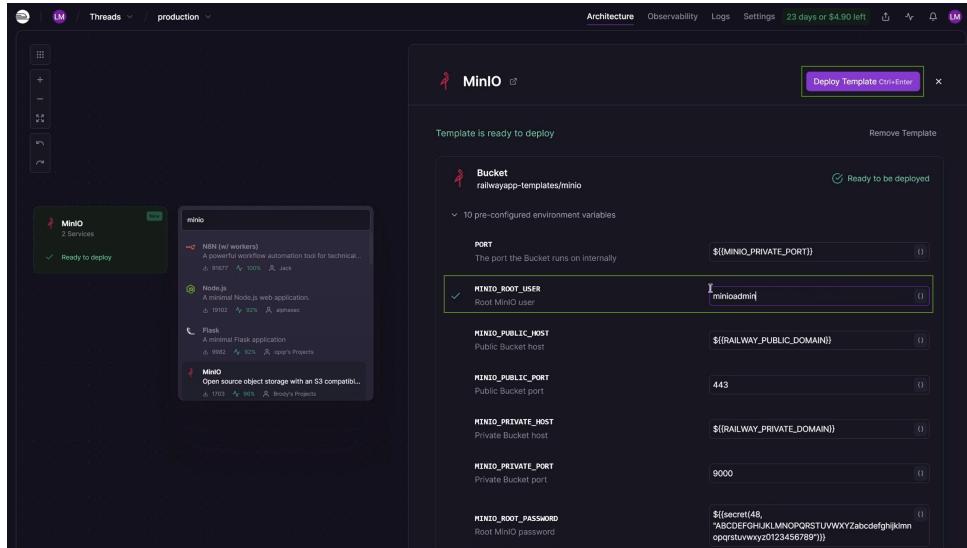


Figure 15: Configuration et déploiement de MinIO

La procédure identique est ensuite appliquée pour le déploiement de **MongoDB**.

Vérification du Service MinIO

Le bon fonctionnement de MinIO est validé via l'accès à son interface web, confirmant sa capacité à gérer les buckets et objets.

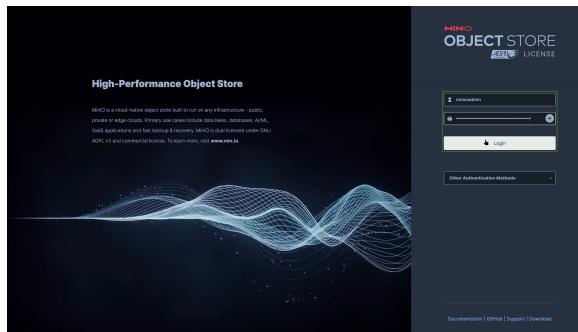


Figure 16: Authentification à la console MinIO

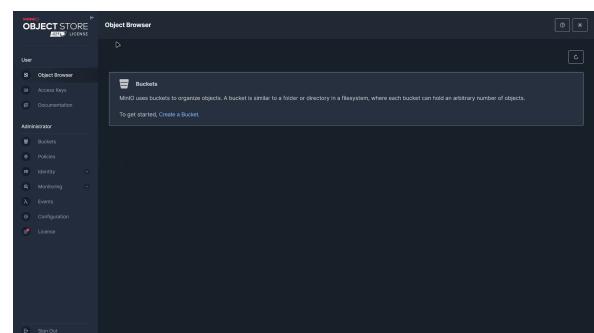


Figure 17: Accès à l'Object Browser

Déploiement des Services Applicatifs

Les services **Backend** et **Frontend** sont créés via l'option « **Empty Service** », permettant une configuration personnalisée depuis le dépôt GitHub.

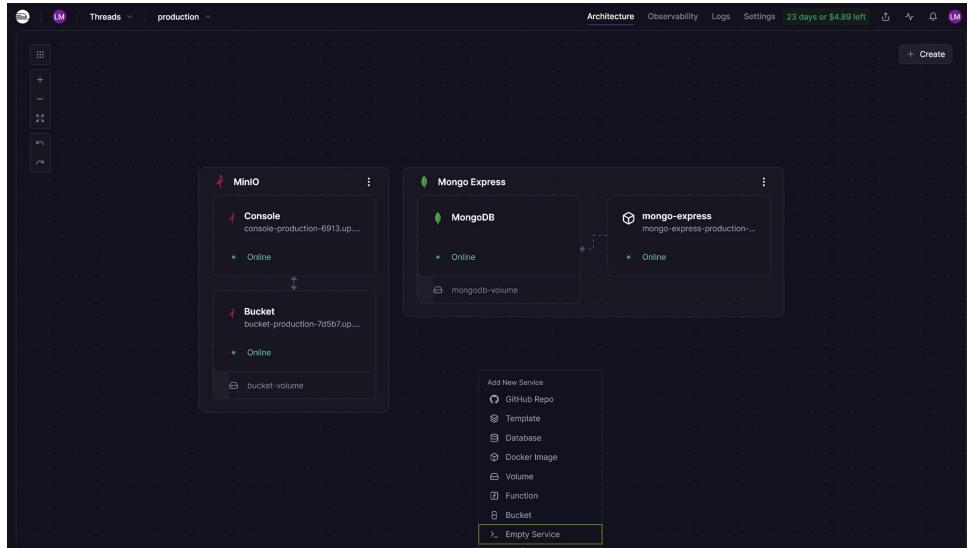


Figure 18: Création d'un service vide sur Railway

Configuration du Backend

La configuration du backend s'articule autour de trois étapes : liaison au dépôt GitHub, définition du répertoire racine (**backend/**), et sélection du mode de build via **Dockerfile**.

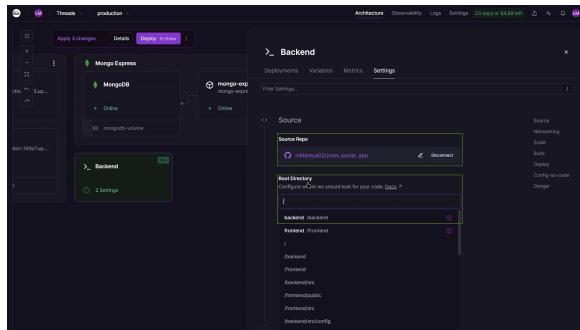


Figure 19: Liaison au dépôt et choix du répertoire

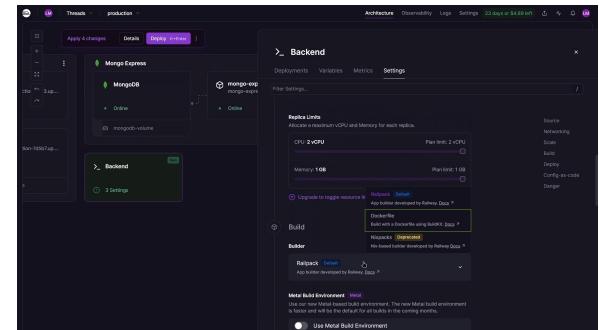


Figure 20: Configuration du build via Dockerfile

Les variables d'environnement nécessaires au fonctionnement de l'API (connexions MinIO et MongoDB, secrets JWT) sont définies via le **Raw Editor**.

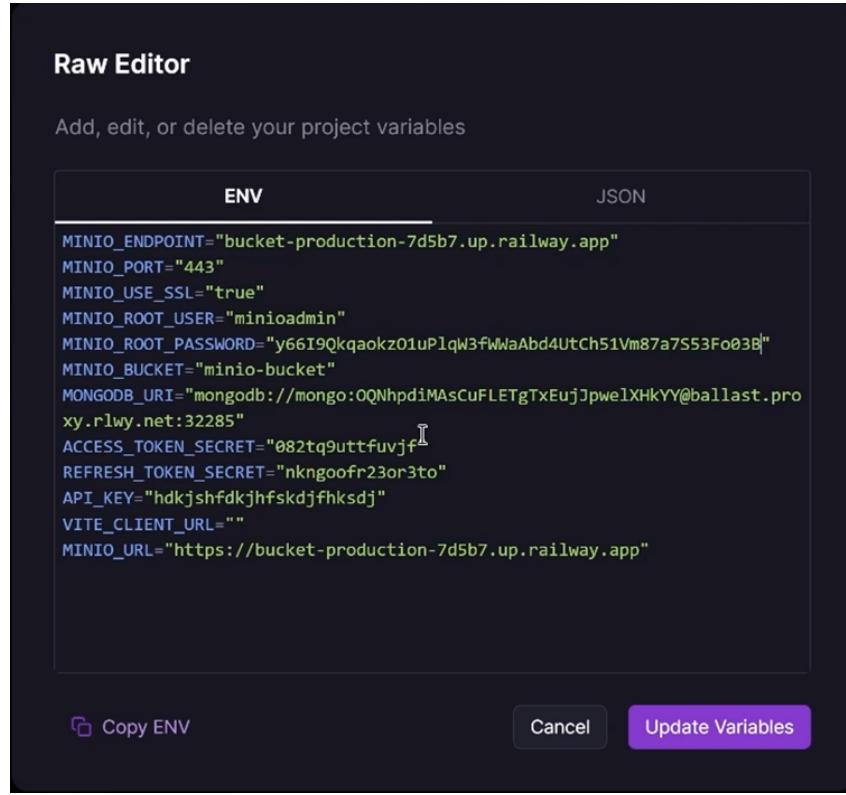


Figure 21: Configuration des variables d'environnement du backend

Exposition Publique du Backend

L'accessibilité publique de l'API est assurée par la génération d'un domaine Railway, permettant la communication avec le frontend.

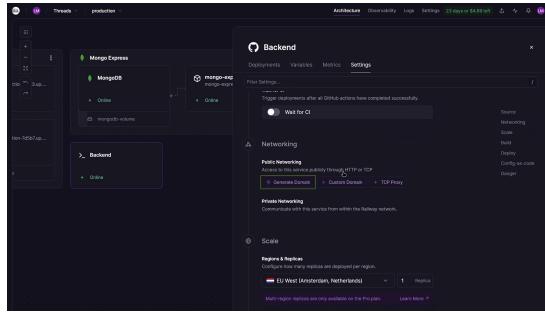


Figure 22: Génération du domaine public

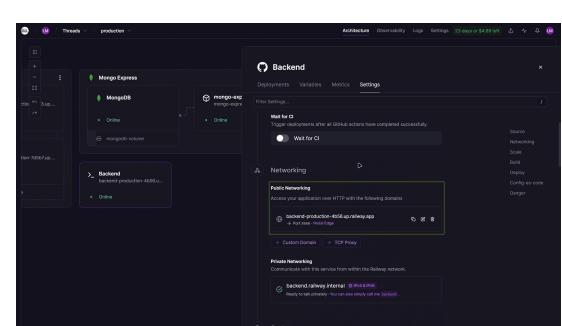


Figure 23: Domaine public généré

Déploiement du Frontend

La procédure suivie pour le backend est reproduite à l'identique pour le frontend, aboutissant à l'obtention de son propre domaine public.

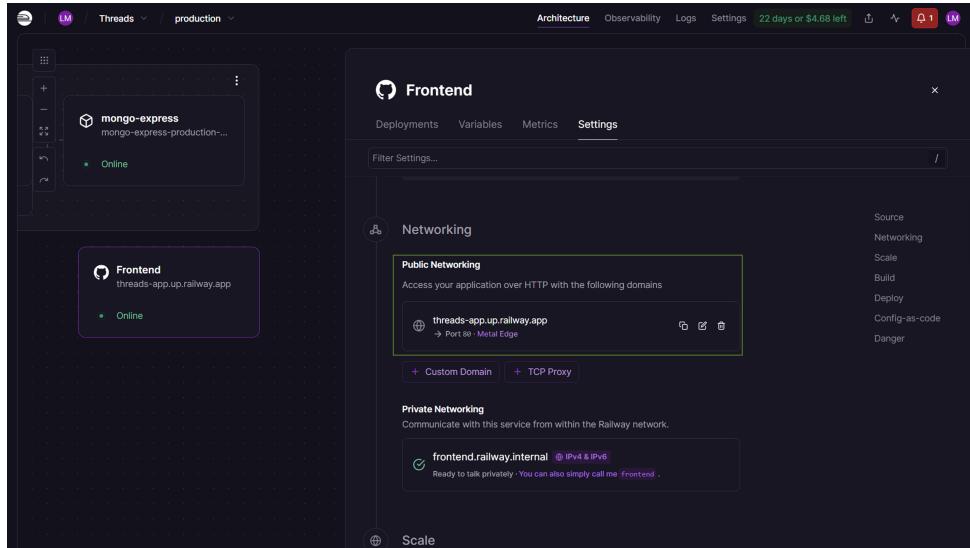


Figure 24: Domaine public du service Frontend

Validation du Déploiement

L'accès à l'interface web via le domaine frontend confirme le succès du déploiement. L'affichage correct du fil d'actualité, des composants de navigation et des fonctionnalités interactives valide l'accessibilité publique de l'application.

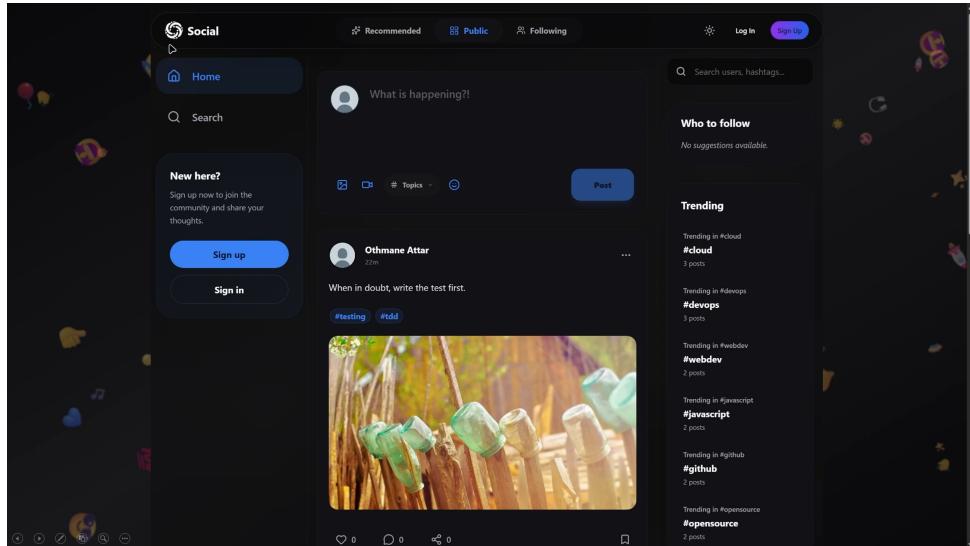


Figure 25: Interface de l'application déployée

10 Démonstration de l'Application

Les interfaces présentées ci-après illustrent les fonctionnalités essentielles de la plateforme, depuis le processus d'authentification jusqu'aux mécanismes d'interaction sociale, en passant par la personnalisation du profil et la gestion des paramètres de confidentialité.

10.1 Authentification

Le processus d'authentification s'appuie sur des interfaces modernes dédiées à l'inscription et à la connexion des utilisateurs.

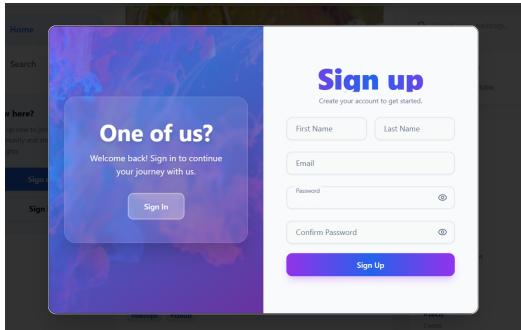


Figure 26: Interface d'inscription

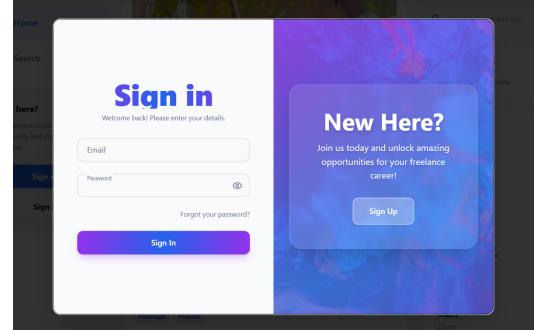


Figure 27: Interface de connexion

L'inscription requiert la saisie des informations essentielles avec validation en temps réel des critères de robustesse. La connexion s'effectue via l'adresse électronique et le mot de passe, déclenchant la génération de tokens JWT sécurisés permettant l'ouverture et le maintien de la session utilisateur.

10.2 Fil d'Actualité et Modes d'Affichage

La page d'accueil organise le fil d'actualité selon les préférences visuelles de l'utilisateur, offrant un basculement fluide entre modes clair et sombre.

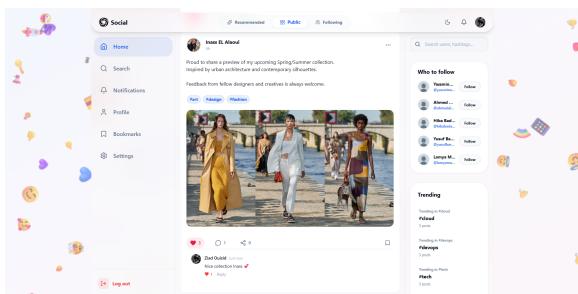


Figure 28: Mode clair

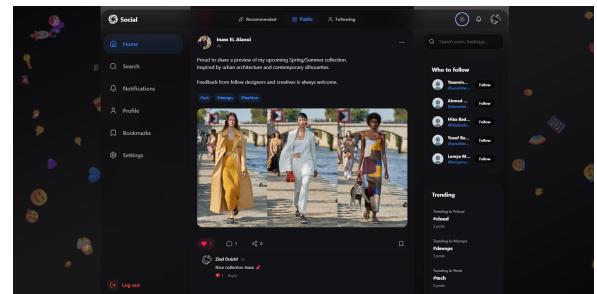


Figure 29: Mode sombre

Le fil agrège les publications des utilisateurs suivis, intégrant contenus multimédias, hashtags, réactions et commentaires. L'architecture de navigation favorise l'accès immédiat aux sections principales : accueil, recherche, notifications, profil, favoris et paramètres.

Filtrage et Organisation du Contenu

Le système propose trois modes de consultation du fil d'actualité, accessibles via des onglets dédiés :

- **Recommended** : fil personnalisé intégrant le système de recommandation basé sur les centres d'intérêt de l'utilisateur et son historique d'interactions ;
- **Public** : fil global regroupant l'ensemble des publications publiques, indépendamment des relations de suivi ;
- **Following** : fil restreint aux publications des utilisateurs suivis, respectant strictement les règles de confidentialité des comptes privés.

Cette segmentation permet à l'utilisateur d'adapter sa navigation selon ses besoins : découverte de nouveaux contenus via le fil recommandé, exploration large via le fil public, ou consultation ciblée des abonnements.

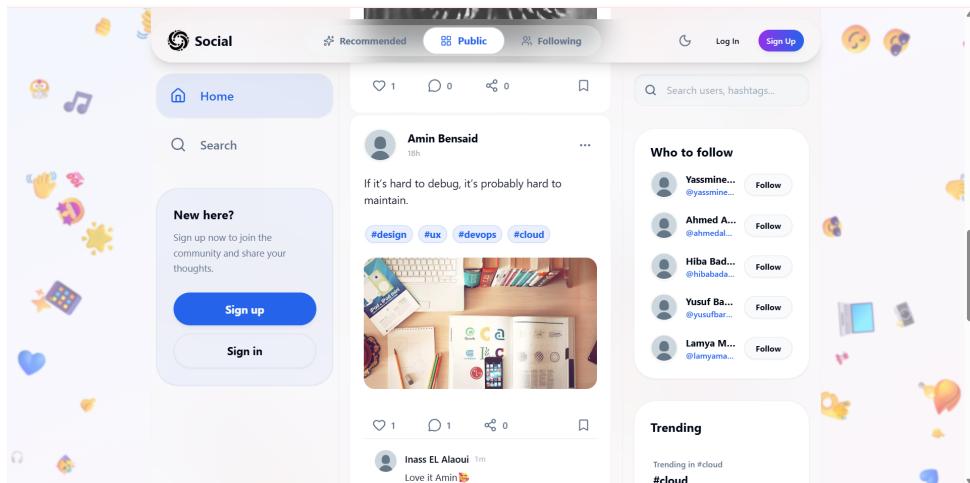


Figure 30: Aperçu du fil public avant authentification

Les visiteurs non authentifiés accèdent au fil public en mode consultation, l'inscription demeurant nécessaire pour activer l'ensemble des fonctionnalités interactives.

10.3 Crédit et Publication de Contenu

La publication de contenus impose la sélection préalable d'au moins un centre d'intérêt (topic) afin d'alimenter le moteur de recommandation. Cette contrainte garantit la pertinence du système de filtrage et améliore la découvrabilité des publications auprès des utilisateurs partageant des affinités thématiques similaires.

Les hashtags extraits automatiquement du contenu textuel complètent cette catégorisation, offrant une indexation fine des publications et facilitant la recherche thématique au sein de la plateforme.

10.4 Gestion du Profil Utilisateur

L'espace profil centralise les données personnelles, les statistiques sociales et l'historique des publications de l'utilisateur.

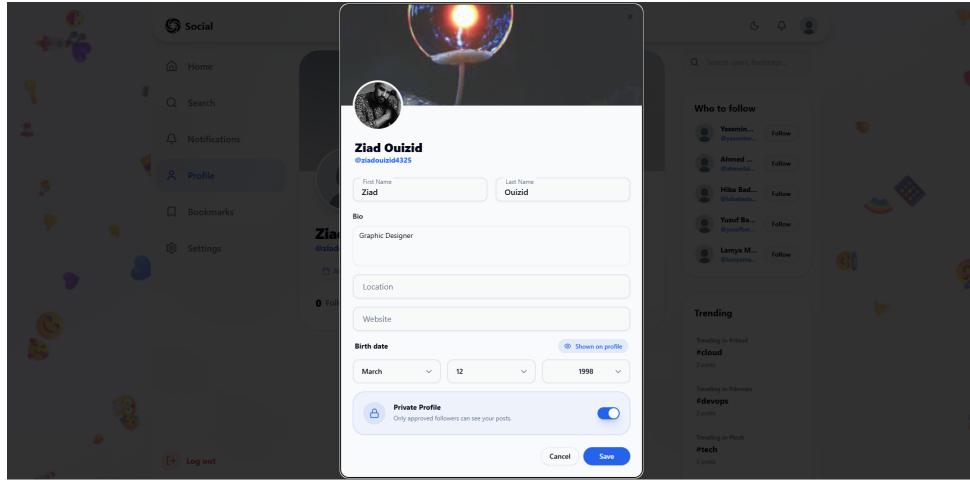


Figure 31: Édition du profil

La consultation du profil révèle l'avatar, la bannière personnalisée, la biographie, les statistiques d'abonnement (following/followers) ainsi que l'intégralité des contenus publiés. L'interface d'édition autorise la modification de l'ensemble de ces éléments, incluant également la localisation géographique, le site web personnel, la date de naissance et le statut de confidentialité du compte.

10.5 Personnalisation et Système de Recommandation

Le système de personnalisation repose sur la sélection explicite de centres d'intérêt, permettant l'ajustement dynamique du contenu proposé.

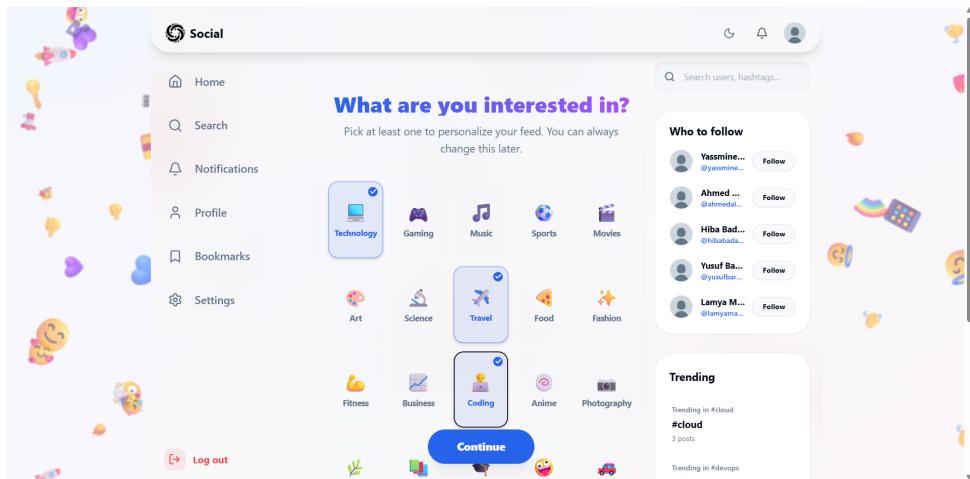


Figure 32: Sélection des centres d'intérêt

L'utilisateur détermine ses préférences thématiques parmi un ensemble de catégories prédéfinies: technologie, gaming, musique, sport, cinéma, art, sciences, voyage, gastronomie, et autres. Ces préférences alimentent le moteur de recommandation, assurant la personnalisation du fil d'actualité en fonction des affinités déclarées.

Le système exploite ces informations pour construire le fil *Recommended*, où sont proposées prioritairement les publications correspondant aux centres d'intérêt sélectionnés.

10.6 Notifications et Gestion des Interactions

Le système de notifications informe l'utilisateur des interactions significatives en temps réel via WebSocket.

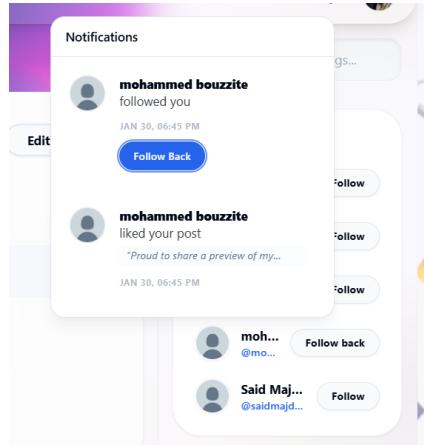


Figure 33: Page dédiée aux notifications

Les événements notifiés comprennent l'acquisition de nouveaux abonnés, les demandes de suivi pour les comptes privés, ainsi que les réactions et commentaires sur les publications. L'accès s'effectue via un modal contextuel ou une page dédiée présentant l'historique complet.

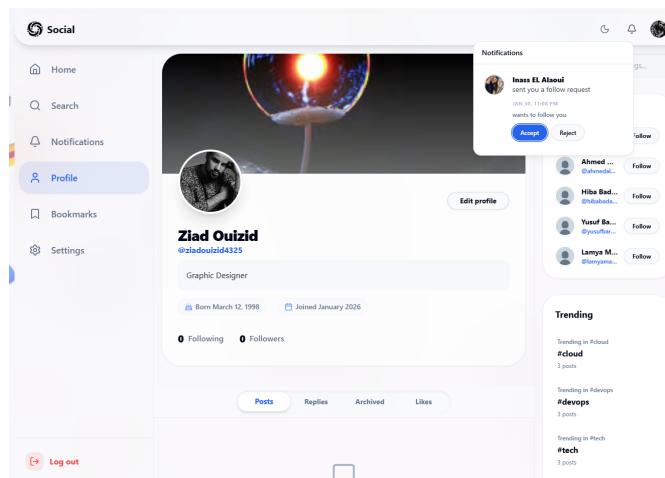


Figure 34: Gestion des demandes de suivi

Les comptes configurés en mode privé imposent une validation manuelle des demandes de suivi, l'utilisateur disposant des options d'acceptation ou de refus.

10.7 Configuration et Paramètres de Sécurité

L'interface de paramétrage offre un contrôle granulaire sur la confidentialité, les préférences d'affichage et la sécurité du compte.

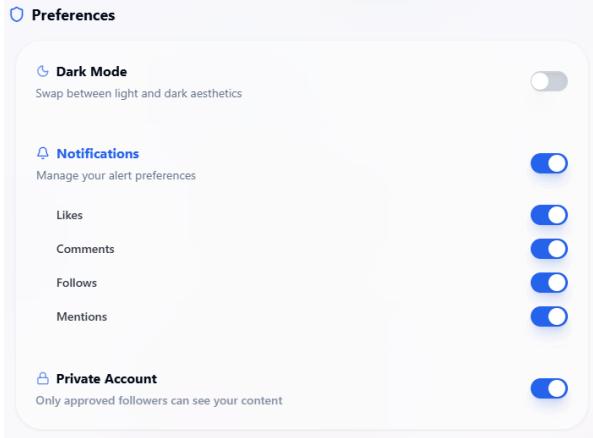


Figure 35: Préférences utilisateur

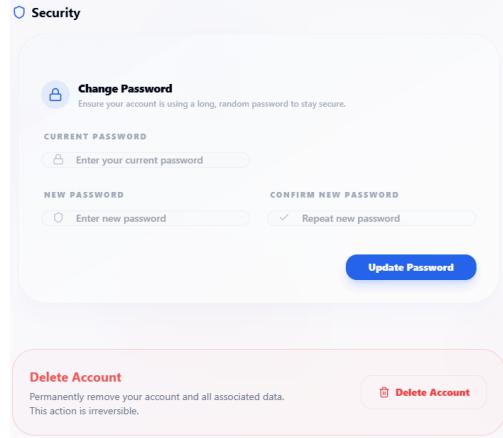


Figure 36: Paramètres de sécurité

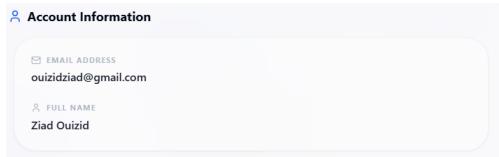


Figure 37: Informations du compte

L'organisation des paramètres s'articule autour de trois sections distinctes :

- **Informations du compte** : consultation de l'adresse électronique et de l'identité complète de l'utilisateur ;
- **Préférences** : activation ou désactivation du mode sombre, configuration fine des notifications par type d'événement (likes, commentaires, abonnements, mentions) et gestion du statut de confidentialité du compte ;
- **Sécurité** : modification du mot de passe avec validation sécurisée des critères de robustesse, ainsi qu'option de suppression définitive et irréversible du compte.

L'architecture responsive garantit une adaptation optimale aux différents formats d'écran, assurant une expérience homogène indépendamment du terminal utilisé.

11 Difficultés Rencontrées et Solutions

Le développement de la plateforme Social a nécessité la prise en compte de plusieurs défis techniques et organisationnels. Ces difficultés ont permis de confronter les choix architecturaux aux contraintes réelles d'une application web moderne et ont conduit à la mise en place de solutions adaptées, robustes et évolutives.

11.1 Synthèse des défis rencontrés

Le tableau suivant récapitule les principaux défis techniques et organisationnels ainsi que les solutions mises en œuvre.

Défi	Solution Mise en Œuvre
Gestion des notifications en temps réel	Intégration Socket.io avec émission ciblée par utilisateur
Stockage scalable des médias	Externalisation via MinIO avec stockage des URL en MongoDB
Contrôle de la confidentialité des contenus	Filtrage MongoDB + middlewares + rendu conditionnel frontend
Prévention des doublons d'engagement	Index unique composé (<code>user</code> , <code>thread</code> , <code>comment</code>)
Optimisation des performances de recherche	Index textuels sur <code>content</code> , <code>hashtags</code> , noms
Synchronisation de l'état entre composants	Redux Toolkit avec store centralisé
Persistance de l'authentification	JWT dans localStorage + mécanisme de réhydratation

Table 8: Synthèse des difficultés rencontrées et solutions apportées

11.2 Défis Techniques

11.2.1 Gestion des notifications en temps réel

L'un des principaux enjeux du projet concernait la livraison instantanée des notifications aux utilisateurs, sans recourir à des mécanismes de polling constant susceptibles d'impacter les performances du système.

Afin de répondre à cette problématique, la solution retenue repose sur l'intégration de **Socket.io**, permettant une communication bidirectionnelle en temps réel via WebSocket. Lorsqu'un événement survient, tel qu'un like ou un commentaire, le serveur émet immédiatement une notification vers la connexion socket de l'utilisateur concerné, assurant ainsi une expérience fluide et réactive.

11.2.2 Stockage scalable des médias

Le stockage des fichiers multimédias a constitué un autre défi majeur. L'enregistrement direct des images et vidéos dans MongoDB entraînait des documents volumineux, susceptibles de dégrader les performances et la maintenabilité de la base de données.

La solution adoptée consiste à externaliser le stockage des médias à l'aide de **MinIO**, un service de stockage d'objets compatible S3. Les fichiers sont stockés sur MinIO, tandis que seules leurs références (URL) sont conservées en base de données. Cette approche permet une séparation claire des responsabilités et favorise une montée en charge indépendante des données et des médias.

11.2.3 Contrôle de la confidentialité des contenus

La gestion de la confidentialité des comptes privés représentait une contrainte fonctionnelle importante. Les publications de ces comptes devaient être accessibles uniquement aux utilisateurs disposant d'une autorisation explicite.

Pour répondre à cette exigence, un mécanisme de filtrage multi-couches a été mis en place :

- filtrage des requêtes MongoDB en fonction du statut de suivi ;
- vérification des permissions via des middlewares dédiés ;
- rendu conditionnel des contenus côté frontend selon la relation entre les utilisateurs.

Cette approche garantit une protection cohérente des données, tant au niveau serveur que côté client.

11.2.4 Prévention des doublons d'engagement

Des problèmes de concurrence pouvaient conduire à des doublons lors des interactions, notamment lorsque plusieurs requêtes de type *like* étaient envoyées simultanément.

La solution retenue repose sur la création d'un index unique composé sur les champs (`user`, `thread`, `comment`) au sein de la collection `Like`. MongoDB assure ainsi l'unicité des engagements directement au niveau de la base de données, éliminant tout risque de duplication.

11.2.5 Optimisation des performances de recherche

La recherche de contenus et d'utilisateurs sur un volume important de données s'est révélée initialement coûteuse en termes de performance.

Afin d'optimiser ce processus, des index textuels ont été mis en place sur les champs `content`, `hashtags`, `firstName` et `lastName`. Ces index permettent la création d'un index inversé, offrant des performances de recherche significativement améliorées.

11.3 Défis Organisationnels

11.3.1 Synchronisation de l'état entre composants

La multiplication des composants nécessitant l'accès aux mêmes données posait un risque d'incohérence et de duplication de l'état.

Pour résoudre ce problème, une gestion centralisée de l'état a été mise en œuvre à l'aide de **Redux Toolkit**. L'ensemble des composants s'appuie sur un store unique, garantissant la cohérence des données et facilitant leur mise à jour.

11.3.2 Persistance de l'authentification

Un autre défi concernait la persistance de la session utilisateur lors du rafraîchissement de la page, entraînant des déconnexions involontaires.

La solution adoptée repose sur un mécanisme de réhydratation de l'authentification :

1. stockage du jeton JWT et des données utilisateur dans le `localStorage` ;
2. vérification de ces données au chargement de l'application ;
3. restauration de l'état Redux si le jeton est valide ;
4. validation systématique du jeton auprès du serveur.

Ce mécanisme assure une expérience utilisateur continue tout en maintenant un niveau de sécurité élevé.

12 Conclusion

Le projet **Social**, consacré à la conception et au développement d'une plateforme de réseau social web, constitue l'aboutissement de la formation *Full Stack Developer* dispensée par **Jobintech** en partenariat avec **Ynov Campus Maroc**. Il illustre la capacité à concevoir, implémenter et déployer une application web moderne en mobilisant l'ensemble des compétences techniques et méthodologiques acquises durant la formation.

12.1 Réalisations et apports techniques

La plateforme développée repose sur la stack **MERN** et intègre les fonctionnalités essentielles d'un réseau social moderne : authentification sécurisée par JWT, gestion des contenus et des médias via MinIO, interactions sociales (likes, commentaires, système de suivi), notifications en temps réel à l'aide de WebSocket, ainsi qu'un mécanisme de confidentialité pour les comptes publics et privés.

Sur le plan architectural, l'application s'appuie sur une API REST sécurisée, une base de données MongoDB optimisée par indexation, et une interface utilisateur responsive développée avec React. Ces choix garantissent une solution robuste, évolutive et maintenable, tout en offrant une expérience utilisateur fluide.

12.2 Compétences consolidées

Ce projet a permis de consolider des compétences clés en développement full stack, notamment la maîtrise de React, Node.js, Express et MongoDB, ainsi que l'application de bonnes pratiques architecturales et de sécurité (JWT, bcrypt, validation des données). L'adoption d'une organisation Agile, l'utilisation de Git pour le travail collaboratif et la production d'une documentation structurée ont renforcé une approche professionnelle du développement logiciel.

12.3 Perspectives d'évolution

Bien que pleinement fonctionnelle, la plateforme peut être enrichie par l'ajout de nouvelles fonctionnalités telles qu'une messagerie privée, des contenus éphémères, un système de recommandation intelligent ou une application mobile dédiée. Ces évolutions permettraient d'étendre l'usage de la plateforme vers un contexte de production plus large.

En conclusion, ce projet démontre la capacité à transformer des besoins fonctionnels en une solution technique cohérente et évolutive, conforme aux standards actuels du développement web, et constitue une base solide pour une insertion professionnelle dans le domaine.

« L'innovation distingue un leader d'un suiveur. »

– Steve Jobs

Annexes

Annexe A : Environnement Technique

Les technologies, bibliothèques et outils mobilisés dans le cadre du projet **Social** sont présentés ci-dessous.

Backend – Serveur et API

Le backend est implémenté en **Node.js (v18+)** avec le framework **Express.js (v5.2.1)**. La persistance des données est assurée par **MongoDB** et l'ODM **Mongoose (v9.0.2)**.

La communication temps réel utilise **Socket.io (v4.8.3)**. La sécurité s'appuie sur l'authentification **JWT**, le hachage **bcrypt**, ainsi que les middlewares **Helmet**, **CORS**, **express-rate-limit**, **express-mongo-sanitize** et **xss-clean**.

La validation des données utilise **Joi**, la gestion des fichiers repose sur **Multer** et le SDK **MinIO**, tandis que la journalisation est assurée par **Morgan**.

Frontend – Interface Utilisateur

L'interface est développée avec **React (v19.2.0)** et l'outil de build **Vite**. La gestion d'état utilise **Redux Toolkit (v2.11.2)** et la navigation s'appuie sur **React Router DOM (v7.11.0)**.

Le design repose sur **Tailwind CSS (v3.4.17)**, les composants **Radix UI** et les icônes **Lucide React**. Les animations utilisent **Framer Motion**, les notifications **React Hot Toast**, et les formulaires **React Hook Form**.

La communication API s'effectue via **Axios**, le temps réel côté client via **Socket.io-client**, et les rendus 3D avec **Three.js**.

Infrastructure et Outils

L'environnement est conteneurisé avec **Docker** et **Docker Compose**. Le stockage des médias utilise **MinIO** (compatible S3), l'administration de MongoDB s'effectue via **Mongo Express**, et les variables d'environnement sont gérées par **Dotenv**.

Le développement collaboratif s'appuie sur **Git/GitHub**. Les outils incluent **Visual Studio Code**, **Postman**, **MongoDB Express**, **Jira**. Le déploiement production est effectué sur **Railway**.

Annexe B : Ressources du projet

Les ressources associées au projet sont listées ci-dessous :

- Dépôt GitHub : <https://github.com/abdo-isfi/web-social-platform.git>
- Application déployée (Railway) : <https://threads-app.up.railway.app>

Annexe C : Guide d'Installation

Ce guide décrit les étapes nécessaires pour mettre en place et exécuter le projet localement.

1. Prérequis

Pour garantir le bon fonctionnement de l'application, les outils suivants doivent être installés :

- Docker et Docker Compose (conteneurisation)
- Node.js v18+ et MongoDB v6+ (optionnel, pour installation manuelle)

2. Installation via Docker

L'utilisation de Docker permet de lancer l'intégralité de l'écosystème (Frontend, Backend, Base de données, Stockage) avec une seule commande.

```
1 # ETAPE 1 : CLONER LE PROJET
2 git clone https://github.com/abdo-isfi/SOCIAL_MEDIA.git
3 cd SOCIAL_MEDIA
4
5
6 # ETAPE 2 : CONFIGURATION ENVIRONNEMENT
7 cd backend
8 cp .env.example .env
9 # Modifier le fichier .env avec les valeurs appropriées
10 cd ..
11
12
13 # ETAPE 3 : LANCEMENT DES CONTENEURS
14 docker-compose up --build -d
15
16 # Vérifier l'état des conteneurs :
17 docker ps
```

Listing 11: Installation complète du projet

3. Accès aux Services

Une fois les conteneurs démarrés, les services sont accessibles aux adresses indiquées dans la section Déploiement (voir tableau des points d'accès, page 32).

Identifiants par défaut :

- **MinIO** : minioadmin / minioadmin
- **Mongo Express** : admin / admin

4. Commandes de Maintenance

```
1 # Logs et surveillance
2 docker-compose logs -f                      # Logs en temps reel
3 docker-compose logs -f backend               # Logs d'un service
   spécifique
4 docker ps                                    # Etat des conteneurs
5
6 # Gestion des services
7 docker-compose down                         # Arreter les conteneurs
8 docker-compose down -v                      # Arreter + supprimer
   volumes
9 docker-compose restart                      # Redemarrer tous les
   services
10 docker-compose restart backend            # Redemarrer un service
11
12 # Reconstruction
13 docker-compose up -d --build              # Reconstruire apres
   modifications
14 docker-compose down --rmi all             # Nettoyer images
15 docker-compose up --build -d              # Reconstruire complètement
```

Listing 12: Commandes utiles pour la gestion du projet

5. Dépannage

En cas de dysfonctionnement lors de l'exécution de l'application, certaines vérifications simples permettent d'identifier rapidement l'origine du problème.

- **Conflits de ports** : vérifier et ajuster les ports exposés dans le fichier **docker-compose.yml**.
- **Connexion à MongoDB** : contrôler les variables d'environnement définies dans **backend/.env**.
- **Stockage des médias** : s'assurer du bon fonctionnement du service MinIO et de l'accessibilité du bucket.

Annexe D : Glossaire

API REST Architecture pour concevoir des services web reposant sur le protocole HTTP.

JWT JSON Web Token, standard utilisé pour l'authentification et la gestion des sessions.

MERN Stack Ensemble technologique composé de MongoDB, Express, React et Node.js.

NoSQL Type de base de données non relationnelle.

WebSocket Protocole de communication bidirectionnelle en temps réel.

Redux Bibliothèque de gestion d'état pour les applications React.

Docker Plateforme de conteneurisation facilitant le déploiement des applications.

MinIO Solution de stockage d'objets compatible avec l'API Amazon S3.

bcrypt Algorithme de hachage utilisé pour la sécurisation des mots de passe.

Socket.io Bibliothèque permettant la communication temps réel entre client et serveur.