Une image contenant capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.Une image contenant capture d’écran, conception, tissu

Le contenu généré par l’IA peut être incorrect.

**Adam LADJAL – Romain DELPECH – Eliote SADAOUI – Tom FREBAULT**

**RAPPORT PROJET WEB AVANCE**

Table des matières

[Contexte et objectifs 3](#_Toc201786032)

[Cahier des charges fonctionnel 3](#_Toc201786033)

[2.1 Regroupement des fonctionnalités en blocs métiers 3](#_Toc201786034)

[2.2 Dépendances logiques 4](#_Toc201786035)

[2.3 Parcours utilisateurs (exemples) 4](#_Toc201786036)

[Architecture technique et infrastructure 5](#_Toc201786037)

[3.1 Microservices 5](#_Toc201786038)

[3.2 Gateway NGINX 5](#_Toc201786039)

[3.3 Communication inter-services 6](#_Toc201786040)

[3.4 Base de données 6](#_Toc201786041)

[3.5 Conteneurisation & déploiement 6](#_Toc201786042)

[3.6 Sécurité & résilience 6](#_Toc201786043)

[Modèle de permissions (rôles) 7](#_Toc201786044)

[Technologies utilisées 7](#_Toc201786045)

[Méthodologie & déroulement du projet 8](#_Toc201786046)

[6.1 Planification & hiérarchisation 8](#_Toc201786047)

[6.2 Cycle de développement 8](#_Toc201786048)

[6.3 Outils & ressources 8](#_Toc201786049)

[Maquettes et schémas 9](#_Toc201786050)

[7.1 - Maquette d’écrans 9](#_Toc201786051)

[7.2 – Modèle de données MongoDB 10](#_Toc201786052)

# Contexte et objectifs

Le projet **Breezy** a pour ambition de proposer un réseau social minimaliste, inspiré de Twitter/X, spécifiquement conçu pour offrir une expérience rapide et fluide même sur des terminaux et connexions à faibles ressources. Bien qu’aucune catégorie d’utilisateurs n’ait été privilégiée a priori, l’application vise à répondre aux usages courants de micro-blogging : publication de messages courts, interactions (likes, commentaires, suivis) et consultation d’un fil d’actualité.

Les principaux critères de réussite reposent sur :

1. **La performance**, mesurée en temps de chargement des écrans et réactivité des actions utilisateur.
2. **La complétude du MVP**, c’est-à-dire l’implémentation de l’ensemble des fonctionnalités obligatoires définies.

# Cahier des charges fonctionnel

## 2.1 Regroupement des fonctionnalités en blocs métiers

| **Bloc métier** | **Fonctionnalités incluses** |
| --- | --- |
| **User (authentification & compte)** | • Création de compte (inscription)  • Connexion / déconnexion / rafraîchissement de token |
| **Profil** | • Consultation d’un profil (affichage bio, photo, stats)  • Recherche de profils• Suivre / se désabonner  • Liste abonnés / abonnements |
| **Posts & interactions** | • Publication de messages courts (≤ 280 caractères, emojis)  • Suppression de ses propres posts  • Flux chronologique (infinite scroll) des comptes suivis  • Like (toggle, compteur)  • Commentaires et réponses sans limite de profondeur |

## 2.2 Dépendances logiques

1. **User → Profil**

* Un profil ne peut être créé / consulté qu’après inscription et connexion.

1. **Profil → Posts**

* On peut voir le fil d’actualité (posts des suivis) uniquement une fois qu’on suit au moins un compte.

1. **Posts → Interactions**

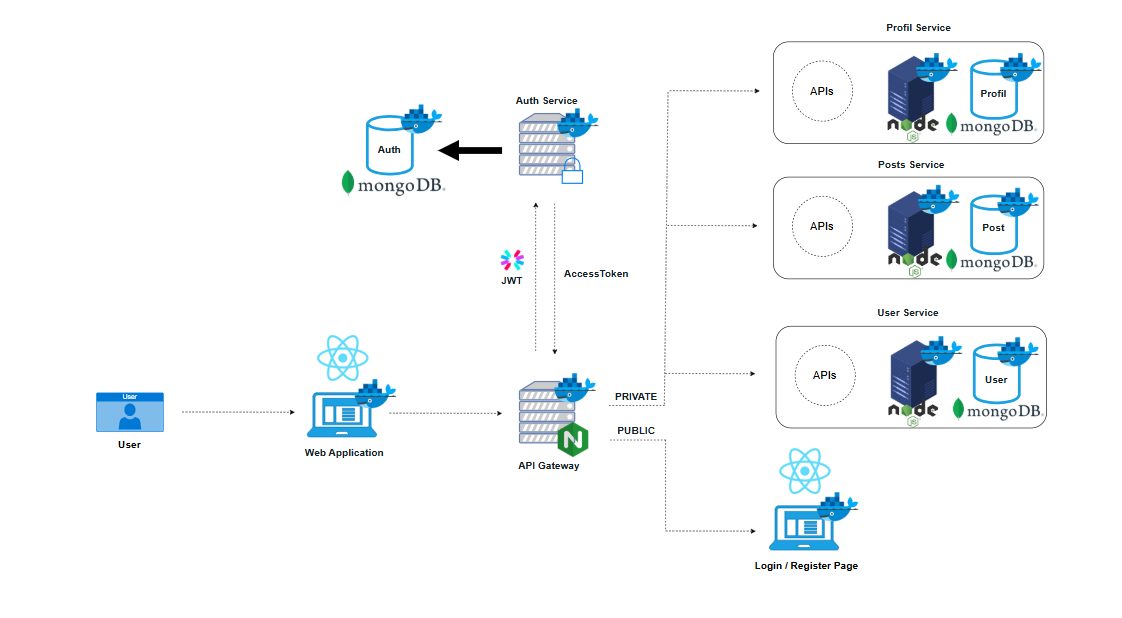
* /\*-+L’édition, la suppression, le like et le commentaire ne sont accessibles que si l’utilisateur est authentifié.

## 2.3 Parcours utilisateurs (exemples)

* **Inscription → Connexion → Création de post**
* **Connexion → Fil d’actualité (infinite scroll) → Like / Commentaire** **→ Réponse imbriquée**
* **Connexion → Recherche de profil → Follow / Unfollow**

*Pour chacun de ces parcours, les écrans comportent : formulaires avec contrôles (mot de passe, longueur de post), boutons activés/désactivés selon validité, messages d’erreur (format invalide, doublon, quota dépassé) et toasts en cas d’échec réseau.*

# Architecture technique et infrastructure



## 3.1 Microservices

Reverse-proxy unique exposé en front :

Routage des requêtes vers les services (/auth-service/…, /user-service/…, /profile-service/…, /post-service/…)

Gestion CORS et cache GET léger

Vérification et délégation JWT (header Authorization ou cookie HttpOnly)

## 3.2 Gateway NGINX

* Reverse-proxy unique exposé en front :
* Routage des requêtes (/auth-service/…, /profile-service/…, /post-service/…)
* Terminaison SSL, gestion CORS, cache GET léger
* Vérification et déléguation JWT (header Authorization ou cookie HttpOnly)

## 3.3 Communication inter-services

• Les services communiquent entre eux via des requêtes HTTP/JSON, grâce à Axios sur le réseau privé Docker (my-network).  
• Découverte de services via DNS Docker Compose (pas de broker externe).

## 3.4 Base de données

• Chaque microservice dispose de sa propre base MongoDB :

Auth\_DB : collection Auth\_User (e-mail, hash, rôle)

User\_DB : collection User (username, e-mail, date de création)

Profile\_DB : collection Profile (bio, photo, followers/following)

Post\_DB : collection Post (contenu, auteur, likes, replies)

• Les références entre documents dans des bases différentes se font explicitement via \_id dans le code, sans usage de .populate().

## 3.5 Conteneurisation & déploiement

* **Docker Compose** (réseau my-network, volumes MongoDB)
* Conteneurs : mongo, auth-service, profile-service, post-service, nginx, client
* **Pipeline CI/CD** à prévoir (GitHub Actions / GitLab CI) : build, tests, push, déploiement (docker-compose ou Kubernetes).

## 3.6 Sécurité & résilience

* Authentification stateless via **JWT** + **cookie HttpOnly**
* **Hash bcrypt** (cost ≥ 10) pour mots de passe
* **Rate-limiting** et **CORS** configurés sur NGINX
* **Health checks** (/health) pour monitoring et redémarrage automatique
* Réplication MongoDB en production pour tolérance aux pannes

# Modèle de permissions (rôles)

| **Fonctionnalité / Rôle** | **Visiteur** | **Utilisateur** | **Modérateur** | **Administrateur** |
| --- | --- | --- | --- | --- |
| Création de comptes (Fx1) | ✓ | ✗ | ✗ | ✓ |
| Authentification (Fx2) | ✗ | ✓ | ✓ | ✓ |
| Publication de messages (Fx3) | ✗ | ✓ | ✓ | ✓ |
| Affichage messages profil (Fx4) | ✗ | ✓ (seuls siens) | ✓ | ✓ |
| Flux chronologique (Fx5) | ✗ | ✓ | ✓ | ✓ |
| Like (Fx6) | ✗ | ✓ | ✓ | ✓ |
| Commenter (Fx7) | ✗ | ✓ | ✓ | ✓ |
| Répondre (Fx8) | ✗ | ✓ | ✓ | ✓ |
| Suivre / unfollow (Fx9) | ✗ | ✓ | ✓ | ✓ |
| Profil utilisateur (Fx10) | ✗ | ✓ | ✓ | ✓ |
| Liste des messages (Fx11) | ✗ | ✓ | ✓ | ✓ |

# Technologies utilisées

| **Domaine** | **Technologies** |
| --- | --- |
| **Back-end** | Node.js · Express.js · JWT · MongoDB · Mongoose |
| **Sécurisation** | JWT stateless · bcrypt · CORS · gestion d’erreurs |
| **Deployment** | Docker · Docker Compose |
| **Front-end** | React.js · Next.js · Tailwind CSS · Axios · React Router |
| **UX/UI & Réactivité** | Mobile-first · Responsive · gestion d’erreurs UI |
| **Sessions** | Stockage JWT (cookie HttpOnly) · redirections post-auth |
|  |  |

# Méthodologie & déroulement du projet

## 6.1 Planification & hiérarchisation

* Phase 1 : maquette UI & schéma de données
* Phase 2 : développement des microservices par bloc (User → Profile → Posts)
* Phase 3 : intégration via Docker Compose et tests end-to-end

## 6.2 Cycle de développement

* **Versioning** : Git + GitHub
* **Tests** : Postman pour APIs, revues de code
* **Intégration** : rebuild & redeploy auto via Docker Compose à chaque push

## 6.3 Outils & ressources

* **Environnement** : Node.js, MongoDB conteneurisé
* **CI/CD projeté** : GitHub Actions / GitLab CI (build, tests, images Docker)
* **Monitoring** : health checks exposés, logs centralisés

# Maquettes et schémas

## 7.1 - Maquette d’écrans

Une image contenant texte, logiciel, Icône d’ordinateur, conception

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, Site web

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

7.2 – Modèle de données MongoDB

Ce diagramme illustre les quatre collections principales et leurs liaisons :

* **Authentication** stocke les creds (email, hash, rôles, tokens) et référence un **User** via userId.
* **User** contient les métadonnées (username, email, statut, date de création).
* **Profile** enrichit le **User** avec bio, photo, followers/following, etc.
* **Post** référence l’auteur (userId), contient le contenu, les likes (liste d’ObjectId) et les replies.

Les flèches montrent les références \_id employées pour peupler les données au runtime.

Une image contenant texte, capture d’écran, diagramme, nombre

Le contenu généré par l’IA peut être incorrect.