DOCUMENT TECHNIQUE

EDOUARD Axel

Promotion : M1 2025

Campus : Campus distanciel

Table des matières

1. Introduction

Ce document décrit la configuration requise, les paramètres à renseigner, le guide de déploiement, les justifications techniques, les diagrammes UML et le schéma de base de données pour l'application SUPRSS.

2. Informations et éléments à renseigner

2.1Prérequis

- Docker et Docker Compose : assurent des déploiements reproductibles.
- Accès réseau à MongoDB : conteneur local ou service managé.
- Identifiants OAuth Google : pour proposer le SSO en plus des comptes locaux.
- Dépôt GitHub du projet : https://github.com/axel1edouard/SUPRSS

2.2 Variables d'environnement (backend)

Nom	Exemple / Valeur	Description
PORT	4000	Port HTTP de l'API Express
MONGODB_URI	mongodb://mongo:27017/suprss	URI MongoDB
JWT_SECRET	change-me	Secret de signature des tokens/cookies
CORS_ORIGIN	http://localhost:5173	Origine autorisée (CORS)
NODE_ENV	production	Mode d'exécution
OAUTH_GOOGLE_CLIENT_ID	xxxx.apps.googleusercontent.com	Client ID Google (optionnel)
OAUTH_GOOGLE_CLIENT_SECRET	*****	Client Secret Google (optionnel)
OAUTH_GOOGLE_REDIRECT_URI	http://localhost:4000/api/auth/google/callback	Callback OAuth (optionnel)

2.3 Variables d'environnement (frontend)

Nom	Exemple / Valeur	Description
VITE_API_BASE	http://localhost:4000	Base URL de l'API

2.4 Sécurité et gestion des secrets

• Ne pas committer les secrets : utiliser des fichiers .env locaux et un coffre-fort de secrets en prod.
• Sessions en cookie httpOnly (Secure en HTTPS), SameSite adapté (Lax par défaut, None si cross-site).
• CORS restreint à l'origine du frontend. Activer helmet, rate limiting et journalisation des accès.
• Valider et assainir toutes les entrées (Joi/Zod) ; journaliser les erreurs sans divulguer d'infos sensibles.

3. Guide de déploiement

Le déploiement s'effectue via Docker Compose. La pile par défaut comprend trois services : frontend, backend et MongoDB.

3.1 Etapes rapides (production)

1) Préparer les .env (backend, frontend) à partir des .env.example et renseigner les valeurs.
2) Lancer le build et l'exécution : docker compose up -d --build
3) Vérifier : API sur http://localhost:4000, Frontend sur http://localhost:5173

3.2 Conteneurs & services

• frontend : build Vite/React servi en statique ; point d'entrée de l'UI.
• backend : API Express (auth, collections, feeds, articles, preview/proxy images).
• mongo : base de données MongoDB (persistance).

3.3 Journalisation & supervision

Les journaux applicatifs sont exposés sur la sortie standard (consultables via docker compose logs -f <service>).
Prévoir des checks de santé légers (ou des probes via reverse-proxy/orchestrateur) pour l'API.
Mettre en place des sauvegardes régulières de MongoDB avec mongodump et une rotation (rétention hebdo/mensuelle).

3.4 Sécurité runtime

Cookies de session **httpOnly** et Secure (HTTPS), SameSite=Lax par défaut (None si cross-site).
CORS restreint à l'origine du frontend. Activer **helmet**, **rate-limit** et la validation des entrées (Joi/Zod).
Les secrets sont injectés via **variables d'environnement** uniquement ; ne pas committer de fichiers .env.

4. Justification des langages et des librairies

• Backend — Node.js + Express : JSON natif, middleware mature (auth, sécurité, CORS), courbe d'apprentissage faible et performance suffisante pour une API orientée I/O.
• Frontend — React + Vite : build rapide, HMR en dev, structure de composants, écosystème vaste.
• Base — MongoDB : schéma souple, adapté aux données hétérogènes des flux RSS et aux évolutions rapides.

Librairies clés : Mongoose (ODM), jsonwebtoken (auth), bcrypt (hash), helmet/cors/rate-limit (sécurité), axios (HTTP), react-router (routing), rss-parser & cheerio (ingestion/preview), node-cron/agenda (planif.).

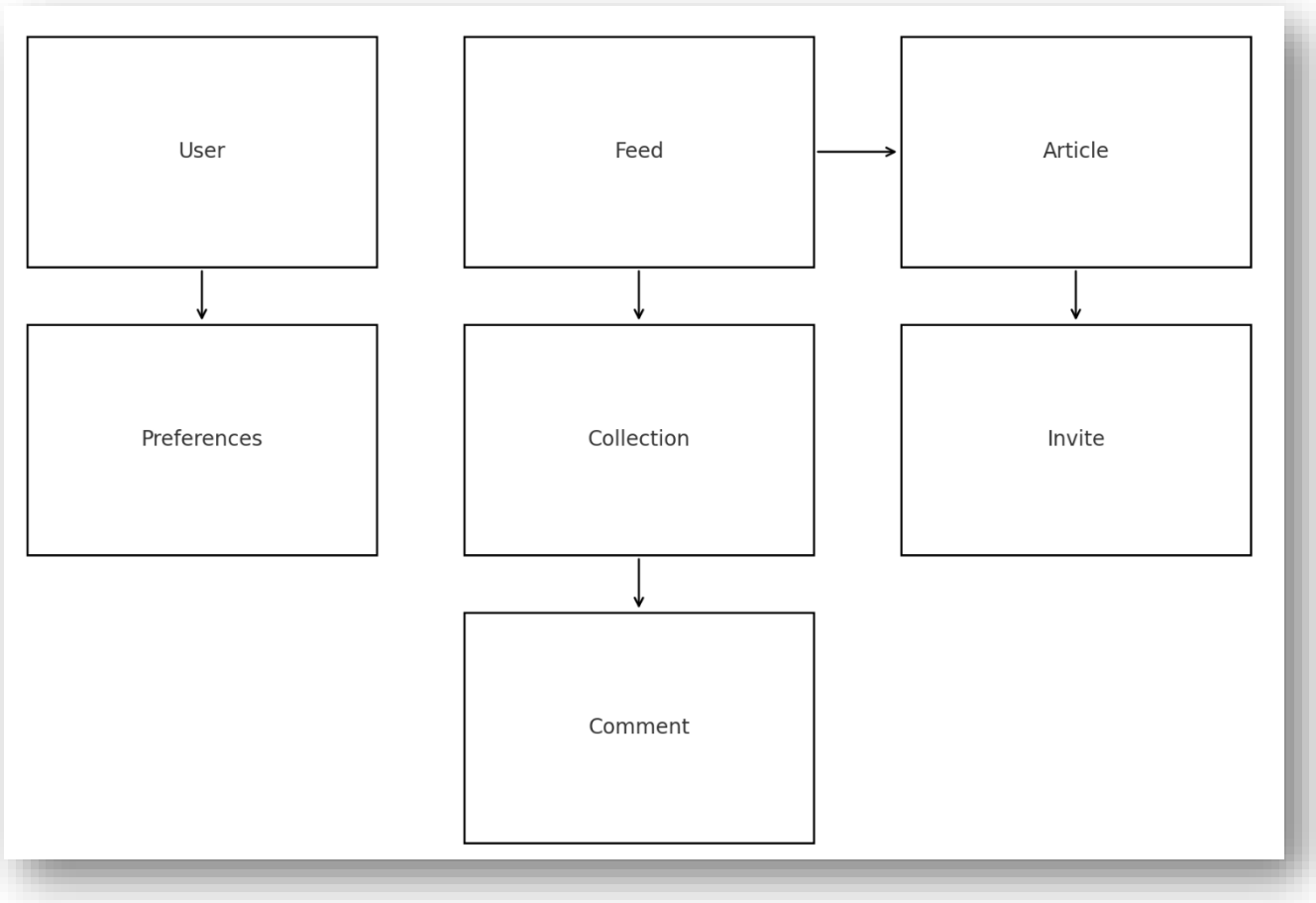
4.1 Considérations sécurité

• Stockage sûr des secrets et obfuscation des erreurs.
• Politiques de mots de passe et hachage fort (bcrypt).
• Contrôles d'accès par middleware et revocation de session.
• Durcissement HTTP (headers Helmet) et CORS strict.

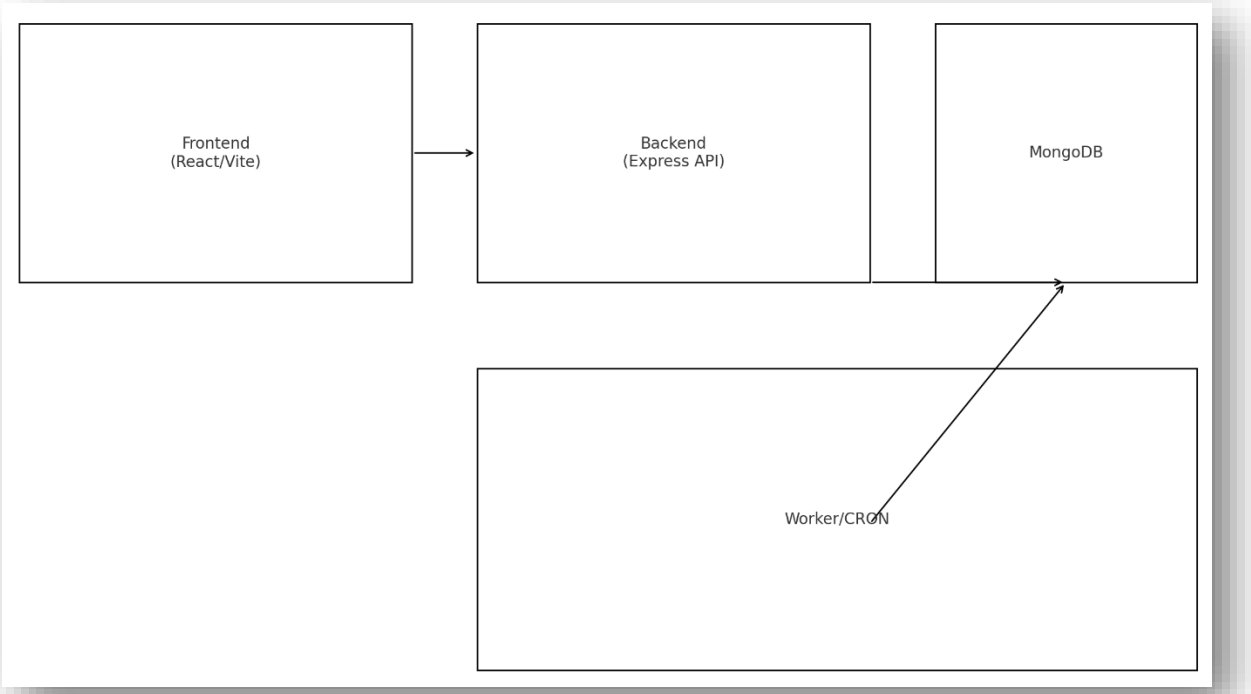
5. Diagrammes UML

Les diagrammes suivants donnent une vision synthétique du domaine (classes), de l'architecture (composants), des principaux scénarios (séquences), des états clés (invitation) et du déploiement.

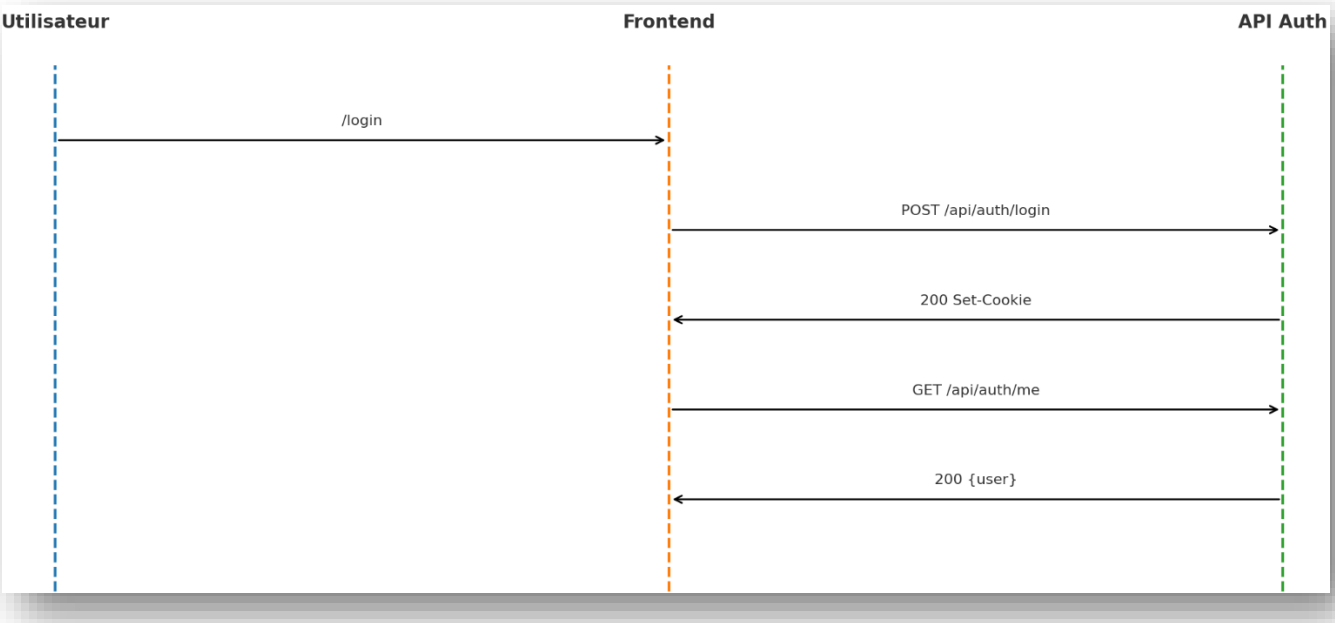
5.1 Modèle de domaine (Classes)



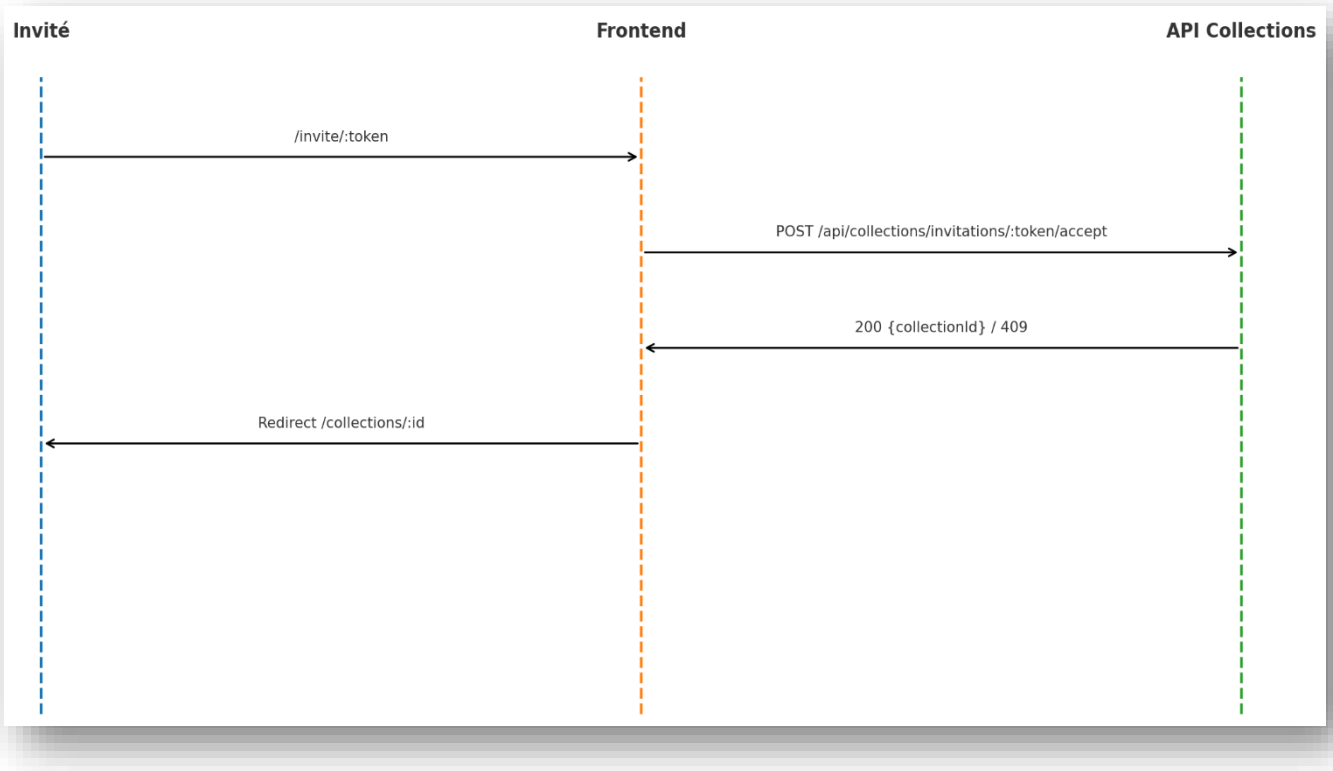
5.2 Composants (Architecture applicative)



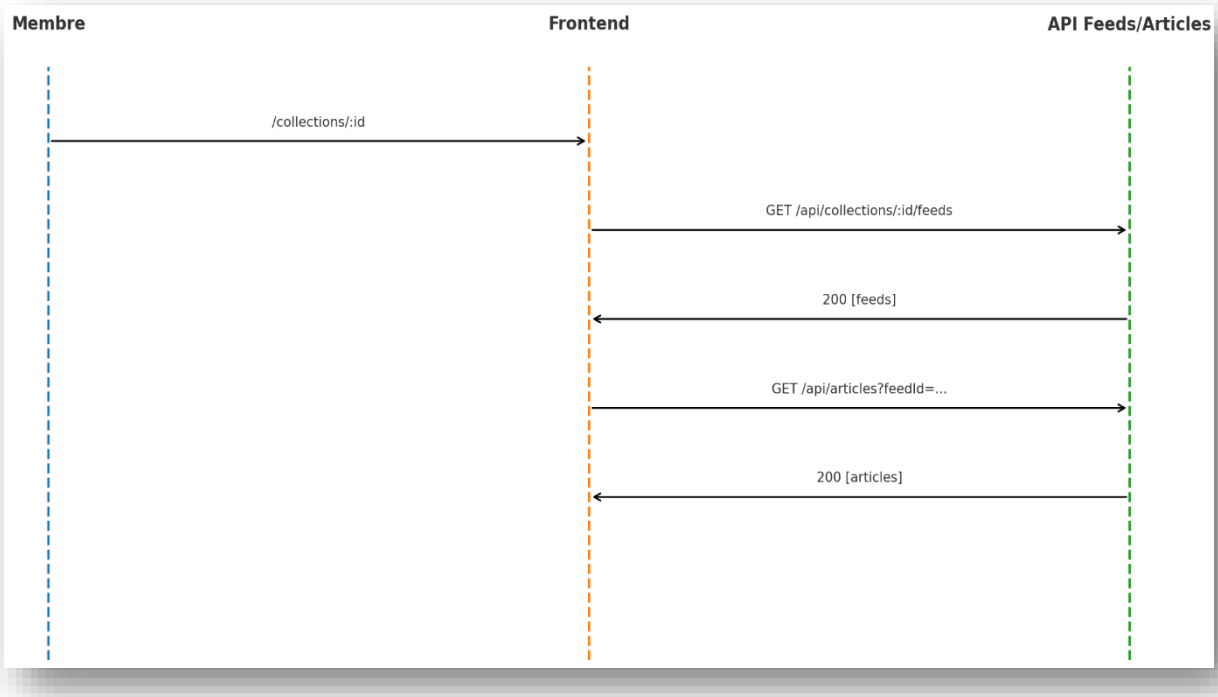
5.3 Séquence — Connexion



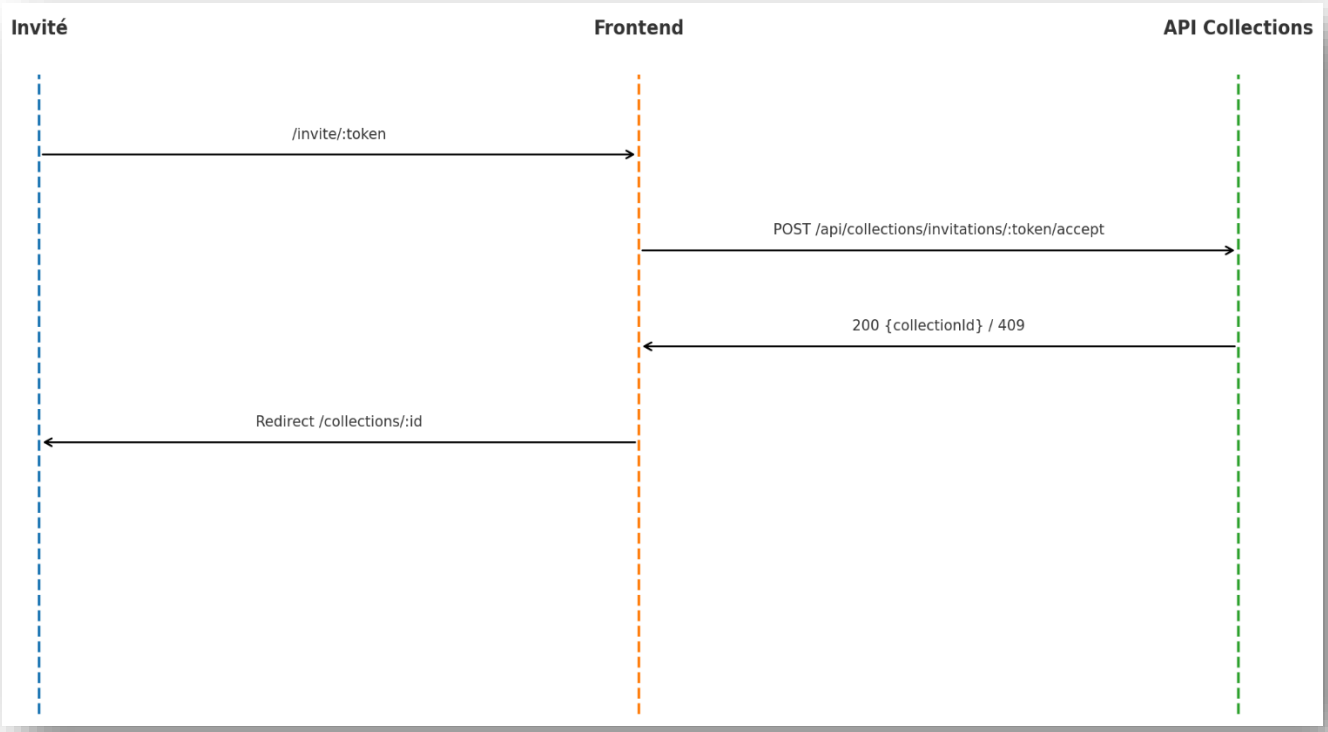
5.4 Séquence — Acceptation d’invitation



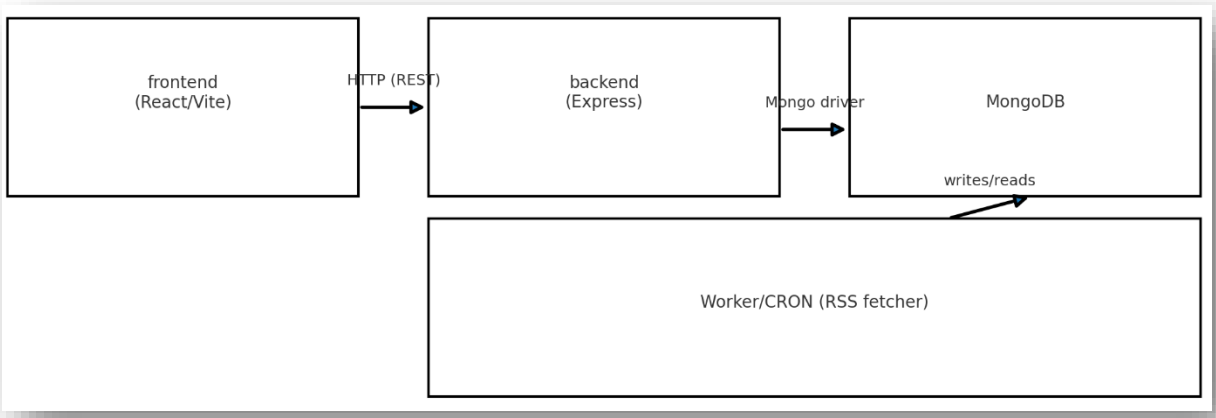
5.5 Séquence — Lecture d’articles d’une collection



5.6 États — Invitation



5.7 Déploiement (Docker)



6. Schéma de la base de données

MongoDB stocke les entités principales : users, preferences, collections, invitations, feeds, articles, comments.

Les index garantissent l'unicité (e-mails, tokens), optimisent les requêtes (tri par date, lookup par feed/collection) et empêchent certains doublons (idempotence des commentaires).

6.1 Collections et index (synthèse)

• users: email (unique), googleId (unique,sparse)
• preferences: userId (unique)
• collections: owner, members (multikey)
• collectionInvites: token (unique), expiresAt (TTL partiel), collectionId
• feeds: unicité contextuelle (url, ownerId) | (url, collectionId)
• articles: unique (feedId, link) ou (feedId, guid), publishedAt
• comments: (collectionId, clientId) unique partiel

6.2 Relations (récapitulatif)

User–Preferences (1–1), User–Feed (1–N) pour les flux personnels, Collection–Feed (1–N) pour les flux partagés, Feed–Article (1–N), Collection–Comment (1–N), Collection–Invite (1–N), Article–User (N–N via états lu/favori).