Wireframe

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

ULM

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Stack Technique

**Backend**

* **FastAPI** (Python 3.11) - API REST
* **PostgreSQL** - Base de données
* **SQLAlchemy** - ORM
* **feedparser** - Parsing RSS
* **bcrypt** - Mots de passe

**Frontend**

* **React.js** - Interface utilisateur
* **Axios** - Appels API
* **Bootstrap** - CSS simple

**Déploiement**

* **Docker + Docker Compose**
* **Git**

Frontend (React) ↔ API (FastAPI) ↔ Database (PostgreSQL)

Structure du projet simplifiée

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Jours 1-2 :** Setup + validation mentor  
**Jours 3-4 :** Backend API fonctionnel  
**Jours 5-6 :** Frontend basique  
**Jour 7 :** Intégration complète

**= Une démo fonctionnelle en 1 semaine !**

A screenshot of a computer

AI-generated content may be incorrect.

**🔧 ÉTAPE 2 : Backend FastAPI complet**

**Base de données (models.py) :**

* ✅ **9 tables** : users, collections, user\_collections, rss\_feeds, articles, user\_articles, categories, feed\_categories, comments, messages
* ✅ **Relations complexes** : One-to-Many, Many-to-Many avec tables d'association
* ✅ **Contraintes** : clés primaires, étrangères, unique
* ✅ **Timestamps** automatiques

**API d'authentification (auth.py + routers/auth.py) :**

* ✅ **POST /auth/register** : Création de compte avec validation
* ✅ **POST /auth/login** : Connexion avec JWT
* ✅ **GET /auth/me** : Profil utilisateur (protégé)
* ✅ **PUT /auth/me** : Mise à jour profil (protégé)
* ✅ **Sécurité** : bcrypt pour mots de passe, JWT avec expiration
* ✅ **Validation** : Pydantic schemas avec règles métier

**Configuration (database.py, main.py) :**

* ✅ **PostgreSQL** avec SQLAlchemy
* ✅ **CORS** configuré pour React
* ✅ **Documentation automatique** Swagger/OpenAPI
* ✅ **Dependency injection** pour DB et auth

**🎨 ÉTAPE 3 : Frontend React complet**

**Authentification (AuthContext.js + api.js) :**

* ✅ **Context React** pour état global
* ✅ **Axios interceptors** pour token automatique
* ✅ **localStorage** pour persistance
* ✅ **Gestion erreurs** 401 avec redirection

**Components React :**

* ✅ **Login.js** : Formulaire de connexion avec validation
* ✅ **Register.js** : Formulaire d'inscription complet
* ✅ **Dashboard.js** : Page d'accueil avec stats et actions
* ✅ **Profile.js** : Gestion du profil utilisateur
* ✅ **Navbar.js** : Navigation avec dropdown Bootstrap

**Routing et protection :**

* ✅ **React Router** : navigation entre pages
* ✅ **ProtectedRoute** : routes nécessitant authentification
* ✅ **PublicRoute** : redirection si déjà connecté
* ✅ **Navigation conditionnelle** selon statut auth

**🐳 ÉTAPE 4 : Déploiement Docker**

**Containerisation :**

* ✅ **3 services** : database (PostgreSQL), backend (FastAPI), frontend (React)
* ✅ **docker-compose.yml** avec dépendances et volumes
* ✅ **Health checks** pour PostgreSQL
* ✅ **Hot reload** en développement
* ✅ **Variables d'environnement** sécurisées

**🧪 ÉTAPE 5 : Tests et validation**

**Fonctionnalités testées :**

* ✅ **Création de compte** via API et interface
* ✅ **Connexion/déconnexion** complète
* ✅ **Persistance de session** avec refresh
* ✅ **Navigation** entre pages protégées
* ✅ **Modification de profil** fonctionnelle
* ✅ **Documentation API** accessible et interactive