Rapport du Projet BAWE

Arthur PINEL - ENSIIE FISA

1. Introduction

Dans le cadre du module *Bases du Web*, j'ai développé un mini réseau social nommé **FEUR**. Le but était de créer un site complet comprenant une partie client, une partie serveur, une base de données, un système de traduction, un thème jour/nuit et des requêtes SQL variées.

J'ai souhaité profiter de ce projet pour explorer et mettre en œuvre des technologies que je n'avais jamais utilisées auparavant, c'est alors ma première fois avec chacune d'entre elles.

2. Technologies utilisées

Backend : Rust avec le framework Axum. Ce choix me permettait d'utiliser un langage moderne, sécurisé et performant, tout en explorant une technologie que je voulais apprendre.

Frontend : Flutter Web. J'ai choisi Flutter pour bénéficier d'une interface moderne ainsi que de sa capacité à se compiler sur diverses plateformes. C'est aussi une technologie que je voulais explorer.

Base de données : PostgreSQL avec trois tables :

- `users` : contient les informations des utilisateurs.
- `posts` : représente les messages publiées.
- `user_likes` : contient les likes éffectué par les utilisateurs.

3. Fonctionnement général du site

L'utilisateur peut intéragir avec l'application via :

- Sur la page via des **boutons** et des **formulaires**,
- Les **deux boutons** de thème et de langue disponible sur le côté droit de la **barre horizontale** en haut de l'écran,
- Le **bouton d'action** en bas à droite de l'écran.

Avec ces outils, le site propose les pages et fonctionnalités suivantes :

- **Accueil :** créer et consulter les publications.
- **Connexion et inscription :** création de compte, login.
- **Profil :** informations de l'utilisateur et suppression du compte.
- **Admin :** permet aux adminstrateur de gérer les comptes.

Accedez au compte adminstrateur :

- login : `admin`
- mot de passe : `admin`

4. Détails techniques du code

Authentification et sécurité :
Le backend utilise un système d'authentification basé sur des jetons `JWT`. Lors de la connexion, le serveur génère un jeton signé contenant l'identifiant de l'utilisateur. Ce jeton est ensuite envoyé au client et stocké en toute sécurité dans le `localStorage`. À chaque requête nécessitant une authentification, le frontend ajoute automatiquement un header `Authorization: Bearer <token>`.

Côté serveur, un middleware Axum intercepte les requêtes, vérifie la validité du JWT, extrait l'identifiant utilisateur et le transmet handler sous forme d'une structure `AuthUser`. Cela permet d'avoir des routes protégées.

Lazy Loading :
Pour optimiser les performances du frontend, un système de *lazy loading* a été mis en place : la page d'accueil ne charge qu'un nombre limité de posts et charge les suivants au fur et à mesure que l'utilisateur fait défiler la page. Cela réduit les temps de chargement initiaux et rend l'application fluide même avec de nombreuses données.

5. Bugs mineures

- Le like peut disparaître lors d'un changement de résolution (mobile → desktop).
- Le thème jour/nuit peut parfois ne pas s'appliquer immédiatement.
- Il est possible de supprimer son propre compte (fonctionnalité volontaire mais perfectible).

6. Justification des Contraintes Techniques

Conformément à la possibilité de justification fournie dans le rapport, le choix technologique impose un léger dépassement du nombre maximal de fichiers sources non-binaires :

- **Backend (Rust) :** Nous comptons **12 fichiers** (au lieu de 10) dans le répertoire `/src`. Ce dépassement est dû à l'architecture de **Rust** qui exige l'utilisation de fichiers `mod.rs` pour la structuration des modules.
- **Frontend (Flutter) :** Nous comptons **16 fichiers** (au lieu de 15) dans le répertoire `/lib`. Ce dépassement est lié à la génération automatique de fichiers sources par **Flutter** pour la gestion centralisée des traductions.

Ce surplus est principalement lié à la gestion de l'**authentification** dans l'architecture des frameworks choisis.

Enfin, des fichiers HTML, JavaScript et CSS sont présents dans le répertoire `/web`, ces derniers sont entièrement **optionnels** et ne servent qu'à créer une interface de chargement personnalisée, sans être nécessaires au fonctionnement de l'application.

7. Conclusion

Ce projet m'a permis de développer une application web complète, **FEUR**, en utilisant une stack moderne : **Rust (Axum)**, **Flutter Web** et **PostgreSQL**.

J'ai trouvé particulièrement intéressant et formateur de travailler avec **Rust** et **Flutter**, deux technologies aux philosophies complètement opposées (sécurité/bas niveau contre développement rapide/multiplateforme). Ce challenge a été très enrichissant et je suis content d'avoir atteint les objectifs fixés pour le projet.

L'ensemble du code a été assurée via **Git**, vous pouvez consulter le dépot via le lien ci-dessous :

`[https://github.com/LePeruvienn/projet-bawe]`

Merci d'avoir lu !

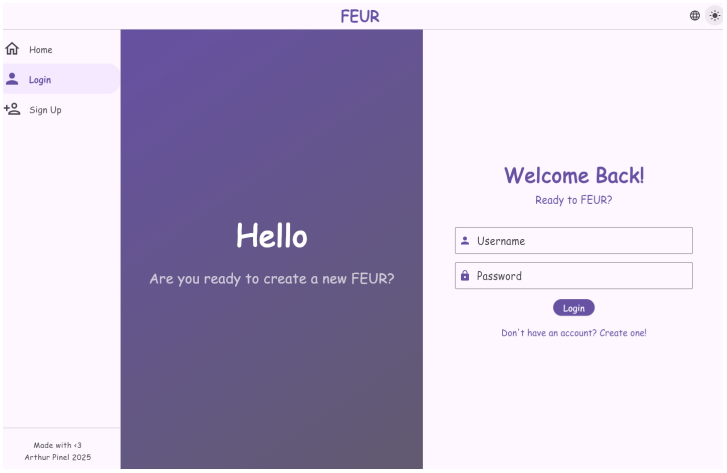

Figure 1: Thème clair de la page de connection

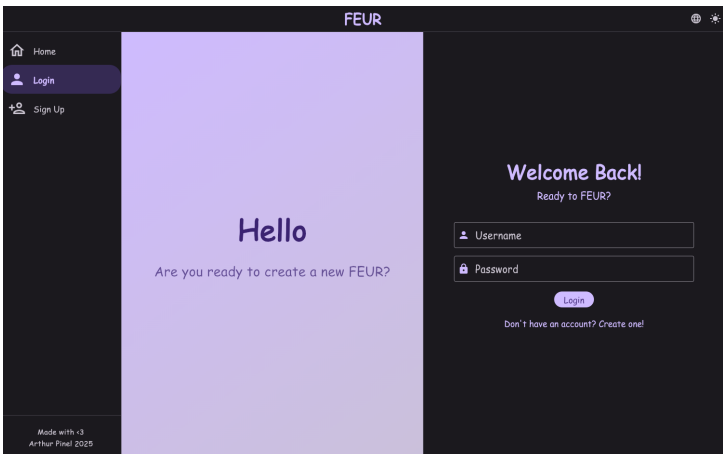

Figure 2: Thème sombre de la page de connection

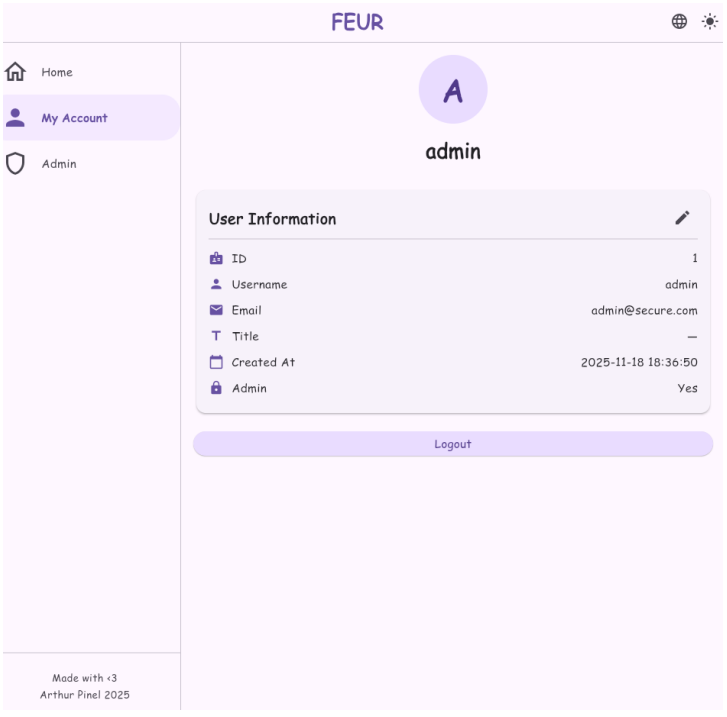

Figure 3: Utilisateur consultant son profil avec le thème clair

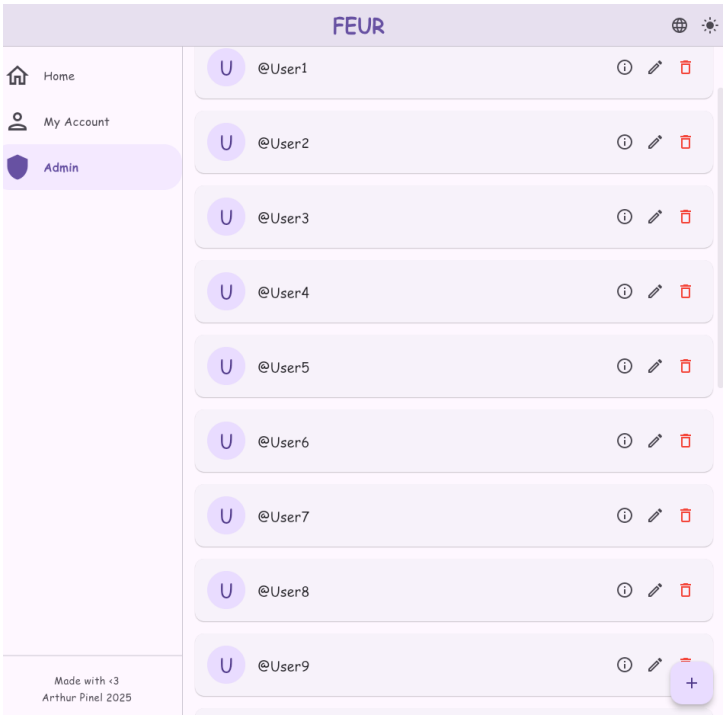

Figure 4: Thème clair de la page d'adminisatration

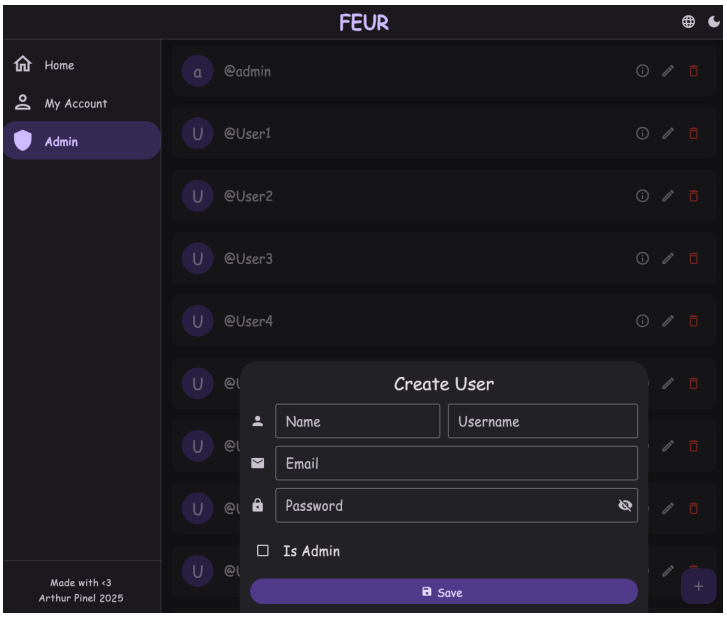

Figure 5: Utilisateur en train de créer un nouvelle utilisateur via la page admin

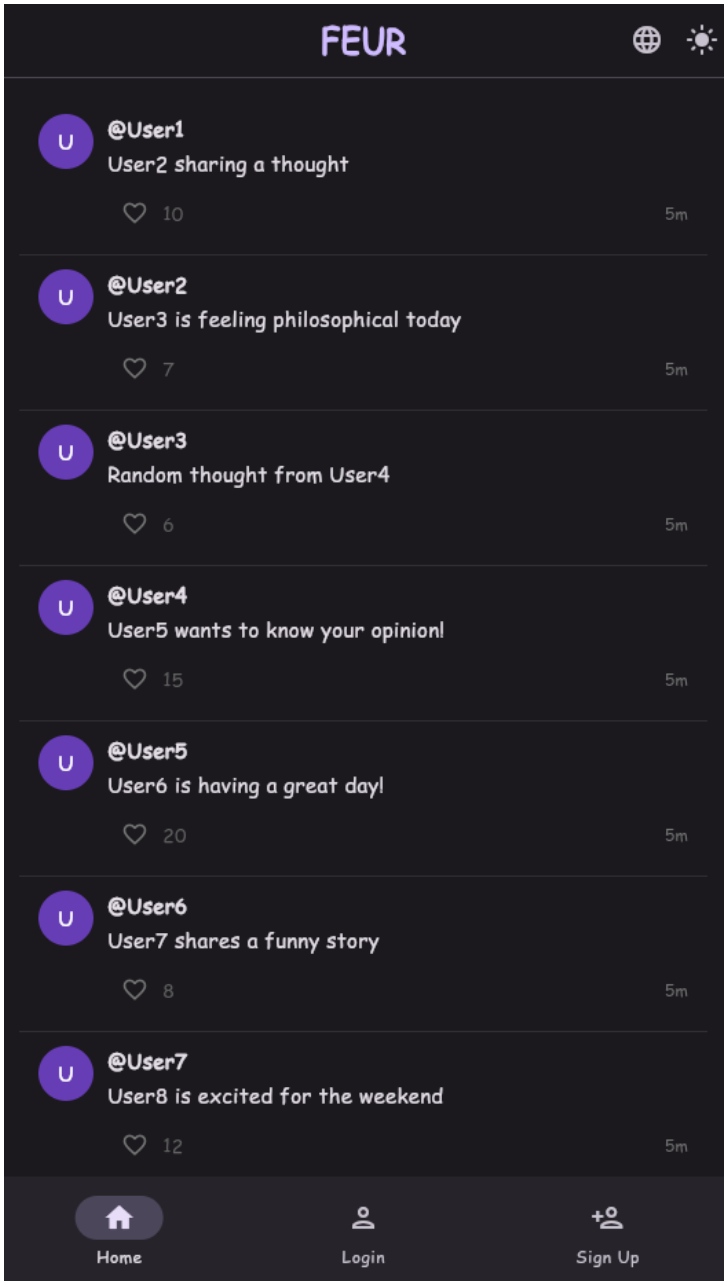

Figure 6: Utilisateur non connecté sur mobile consultant les publication avec le thème sombre