ARGENTBANK

# Utilisez une API pour un compte utilisateur bancaire avec React

# Sommaire

# La mission

- **Phase 1 : Authentification des utilisateurs** - Création d'une application web permettant aux clients de se connecter et de gérer leurs comptes et leur profil.
- **Phase 2 : Transactions** - Il s’agirait de spécifier les endpoints d’API nécessaires pour une éventuelle deuxième mission une fois que nous aurons terminé la première.

# Phase 1 : Authentification des utilisateurs

- Convertir le HTML statique et le CSS pour la page d'accueil, la page de connexion et la page de profil en application react.
- Utiliser Redux pour gérer le state de l'ensemble de l'application.
- Ce que doit faire l'application (voir les détails pour chacune sur nos modèles de GitHub Issues) :
  - L'utilisateur peut visiter la page d'accueil
  - L'utilisateur peut se connecter au système
  - L'utilisateur peut se déconnecter du système
  - L'utilisateur ne peut voir les informations relatives à son propre profil qu'après s'être connecté avec succès
  - L'utilisateur peut modifier le profil et conserver les données dans la base de données

# Les outils utilisés

- Vite : Plus rapide que CRA
- Typescript : afin d’éviter les erreurs cachées et de faciliter le développement grâce à sa prise en charge par les IDE (lisibilité, auto-completion etc…).
- ESLint et Prettier : afin de trouver et de corriger les erreurs et aussi de formater le code.
- React router, RTK, react-hook-form, axios

# L’authentification

- Stockage du token dans le local storage avec une date d’expiration dépendante du choix « remember me »
- Utilisation d’axios interceptor pour ajouter ce token dans les en-tête des requêtes.
- Création des services d’authentification (login et logout).
- Utilisation d’un état « auth » pour gérer l’état d’authentification dans l’application.
- HOC pour les routes protégées.

# Le logout

- Suppression du token dans le local storage
- Réinitialisation des états
- Pour une application banquaire, il y aura probablement un logout asynchrone coté server. C’est pourquoi j’ai  d'ores et déjà implémenté un logout asynchrone dans l’application,

# La page profil

- Utilisation d’un état « user »
- Création des services (getProfile et updateProfile).
- Création du formulaire d’édition du profil.

# Phase 2 : Transactions

- Fournir un document décrivant les API proposées pour les transactions, en suivant les directives de Swagger.
- Spécifier pour chaque endpoint de l'API :
  - La méthode HTTP (ex. : GET, POST, etc.
  - La route (ex. : /store/inventory)
  - La description de ce à quoi correspond l'endpoint (ex. : Liste des transactions)
  - Les paramètres possibles pour tenir compte des différents scénarios (ex. : itemId (facultatif) = ID de l'article spécifique à demander à la base de données d'inventaire).
  - Les différentes réponses avec les codes de réponse correspondants qui ont un sens pour cet endpoint (ex. : 404 : réponse d'erreur d'article inconnu).

# Les endpoints

- Liste des transactions :
  - Requête post vers /user/transactions
- Transaction détaillé :
  - Requête post vers /user/transaction avec l'id dans le body
- Modification d'une transaction
  - Requête put vers /user/transaction avec l'id et les données à modifier dans le body