Rapport de Projet : Application E-Banking

Architecture des Composants d'Entreprise

Réalisé par : **Amine içame - Salma BenOmar - Mohamed lakhessassi**
Encadré par : Pr. Oussama HAMAL

Année Universitaire 2024-2025

Table des matières

Chapitre 1

Introduction

1.1 Contexte du projet

Dans le cadre du module "Architecture des Composants d'Entreprise", nous avons réalisé une application bancaire complète "Full Stack". L'objectif était de mettre en œuvre une architecture distribuée robuste, sécurisée et modulaire.

1.2 Objectifs techniques

Le projet repose sur les technologies standards de l'industrie :
— **Backend :** Spring Boot 3 (Java 17), Spring Security 6, Spring Data JPA.
— **Frontend :** React JS 18, Axios, Bootstrap/Material UI.
— **Base de données :** MySQL 8.
— **Sécurité :** Authentification Stateless via JWT (JSON Web Tokens).

1.3 Fonctionnalités réalisées

L'application permet de gérer deux types d'acteurs :

1. **Agent (Admin) :** Gestion des clients, des comptes bancaires et validation des nouvelles inscriptions.
2. **Client (User) :** Consultation de solde, historique des opérations, virements.

Chapitre 2

Conception et Architecture Backend

2.1 Architecture en Couches

Nous avons respecté une architecture propre (Clean Architecture) pour assurer la maintenabilité :

- **Web Layer (Controllers) :** Expose les APIs REST.
- **Service Layer :** Contient la logique métier (Virements, Activation de compte).
- **DAO Layer (Repositories) :** Gère l'accès aux données via Spring Data JPA.
- **Entities :** Représente le schéma de la base de données.
- **DTOs :** Objets de transfert pour découpler les entités de l'API.

2.2 Modèle de Données (Entités)

Les principales entités du système sont :

- **AppUser / AppRole :** Gestion de la sécurité et des utilisateurs.
- **Customer :** Informations personnelles du client bancaire.
- **BankAccount :** Compte bancaire (relié à un Customer).
- **AccountOperation :** Historique des transactions (Débit/Crédit).

2.3 Implémentation des Services Métiers

Le service `BankAccountServiceImpl` gère la complexité transactionnelle. Exemple de la méthode de virement :

```
@Override
public void transfer(String idSource, String idDest, double amount) {
    debit(idSource, amount, "Transfert vers " + idDest);
    credit(idDest, amount, "Transfert de " + idSource);
}
```

Listing 2.1 – Logique de virement transactionnel

Chapitre 3

Sécurité Avancée et Workflow

La sécurité est le cœur de ce projet. Nous avons implémenté une sécurité **Stateless** basée sur JWT.

3.1 Configuration Spring Security

Nous avons configuré une chaîne de filtres (`SecurityFilterChain`) pour :

1. Désactiver le CSRF (inutile en Stateless).
2. Autoriser CORS (pour React).
3. Imposer l'authentification pour toutes les requêtes sauf `/auth/login` et `/auth/register`.

3.2 Authentification par Email

Pour plus de réalisme, nous avons modifié `UserDetailsServiceImpl` pour authentifier les utilisateurs via leur **Email** plutôt que leur nom d'utilisateur.

3.3 Workflow d'Inscription et Validation

Nous avons mis en place un processus d'inscription réaliste (KYC - Know Your Customer) :

1. **Inscription :** Le visiteur remplit le formulaire. Le compte est créé avec `active = false`.
2. **Attente :** Si l'utilisateur tente de se connecter, le système refuse l'accès ("Compte en attente de validation").
3. **Validation (Admin) :** L'agent consulte la liste des inactifs et valide le compte.
4. **Création Automatique :** Lors de la validation, le système déclenche automatiquement :
   — L'activation du login.
   — La création du profil `Customer`.
   — La création d'un `BankAccount` initial.

```java
@PutMapping("/activate/{username}")
    public void activate(@PathVariable String username){
        // 1. Activation de l'utilisateur (S curit )
        accountService.activateUser(username);

        // 2. Automatisme : Cr ation du Client et du Compte Bancaire
        try {
            // A. On r cup re les infos de l'utilisateur inscrit
            AppUser appUser = appUserRepository.findByUsername(username)
    ;

            // B. On pr pare le Client Bancaire (Customer)
            CustomerDTO customerDTO = new CustomerDTO();
            customerDTO.setFirstname(appUser.getUsername());
            customerDTO.setLastname("Nouveau Client"); // Valeur par
    d faut
            customerDTO.setEmail(appUser.getEmail());
            customerDTO.setIdentityNumber("CIN-" + appUser.getUserId().
    substring(0, 8)); // Faux CIN bas  sur l'ID

            // C. On sauvegarde (Gr ce  notre modif,  a  ne plantera
    pas si l'email existe d j  !)
            CustomerDTO savedCustomer = bankAccountService.saveCustomer(
    customerDTO);

            // D. On v rifie s'il a d j  un compte pour ne pas en
    cr er un deuxi me inutilement
            // On utilise la m thode qu'on a cr  e tout  l'heure
    pour le Dashboard
            try {
                var existingAccounts = bankAccountService.
    getBankAccountByEmail(savedCustomer.getEmail());
                if(existingAccounts.isEmpty()) {
                    // E. Cr ation du compte SEULEMENT s'il n'en a pas
                    bankAccountService.saveBankAccount(0, 0,
    savedCustomer.getId());
                    System.out.println("Compte bancaire cr  pour " +
    username);
                } else {
                    System.out.println("L'utilisateur " + username + " a
     d j  un compte, on ne fait rien.");
                }
            } catch (Exception e) {
                // Fallback si la v rification  choue , on tente quand
    m me la cr ation
                bankAccountService.saveBankAccount(0, 0, savedCustomer.
    getId());
            }

        } catch (Exception e) {
            // On log l'erreur mais on ne bloque pas l'activation du
    compte utilisateur
            System.err.println("Erreur Warning : Le compte bancaire n'a
    pas pu  tre  cr  auto : " + e.getMessage());
            e.printStackTrace();
        }
```

Listing 3.1 – Extrait du contrôleur d'activation

Chapitre 4

Développement Frontend (React)

4.1 Architecture React

L'application React est structurée en composants modulaires et utilise les Hooks modernes (`useState`, `useEffect`).

- **Axios Interceptor :** Injecte automatiquement le Token JWT dans le header `Authorization` de chaque requête HTTP.
- **AuthService :** Service utilitaire pour décoder le JWT et vérifier les rôles (`isAdmin`).

4.2 Gestion des Rôles et Interfaces Dynamiques

L'interface s'adapte dynamiquement selon le rôle de l'utilisateur connecté :

4.2.1 Pour l'Administrateur

- Accès au menu "Gestion Clients", "Comptes" et "Validations".
- Visualisation des boutons "Supprimer" et "Nouveau Client".

4.2.2 Pour le Client

- Redirection automatique vers son **Tableau de Bord Personnel**.
- Accès restreint à l'API `/accounts/me` (ne voit que ses propres comptes).
- Interface simplifiée montrant le solde total et l'historique.

```
{isAdmin() ? (
    <>
       <Route path="/customers" element={<Customers />} />
       <Route path="/admin-dashboard" element={<AdminDashboard />} />
    </>
) : (
    <Route path="/client-dashboard" element={<ClientDashboard />} />
)}
```

Listing 4.1 – Protection des Routes dans App.js

Chapitre 5

Interfaces Graphiques

Cette section présente les interfaces finales de l'application, illustrant le design moderne (Material Design) et l'expérience utilisateur.

5.1 Authentification

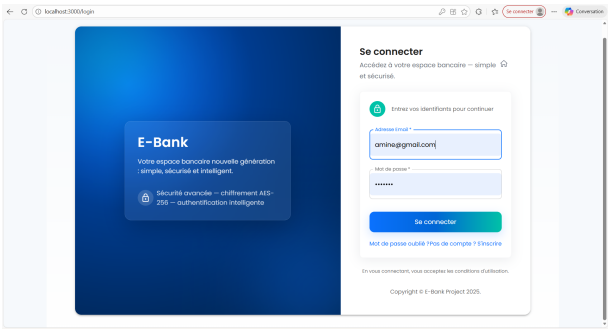

FIGURE 5.1 – Page de Connexion (Login) sécurisée

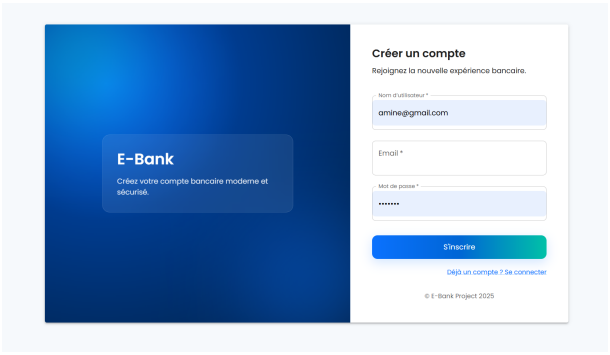

FIGURE 5.2 – Formulaire d'inscription pour les nouveaux membres

5.2 Espace Administrateur

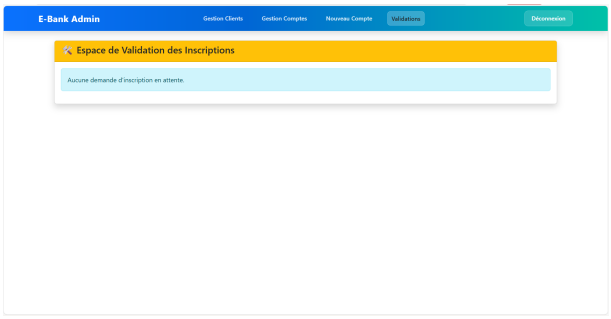

FIGURE 5.3 – Tableau de bord de validation des inscriptions

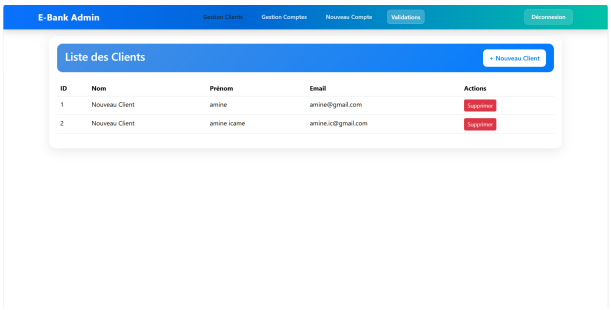

FIGURE 5.4 – Gestion des Clients (Vue Admin)

5.3 Espace Client

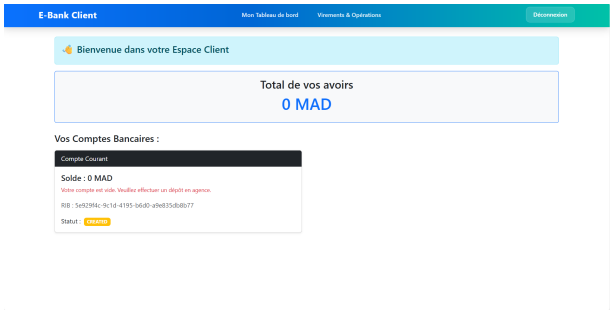

FIGURE 5.5 – Tableau de bord personnel du Client (Comptes et Solde)

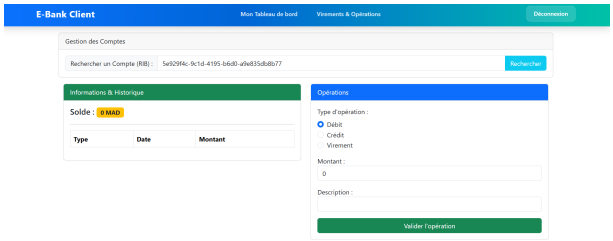

FIGURE 5.6 – operation client

Chapitre 6

Conclusion

Ce projet nous a permis de maîtriser l'ensemble de la chaîne de développement d'une application d'entreprise sécurisée.

Nous avons réussi à implémenter :

— Une architecture Backend scalable avec Spring Boot.

— Une sécurité robuste conforme aux standards (JWT, Stateless).

— Une interface Frontend réactive et adaptée aux rôles des utilisateurs.

— Une logique métier complexe d'activation de compte.

Les perspectives d'évolution incluent l'ajout de notifications par email lors de la validation et l'intégration d'une authentification à deux facteurs (2FA).