

REPUBLIQUE DU SENEGAL

Un peuple – Un but – Une foi



Université Assane SECK

L'excellence, ma référence

UFR SCIENCES ET TECHNOLOGIES

Licence Ingénierie Informatique

Licence - 3

Rapport de projet

INF3651 : STAGE / PROJET OPERATIONNEL

SYSTEME DE GESTION BANCAIRE (Partie Conception)

NOMS

1. Fallou DIOUCK
2. El hadji Malick MBENGUE

Matricules

202101663
202101109

RESUME

Dans le cadre de notre formation en Licence d'Ingénierie Informatique à l'Université Assane Seck de Ziguinchor (UASZ), nous avons réalisé ce projet en groupe de quatre personnes. Ce travail s'inscrit dans une démarche d'apprentissage et d'application des connaissances acquises au cours de notre cursus universitaire.

L'objectif de ce projet est de **concevoir et développer un système de gestion bancaire** permettant de digitaliser et sécuriser les services bancaires. Il vise à proposer une solution innovante et fonctionnelle pour la gestion des **comptes bancaires, agences, transactions et cartes bancaires**, en répondant aux exigences de fiabilité et d'accessibilité des institutions financières et de leurs clients.

Pour sa mise en œuvre, nous avons adopté une approche méthodologique rigoureuse, combinant **analyse des besoins, conception et développement**. Nous avons utilisé des technologies modernes adaptées au contexte du projet, garantissant ainsi sa fiabilité et son efficacité.

Ce projet constitue une opportunité précieuse pour renforcer nos compétences en **développement web, gestion de bases de données, programmation orientée objet et sécurité informatique**. Il nous a permis d'appliquer concrètement les concepts théoriques étudiés en cours et de développer une solution répondant aux besoins réels du secteur bancaire.

REMERCIEMENTS

Nous souhaitons exprimer notre profonde gratitude envers toutes les personnes qui nous ont soutenus tout au long de la réalisation de ce projet.

Tout d'abord, nous remercions nos enseignants de l'Université Assane Seck de Ziguinchor (UASZ) pour la qualité de leur enseignement et leurs précieux conseils qui nous ont permis d'acquérir les compétences nécessaires à la réalisation de ce travail.

Un grand merci à nos familles et amis pour leur soutien moral et leur encouragement constant.

Nous tenons aussi à exprimer notre reconnaissance à nos camarades de la **deuxième promotion de la Licence en Ingénierie Informatique** à l'UASZ. Leur entraide, leur esprit de partage et les échanges constructifs que nous avons eus tout au long de notre formation ont été une véritable source de motivation et d'apprentissage.

Enfin, nous soulignons l'excellente collaboration entre nous, dans le cadre de ce projet réalisé en groupe de quatre personnes. Cette expérience nous a permis de renforcer nos compétences techniques et notre esprit d'équipe.

SOMMAIRE

INTRODUCTION	p.7.
I. Analyse des Besoins	p.9.
1. Fonctionnalités Principales	p.9.
2. Acteurs et Rôles	p.11.
3. Diagramme de cas d'utilisations.....	p.12.
4. Exigences Fonctionnelles et Non Fonctionnelles	p.15.
II. Conception du Système.....	p.16.
1. Architecture Globale du Système.....	p.16.
2. Choix Technologiques.....	p.17.
3. Diagramme de Classes.....	p.17.
4. Modèle de Données.....	p.22.
5. Sécurité.....	p.24.
CONCLUSION.....	p.27.
DOCUMENTATIONS.....	p.28

LISTE DES FIGURES

Figure 1 : Diagramme de Cas d'Utilisation du Système de Gestion Bancaire	p.14.
Figure 2 : Architecture Client-Serveur avec React, Spring Boot et MySQL	p.16.
Figure 3 : Diagramme de Classes du Système de Gestion Bancaire	p.21.
Figure 4 : Schéma de la Base de Données du Système de Gestion Bancaire	p.23.

ABRÉVIATIONS

JSON	: JavaScript Object Notation
UML	: Unified Modeling Language
HTML	: Hypertext Markup Language
SQL	: Structured Query Language
HTTP	: Hypertext Transfer Protocol
API	: Application Programming Interface
UX	: Expérience Utilisateur
UI	: Interface Utilisateur
JWT	: JSON Web Token
CSRF	: Cross-Site Request Forgery
XSS	: Cross-Site Scripting
CORS	: Cross-Origin Resource Sharing

INTRODUCTION

Contexte du projet

Le projet de **système de gestion bancaire** a été conçu pour répondre aux besoins croissants des institutions financières en matière de digitalisation et de sécurité. Dans un contexte où les clients exigent des services bancaires accessibles et fiables, cette application offre une solution complète pour la gestion des comptes, des agences, des transactions, et des cartes bancaires. Ce projet s'inscrit dans le cadre d'un projet stage et vise à démontrer l'application des compétences techniques et méthodologiques acquises au cours de la formation.

Objectifs du projet

Les objectifs principaux de ce projet sont les suivants :

- Fournir une UI intuitive et responsive pour la gestion des comptes bancaires, des transactions, et des cartes.
- Assurer la sécurité des données sensibles grâce à des technologies modernes comme JWT et Spring Security.
- Permettre une gestion multi-rôles, avec des fonctionnalités adaptées aux administrateurs, directeurs, caissiers, et clients.
- Offrir des fonctionnalités avancées telles que les virements programmés et le suivi des transactions en temps réel.

Problématique

Le projet vise à résoudre plusieurs problématiques liées à la gestion bancaire moderne :

- **Centralisation des opérations** : Les banques doivent gérer un grand nombre de comptes, de transactions, et de cartes, ce qui nécessite une solution centralisée et efficace.
- **Sécurité des données** : Les transactions bancaires impliquent des données sensibles qui doivent être protégées contre les accès non autorisés et les fraudes.
- **UX** : Les clients attendent des interfaces simples et intuitives pour gérer leurs comptes et effectuer des transactions en toute sécurité.

- **Gestion multi-rôles :** Les banques doivent offrir des fonctionnalités adaptées à différents types d'utilisateurs, des clients aux administrateurs.

I. Analyse des Besoins

1. Fonctionnalités Principales

Le système offre les fonctionnalités suivantes :

a. Gestion Multi-Rôles :

- **Admin** : Gestion globale du système (utilisateurs, agences).
- **Directeur** : Supervision des agences et des opérations.
- **Caissier** : Gestion des transactions quotidiennes (dépôts, retraits, virements) et des comptes clients.
- **Client** : Accès aux services bancaires personnels (solde, virements, cartes).

b. Gestion des Utilisateurs du Système :

- **Description** : Permet de créer, modifier et Supprimer des utilisateurs. Chaque utilisateur a un nom d'utilisateur, un mot de passe, un email, un nom complet, un type (Administrateur, Directeur, Caissier, Utilisateur), une Adresse, un Telephone.
- **Acteurs concernés** : Admin.
- **Importance** : Assure un contrôle total sur les accès au système et garantit la sécurité des données des utilisateurs.

c. Création des Comptes Bancaires :

- **Description** : Permet la création des comptes bancaires. Chaque compte est associé à un utilisateur et peut avoir un solde, un numéro de compte unique, et un statut (actif, inactif).
- **Acteurs concernés** : Admin.
- **Importance** : Essentielle pour permettre aux clients de gérer leurs finances et aux administrateurs de superviser les comptes.

d. Gestion des Agences :

- **Description** : Permet aux administrateurs de créer, modifier et supprimer des agences bancaires. Chaque agence a un code unique, un nom, une adresse, un telephone, un email et un directeur.
- **Acteurs concernés** : Admin.
- **Importance** : Centralise la gestion des agences pour une meilleure organisation.

e. Gestion des Comptes Clients :

- **Description** : Permet l'ouverture et la clôture des comptes clients. Chaque compte a un nom complet, un email, un telephone, une Adresse, un nom utilisateur, un mot de passe et un solde initial.
- **Acteurs concernés** : Caissier.
- **Importance** : Facilite l'accès aux services bancaires pour les clients et permet une gestion fluide des comptes.

f. Gestion du personnel :

- **Description** : Permet d'ajouter, modifier, supprimer et lister les caissiers d'une agence.
- **Acteurs concernés** : Directeur.
- **Importance** : Centralise la gestion des caissiers pour une meilleure organisation.

g. Effectuer Transactions :

- **Description** : Permet aux caissiers et directeurs d'effectuer des transactions (Dépôt, Retrait, Virement) aux clients. Il permet aux clients d'effectuer des virements entre comptes. l'historique des transactions peut être consulté et des notifications doivent être envoyées en temps réel.
- **Acteurs concernés** : Client, Caissier, Directeur.
- **Importance** : Facilite les transferts d'argent et assure la transparence des opérations.

h. Gestion des Cartes Bancaires

- **Description** : Permet aux caissiers d'ajouter et supprimer des cartes bancaires aux clients. Les cartes peuvent être de type **VISA** ou **MasterCard** et possèdent :
 - Un numéro unique
 - Une date d'expiration
- **Acteurs concernés** : Caissier.
- **Importance** : Offre une gestion sécurisée des moyens de paiement.

i. Virements Programmés

- **Description** : Permet aux clients de planifier des virements à une date ultérieure. Les virements seront exécutés à la date et heures programmés.
- **Acteurs concernés** : Client.

- **Importance** : Simplifie la gestion des finances personnelles en automatisant les transferts.

j. **Suivi des Transactions en Temps Réel**

- **Description** : Offre un tableau de bord (personnalisé par rôle) pour visualiser les transactions en temps réel.
- **Acteurs concernés** : Client, Directeur, Admin, Client.
- **Importance** : Permet une surveillance continue des activités bancaires.

k. **Génération de relevés de comptes bancaires :**

- **Description** : Permet aux clients d'obtenir un relevé détaillé de leurs transactions sur une période donnée. Le relevé inclut :
 - Les transactions entrantes et sortantes.
 - Les dates et montants.
- **Acteurs concernés** : Client.
- **Importance** : Permet un suivi précis des finances et sert de justificatif pour diverses transactions.

l. **UI :**

- Dashboard personnalisé par rôle.
- Statistiques et graphiques en temps réel.
- Notifications des opérations importantes.

2. Acteurs et Rôles

Le système a plusieurs acteurs et plusieurs rôles

a. **Admin**

- **Rôle** : Gestion globale du système.
- **Responsabilités** :
 - Créer, modifier et supprimer des utilisateurs du système.
 - Créer, modifier et supprimer des agences bancaires.
 - Créer des Comptes bancaires aux utilisateurs.

b. **Directeur**

- **Rôle** : Supervision des agences et des opérations.
- **Responsabilités** :

- Superviser les agences et les opérations, avec des statistiques et graphiques en temps réel.
- Gérer les Caissiers de l'agence (Création, Suppression, Modification).
- Effectuer des retraits et des dépôts pour les clients.

c. Caissier

- **Rôle** : Gestion des transactions quotidiennes.
- **Responsabilités** :
 - Effectuer des opérations bancaires (dépôts, retraits et virements).
 - Gérer les cartes bancaires (Création et Suppression).
 - Gérer les comptes clients (Ouverture , Clôture et modification).

d. Client

- **Rôle** : Utilisateur final des services bancaires.
- **Responsabilités** :
 - Consulter les soldes et les transactions.
 - Effectuer des virements en temps réel et des virements programmés.
 - Voir l'historique de ces transactions en temps réel.
 - Générer des rapports de compte bancaires.
 - Effectuer des paiements.

3. Diagramme de Cas d'Utilisation (Use Case)

a. Acteurs Principaux

- **Admin** : Gère les utilisateurs, les agences.
- **Directeur** : Supervise les agences et les opérations.
- **Caissier** : Gère les transactions quotidiennes.
- **Client** : Accède à ses comptes, effectue des opérations bancaires.

b. Cas d'Utilisation

- **Gérer les Utilisateurs** : Créer, modifier, supprimer des comptes.
- **Superviser les agences** : Consulter les informations des agences et leur statistique en temps réel.

- **Superviser les Operations** : Visualiser les transactions effectuées dans les agences.
- **Gérer le personnel** : Ajouter, modifier, et supprimer des employés comme les caissiers d'une agence.
- **Gérer les Comptes Clients** : Ouvrir et clôturer des comptes clients.
- **Effectuer des Transactions** : Dépôts, retraits et transferts d'argent entre comptes bancaires.
- **Gérer les Cartes Bancaires** : Ajouter, Supprimer des cartes.
- **Planifier des Virements** : Programmer des virements à une date ultérieure.
- **Consulter les Transactions** : Visualiser l'historique des transactions.
- **Consulter le Solde** : Vérifier le solde disponible sur un compte bancaire.
- **Gérer les Agences** : Créer, modifier, supprimer des agences.
- **Effectuer des paiements** : Payer des factures sur différents catégories de dépenses (ex : Eau, Electricité, impôts, Assurances, Telephone, Internet)
- **Télécharger un relevé en format PDF** : Télécharger un relevé détaillant toutes les transactions effectuées sur une période donnée.

c. Diagramme de Cas d'Utilisation du Système avec UML

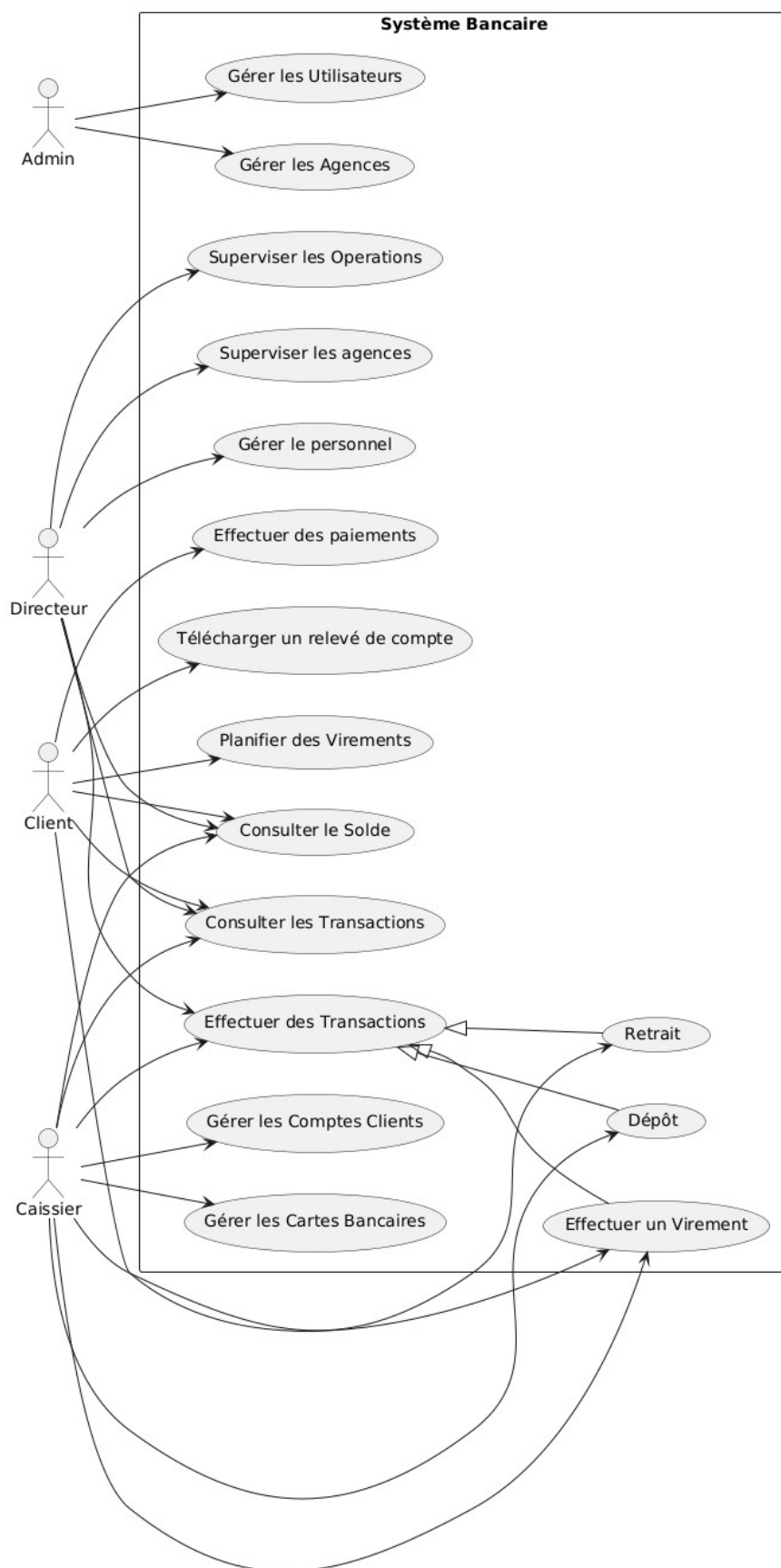


Figure 1 : Diagramme de Cas d'Utilisation du Système de Gestion Bancaire

4. Exigences Fonctionnelles et Non Fonctionnelles

a. Exigences Fonctionnelles

- Le système doit permettre la gestion des utilisateurs (création, modification, suppression, attribution des rôles).
- Le système doit permettre les transactions (Dépôts, Retraits et Virements) entre comptes.
- Le système doit offrir un tableau de bord (Personnalisé pour chaque rôle) pour visualiser les transactions en temps réel.
- Le système doit gérer les cartes bancaires (Ajout , Suppression).
- Le système doit permettre la planification de virements.
- Suivre l'historique des transactions.
- Le système doit permettre l'ouverture et la clôture des comptes clients.
- Le système doit permettre la gestion des agences bancaires (création, modification, suppression).
- Le système doit générer des relevés de comptes pour les clients.
- Le système doit envoyer des notifications en temps réel pour informer les utilisateurs des opérations importantes.

b. Exigences Non Fonctionnelles

- **Sécurité** : Le système doit utiliser JWT pour l'authentification et Spring Security pour l'autorisation.
- **Performance** : Le système doit répondre en moins de 2 secondes pour 95 % des requêtes.
- **Disponibilité** : Le système doit être disponible 99,9 % du temps.
- **Évolutivité** : Le système doit pouvoir supporter un grand nombre d'utilisateurs simultanés.

II. Conception du Système

1. Architecture Globale du Système

Le système est basé sur une architecture **3-tiers** :

a. **Frontend** : UI développée avec React et TailwindCSS.

- **Technologies** : React, TypeScript, TailwindCSS.
- **Rôle** : Fournir une UI intuitive et responsive.
- **Communication** : Le frontend communique avec le backend via des API RESTful en utilisant Axios.

b. **Backend** : API RESTful développée avec Spring Boot.

- **Technologies** : Java Spring Boot, Spring Security, JWT, JPA/Hibernate.
- **Rôle** : Gérer la logique métier, l'authentification, et les opérations bancaires.
- **Endpoints API** : Exemples :
 - POST /api/auth/login : Authentification de l'utilisateur.
 - GET /api/accounts : Récupérer la liste des comptes.
 - GET /api/transactions : Récupérer la liste des transactions.

c. **Base de Données** : MySQL pour le stockage des données.

- **Technologie** : MySQL.
- **Rôle** : Stocker les données de manière persistante (utilisateurs, comptes, transactions, etc.).

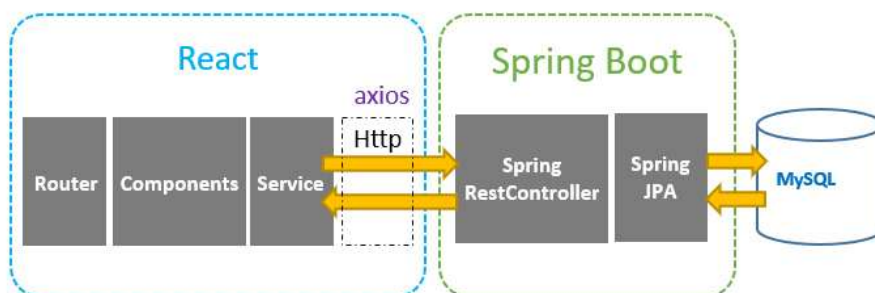


Figure 2 : Architecture Client-Serveur avec React, Spring Boot et MySQL

2. Choix Technologiques

a. Frontend :



React : Pour une interface dynamique et réactive.



TypeScript : Pour un code plus robuste et maintenable.



TailwindCSS : Pour un design responsive et moderne.

b. Backend :

- **Spring Boot** : Pour des API RESTful robustes et évolutives.
- **Spring Security** : Pour la gestion de l'authentification et des autorisations.
- **JWT** : Pour sécuriser les échanges entre le frontend et le backend.

c. Base de Données :

- **MySQL** : Pour le stockage relationnel des données.
- **JPA/Hibernate** : Pour simplifier les interactions avec la base de données.

3. Diagramme de Classes

a. Classes principales :

○ User :

- **Description** : La classe User représente un utilisateur du système. Elle contient les informations personnelles et les détails de connexion de l'utilisateur, ainsi que son rôle dans le système (admin, directeur, caissier, client).

➤ **Attributs** :

- ✓ id : Identifiant unique de l'utilisateur.
- ✓ username : Nom d'utilisateur pour la connexion.
- ✓ password : Mot de passe haché pour la sécurité.
- ✓ role : Rôle de l'utilisateur (admin, directeur, caissier, client).
- ✓ email : Adresse email de l'utilisateur.

- ✓ fullName : Nom complet de l'utilisateur.
- ✓ address : Adresse physique de l'utilisateur.
- ✓ phone : Numéro de téléphone de l'utilisateur.
- **Relations :**
 - ✓ OneToMany avec Account : Un utilisateur peut avoir plusieurs comptes.
- **Account :**
 - **Description :** La classe Account représente un compte bancaire. Elle contient les informations relatives au compte, telles que le numéro de compte, le solde et le statut.
 - **Attributs :**
 - ✓ id : Identifiant unique du compte.
 - ✓ accountNumber : Numéro de compte unique.
 - ✓ balance : Solde du compte.
 - ✓ status : Statut du compte (actif, inactif, bloqué).
 - **Relations :**
 - ✓ ManyToOne avec User : Un compte appartient à un utilisateur.
 - ✓ OneToMany avec Transaction : Un compte peut avoir plusieurs transactions.
- **Transaction :**
 - **Description :** La classe Transaction représente une opération bancaire, telle qu'un virement, un dépôt ou un retrait. Elle contient les détails de la transaction, y compris le montant, le type et le statut.
 - **Attributs :**
 - ✓ id : Identifiant unique de la transaction.
 - ✓ amount : Montant de la transaction.
 - ✓ type : Type de transaction (virement, dépôt, retrait).
 - ✓ description : Description de la transaction.
 - ✓ date : Date et heure de la transaction.
 - ✓ status : Statut de la transaction (en attente, validée, refusée).
 - **Relations :**

- ✓ ManyToOne avec Account : Une transaction est liée à un compte.
- **BankCard :**
 - **Description :** La classe BankCard représente une carte bancaire associée à un compte. Elle contient les informations de la carte, telles que le numéro de carte, le type et la date d'expiration.
 - **Attributs :**
 - ✓ id : Identifiant unique de la carte.
 - ✓ cardNumber : Numéro de la carte bancaire.
 - ✓ cardType : Type de carte (VISA, MASTERCARD).
 - ✓ expirationDate : Date d'expiration de la carte.
 - ✓ cvc : Code de sécurité de la carte.
 - **Relations :**
 - ✓ ManyToOne avec Account : Une carte est liée à un compte.
- **Agency :**
 - **Description :** La classe Agency représente une agence bancaire. Elle contient les informations relatives à l'agence, telles que son code, son nom, son adresse et son statut.
 - **Attributs :**
 - ✓ id : Identifiant unique de l'agence.
 - ✓ code : Code unique de l'agence.
 - ✓ name : Nom de l'agence.
 - ✓ address : Adresse de l'agence.
 - ✓ phone : Numéro de téléphone de l'agence.
 - ✓ email : Adresse email de l'agence.
 - ✓ createdAt : Date de création de l'agence.
 - ✓ isActive : Statut de l'agence (active ou inactive).
 - **Relations :**
 - ✓ OneToMany avec User : Une agence peut avoir plusieurs utilisateurs.
 - ✓ OneToOne avec User (directeur) : Une agence a un directeur.
- **VirementProgramme :**

- **Description :** La classe `VirementProgramme` représente un virement programmé (transfert d'argent planifié). Elle contient les détails du virement, y compris le compte source, le compte destinataire, le montant et la date d'exécution.
- **Attributs :**
 - ✓ `id` : Identifiant unique du virement programmé.
 - ✓ `compteSource` : Référence au compte source.
 - ✓ `numeroCompteDestination` : Numéro du compte destinataire.
 - ✓ `beneficiaireName` : Nom du bénéficiaire.
 - ✓ `montant` : Montant à transférer.
 - ✓ `dateExecution` : Date et heure programmées pour le virement.
 - ✓ `executed` : Indique si le virement a été exécuté.
 - ✓ `status` : Statut du virement (`EN_ATTENTE`, `EXECUTE`, `REFUSE`).
 - ✓ `refusReason` : Raison du refus (si applicable).
- **Relations :**
 - ✓ ManyToOne avec `Account` : Un virement programmé est lié à un compte source.
- **ExpenseCategory :**
 - **Description :** La classe `ExpenseCategory` représente une catégorie de dépenses pour les transactions (ex : Eau, Impôts, Electricité, Assurances, Telephone, Internet). Elle est utilisée pour classer les transactions par type de dépense.
 - **Attributs :**
 - ✓ `id` : Identifiant unique de la catégorie.
 - ✓ `name` : Nom de la catégorie (ex : Electricité, Impôt, Téléphone, Assurances, Eau, Internet).
 - ✓ `color` : Couleur associée à la catégorie (pour l'affichage dans l'UI).
 - **Relations :**
 - ✓ OneToMany avec `Transaction` : Une catégorie peut être associée à plusieurs transactions.
- **CashierLog :**

- Description : La classe CashierLog représente les logs des opérations effectuées par un caissier. Elle est utilisée pour tracer toutes les activités des caissiers.
- Attributs :
 - ✓ id : Identifiant unique du log.
 - ✓ type : Type d'opération (ex : dépôt, retrait).
 - ✓ description : Description de l'opération.
 - ✓ date : Date et heure de l'opération.
 - ✓ amount : Montant de l'opération.
 - ✓ accountNumber : Numéro de compte concerné par l'opération.
 - ✓ userName : Nom de l'utilisateur concerné.
 - ✓ status : Statut de l'opération (réussie, échouée).
 - ✓ details : Détails supplémentaires sur l'opération.
 - ✓ cashier : Référence à l'utilisateur caissier qui a effectué l'opération.
- Relations :
 - ✓ ManyToOne avec User : Un log est associé à un caissier.

b. Diagramme de Classe du Système avec UML

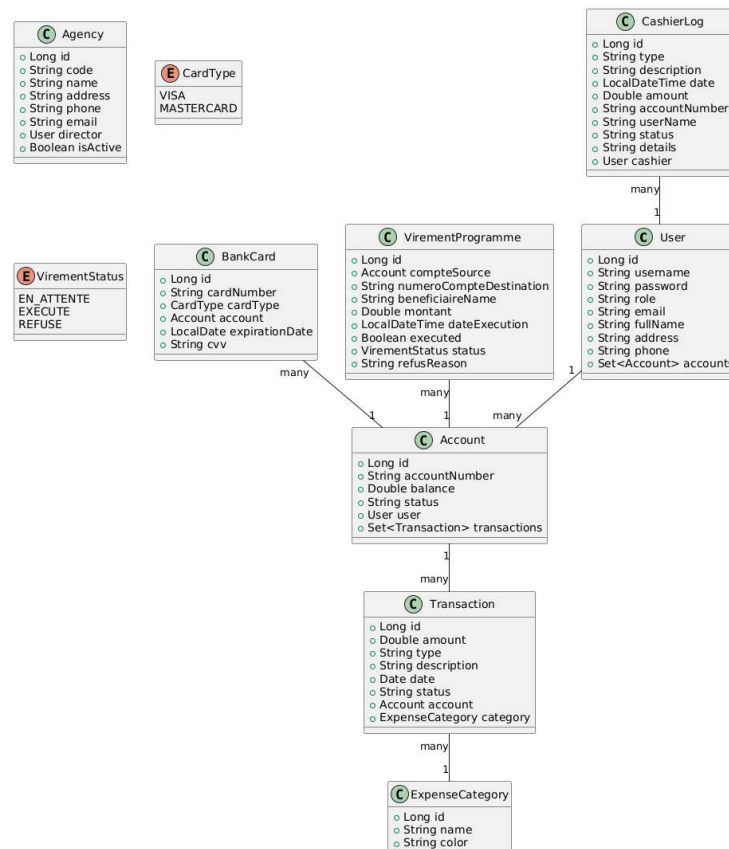


Figure 3 : Diagramme de Classes du Système de Gestion Bancaire

4. Modèle de Données

- a. Le modèle de données est conçu pour garantir une gestion efficace des informations.

Voici les tables

○ **User :**

- Colonnes: id, username, password, role, email, fullName, address, phone, agency_id.
- Relations : User a une clé étrangère vers Agency.

○ **Account :**

- Colonnes : id, accountNumber, balance, status, user_id.
- Relations :
 - ✓ Un compte appartient à un utilisateur (clé étrangère user_id vers User).
 - ✓ Un compte peut avoir plusieurs transactions (relation avec Transaction via account_id).
 - ✓ Un compte peut avoir plusieurs cartes bancaires (relation avec BankCard via account_id).

○ **Transaction :**

- Colonnes : id, amount, type, description, date, status, account_id.
- Relations :
 - ✓ Une transaction est associée à un compte (clé étrangère account_id vers Account).
 - ✓ Une transaction peut être catégorisée (clé étrangère category_id vers ExpenseCategory).

○ **ExpenseCategory:**

- Colonnes: id, name, color.
- Relations : Une catégorie de dépense peut être associée à plusieurs transactions.





○ **BankCard :**

- Colonnes : id, cardNumber, cardType, expirationDate, cvv, account_id.
- Relations : BankCard a une clé étrangère vers Account.

○ **Agency :**

- Colonnes: id, code, name, address, phone, email, createdAt, isActive, director or_id.
- Relations : Agency a une clé étrangère vers User (directeur).
- **CashierLog:**
 - Colonnes : id, type, description, date, amount, accountNumber, username, status, details, cashier_id;
 - Relations: Un journal de caisse est associé à un caissier (clé étrangère cashier_id vers User).
- **VirementProgramme :**
 - Colonnes : id, CompteSource_id, numeroCompteDestination, beneficiaireName, montant, dateExecution, executed, status, refusReason.
 - Relations: Un virement programmé provient d'un compte source (clé étrangère compteSource_id vers Account).

b. Schéma de Base de données

User	Account	Transaction	ExpenseCategory
id  Long	id  Long	id  Long	id  Long
username String	accountNumber String	amount Double	name String
password String	balance Double	type String	color String
role String	user_id Long	description String	
email String	status String	date Date	
fullName String		account_id Long	
address String		category_id Long	
phone String			





BankCard	Agency	CashierLog	VirementProgramme
id  Long	id  Long	id  Long	id  Long
cardNumber String	code String	type String	compteSource_id Long
cardType String	name String	description String	numeroCompteDestination String
account_id Long	address String	date LocalDateTime	beneficiaireName String
expirationDate Date	phone String	amount Double	montant Double
cw String	email String	accountNumber String	dateExecution LocalDateTime
	director_id Long	userName String	executed Boolean
	isActive Boolean	status String	status String
		details String	refusReason String
		cashier_id Long	

Figure 4 : Schéma de la Base de Données du Système de Gestion Bancaire

5. Sécurité

La sécurité est une priorité dans la conception du système. Les mesures suivantes sont prévues :

- a. **Authentification** : L'authentification est le processus de vérification de l'identité d'un utilisateur. Pour ce projet, nous utiliserons **JWT** pour gérer l'authentification de manière sécurisée.
 - **Fonctionnement de JWT** :
 - Lorsqu'un utilisateur se connecte, le backend génère un JWT contenant des informations sur l'utilisateur (comme son ID et son rôle).
 - Ce token est signé avec une clé secrète pour garantir son intégrité.
 - Le token est renvoyé au frontend et stocké localement (dans le localStorage ou les cookies).
 - Pour chaque requête ultérieure, le frontend envoie le token dans l'en-tête HTTP Authorization.
 - Le backend vérifie la validité du token avant de traiter la requête.
 - **Avantages de JWT** :
 - **Stateless** : Le serveur n'a pas besoin de stocker les sessions, ce qui améliore la scalabilité.
 - **Sécurisé** : Les tokens sont signés et peuvent être chiffrés pour une sécurité accrue.
 - **Flexible** : Les tokens peuvent contenir des informations supplémentaires (comme le rôle de l'utilisateur).
- b. **Autorisation** : L'autorisation détermine ce qu'un utilisateur est autorisé à faire une fois authentifié. Nous utiliserons **Spring Security** pour gérer les rôles et les permissions.
 - **Rôles** :
 - **Admin** : Accès complet au système.
 - **Directeur** : Accès aux données de son agence.
 - **Caissier** : Accès limité aux transactions quotidiennes et aux clients de l'agence auquel il travaille.
 - **Client** : Accès à ses propres comptes et transactions.
 - **Configuration de Spring Security** :

- Les endpoints sont protégés par des règles d'autorisation basées sur les rôles.
- c. **Hachage des mots de passe** : Les mots de passe sont stockés de manière sécurisée dans la base de données grâce au hachage.
 - **Algorithme de hachage : BCrypt.**
 - BCrypt est un algorithme de hachage robuste qui inclut un "sel" (salt) pour renforcer la sécurité.
 - Il est résistant aux attaques par force brute et aux attaques par table arc-en-ciel.
- d. **Validation des données** : La validation des données est essentielle pour prévenir les attaques courantes comme les injections SQL et les attaques XSS.
 - **Validation côté backend** :
 - Utilisation des annotations de validation de Spring Boot (@NotBlank, @Email, @Pattern, etc.).
 - **Validation côté frontend** :
 - Utilisation de bibliothèques comme Formik et Yup pour valider les formulaires avant soumission.
- e. **Protection contre les attaques courantes** : Plusieurs mesures seront mises en place pour protéger l'application contre les attaques courantes.
 - **CSRF** :
 - Désactivé dans Spring Security car les tokens JWT sont utilisés pour l'authentification.
 - Les tokens JWT sont inclus dans l'en-tête Authorization, ce qui rend les attaques CSRF inefficaces.
 - **XSS** :
 - Les données entrantes sont validées et nettoyées avant d'être affichées.
 - Utilisation de bibliothèques comme DOMPurify pour nettoyer le HTML côté frontend.
 - **Injection SQL** :

- Utilisation de JPA/Hibernate pour les requêtes SQL, ce qui prévient les injections SQL.
- Les requêtes natives sont évitées autant que possible.
- **CORS :**
 - Configuration stricte des en-têtes CORS pour limiter les requêtes à des domaines autorisés.

Conclusion

La phase de conception du **Système de Gestion Bancaire** a été une étape cruciale pour définir les fondations techniques et fonctionnelles du projet. Cette section résume les réalisations, les défis rencontrés, et les perspectives d'amélioration liées à la conception du système.

Objectifs atteintes

- **Modélisation Complète** : La conception du système a permis de modéliser de manière exhaustive les entités principales (utilisateurs, comptes, transactions, agences, etc.) et leurs relations.
- **Architecture Robustes** : L'architecture 3-tier (frontend, backend, base de données) a été soigneusement définie pour garantir une séparation claire des responsabilités et une scalabilité future.

Sécurité Intégrée : Les mécanismes de sécurité (JWT, Spring Security, BCrypt, Validation des données, Protection contre les attaques courantes) ont été intégrés dès la phase de conception pour garantir la protection des données sensibles.

Perspectives d'Amélioration

Pour améliorer la scalabilité et la maintenabilité, le système pourrait être décomposé en microservices. Par exemple :

- Un service dédié à l'authentification et à l'autorisation.
- Un service pour la gestion des transactions.
- Un service pour la gestion des comptes et des cartes bancaires.
- Un service pour la gestion des agences.

Documentations

1. Conception de Systèmes

- Université de Paris 13 - IUT Villetaneuse. Introduction à UML 2 [PDF].
<http://lipn.univ-paris13.fr/~gerard/docs/cours/uml-cours-slides.pdf>.

2. Documentations officielles

- *Pivotal Software*. (2023). Spring Security Reference. <https://docs.spring.io/spring-security/reference/>
Documentation officielle de Spring Security, couvrant l'authentification, l'autorisation et la sécurisation des applications.

3. Guides en ligne

- *Auth0*. (2023). Introduction to JSON Web Tokens (JWT). <https://jwt.io/introduction/>
Guide d'introduction à JWT, incluant des explications sur la structure et l'utilisation des tokens.