ENSET
ÉCOLE NORMALE SUPÉRIURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Architecture Distribuée et Middleware

Rapport du Projet :

Application de Gestion des crédits bancaires

EXAMEN FINAL

Fait par :

Amine ARFAOUI

Encadré par :

M. **Mohamed** Youssfi

GLSID

2024 -2025

Rapport de Projet: Système de Gestion de Crédit Bancaire

Développé par l'équipe Credit-Bancaire

19 mai 2025

Table des matières

1 Introduction **2**

2 Description du Projet **2**
2.1 Objectifs du Projet 2

3 Architecture Technique **2**
3.0.1 Backend 2
3.0.2 Frontend 2
3.0.3 Backend 3
3.0.4 Frontend 3

4 Modèle de Données **3**
4.1 Entités Principales 3
4.1.1 Client 3
4.1.2 Credit 3
4.1.3 Types de Crédit 4
4.1.4 Remboursement 4

5 Services et Fonctionnalités Backend **5**
5.1 Gestion des Clients 5
5.2 Gestion des Crédits 5
5.3 Gestion des Remboursements 5

6 Interface Utilisateur (Frontend) **5**
6.1 Fonctionnalités de l'Interface 5
6.2 Captures d'écran 5
6.3 Communication Frontend-Backend 6

7 Sécurité **6**
7.1 Authentification et Autorisation 6

8 Pattern Data Transfer Object (DTO) **7**

9 Conclusion **7**

1 Introduction

Ce rapport présente une analyse technique détaillée du projet de gestion de crédit bancaire. Ce système vise à automatiser et simplifier la gestion des crédits bancaires, y compris les demandes, les suivis et les remboursements, pour une institution financière. Le projet est développé en utilisant des technologies modernes du monde Java pour le backend et Angular pour le frontend, avec une architecture orientée services.

2 Description du Projet

Le projet "Crédit Bancaire" est une application de gestion complète qui permet aux institutions financières de gérer efficacement les demandes de crédit, le suivi des remboursements et la gestion des clients. L'application prend en charge différents types de crédits (personnel, immobilier, professionnel) et offre une interface pour gérer l'ensemble du cycle de vie d'un crédit, de la demande initiale jusqu'au remboursement final.

2.1 Objectifs du Projet

Les objectifs principaux du système sont :
— Gestion des clients et de leurs informations personnelles
— Traitement des demandes de crédit de différentes natures
— Suivi de l'état des crédits (en cours, accepté, rejeté)
— Gestion des échéances de remboursement
— Génération de rapports sur l'état des crédits
— Interface REST pour l'interaction avec d'autres systèmes
— Interface utilisateur moderne et intuitive

3 Architecture Technique

3.0.1 Backend

Le backend du projet est développé avec les technologies suivantes :
— Java comme langage de programmation principal
— Spring Boot comme framework d'application
— JPA/Hibernate pour la persistance des données
— Spring Data pour l'accès aux données
— Spring REST pour l'exposition des services web
— Lombok pour réduire le code boilerplate
— JWT pour l'authentification et l'autorisation

3.0.2 Frontend

Le frontend est développé avec :
— Angular 19 comme framework principal
— Bootstrap 5 pour les composants UI
— Bootstrap Icons pour les icônes
— Reactive Forms pour la gestion des formulaires
— RxJS pour la programmation réactive

— SCSS pour les styles
— Angular HttpClient pour la communication avec le backend

3.0.3 Backend

L'application backend suit une architecture en couches classique :
— **Couche Entités** : Représente les objets du domaine et leurs relations
— **Couche Repository** : Gère l'accès aux données
— **Couche Service** : Contient la logique métier
— **Couche Controller** : Expose les API REST
— **Couche DTO** : Objets de transfert de données entre le frontend et le backend

3.0.4 Frontend

L'architecture frontend suit les recommandations Angular :
— **Composants** : Éléments d'interface utilisateur réutilisables
— **Services** : Logique métier et communication avec le backend
— **Modèles** : Interfaces TypeScript pour les données
— **Guards** : Protection des routes pour l'authentification
— **Interceptors** : Interception des requêtes HTTP pour l'ajout de token JWT

4 Modèle de Données

4.1 Entités Principales

4.1.1 Client

La classe `Client` représente les utilisateurs du système bancaire qui peuvent demander des crédits :

```
@Entity
public class Client {
    @Id @GeneratedValue
    private Long id;
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
    private String adresse;
    private String dateNaissance;
    private String cin;

    @OneToMany(mappedBy = "client")
    private List<Credit> credits;
}
```

Listing 1 – Entité Client

4.1.2 Credit

La classe abstraite `Credit` est la base pour tous les types de crédit :

```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "type_credit")
public abstract class Credit {
    @Id @GeneratedValue
    private Long id;
    private LocalDate dateDemande;
    @Enumerated(EnumType.STRING)
    private StatutCredit statut;
    private LocalDate dateAcceptation;
    private double montant;
    private int dureeRemboursement;
    private double tauxInteret;

    @ManyToOne
    private Client client;

    @OneToMany(mappedBy = "credit")
    private List<Remboursement> remboursements;
}
```

Listing 2 – Entité Credit

4.1.3 Types de Crédit

Le système gère trois types de crédits spécifiques :

- **CreditPersonnel** : Pour les besoins personnels des clients
- **CreditImmobilier** : Pour l'achat de biens immobiliers
- **CreditProfessionnel** : Pour les projets professionnels ou d'entreprise

4.1.4 Remboursement

La classe `Remboursement` représente les paiements effectués sur un crédit :

```java
@Entity
public class Remboursement {
    @Id @GeneratedValue
    private Long id;
    private LocalDate dateEcheance;
    private LocalDate datePaiement;
    private double montant;
    private boolean paye;

    @Enumerated(EnumType.STRING)
    private TypeRemboursement type;

    @ManyToOne
    private Credit credit;
}
```

Listing 3 – Entité Remboursement

Le diagramme de classes montre les relations entre les différentes entités du système. Les principales relations sont :

— Un Client peut avoir plusieurs Crédits (relation 1-n)
— Un Crédit appartient à un seul Client (relation n-1)
— Un Crédit peut avoir plusieurs Remboursements (relation 1-n)
— Un Remboursement appartient à un seul Crédit (relation n-1)
— Héritage des différents types de Crédit depuis la classe abstraite Credit

5 Services et Fonctionnalités Backend

5.1 Gestion des Clients

Le service `ClientService` offre les fonctionnalités suivantes :
— Listing de tous les clients
— Recherche de clients par ID, nom ou email
— Création, mise à jour et suppression de clients

5.2 Gestion des Crédits

Le service `CreditService` permet :
— Création de différents types de crédits (Personnel, Immobilier, Professionnel)
— Consultation des crédits par ID, par client ou par statut
— Acceptation ou rejet des demandes de crédit
— Modification et suppression des crédits

5.3 Gestion des Remboursements

Le service `RemboursementService` gère :
— Enregistrement des remboursements programmés
— Suivi des paiements effectués
— Consultation des échéanciers par crédit
— Modification et suppression des remboursements

6 Interface Utilisateur (Frontend)

6.1 Fonctionnalités de l'Interface

L'interface utilisateur du système offre les fonctionnalités suivantes :
— **Authentification** : Connexion sécurisée avec token JWT
— **Tableau de bord** : Vue d'ensemble des statistiques et activités
— **Gestion des clients** : Recherche, création, édition et suppression
— **Gestion des crédits** : Consultation, création et suivi des crédits
— **Gestion des remboursements** : Consultation des échéanciers et enregistrement des paiements
— **Rapports** : Génération et exportation de rapports sur les crédits

6.2 Captures d'écran

— `/auth/login` - Authentification et génération de token JWT
— `ClientRestController` : Endpoints pour la gestion des clients

— `CreditRestController` : Endpoints pour la gestion des crédits
— `RemboursementRestController` : Endpoints pour la gestion des remboursements

6.3 Communication Frontend-Backend

Le frontend communique avec le backend via des services Angular qui consomment les APIs REST :

```
@Injectable({
  providedIn: 'root'
})
export class ClientService {
  private readonly API_URL = 'http://localhost:8085';

  constructor(private http: HttpClient) { }

  getClients(keyword: string = ""): Observable<Client[]> {
    return this.http.get<Client[]>(`${this.API_URL}/customers?keyword=${keyword}`);
  }

  getClientById(id: number): Observable<Client> {
    return this.http.get<Client>(`${this.API_URL}/customers/${id}`);
  }

  saveClient(client: Client): Observable<Client> {
    return this.http.post<Client>(`${this.API_URL}/customers`, client);
  }

  updateClient(client: Client): Observable<Client> {
    return this.http.put<Client>(`${this.API_URL}/customers/${client.id}`, client);
  }

  deleteClient(id: number): Observable<any> {
    return this.http.delete(`${this.API_URL}/customers/${id}`);
  }
}
```

Listing 4 – Service Angular pour les clients

7 Sécurité

7.1 Authentification et Autorisation

Le système utilise JWT (JSON Web Tokens) pour l'authentification et l'autorisation :
— **Backend** : Spring Security avec JWT pour sécuriser les APIs
— **Frontend** : Intercepteurs HTTP pour ajouter le token aux requêtes et guards de routes pour protéger les pages

```typescript
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private authService: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<
   HttpEvent<any>> {
    const token = this.authService.getToken();

    if (token) {
      const authReq = req.clone({
        headers: req.headers.set('Authorization', `Bearer ${token
   }`)
      });
      return next.handle(authReq);
    }

    return next.handle(req);
  }
}
```

Listing 5 – Intercepteur HTTP pour l'authentification

8 Pattern Data Transfer Object (DTO)

Le projet utilise le pattern DTO pour séparer les entités de persistance des objets utilisés pour la communication avec les clients. Par exemple, la classe `ClientDTO` est utilisée pour transférer les données des clients sans exposer directement les entités JPA.

```java
public class ClientDTO {
    private Long id;
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
    private String adresse;
    private String dateNaissance;
    private String cin;

    // Getters, setters, constructors
}
```

Listing 6 – Exemple de DTO pour les clients

9 Conclusion

Le système de gestion de crédit bancaire présente une architecture robuste et extensible qui répond aux besoins d'une institution financière moderne. L'utilisation des technologies Spring Boot et Angular assure une base solide pour le développement futur et la maintenance.

La combinaison d'un backend robuste avec une interface utilisateur moderne et intuitive permet aux utilisateurs de gérer efficacement les crédits bancaires, depuis la demande initiale jusqu'au remboursement final.

Les prochaines étapes du projet pourraient inclure :

— Implémentation de calculs financiers avancés
— Intégration avec des systèmes de scoring de crédit
— Mise en place d'un système de notification
— Ajout de fonctionnalités de reporting et d'analyse avancées
— Intégration avec d'autres systèmes bancaires
— Applications mobiles natives pour iOS et Android