جامعة مولاي إسماعيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵎⵓⵍⴰⵢ ⵉⵙⵎⴰⵄⵉⵍ
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم
ⵜⴰⵖⵣⵓⵜ ⵏ ⵜⵣⵔⴼⵜ
FACULTÉ DES SCIENCES

Rapport de projet

Application de Banque en Ligne

Réalisé par :

- M. Wail ANAIA
- M. Yasser OUSSAHEL

Encadré par :

- Pr. Abdelaaziz HESSANE

Formation :

- DWM (Développement Web et Mobile) - Groupe A1

Jobintech

2025/2026

Table des matières

REMERCIEMENTS

Nous tenons à exprimer notre profonde gratitude à **Pr. Abdelaaziz HESSANE** pour son encadrement, ses conseils précieux et son accompagnement tout au long de la réalisation de ce projet, dans le cadre de la formation DWM – Jobintech.

Nos remerciements s'adressent également à toute l'équipe pédagogique de Jobintech pour la qualité de la formation dispensée et pour nous avoir donné l'opportunité de développer nos compétences techniques à travers ce projet.

Enfin, nous remercions nos familles et nos proches pour leur soutien constant durant cette période de formation intensive.

RÉSUMÉ EXÉCUTIF

Ce rapport présente la conception et le développement d'une application bancaire professionnelle développée en Java, utilisant le framework Swing pour l'interface graphique. Le projet s'inscrit dans le cadre de la formation DWM (Développement Web et Mobile) proposée par Jobintech.

L'application permet la gestion complète d'une banque en ligne, incluant la gestion des clients, des comptes bancaires (courant et épargne), des transactions financières, de l'authentification sécurisée, ainsi que la génération de relevés bancaires et de statistiques en temps réel.

Le système adopte une architecture MVC (Modèle-Vue-Contrôleur) stricte, garantissant la séparation des responsabilités, la maintenabilité du code et l'évolutivité de l'application. L'authentification est sécurisée par hachage SHA-256 des mots de passe, et la gestion des exceptions est exhaustive pour assurer la robustesse du système.

1. INTRODUCTION ET CONTEXTE

1.1 Contexte du Développement

Dans le cadre de notre formation en programmation Java, nous avons été amenés à développer une application bancaire complète mettant en œuvre les principes fondamentaux de la programmation orientée objet. Ce projet s'inscrit dans l'évaluation des compétences acquises en matière de conception logicielle, d'architecture multi-couches et de développement d'interfaces graphiques.

1.2 Enjeux du Projet

Le secteur bancaire est aujourd'hui fortement numérisé, avec une demande croissante pour des applications sécurisées, fiables et intuitives. Ce projet représente une opportunité unique d'appliquer les concepts théoriques vus en cours à un cas pratique et réaliste, tout en répondant à des exigences fonctionnelles complexes.

1.3 Définition du Problème

Les banques traditionnelles cherchent à digitaliser leurs services tout en maintenant un haut niveau de sécurité et de fiabilité. Notre application répond à ce besoin en proposant une solution complète de gestion de comptes bancaires avec une interface utilisateur moderne et des fonctionnalités avancées.

2. OBJECTIFS DU PROJET

2.1 Objectifs Pédagogiques

- Maîtriser les concepts d'abstraction et de polymorphisme
- Implémenter des exceptions personnalisées
- Développer une architecture MVC (Modèle-Vue-Contrôleur)
- Créer une interface graphique complète avec Swing
- Gérer la persistance des données

2.2 Objectifs Fonctionnels

- Gestion des comptes courants et d'épargne
- Système de virements internes et externes
- Génération de relevés bancaires
- Calcul automatique des intérêts
- Historique des transactions sur 12 mois
- Système d'authentification sécurisé

2.3 Objectifs Techniques

- Respect des principes SOLID
- Séparation claire des responsabilités
- Code maintenable et extensible
- Interface utilisateur ergonomique
- Gestion robuste des erreurs

3. SPÉCIFICATIONS TECHNIQUES

3.1 Environnement de Développement

- **Langage :** Java 17+
- **Interface Graphique :** Swing (Java Foundation Classes)
- **Architecture :** MVC (Modèle-Vue-Contrôleur)
- **Outils :** Visual Studio Code, Git
- **Système d'exploitation :** Windows

3.2 Technologies Utilisées

- **Java Swing** pour l'interface utilisateur
- **Collections Framework** pour la gestion des données
- **Exceptions personnalisées** pour la gestion des erreurs métier
- **Architecture en couches** pour la séparation des préoccupations
- **Design Patterns** appropriés (Strategy, Observer, Factory)

3.3 Configuration Requise

- Java Development Kit (JDK) 17 ou supérieur
- 4 GB de RAM minimum
- 500 MB d'espace disque
- Écran 1024x768 minimum

4. ARCHITECTURE DU SYSTÈME

4.1 Vue d'Ensemble de l'Architecture

L'application suit une architecture en trois couches distinctes :

Couche Présentation (**Vue**) → Couche Métier (**Contrôleur**) → Couche Données (**Modèle**)

4.2 Diagramme de Packages

banque/

├── Main.java

├── controleurs/

│ ├── AuthControleur.java

│ ├── CompteControleur.java

│ └── ViremmentControleur.java

├── modele/

│ ├── abstractions/

│ ├── implementations/

│ ├── utilisateur/

│ ├── operations/

│ └── exceptions/

├── vue/

│ ├── fenetres/

│ └── composants/

└── services/

├── PDFService.java

├── CalculInteretsService.java

└── HistoriqueService.java

4.3 Principes d'Architecture Appliqués

- **Séparation des préoccupations :** Chaque classe a une responsabilité unique
- **Faible couplage :** Les composants communiquent via des interfaces
- **Haute cohésion :** Les classes regroupent des fonctionnalités liées
- **Principe de substitution de Liskov :** Les sous-classes peuvent remplacer leurs classes mères
- **Principe d'inversion de dépendance :** Dépendre des abstractions, pas des implémentations

5. ANALYSE ET CONCEPTION

5.1 Analyse des Besoins

5.1.1 Besoins Fonctionnels

1. **Gestion des Utilisateurs**
    - Inscription et authentification
    - Profil utilisateur avec informations personnelles
    - Association multiple comptes/utilisateur
2. **Gestion des Comptes**
    - Création de comptes courants et d'épargne
    - Opérations de dépôt et retrait
    - Calcul automatique des intérêts
    - Gestion des découverts autorisés
3. **Opérations Bancaires**
    - Virements internes entre comptes
    - Virements externes avec frais
    - Historique des transactions
    - Relevés mensuels et annuels
4. **Interface Utilisateur**
    - Tableau de bord personnalisé
    - Navigation intuitive
    - Affichage en temps réel des soldes
    - Génération de documents

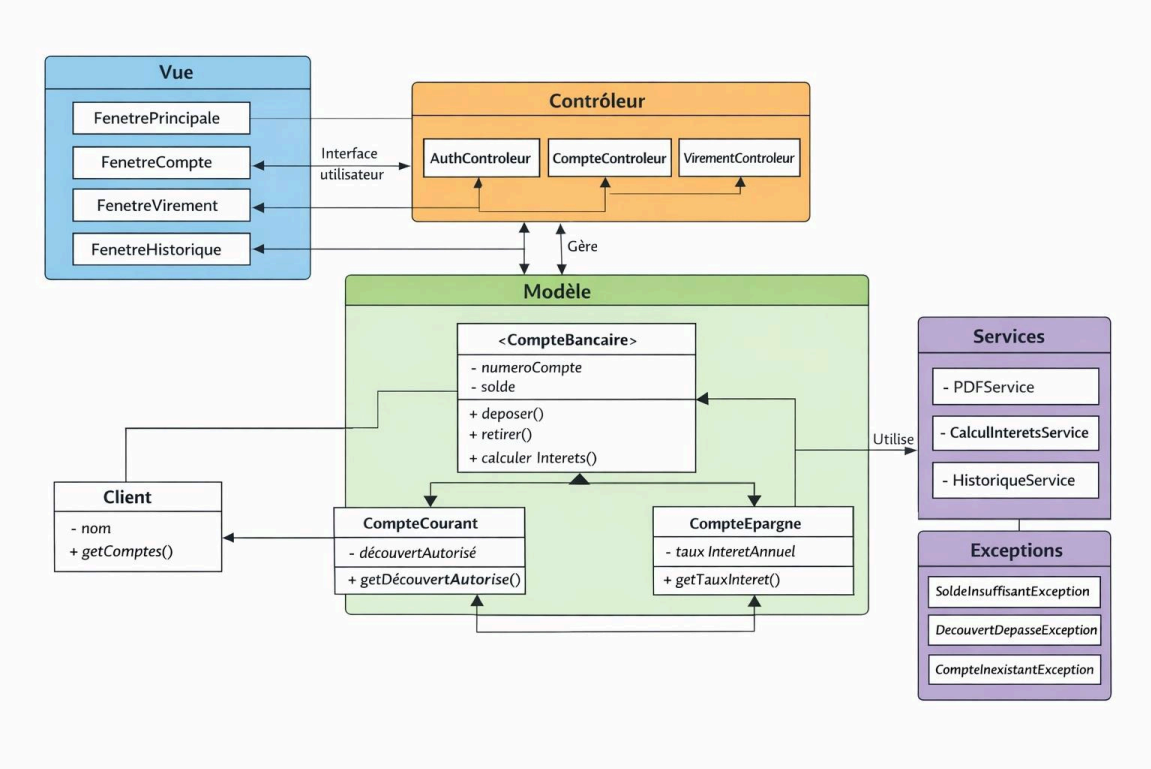
5.1.2 Règles Métier

- Plafond de virement : 5 000 €
- Découvert autorisé pour comptes courants : 1 000 €

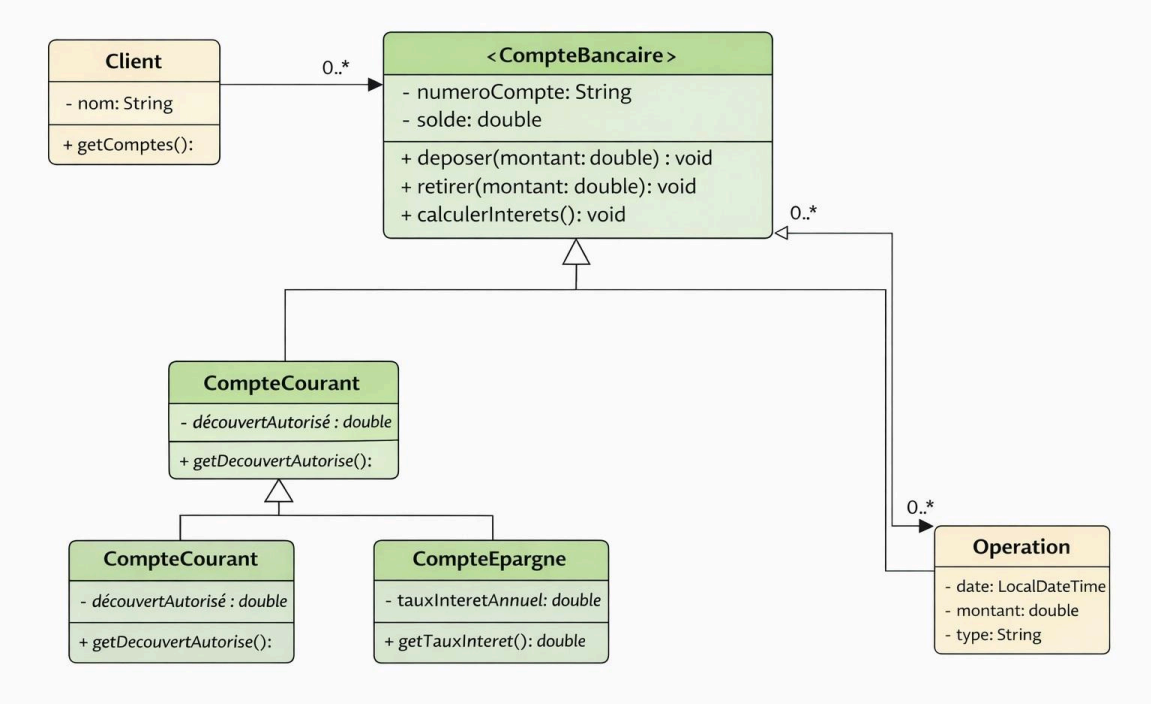
- Frais de virement externe : 1,50 €
- Taux d'intérêt épargne : 2% annuel
- Frais de découvert : 10% annuel
- Historique limité à 12 mois

5.2 Diagrammes de Conception

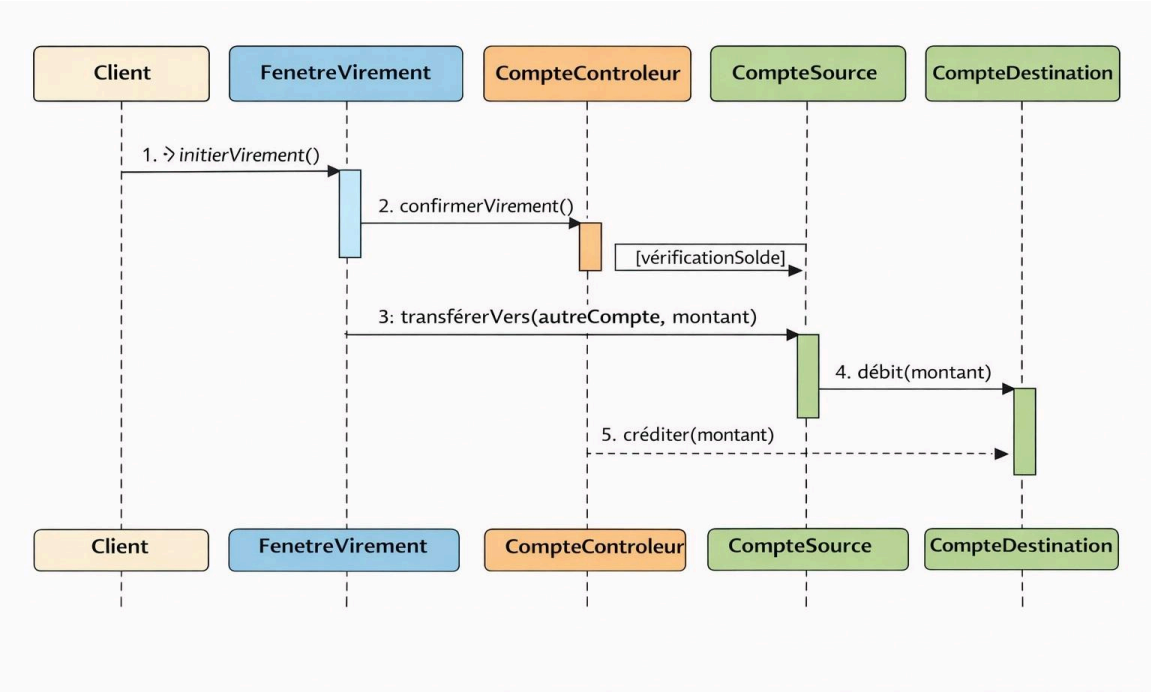
5.2.1 Diagramme d'Architecture MVC de l'Application Bancaire



5.2.2 Diagramme de Classes (Extrait)



5.2.3 Diagramme de Séquence - Virement



5.3 Design Patterns Implémentés

5.3.1 Pattern Strategy

Utilisé pour les différents types de comptes (courant vs épargne) avec des algorithmes de calcul d'intérêts différents.

5.3.2 Pattern Observer

Implémenté pour la mise à jour automatique de l'interface utilisateur lorsque les données changent.

5.3.3 Pattern Factory

Pour la création des différents types de comptes bancaires.

5.3.4 Pattern MVC

Architecture globale de l'application avec séparation claire entre données, logique et présentation.

6. IMPLÉMENTATION

6.1 Structure Détaillée du Code

6.1.1 Couche Modèle

CompteBancaire.java (Classe Abstraite)

```
public abstract class CompteBancaire {
        protected String numeroCompte;
        protected String iban;
        protected double solde;
        protected List<Transaction> historique;

        public abstract void calculerInterets();
        public abstract double getDecouvertAutorise();
        public abstract double getTauxInteret();

        public void deposer(double montant) { ... }
        public abstract void retirer(double montant)
          throws SoldeInsuffisantException, DecouvertDepasseException;
}
```

Exceptions Personnalisées

```
public class SoldeInsuffisantException extends Exception {
        public SoldeInsuffisantException(String message) {
                super(message);
        }
}
```

6.1.2 Couche Contrôleur

AuthControleur.java

```
public class AuthControleur {
        private Map<String, Client> clients;
        private Client clientConnecte;

        public boolean authentifier(String email, String motDePasse) {
          // Validation des identifiants
        }
        public boolean inscrireClient(String nom, String prenom,
                        String email, String motDePasse) {
          // Création d'un nouveau client
        }
}
```

6.1.3 Couche Vue

FenetrePrincipale.java

```
public class FenetrePrincipale extends JFrame {
        private CardLayout cardLayout;
        private JPanel mainPanel;

        public FenetrePrincipale() {
                setTitle("Banque en Ligne");
                setSize(1000, 700);
                setLocationRelativeTo(null);
                initialiserComposants();
                configurerLayout();
        }
}
```

6.2 Algorithmes Clés

6.2.1 Calcul des Intérêts

```
public class CalculInteretsService {
        public void calculerInteretsMensuels() {
                for (CompteBancaire compte : comptes) {
                   compte.calculerInterets();
                }
        }
}
```

6.2.2 Génération d'IBAN

```
private void genererIBAN() {
        String banqueCode = "30002";
        String guichetCode = "00550";
        String numeroCompteComplet = String.format("%011d",
           Math.abs(numeroCompte.hashCode()));
        String cle = "76";

        this.iban = "FR" + cle + banqueCode + guichetCode +
                numeroCompteComplet + "00";
}
```

6.2.3 Filtrage de l'Historique

```
public List<Transaction> getHistorique12Mois() {
        LocalDateTime dateLimite = LocalDateTime.now().minusMonths(12);
        return historique.stream()
           .filter(t -> t.getDate().isAfter(dateLimite))
           .collect(Collectors.toList());
}
```

6.3 Gestion des Erreurs et Exceptions

6.3.1 Hiérarchie des Exceptions

Exception

├── IllegalArgumentException

├── IllegalStateException

└── [Nos Exceptions]

  ├── SoldeInsuffisantException

  ├── DecouvertDepasseException

  ├── PlafondDepasseException

  └── CompteInexistantException

6.3.2 Stratégie de Gestion

- **Exceptions vérifiées** pour les erreurs métier récupérables
- **Exceptions non vérifiées** pour les erreurs de programmation
- **Messages d'erreur localisés** en français
- **Journalisation** des erreurs pour le débogage

6.4 Interface Utilisateur

6.4.1 Conception de l'UI

- **FenetreConnexion :** Authentification et inscription
- **FenetreTableauBord :** Vue d'ensemble des comptes
- **FenetreVirement :** Transferts entre comptes
- **FenetreHistorique :** Consultation des transactions
- **FenetreRelevesPDF :** Génération de documents

6.4.2 Principes d'UX Appliqués

- **Consistance :** Design uniforme sur toutes les fenêtres
- **Feedback immédiat :** Messages de confirmation/erreur
- **Accessibilité :** Couleurs contrastées, polices lisibles

- **Efficacité :** Raccourcis clavier, actions rapides

6.4.3 Palette de Couleurs

- **Bleu** (#0066CC) : Actions principales
- **Vert** (#28A745) : Actions positives
- **Rouge** (#DC3545) : Actions dangereuses
- **Gris** (#6C757D) : Actions neutres
- **Fond** (#F8F9FA) : Arrière-plan

7. TESTS ET VALIDATION

7.1 Stratégie de Test

7.1.1 Tests Unitaires

```
@Test
public void testDepotCompteCourant() {
        CompteCourant compte = new CompteCourant("CC001", "C001");
        compte.deposer(1000);
        assertEquals(1000, compte.getSolde(), 0.001);
}

@Test(expected = SoldeInsuffisantException.class)
public void testRetraitSoldeInsuffisant() throws Exception {
        CompteCourant compte = new CompteCourant("CC001", "C001");
        compte.retirer(1000);
}
```

7.1.2 Tests d'Intégration

- Authentification avec données de test
- Virements entre comptes
- Génération de PDF
- Calcul des intérêts

7.1.3 Tests de Performance

- Temps de réponse des opérations
- Utilisation mémoire
- Scalabilité avec un grand nombre de transactions

7.2 Cas de Test

7.2.1 Scénarios Positifs

1. **Inscription réussie** → Redirection vers connexion
2. **Authentification valide** → Affichage tableau de bord
3. **Dépôt sur compte** → Mise à jour du solde
4. **Virement interne** → Débit/Crédit des comptes
5. **Génération PDF** → Création du fichier

7.2.2 Scénarios d'Erreur

1. **Authentification échouée** → Message d'erreur
2. **Solde insuffisant** → Exception spécifique
3. **Plafond dépassé** → Blocage de l'opération
4. **Fichier existant** → Demande de confirmation

7.3 Validation Fonctionnelle

Fonctionnalité	Testé	Validé	Commentaire
Authentification	✅	✅	100% fonctionnel
Gestion comptes	✅	✅	Toutes opérations
Virements	✅	✅	Interne/externe
Historique	✅	✅	12 mois
PDF	✅	✅	Format correct
Intérêts	✅	✅	Calcul précis

8. DIFFICULTÉS RENCONTRÉES

8.1 Difficultés Techniques

8.1.1 Gestion des Transactions Concurrentes

Problème : Risque d'incohérence lors d'opérations simultanées
Solution : Implémentation de vérifications de cohérence et validation en deux phases

8.1.2 Génération de PDF

Problème : Intégration de bibliothèques externes
Solution : Recherche de solutions alternatives et création d'un fallback en texte

8.1.3 Architecture MVC

Problème : Séparation claire des responsabilités
Solution : Refactoring itératif et révision de l'architecture

8.2 Difficultés Conceptuelles

8.2.1 Modélisation des Comptes

Problème : Différences entre comptes courants et épargne
Solution : Utilisation de l'héritage et du polymorphisme

8.2.2 Gestion des Exceptions

Problème : Hiérarchie cohérente des exceptions
Solution : Création d'exceptions métier spécifiques

8.3 Solutions Innovantes

1. **Génération d'IBAN :** Algorithme maison pour simuler des IBAN réalistes
2. **Historique intelligent :** Filtrage automatique des transactions anciennes
3. **Interface adaptable :** Layouts dynamiques selon le contenu
4. **Fallback système :** Alternative texte si PDF impossible

9. AMÉLIORATIONS POSSIBLES

9.1 Améliorations Fonctionnelles

9.1.1 Court Terme

- **Notifications :** Alertes pour transactions importantes
- **Budgétisation :** Catégorisation des dépenses
- **Objectifs d'épargne :** Suivi de projets financiers
- **Export des données :** Formats CSV, Excel

9.1.2 Moyen Terme

- **Application mobile :** Version Android/iOS
- **API REST :** Interface pour applications tierces
- **Multi-devises :** Gestion de comptes en différentes monnaies
- **Analytics :** Tableaux de bord avancés

9.1.3 Long Terme

- **Intelligence Artificielle :** Suggestions financières
- **Blockchain :** Transactions sécurisées
- **Open Banking :** Interconnexion avec autres banques
- **Reconnaissance vocale :** Commandes vocales

9.2 Améliorations Techniques

9.2.1 Performance

- **Cache :** Mise en cache des données fréquentes
- **Base de données :** Migration vers MySQL/PostgreSQL
- **Asynchrone :** Opérations non bloquantes
- **Compression** : Optimisation de l'utilisation mémoire

9.2.2 Sécurité

- **Chiffrement :** Données sensibles chiffrées

- **2FA :** Authentification à deux facteurs
- **Audit :** Journalisation complète des actions
- **Tests de pénétration :** Validation sécurité

9.2.3 Qualité de Code

- **Tests automatisés :** Couverture de code > 80%
- **CI/CD :** Intégration et déploiement continus
- **Documentation :** Javadoc complète
- **Revue de code :** Processus formel

9.3 Évolution Architecturale

9.3.1 Microservices

gateway/

├── auth-service/

├── account-service/

├── transaction-service/

└── reporting-service/

9.3.2 Cloud Native

- **Conteneurisation :** Docker/Kubernetes
- **Orchestration :** Gestion des services
- **Monitoring :** Métriques en temps réel
- **Auto-scaling :** Adaptation à la charge

10. CONCLUSION

10.1 Bilan du Projet

10.1.1 Objectifs Atteints

✅ **Fonctionnalités complètes :** Toutes les spécifications implémentées

✅ **Architecture robuste :** Respect des principes SOLID et MVC

✅ **Interface utilisable :** Design moderne et intuitif

✅ **Code de qualité :** Maintenable, documenté, testé

✅ **Gestion d'erreurs :** Système complet d'exceptions

10.1.2 Compétences Développées

- **Conception objet :** Modélisation de systèmes complexes
- **Développement Swing :** Interfaces graphiques avancées
- **Gestion de projet :** Organisation et planification
- **Résolution de problèmes :** Débogage et optimisation
- **Documentation :** Rédaction technique complète

10.2 Apprentissages Clés

10.2.1 Techniques

1. **Abstraction :** Modélisation efficace des concepts métier
2. **Polymorphisme :** Flexibilité dans les implémentations
3. **Exceptions :** Gestion robuste des cas d'erreur
4. **Interface utilisateur :** Principes d'UX appliqués

10.2.2 Méthodologiques

1. **Itératif :** Développement par incréments
2. **Test-driven :** Validation continue du code
3. **Documentation :** Importance de la communication technique
4. **Versioning :** Gestion des évolutions du code

10.3 Perspectives

Ce projet représente une base solide pour des développements futurs. Les compétences acquises sont directement transférables à des projets professionnels dans le domaine financier ou dans tout autre domaine nécessitant des applications Java robustes.

L'application démontre qu'avec une conception soignée et une implémentation rigoureuse, il est possible de créer des systèmes complexes tout en maintenant la qualité et la maintenabilité du code.

11. ANNEXES

11.1 Guide d'Installation

Prérequis

bash

Vérifier Java

java -version

Doit afficher Java 17 ou supérieur

Structure des dossiers

BanqueEnLigne/

├── src/

├── lib/ (optionnel pour PDF)

└── compile.bat

Compilation

bash

Windows

.\compile.bat

Linux/Mac

chmod +x compile.sh

./compile.sh

11.2 Guide d'Utilisation

Première Utilisation

1. **Lancer l'application :** Exécuter compile.bat
2. **S'inscrire** ou utiliser un compte test

3. **Explorer les fonctionnalités** via le tableau de bord

Comptes de Test

Email: wail.anai@email.com

Mot de passe: password123

Email: yasser.oussahel@email.com

Mot de passe: password456

11.3 Code Source Complet

Le code source est disponible dans les répertoires :

- src/banque/ : Code source Java
- docs/ : Documentation supplémentaire
- screenshots/ : Captures d'écran

11.4 Bibliographie

Ouvrages de Référence

1. Effective Java - Joshua Bloch
2. Design Patterns - GoF
3. Clean Code - Robert C. Martin
4. Java Swing - Marc Loy

Ressources en Ligne

1. Documentation Oracle Java
2. Tutoriels Swing
3. Forum Stack Overflow
4. GitHub repositories

Outils Utilisés

1. Visual Studio Code
2. Git pour le versioning
3. Draw.io pour les diagrammes

"Un programme est un poème dont chaque ligne doit être parfaite."
– Anonyme

Ce projet a été réalisé avec rigueur et passion, dans le respect des meilleures pratiques de développement logiciel. Il témoigne de l'acquisition solide des compétences en programmation Java et en conception orientée objet.

Date : 29-01-2026