Rapport de Projet - Système de Transfert d'Argent

Table des matières

Introduction

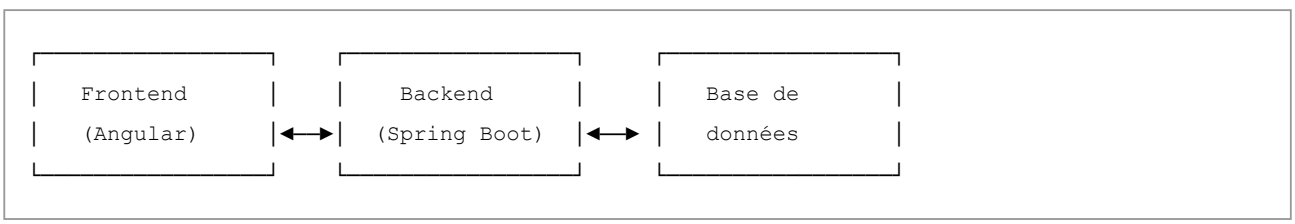
Ce projet consiste en la développement d'une application de transfert d'argent complète, utilisant une architecture moderne basée sur des microservices. L'application permet aux utilisateurs d'effectuer des opérations bancaires de base telles que les transferts, dépôts et retraits, tout en gérant différents niveaux d'abonnement (Free/Premium).

Objectifs du projet

- Développer une plateforme sécurisée de transfert d'argent
- Implémenter un système d'authentification robuste avec JWT
- Mettre en place une infrastructure DevOps complète
- Assurer la scalabilité grâce à Kubernetes
- Automatiser les déploiements avec CI/CD

Architecture générale

L'architecture du système suit un modèle en couches avec séparation claire entre le frontend et le backend :



Composants principaux

- **Frontend Angular** : Interface utilisateur responsive
- **Backend Spring Boot** : API RESTful et logique métier
- **Base de données** : Stockage des données utilisateurs et transactions
- **Services externes** : Notification par email, authentification

---

Technologies utilisées

Backend

- **Java Spring Boot** : Framework principal pour l'API REST
- **Spring Security** : Gestion de la sécurité et authentification
- **JWT (JSON Web Tokens)** : Authentification stateless
- **JPA/Hibernate** : ORM pour la gestion des données
- **Maven** : Gestionnaire de dépendances

Frontend

- **Angular** : Framework frontend SPA
- **TypeScript** : Langage de développement
- **Angular Material** : Composants UI
- **RxJS** : Programmation réactive

Infrastructure et DevOps

- **Docker** : Containerisation des applications
- **Kubernetes** : Orchestration des conteneurs
- **Jenkins** : Intégration continue (CI)
- **ArgoCD** : Déploiement continu (CD)
- **SMTP4Dev** : Simulation serveur SMTP

Backend - Spring Boot

Architecture du backend

Le backend suit une architecture en couches bien structurée :

```
Controllers → Services → Repository → Database
     ↓           ↓
   DTOs      Entities
     ↓           ↓
  Mappers   Validations
     ↓           ↓
Exceptions   Enums
     ↓           ↓
Security   Configuration
     ↓           ↓
Annotations
```

Structure du projet backend

- **Annotations** : Annotations personnalisées pour la validation et la sécurité
- **Configuration** : Classes de configuration Spring (Security, CORS, etc.)
- **Controllers** : Endpoints REST, gestion des requêtes HTTP
- **DTOs** : Data Transfer Objects pour les échanges client-serveur
- **Entities** : Entités JPA représentant les tables de base de données
- **Enums** : Énumérations (statuts de transaction, types de comptes, etc.)
- **Exceptions** : Gestion centralisée des exceptions personnalisées
- **Mappers** : Conversion entre Entities et DTOs (MapStruct ou mapping manuel)
- **Repository** : Couche d'accès aux données avec Spring Data JPA
- **Security** : Configuration JWT, filtres de sécurité
- **Services** : Logique métier et orchestration des opérations
- **Validations** : Validateurs personnalisés pour les données métier

Détail des couches backend

Controllers

Gestion des endpoints REST avec validation des entrées :

DTOs (Data Transfer Objects)

Objects pour l'échange de données entre client et serveur :

- `TransferRequestDTO` : Données de demande de transfert
- `TransferResponseDTO` : Réponse après transfert
- `UserDTO` : Informations utilisateur
- `AccountDTO` : Données de compte

Entities

Entités JPA représentant le modèle de données :

- `User` : Utilisateur de la plateforme
- `Account` : Compte bancaire

- `Transaction` : Historique des transactions
- `Subscription` : Abonnements Free/Premium

Services

Logique métier encapsulée dans des services :

- `TransferService` : Gestion des transferts
- `AccountService` : Gestion des comptes
- `SubscriptionService` : Gestion des abonnements
- `NotificationService` : Envoi d'emails

Repository

Couche d'accès aux données avec Spring Data JPA :

Mappers

Conversion entre Entities et DTOs :

Validations

Validateurs personnalisés pour les règles métier :

- Validation des montants de transfert
- Vérification des limites d'abonnement
- Contrôle de la disponibilité des fonds

Exceptions

Gestion centralisée des erreurs :

- `InsufficientFundsException` : Fonds insuffisants
- `AccountNotFoundException` : Compte introuvable
- `InvalidTransactionException` : Transaction invalide

Gestion des comptes

- Création et gestion des comptes utilisateurs
- Authentification et autorisation
- Gestion des profils utilisateurs

Opérations financières

- **Transferts d'argent** : Entre comptes utilisateurs
- **Dépôts** : Alimentation des comptes
- **Retraits** : Extraction de fonds
- **Historique des transactions** : Traçabilité complète

Système d'abonnement

- **Abonnement Free** : Fonctionnalités de base
- **Abonnement Premium** : Fonctionnalités avancées

    - Limites de transfert plus élevées
    - Frais réduits
    - Support prioritaire

Structure des endpoints

Authentification - `/api/v1/auth`

- `POST /register` - Inscription d'un nouvel utilisateur
- `POST /login` - Connexion utilisateur
- `POST /logout` - Déconnexion utilisateur

Gestion des comptes - `/api/v1/account`

- `POST /verify-pin` - Vérification du code PIN
- `PUT /update-subscription/{id}` - Mise à jour de l'abonnement (Free → Premium)

Transactions - `/api/v1/transaction`

- `POST /` - Gestionnaire de transaction générique (legacy)
- `POST /deposit` - Opération de dépôt d'argent
- `POST /withdraw` - Opération de retrait d'argent
- `POST /transfer` - Opération de transfert entre comptes

Gestion des mots de passe - `/api/v1/password`

- `POST /forgot` - Demande de réinitialisation de mot de passe
- `POST /validate-code` - Validation du code de réinitialisation
- `POST /reset` - Réinitialisation effective du mot de passe

Reçus et justificatifs - `/api/v1/receipts`

- `GET /{receiptNumber}` - Récupération d'un reçu par numéro
- `GET /by-transaction/{transactionNumber}` - Reçu par numéro de transaction
- `GET /by-user/{userId}` - Tous les reçus d'un utilisateur

Frontend - Angular

Architecture frontend

L'application Angular utilise une architecture basée sur des modules :

- **Core Module** : Services partagés, intercepteurs
- **Shared Module** : Composants réutilisables
- **Feature Modules** : Fonctionnalités spécifiques (auth, dashboard, transactions)

Composants principaux

Dashboard utilisateur

- Vue d'ensemble du solde
- Transactions récentes
- Accès rapide aux opérations

Module de transfert

- Formulaire de transfert sécurisé
- Validation en temps réel
- Confirmation des transactions

Gestion des abonnements

- Comparaison Free vs Premium
- Processus de mise à niveau
- Facturation et historique

Services Angular

- **AuthService** : Gestion de l'authentification
- **TransactionService** : Opérations financières
- **NotificationService** : Alertes et notifications
- **SubscriptionService** : Gestion des abonnements

Sécurité et authentification

JWT Stateless

L'application utilise une approche d'authentification stateless avec JWT :

Avantages

- Scalabilité : Pas de stockage de session côté serveur
- Performance : Validation locale des tokens
- Flexibilité : Support multi-domaines

Implémentation

1. **Login** : Génération du JWT après authentification
2. **Stockage** : Token stocké côté client (localStorage/sessionStorage)
3. **Validation** : Vérification automatique à chaque requête
4. **Expiration** : Gestion automatique de l'expiration

Mesures de sécurité

- **Chiffrement des mots de passe** : BCrypt
- **Validation des entrées** : Annotations de validation personnalisées
- **CORS configuré** : Protection contre les requêtes cross-origin malveillantes

- **Gestion centralisée des exceptions** : Réponses d'erreur uniformes
- **Configuration sécurisée** : Séparation des configurations par environnement

---

Containerisation et orchestration

Docker

Chaque composant de l'application est containerisé :

Backend Dockerfile

FROM eclipse-temurin:17-jdk-alpine WORKDIR /app COPY pom.xml . COPY mvnw . COPY .mvn .mvn COPY src src RUN ./mvnw clean package -DskipTests EXPOSE 8080 CMD ["java", "-jar", "target/caly-back-v2-0.0.1-SNAPSHOT.jar"]

Docker Compose

Le fichier `docker-compose.yml` orchestre tous les services :

```
version: '3.1'

services:
  smtp4dev:
    image: rnwood/smtp4dev:v3
    restart: always
    ports:
      - '9081:80'
      - '9025:25'
```

CI/CD avec Jenkins et ArgoCD

Architecture CI/CD

```
Code → Jenkins (CI) → Docker Registry → ArgoCD (CD) → Kubernetes
```

Jenkins Pipeline

Jenkinsfile avec Shared Library

Le projet utilise une shared library Jenkins pour réutiliser les pipelines :

```
@Library('M1\_uvs\_sharedlib') \_

pipeline {
agent any

environment {
    APP_NAME = 'transfert-back'
    ENV = 'dev'
    REGISTRY = 'Transfertregistry:30500'
}

stages {
    stage('Checkout') {
        steps {
            checkoutRepo(
                repoUrl: 'https://github.com/CybeRooT-01/transfert-recherche-operationnel.git',
                branch: 'main'
            )
        }
    }

    stage('Build App') {
        steps {
            buildBackendApp()
        }
    }

    stage('Docker Build & Push') {
        steps {
            dockerBuildAndPush(
                registry: "${REGISTRY}",
                imageName: "${APP_NAME}"
            )
        }
    }

    stage('Deploy') {
        steps {
            deployArgoCD(appName: "${APP_NAME}")
        }
    }
}

post {
    always {
        postAction(currentBuild.currentResult)
    }
```

```
}

}
```

Configuration Jenkins

- **Master** : Orchestration des builds
- **Agent** : Exécution des tâches de build
- **Self-hosted Registry** : Stockage des images Docker

ArgoCD pour le déploiement continu

ArgoCD surveille le repository Git et déploie automatiquement les changements :

Avantages

- **GitOps** : Source de vérité dans Git
- **Synchronisation automatique** : Déploiement sans intervention
- **Rollback facile** : Retour aux versions précédentes
- **Monitoring** : Suivi en temps réel des déploiements

Infrastructure Kubernetes

Organisation des manifests

Le dossier `kubernetes/` contient tous les manifests nécessaires :

```
kubernetes/
├── manifests/
│   ├── deployment-registry.yaml
│   ├── exposeArgoCD.yaml
│   ├── pvc-pv.yaml
│   ├── backend-service.yaml
│   ├── service-for-registry.yaml
└── scripts/
    ├── setupMasterNode.sh
    ├── setupArgoCd.sh
    └── setupWorkers.sh
```

Composants Kubernetes déployés

Deployments

- **Backend Deployment** : Application Spring Boot
- **Frontend Deployment** : Application Angular
- **Database StatefulSet** : Base de données PostgreSQL

Services

- **Backend Service** : Exposition de l'API via NodePort ou LoadBalancer
- **Frontend Service** : Exposition du frontend via NodePort
- **Database Service** : Accès interne à la base de données (ClusterIP)

Exposition des services

Les services sont exposés de manière simple sans Ingress :

```
# Backend Service
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30080
  selector:
    app: backend
```

Scripts d'infrastructure

Tout les scripts et manifest sont dans le github, dans le dossier /kubernetes

Simulation SMTP avec Docker

SMTP4Dev

Pour le développement et les tests, nous utilisons SMTP4Dev qui simule un serveur SMTP :

Caractéristiques

- **Interface web** : Consultation des emails envoyés

Configuration Docker

```
version: '3.1'
services:
  smtp4dev:
    image: rnwood/smtp4dev:v3
    restart: always
    ports:
      - '9081:80'
      - '9025:25'
```

Monitoring et observabilité

On a juste mis en place metric-server sur le cluster. Pas de prometheus ou d'outil d'analyse pousser.

Qualité du code

- **SonarQube** : Analyse statique du code
- **Checkstyle** : Respect des conventions Java
- **ESLint** : Linting du code TypeScript
- **Couverture de code** : JaCoCo pour Java, Istanbul pour Angular

Sécurité et conformité

Conformité réglementaire

- **PCI DSS** : Standards pour le traitement des paiements
- **RGPD** : Protection des données personnelles
- **KYC/AML** : Procédures de connaissance client

Audit

- **Archivage** : Conservation des données légalement requises d'où l'utilisation un softdelete

Conclusion

Ce projet de transfert d'argent représente une implémentation complète d'une application financière moderne, intégrant les meilleures pratiques en matière de :

Points forts du projet

1. **Architecture robuste** : Séparation claire des responsabilités
2. **Sécurité avancée** : JWT stateless, chiffrement, validation
3. **DevOps mature** : CI/CD automatisé avec Jenkins et ArgoCD
4. **Scalabilité** : Infrastructure Kubernetes prête pour la production
5. **Observabilité** : Monitoring et logs centralisés

Apprentissages clés

- L'importance de la sécurité dans les applications financières
- La valeur d'une infrastructure automatisée pour la productivité
- L'impact des patterns architecturaux sur la maintenabilité
- La nécessité du monitoring pour les applications critiques

Impact business

Le projet établit une base solide pour :

- Expansion rapide du nombre d'utilisateurs
- Ajout de nouvelles fonctionnalités
- Conformité réglementaire
- Confiance des utilisateurs dans la sécurité

L'architecture mise en place permet une évolution agile de la plateforme tout en maintenant les standards de sécurité et de performance requis dans le domaine financier.

Annexes

Glossaire technique

- **JWT** : JSON Web Token - Standard pour l'authentification
- **CI/CD** : Continuous Integration/Continuous Deployment
- **ArgoCD** : Outil de déploiement continu GitOps
- **SMTP4Dev** : Simulateur de serveur SMTP pour le développement
- **StatefulSet** : Objet Kubernetes pour les applications avec état

Références

- Spring Boot Documentation (https://spring.io/projects/spring-boot)
- Angular Documentation (https://angular.io/docs)
- Kubernetes Documentation (https://kubernetes.io/docs/)
- Jenkins Shared Libraries (https://www.jenkins.io/doc/book/pipeline/shared-libraries/)
- ArgoCD Documentation (https://argo-cd.readthedocs.io/)

Rapport de projet recherche opérationnel realise par:

Thierno birahim Gueye (N00686020201) Elhadji Malick Ndao (N06603120201) Siabatou Sane (N04163320202)