

# DIGITAL BANKING

DOCUMENTATION TECHNIQUE

## RAPPORT COMPLET

Architecture, Fonctionnalités & API

**Youssef Faik**

June 2025



Angular



Spring Boot



MySQL

JWT Auth

## Résumé

## Résumé

Ce rapport détaille les fonctionnalités clés de l'application **Digital Banking**, une plateforme bancaire moderne conçue pour offrir une expérience numérique intuitive et efficace.

Le document commence par une présentation détaillée de l'architecture en couches de l'application, suivie d'une exploration approfondie des processus d'authentification sécurisée, de la gestion complète des clients et des comptes bancaires, ainsi que de l'ensemble des opérations bancaires courantes (débit, crédit, virement).

Une section est également consacrée à la documentation API interactive générée avec Swagger/OpenAPI, permettant aux développeurs de comprendre et de tester facilement l'ensemble des services backend.

Chaque fonctionnalité est illustrée par des captures d'écran détaillées pour une compréhension visuelle immédiate des interfaces et des flux utilisateur.

## TABLE DES MATIÈRES

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Objectifs de l'Application	2
1.2	Technologies et Stack Technique	2
<b>2</b>	<b>Architecture Générale de l'Application</b>	<b>3</b>
2.1	Description Détaillée de l'Architecture	5
2.1.1	Frontend (Angular)	5
2.1.2	Communication (HTTP/REST)	5
2.1.3	Backend (Spring Boot)	5
2.1.4	Persistence Layer	6
2.1.5	Transversal Concerns	6
<b>3</b>	<b>Authentification</b>	<b>8</b>
3.1	Page de Connexion Initiale	8
3.2	Gestion des Erreurs d'Authentification	8
3.3	Accès Réussi au Tableau de Bord	9
<b>4</b>	<b>Tableau de Bord</b>	<b>10</b>
4.1	Vue d'Ensemble Interactive	10
<b>5</b>	<b>Gestion des Clients</b>	<b>11</b>
5.1	Consultation de la Base Clients	11
5.2	Création de Nouveaux Profils	11
5.2.1	Formulaire de Saisie	12
5.2.2	Formulaire Complété	12
5.3	Modification et Suppression de Clients	12
<b>6</b>	<b>Documentation API avec Swagger</b>	<b>14</b>
6.1	Vue Générale de Swagger UI	14
6.2	Exploration et Test des Endpoints	15
<b>7</b>	<b>Conclusion</b>	<b>18</b>

# INTRODUCTION

## Présentation générale

L'application **Digital Banking** représente une solution technologique avancée visant à moderniser et simplifier l'interaction des utilisateurs avec les services bancaires traditionnels.

Cette plateforme numérique offre une interface utilisateur intuitive et responsive, développée avec les technologies web les plus récentes, permettant aux utilisateurs d'effectuer l'ensemble de leurs opérations bancaires de manière sécurisée et efficace.

Le présent document a pour objectif de fournir une documentation technique complète de l'application, expliquant son architecture, ses fonctionnalités principales et la manière dont ses API sont exposées et documentées.

## 1.1 Objectifs de l'Application

### Une Plateforme Bancaire Intégrée et Moderne

- **Modernisation** : Proposer une interface utilisateur contemporaine, intuitive et ergonomique suivant les meilleures pratiques UX/UI
- **Sécurité** : Garantir la protection des données et des transactions via des mécanismes d'authentification et d'autorisation robustes basés sur JWT
- **Simplicité** : Offrir une expérience utilisateur fluide et intuitive, minimisant la courbe d'apprentissage
- **Complétude** : Couvrir l'ensemble des besoins bancaires quotidiens des utilisateurs dans une interface unifiée
- **Transparence API** : Fournir une documentation API claire et interactive via Swagger pour faciliter l'intégration et l'évolution
- **Modularité** : Adopter une architecture en couches clairement définie pour favoriser la maintenabilité et l'extensibilité du code

## Note technique

L'application a été conçue selon les principes de l'architecture orientée services (SOA), avec une séparation nette entre le frontend Angular et le backend Spring Boot. Cette approche permet d'isoler les préoccupations et d'optimiser indépendamment les différentes parties du système.

## 1.2 Technologies et Stack Technique

### Stack Technique

#### Frontend

- Angular 17
- TypeScript 5
- HTML5/SCSS
- Bootstrap 5
- Chart.js
- RxJS

#### Backend

- Spring Boot 3.4
- Spring Security
- Spring Data JPA
- Java 21
- MySQL 8
- Swagger/OpenAPI 3

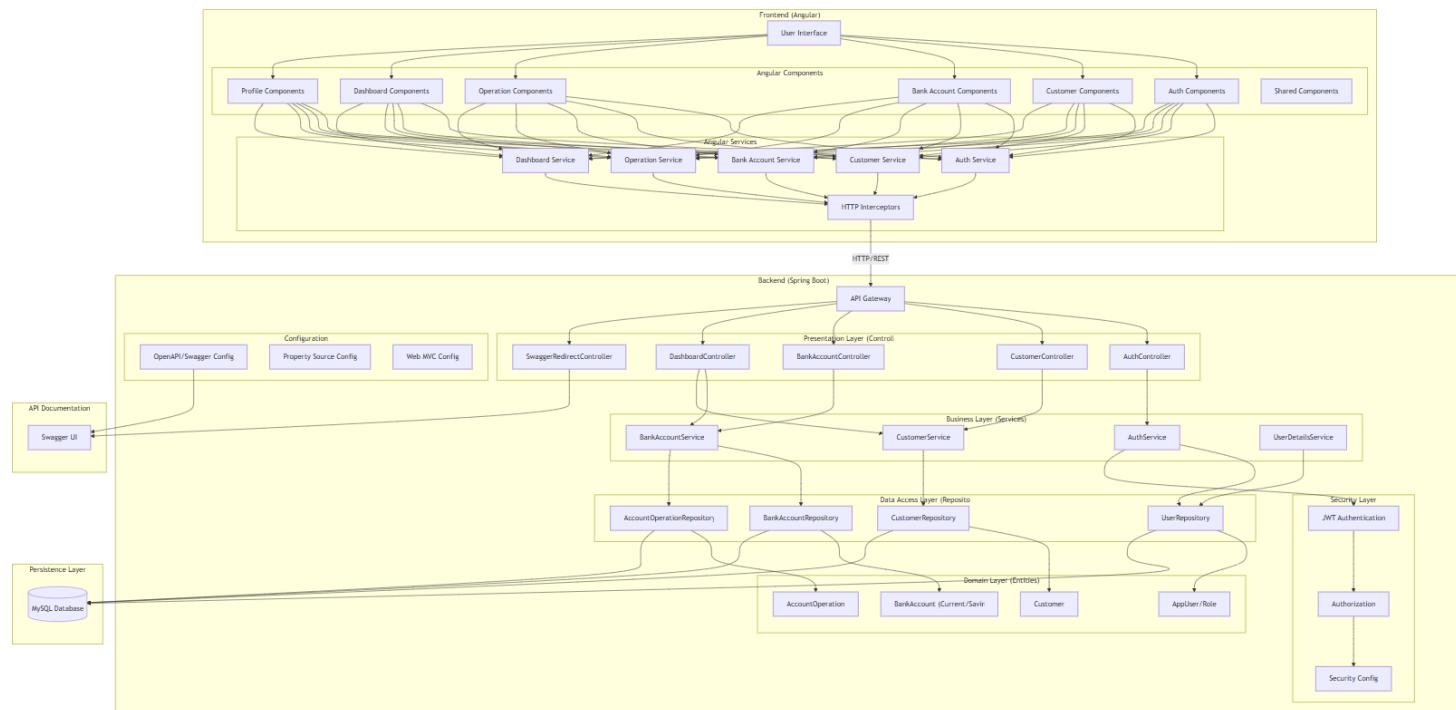
## ARCHITECTURE GÉNÉRALE DE L'APPLICATION



### Vue d'ensemble

L'application Digital Banking est conçue selon une architecture client-serveur moderne, tirant parti des frameworks [Angular](#) pour le frontend et [Spring Boot](#) pour le backend. Cette séparation claire des préoccupations permet une meilleure maintenabilité, évolutivité et des cycles de développement indépendants pour l'interface utilisateur et la logique métier.

Le diagramme présenté en Figure 1 illustre les principaux composants et leurs interactions.



**Figure 1** Diagramme de l'Architecture Générale de l'Application Digital Banking

## 2.1 Description Détaillée de l'Architecture

### Organisation de l'Architecture






L'architecture de Digital Banking est organisée en couches distinctes avec des responsabilités clairement définies, suivant les principes de conception modernes pour assurer la séparation des préoccupations et faciliter la maintenance.

#### 2.1.1 Frontend (Angular)

##### Couche Présentation

La partie cliente de l'application est développée avec Angular. Elle est responsable de l'interface utilisateur (UI) et de l'expérience utilisateur (UX).

##### Composants Principaux

-  Composants de profil utilisateur
- Composants du tableau de bord
- Composants des opérations bancaires
-  Composants de gestion des comptes
-  Composants de gestion des clients
-  Composants d'authentification
-  Composants partagés réutilisables

##### Services Angular

- DashboardService
- OperationService
- BankAccountService
- CustomerService
- AuthService

##### Intercepteurs HTTP

- AuthInterceptor
- ErrorInterceptor

#### 2.1.2 Communication (HTTP/REST)

##### Communication RESTful

Le frontend communique avec le backend via des requêtes HTTP, suivant les principes de l'architecture REST (Representational State Transfer). Les données sont échangées au format JSON.

Méthode HTTP	Utilisation
GET	Récupération de données
POST	Création de nouvelles ressources
PUT	Mise à jour complète de ressources
PATCH	Mise à jour partielle de ressources
DELETE	Suppression de ressources

#### 2.1.3 Backend (Spring Boot)

##### Architecture en couches

## Couches applicatives

— **API Gateway** : `APIGateway`

— **Étape 3**: Accès aux données (Repositories)

## Avantages de cette architecture

- Séparation claire des responsabilités
- Testabilité accrue de chaque couche
- Facilité de maintenance et d'évolution
- Réutilisabilité des composants
- Robustesse face aux changements

## Détail des couches backend

- **Controllers** : `DashboardController`, `BankAccountController`, `CustomerController`, `AuthController`, `SwaggerRedirectController`
- **Services** : `BankAccountService`, `CustomerService`, `AuthService`, `UserDetailsService`
- **Repositories** : `AccountOperationRepository`, `BankAccountRepository`, `CustomerRepository`, `UserRepository`
- **Entities** : `AccountOperation`, `BankAccount` (avec `CurrentAccount` et `SavingAccount`), `Customer`, `AppUser`, `Role`

## 2.1.4 Persistence Layer

## Système de persistance

**MySQL Database** est utilisée comme système de gestion de base de données relationnelle pour stocker toutes les données persistantes de l'application :

- Informations des clients
- Détails des comptes bancaires
- Historique des transactions
- Utilisateurs et rôles du système

## 2.1.5 Transversal Concerns

## Sécurité

- **JWT Authentication** : Authentification par jetons
- **Authorization** : Gestion des droits d'accès
- **Security Config** : Règles de sécurité centralisées

## Documentation API

- **Swagger UI** : Interface d'exploration de l'API
- **OpenAPI** : Spécification standardisée
- **Annotations** : Documentation intégrée au code

## Note technique

Cette architecture multicouche favorise le découplage, la testabilité et la clarté du code, des aspects essentiels pour le développement et la maintenance d'une application complexe comme Digital Banking.

**Point clé**

En résumé, l'architecture de Digital Banking illustre une application moderne, tirant parti des meilleures pratiques pour créer un système bancaire numérique robuste, sécurisé et maintenable.



## AUTHENTIFICATION

### Sécurité renforcée

L'accès à l'application est protégé par un système d'authentification robuste basé sur JSON Web Tokens (JWT), garantissant que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités bancaires.

Protection contre les attaques par force brute

🔑 Gestion sécurisée des sessions utilisateur

🔒 Contrôle d'accès basé sur les rôles

### 3.1 Page de Connexion Initiale

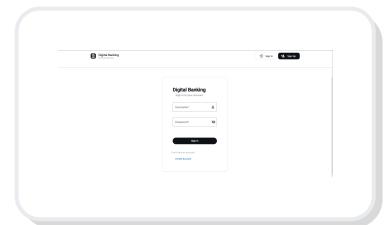
#### Interface d'authentification

Au lancement de l'application, l'utilisateur est accueilli par une interface de connexion épurée et professionnelle. Cette page constitue le point d'entrée sécurisé vers l'ensemble des fonctionnalités bancaires.

👤 Champ pour identifiant unique

🔒 Champ de mot de passe masqué

Bouton de connexion intuitif



#### Flux d'authentification sécurisé



#### POST /api/auth/login

**Description:** Endpoint d'authentification qui valide les identifiants et génère un JWT

**Réponse:** {token: "eyJhbGciOiJIUzI1...", username: "admin\_user", roles: ["ADMIN"]}

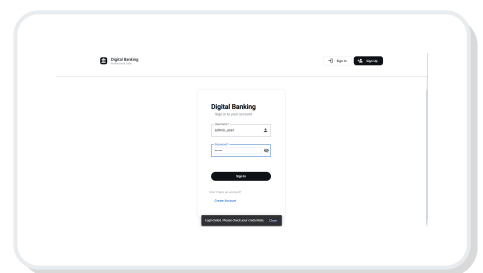
### 3.2 Gestion des Erreurs d'Authentification

#### Mécanisme de sécurité proactif

Le système intègre une gestion d'erreurs claire et informative. Lorsque des identifiants incorrects sont saisis, un message d'erreur explicite guide l'utilisateur sans compromettre la sécurité du système.

Les techniques implémentées incluent :

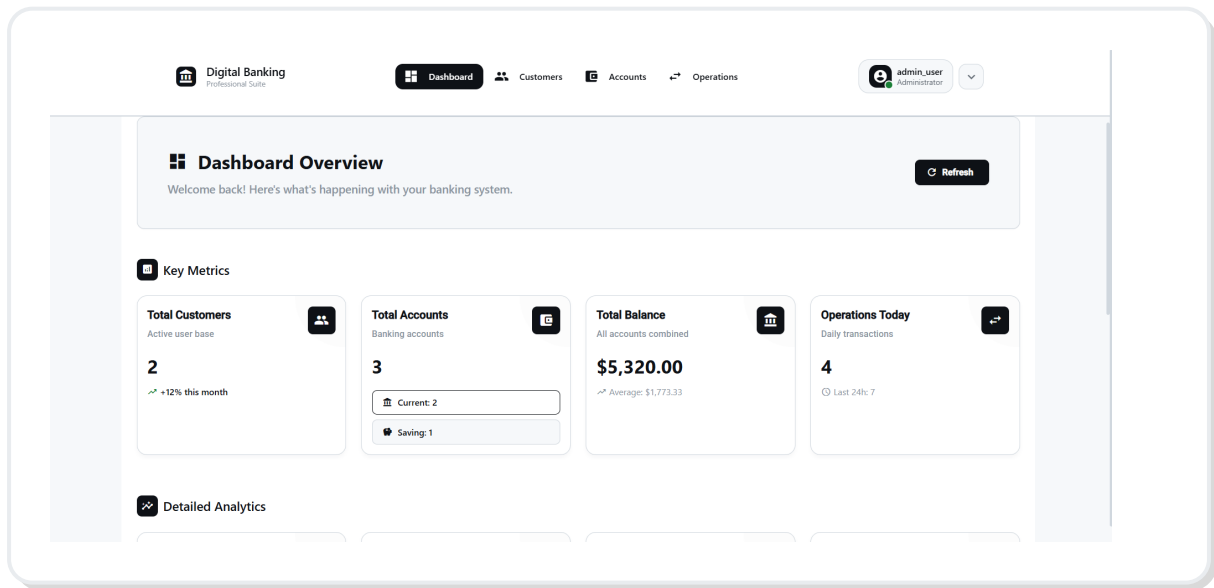
- Messages d'erreur génériques ne révélant pas d'informations sensibles
- Temporisation pour contrer les attaques par force brute
- Journalisation des tentatives d'accès échouées



### 3.3 Accès Réussi au Tableau de Bord

#### Authentification réussie

Après une authentification réussie avec les identifiants valides (`admin_user` / `123456`), l'utilisateur est automatiquement redirigé vers le tableau de bord principal, offrant une vue d'ensemble immédiate de son espace bancaire.



**Figure 2** Redirection post-authentification - Accès direct au tableau de bord personnalisé

#### Note technique

Le système d'authentification utilise Spring Security côté backend et des intercepteurs HTTP côté frontend pour garantir que chaque requête est accompagnée du JWT valide. Cette approche sans état (stateless) permet une meilleure scalabilité de l'application.

## TABLEAU DE BORD

### Centre de contrôle utilisateur

Le tableau de bord constitue le centre névralgique de l'application, offrant une vue synthétique et interactive des données financières de l'utilisateur. Il permet une prise de décision éclairée grâce à la visualisation immédiate des informations importantes.

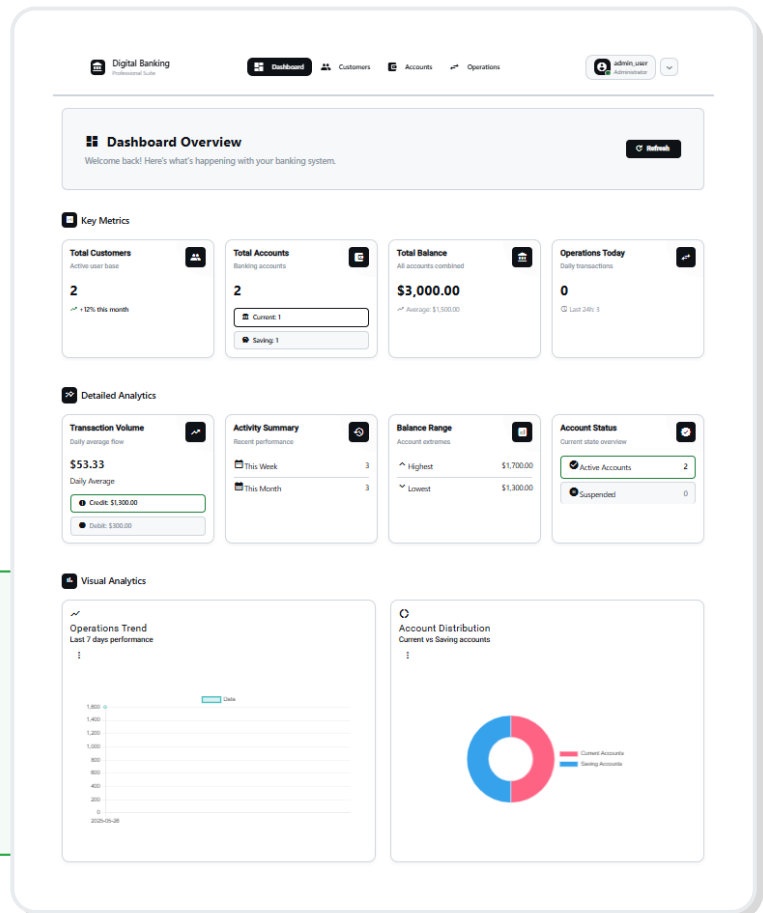
### 4.1 Vue d'Ensemble Interactive

#### Fonctionnalités clés

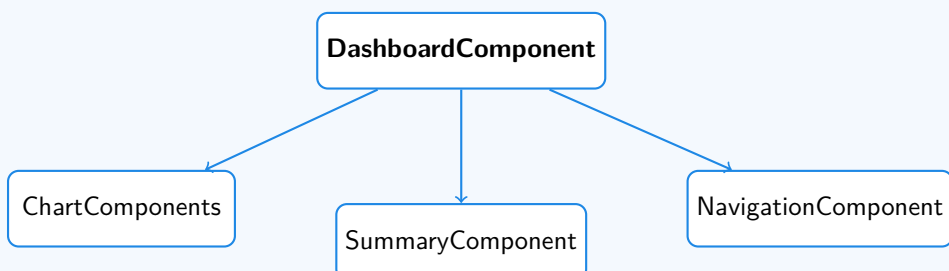
- Distribution des comptes par type
- Tendances des opérations financières
- Indicateurs en temps réel sur l'activité
- Navigation intuitive entre les sections
- Résumé des dernières opérations

#### Avantages

- Visualisation immédiate des données
- Accès rapide aux fonctionnalités
- Vision globale de l'activité
- Alertes et notifications importantes



### Architecture des composants du tableau de bord



**Note technique**

Les graphiques du tableau de bord sont générés dynamiquement à partir des données récupérées via l'API REST. La bibliothèque Chart.js est utilisée pour le rendu des visualisations, offrant une expérience interactive et réactive aux utilisateurs.

## GESTION DES CLIENTS

**Module de gestion clientèle**

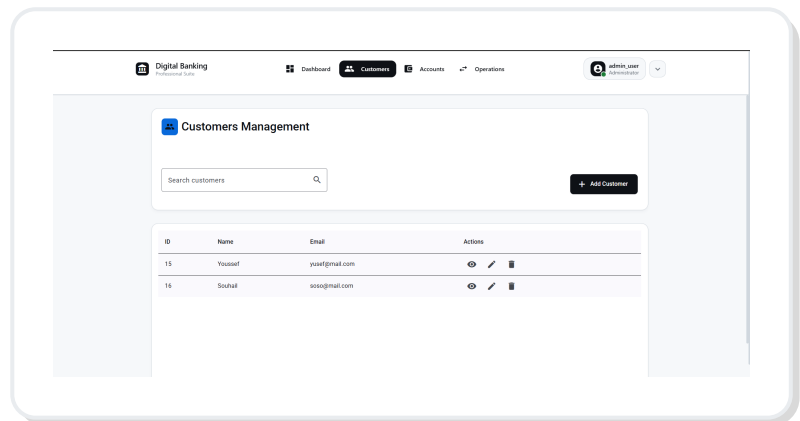
Ce module complet est dédié à l'administration des profils clients avec fonctionnalités CRUD (Créer, Lire, Mettre à jour, Supprimer) intégrées dans une interface intuitive et réactive.

La gestion des clients est un élément central de l'application, permettant aux administrateurs de maintenir une base de données à jour et d'assurer la qualité des informations utilisées pour toutes les opérations bancaires.

### 5.1 Consultation de la Base Clients

**Fonctionnalités de consultation**

- **Recherche** par nom ou identifiant
- **Pagination** pour navigation efficace
- **Tri** par colonnes multiples
- **Actions rapides** sur chaque ligne
- **Vue responsive** adaptable

**GET /api/customers**

**Description:** Récupère la liste paginée des clients avec options de tri et filtrage

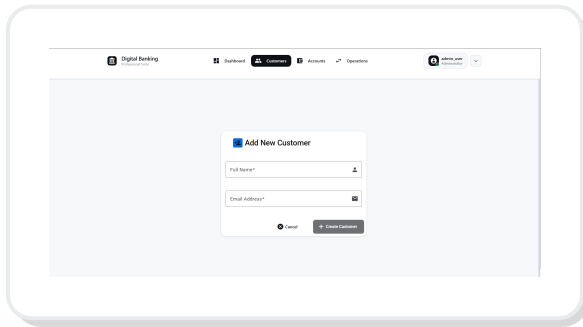
**Réponse:** {"customers": [...], "totalPages": 5, "currentPage": 0, "totalItems": 42}

### 5.2 Création de Nouveaux Profils

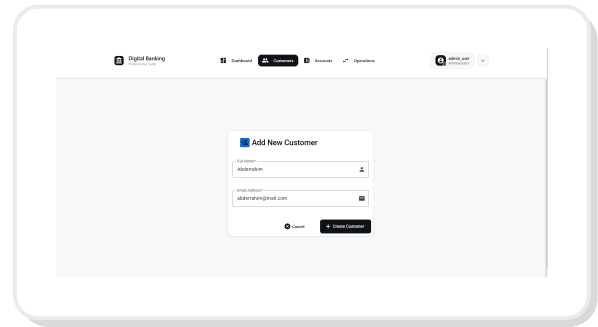
**Processus de création de client**

L'ajout de nouveaux clients s'effectue via un formulaire intuitif qui guide l'utilisateur à travers la saisie de toutes les informations nécessaires. La validation en temps réel garantit l'intégrité des données avant leur soumission.

### 5.2.1 Formulaire de Saisie



### 5.2.2 Formulaire Complété



#### Modèle Angular du Client

```
export class Customer { id: string; name: string; email: string; phoneNumber?: string;
address?: string; createdAt: Date;
// Relations accounts?: BankAccount[]; }
```

#### Note technique

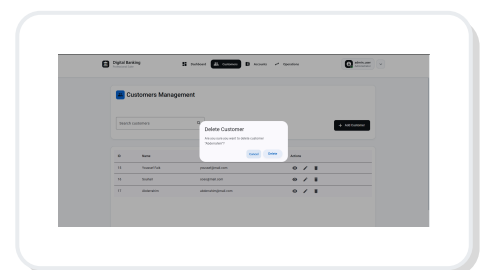
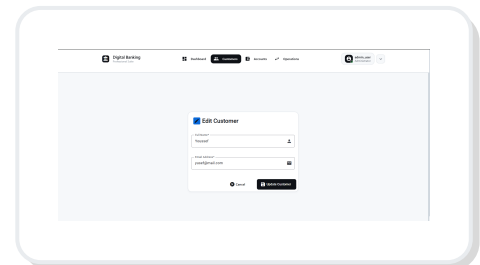
La création de client utilise une approche réactive avec validation côté client et côté serveur. La requête est envoyée via une requête POST au backend, qui valide les données avant de les persister dans la base de données. Une réponse de confirmation avec l'ID généré est retournée au frontend.

## 5.3 Modification et Suppression de Clients

#### Cycle de vie complet des clients

L'application offre un cycle de vie complet pour la gestion des clients, incluant non seulement la création mais aussi la modification des informations existantes et la suppression lorsque nécessaire.

- 1 — Sélection du client à modifier
- 2 — Formulaire pré-rempli avec données existantes
- 3 — Modification des champs nécessaires
- 4 — Validation et enregistrement des changements



### Précautions pour la suppression de clients

- **Confirmation obligatoire** avant toute suppression définitive
- **Vérification des comptes associés** pour éviter les suppressions problématiques
- **Journalisation des suppressions** pour maintenir un audit trail complet
- **Accès restreint** à cette fonctionnalité critique selon les rôles utilisateurs

**DELETE** /api/customers/{id}

**Description:** Supprime un client après vérification des dépendances et contraintes

**Réponse:** {"status": "SUCCESS", "message": "Customer deleted successfully"}

#### Point clé

La gestion complète des clients constitue la fondation de l'application Digital Banking, permettant aux administrateurs de maintenir une base de données clients fiable et à jour.

## DOCUMENTATION API AVEC SWAGGER

**Transparence et Interactivité :** Pour faciliter la compréhension, l'intégration et le test de l'API backend, une documentation interactive est fournie grâce à Swagger (conforme à la spécification OpenAPI). Elle permet aux développeurs de visualiser l'ensemble des endpoints, de comprendre les modèles de données (schémas) et même de tester les requêtes directement depuis une interface web.

### 6.1 Vue Générale de Swagger UI

---

L'interface Swagger UI offre une vue d'ensemble claire et structurée de toutes les ressources de l'API. Les endpoints sont regroupés par contrôleurs (ou tags), facilitant la navigation et la découverte des fonctionnalités disponibles. Chaque endpoint est listé avec sa méthode HTTP (GET, POST, PUT, DELETE) et un résumé de son objectif.

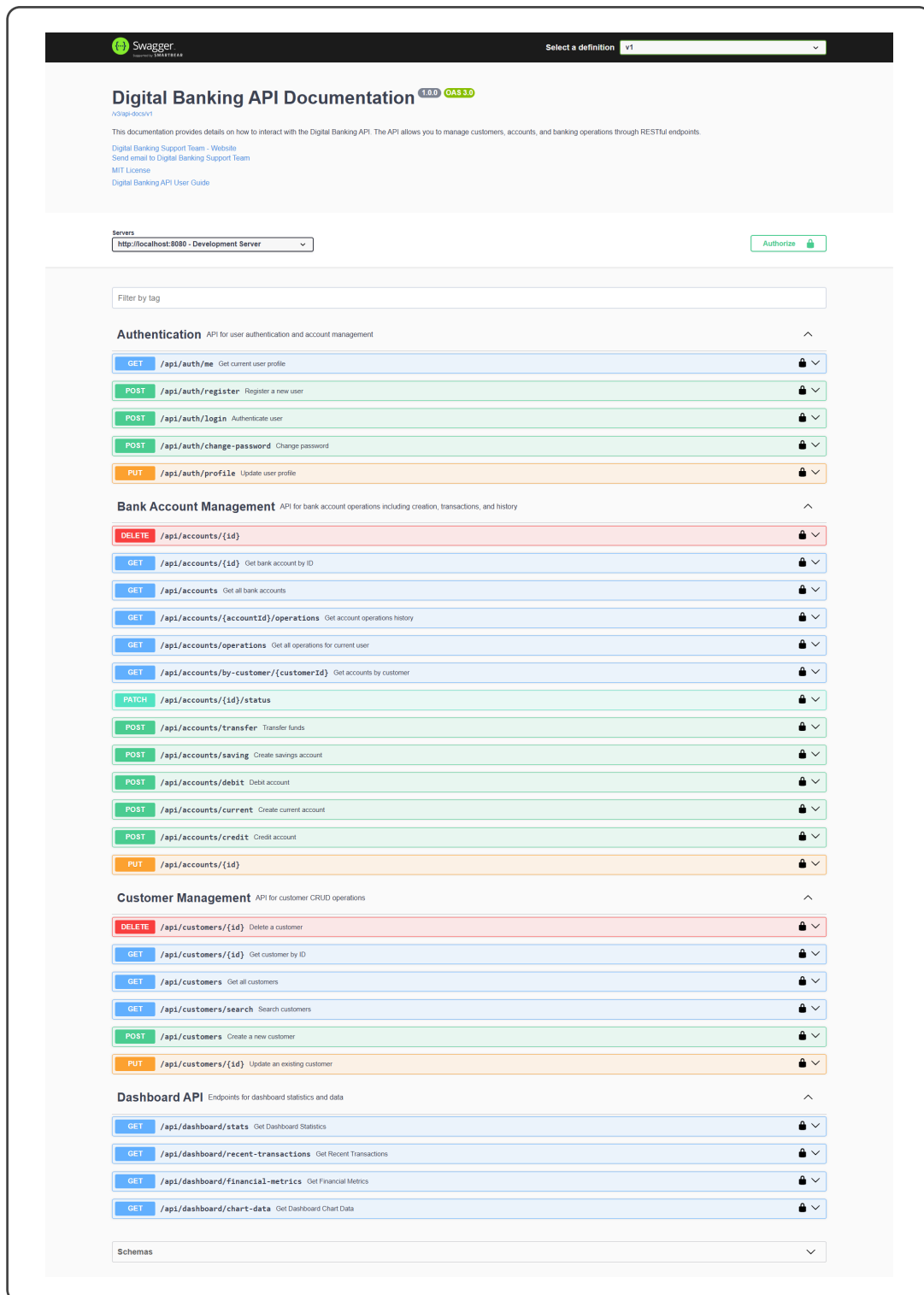


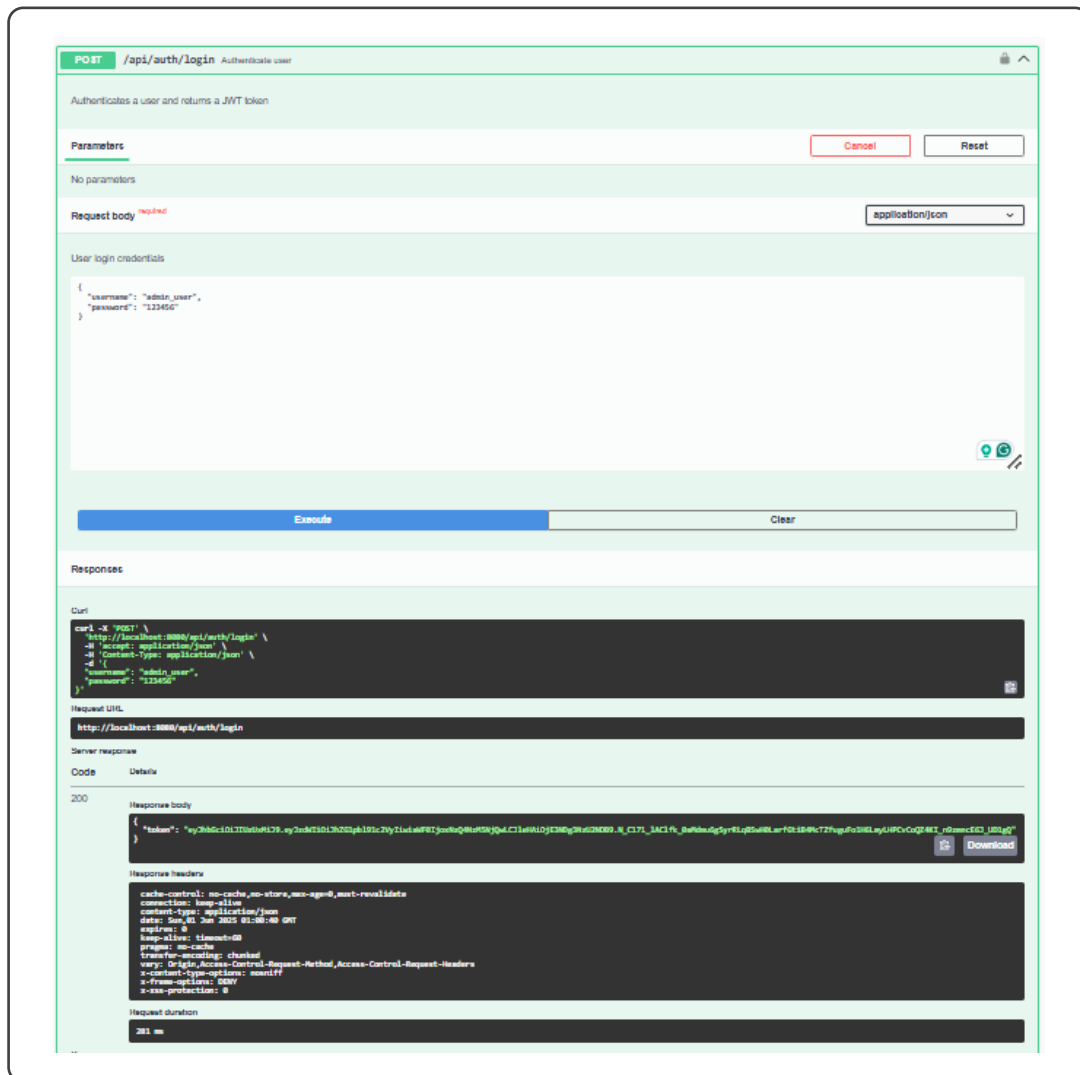
Figure 3 Interface principale de Swagger UI - Liste des contrôleurs et endpoints de l'API

## 6.2 Exploration et Test des Endpoints

En sélectionnant un endpoint spécifique, l'utilisateur peut accéder à une documentation détaillée incluant la description de la fonctionnalité, les paramètres requis et optionnels (path, query, body), les schémas des corps de requête et de réponse, ainsi que les codes de statut HTTP possibles. La fonctionnalité "Try it out" permet d'exécuter l'endpoint directement depuis l'interface. Ci-dessous, quelques exemples illustratifs :

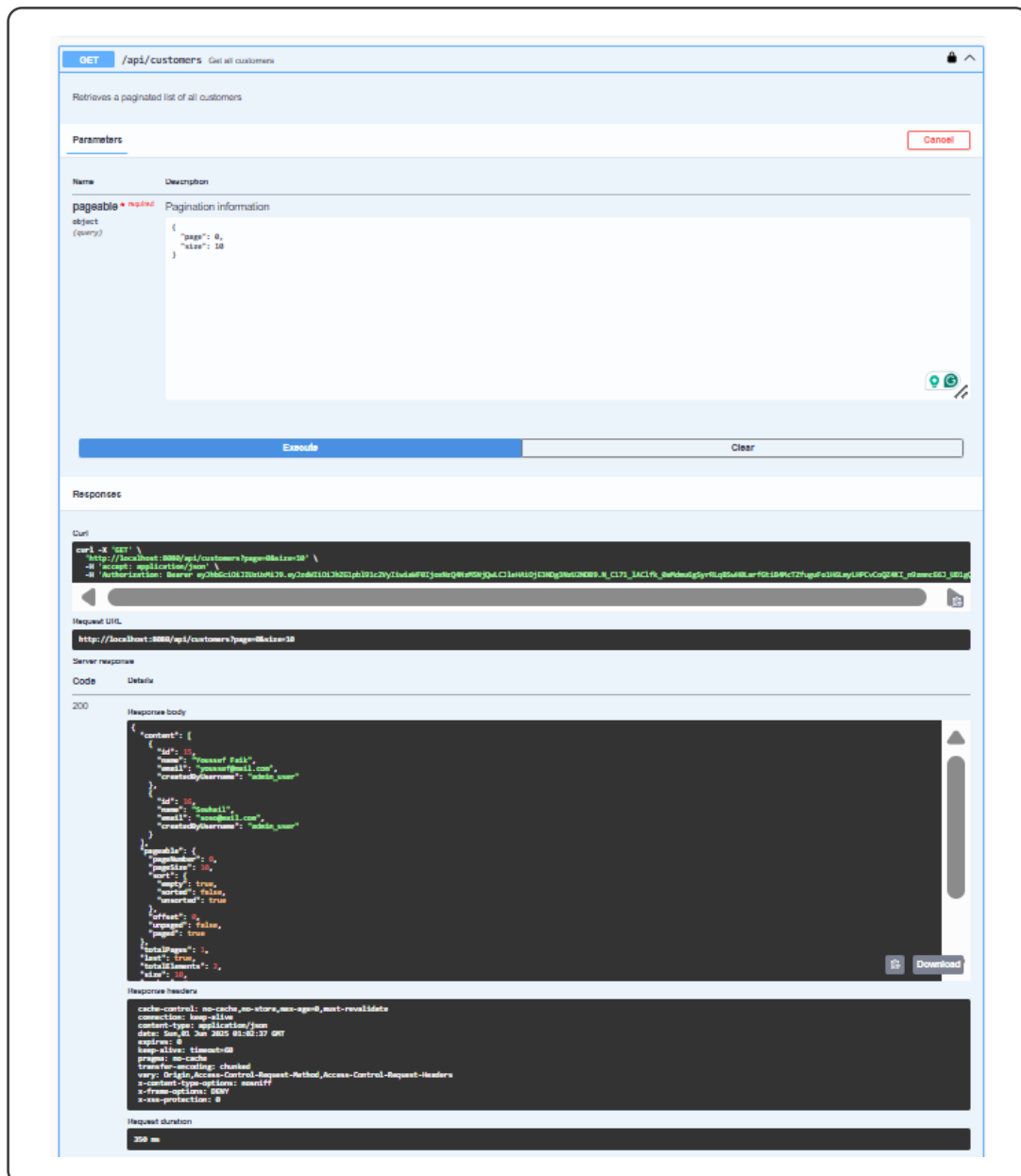


**Exemple 1 : Endpoint d'Authentification** L'authentification est gérée via un endpoint dédié, typiquement une requête POST vers un chemin comme `/api/auth/login`. L'utilisateur fournit ses identifiants (nom d'utilisateur et mot de passe) dans le corps de la requête. En retour, l'API délivre un jeton (par exemple, JWT) qui sera utilisé pour autoriser les requêtes subséquentes.



**Figure 4** Exemple d'endpoint d'authentification (POST) - Paramètres de requête (identifiants) et description de la réponse (jeton d'accès)

**Exemple 2 : Endpoint de Récupération de Données (GET)** Pour récupérer des informations spécifiques, comme les détails d'un client, un endpoint GET utilisant un paramètre de chemin (path parameter) est souvent utilisé, par exemple `/api/customers/{customerId}`. Swagger UI permet de spécifier la valeur de `customerId` pour tester l'appel.



**Figure 5** Exemple d'endpoint de récupération de données (GET) - Utilisation d'un paramètre de chemin et visualisation du schéma de réponse



L'intégration de Swagger améliore significativement la maintenabilité et la facilité d'utilisation de l'API, en fournissant une source de vérité unique et toujours à jour pour sa documentation.

## CONCLUSION



### Synthèse du Projet

#### Une Solution Bancaire Digitale Complète

L'application **Digital Banking** représente un écosystème bancaire numérique complet et moderne, basé sur une architecture robuste et bien structurée, comme détaillé dans ce rapport. Elle démontre l'efficacité des technologies contemporaines ( Angular,  Spring Boot) pour créer une expérience utilisateur fluide et sécurisée. La documentation exhaustive des API via Swagger, illustrée par des exemples concrets d'endpoints, vient renforcer la robustesse et la maintenabilité du système.

### Points forts de l'application

- **Architecture moderne** : Approche microservices avec séparation claire front/back
- **Sécurité renforcée** : Authentification JWT, autorisations basées sur les rôles
- **Interface intuitive** : Expérience utilisateur fluide et responsive
- **Traçabilité** : Historique complet des opérations bancaires
- **API documentée** : Interface Swagger complète facilitant l'intégration
- **Maintenabilité** : Code modulaire et bien structuré

### Perspectives d'évolution

Cette application pourrait être enrichie par :

- L'ajout de fonctionnalités avancées comme les virements programmés
- L'intégration de solutions d'analyse de données pour personnaliser l'expérience
- Le déploiement d'une version mobile native
- L'implémentation de notifications en temps réel

✓ Cette documentation technique illustre la capacité de l'application à répondre aux besoins opérationnels d'une institution bancaire moderne, avec un focus particulier sur la sécurité, l'ergonomie, la traçabilité des opérations et la clarté de son architecture de services. ✓

### Technologies Utilisées

 Spring Boot  Angular  MySQL  JWT  RESTful API  
 Bootstrap