

**DÉPARTEMENT MATHÉMATIQUES ET INFORMATIQUE**

**« Génie du Logiciel et des Systèmes Informatiques Distribués »**

**Architecture JEE et Middlewares**

**Projet JEE Spring Angular**

**Digital Banking**



Achraf TAFFAH  
06 84 13 47 82

**Encadré par :**

- Pr. Mohamed YOUSSEFI

Année Universitaire : 2022 - 2023

# Plan

<b>Introduction.....</b>	<b>3</b>
I. Spécification des besoins.....	4
II. Conception.....	5
III. Test et documentation.....	6
IV. Demonstration.....	9
<b>Conclusion.....</b>	<b>13</b>

# Introduction

Dans ce rapport, je présente mon projet réalisé dans le cadre du module d'Architecture JEE et Middlewares. L'objectif était de développer une application de banque numérique en utilisant Spring Boot et Angular.

Mon application permet la gestion de comptes bancaires pour les clients. Les comptes peuvent subir des opérations de débit ou crédit et sont catégorisés en comptes courants et comptes épargne.

J'ai commencé par créer un projet Spring Boot et développé les entités JPA telles que Customer, BankAccount, Saving Account, CurrentAccount et AccountOperation pour représenter les éléments du système bancaire.

En utilisant les interfaces JPA Repository basées sur Spring Data, j'ai facilité l'accès et la manipulation des données.

J'ai effectué des tests unitaires pour valider la couche DAO de l'application, en m'assurant du bon fonctionnement des entités et interfaces Repository.

J'ai ensuite travaillé sur la couche services, les DTO (Data Transfer Objects) et les mappers pour gérer les opérations métier et la conversion des objets pour les API.

La couche Web a été développée avec les RestControllers pour exposer les fonctionnalités de l'application via des API REST, tandis que le frontend a été réalisé avec Angular, offrant une interface utilisateur conviviale et réactive.

Pour renforcer la sécurité, j'ai intégré Spring Security et JWT pour l'authentification et l'autorisation des utilisateurs.

Ce rapport sera accompagné des répertoires Backend et Frontend contenant les codes sources, ainsi qu'une vidéo de démonstration de 5 minutes illustrant les fonctionnalités principales de l'application.

J'ai respecté les délais du projet et je suis confiant que ce travail démontre mes compétences dans la conception et la réalisation d'une application JEE avec Spring Boot et Angular pour la gestion de comptes bancaires numériques.

## **I. Spécification des besoins**

### **1) Besoins fonctionnel :**

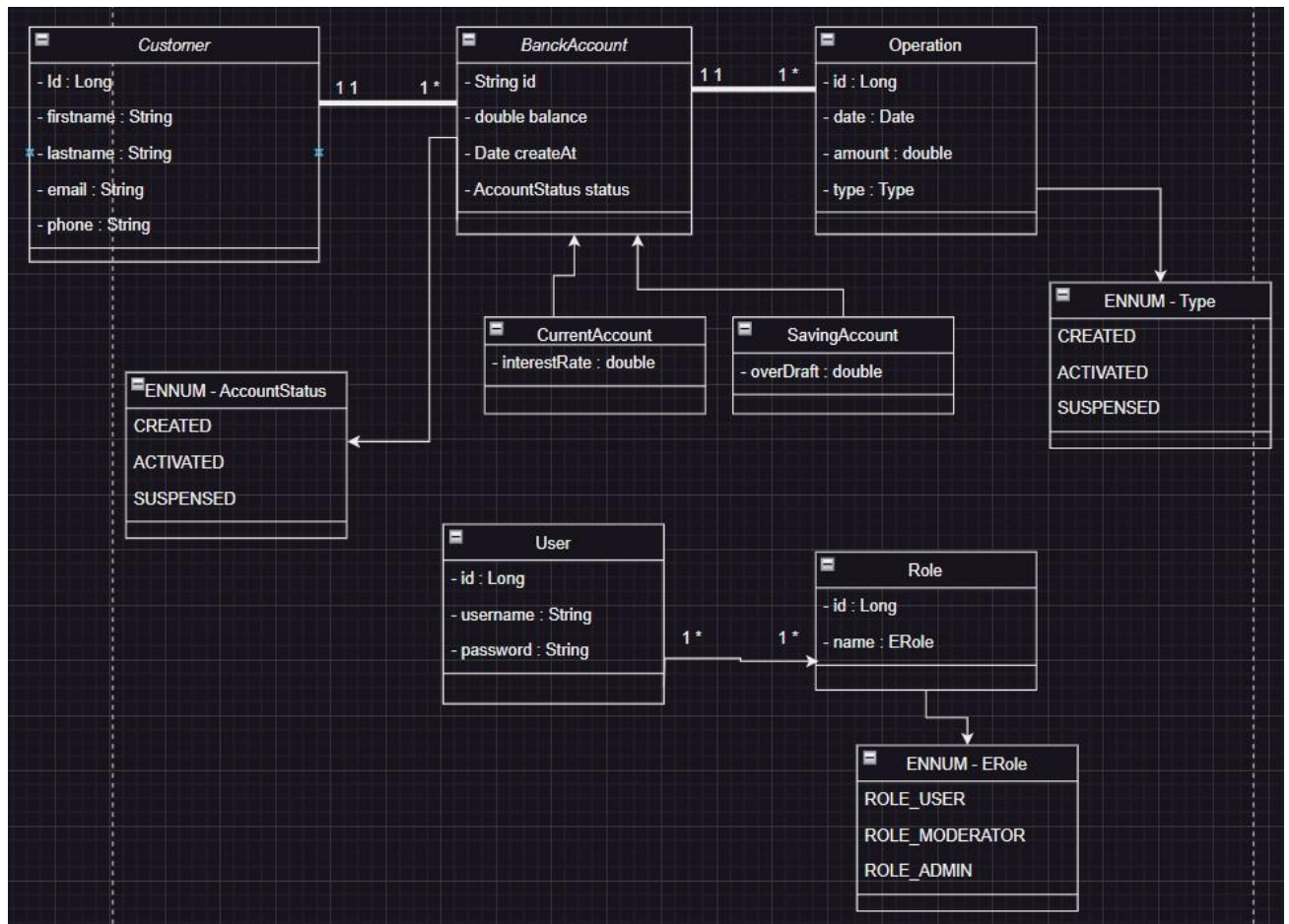
- Gestion des comptes bancaires : L'application doit permettre la création, la consultation, la modification et la suppression des comptes bancaires pour les clients.
- Opérations de débit et de crédit : Les utilisateurs doivent pouvoir effectuer des opérations de débit et de crédit sur les comptes bancaires.
- Catégorisation des comptes : Les comptes bancaires doivent être catégorisés en comptes courants et comptes épargne, avec des fonctionnalités spécifiques pour chaque type.
- Gestion des clients : L'application doit permettre la gestion des clients, incluant la création, la consultation et la modification des informations personnelles.
- Historique des opérations : Les utilisateurs doivent pouvoir consulter l'historique des opérations effectuées sur un compte bancaire, y compris les détails tels que la date, le montant et le type d'opération.

### **2) Besoins non fonctionnel :**

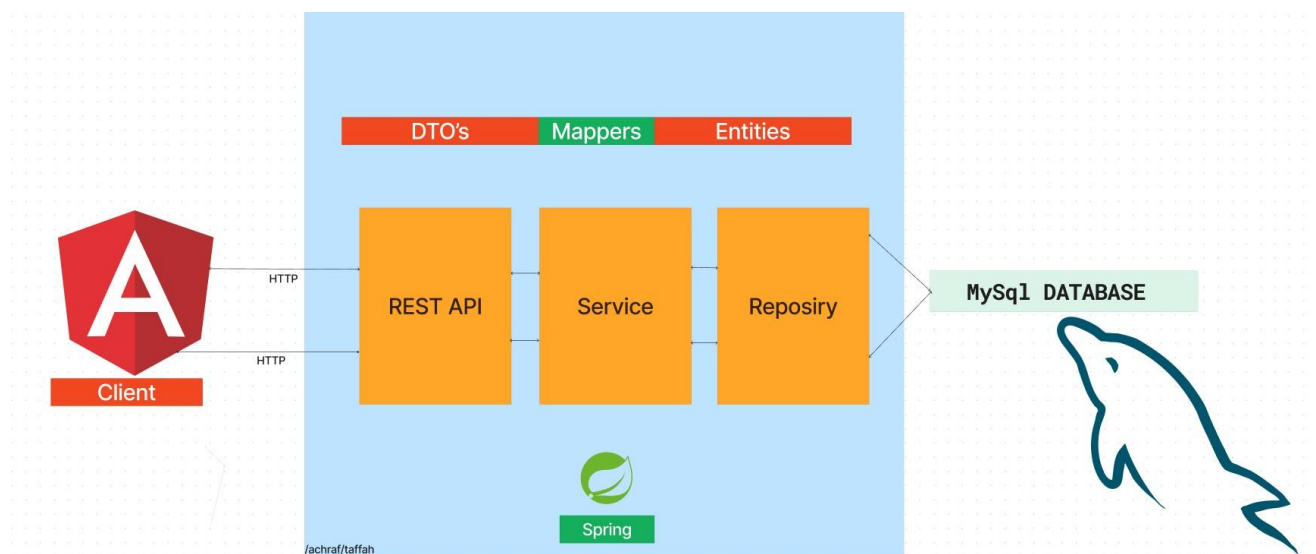
- Sécurité : L'application doit garantir la sécurité des données sensibles, notamment en mettant en place un système d'authentification et de contrôle d'accès.
- Performance : L'application doit être performante, en assurant une réponse rapide aux requêtes des utilisateurs et en gérant efficacement les opérations sur les comptes bancaires.
- Convivialité de l'interface utilisateur : L'interface utilisateur doit être conviviale, intuitive et facile à utiliser, afin de permettre aux utilisateurs de naviguer et d'effectuer des opérations sans difficulté.
- Scalabilité : L'application doit être conçue de manière à pouvoir gérer un nombre croissant de clients et de transactions, en permettant une évolutivité facile.
- Fiabilité : L'application doit être fiable et stable, minimisant les risques de perte de données ou de dysfonctionnements.

## II. Conception

### 1) Diagramme de classe.



### 2) Diagramme d'architecture technique du projet



### III. Test et documentation

Pour la documentation du projet, j'ai utilisé Swagger au niveau du backend et Compodoc au niveau du frontend. Swagger est un outil populaire pour la documentation des API REST. Il permet de générer automatiquement une documentation détaillée des API, en incluant les routes, les paramètres, les réponses attendues, ainsi que des exemples d'utilisation. J'ai intégré Swagger dans mon projet backend, développé avec Spring Boot, afin de fournir une documentation complète et facilement accessible pour les développeurs et les utilisateurs.

Concernant le frontend, j'ai utilisé Compodoc, un outil spécifique à Angular, pour générer la documentation du code source. Compodoc facilite la documentation des composants, des modules, des services et autres éléments du code Angular. Il permet de générer une documentation interactive et bien structurée, qui aide à comprendre l'architecture et le fonctionnement du frontend.

Pour les tests, j'ai utilisé Postman. Postman est un outil de développement d'API qui permet d'envoyer des requêtes HTTP et de vérifier les réponses. J'ai utilisé Postman pour tester mes API au niveau du backend et m'assurer qu'elles fonctionnent correctement. Il m'a permis de vérifier les différentes routes, les paramètres et les réponses des API, ce qui a contribué à assurer la qualité et la fiabilité de mon application.

En résumé, j'ai utilisé Swagger pour la documentation du backend, Compodoc pour la documentation du frontend et Postman pour les tests au niveau du backend. Ces outils ont joué un rôle essentiel dans le développement, la documentation et la vérification de mon projet de banque numérique, en assurant sa cohérence, sa qualité et sa facilité d'utilisation.



## 1) Le test

http://localhost:8085/api/accounts/

GET http://localhost:8085/api/accounts/ Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Type Bearer Token

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Status: 200 OK Time: 187 ms Size: 3154 KB Save as Example

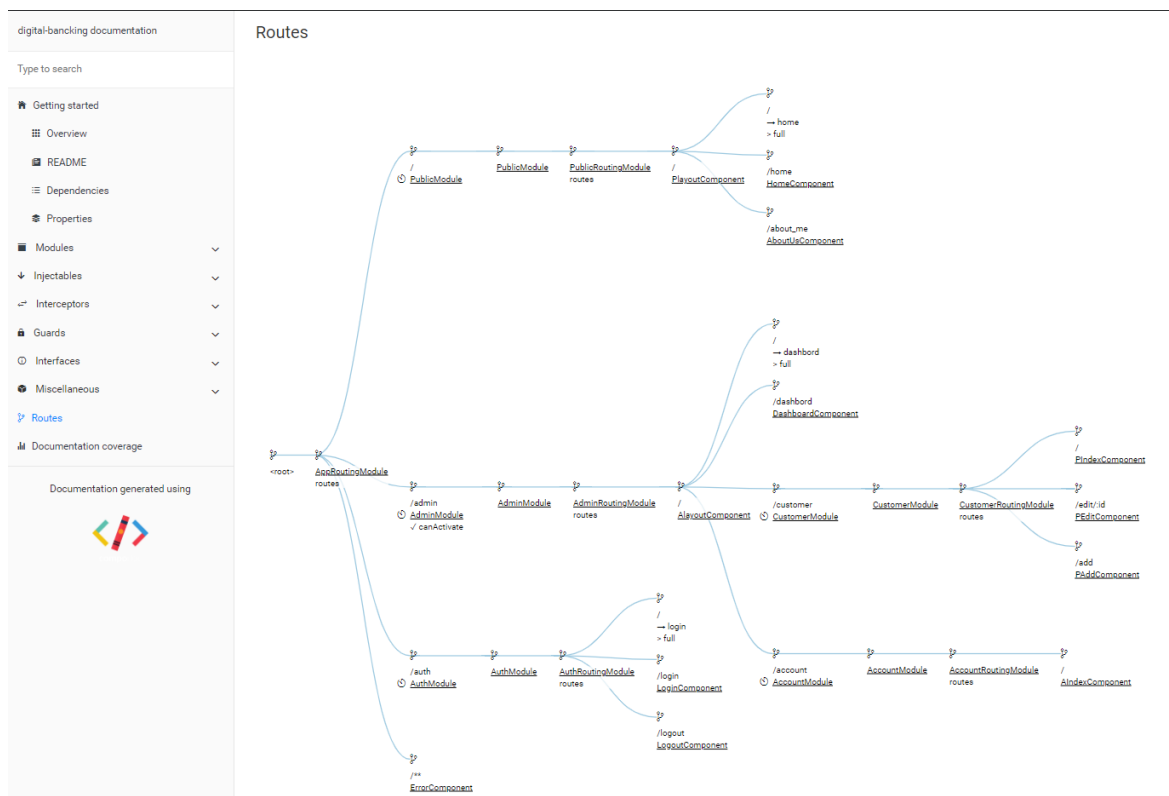
Body Cookies Headers (14) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "type": "CurrentAccount",
3   "id": "03c5fe4a-63b0-4187-b224-00243931e234",
4   "balance": 9624621.77372446,
5   "createdAt": "2023-05-24T01:37:38.000+00:00",
6   "status": null,
7   "customerDTO": {
8     "id": 3,
9     "name": "Mohamed",
10    "email": "Mohamed@gmail.com"
11  },
12  "overDraft": 9000.0
13 },
14 {
15   "type": "CurrentAccount",
16   "id": "05f7bba9-8e03-41fa-9ca4-edf45698156d",
17   "balance": 9897364.418467376,
18   "createdAt": "2023-05-24T01:41:18.000+00:00",
19   "status": null,
20   "customerDTO": {
21     "id": 3,
22     "name": "Mohamed",
23   }
```

## 2) La documentation frontend avec compodoc

### - Le system de routage




### - Les dependances

Type to search

- Getting started
- Overview
- README
- Dependencies
- Properties
- Modules
- Injectables
- Interceptors
- Guards
- Interfaces
- Miscellaneous
- Routes
- Documentation coverage

Documentation generated using



## Dependencies

```
@angular/animations: ^16.0.0
@angular/common: ^16.0.0
@angular/compiler: ^16.0.0
@angular/core: ^16.0.0
@angular/forms: ^16.0.0
@angular/platform-browser: ^16.0.0
@angular/platform-browser-dynamic: ^16.0.0
@angular/router: ^16.0.0
@ng-bootstrap/ng-bootstrap: ^14.1.1
@types/chart.js: ^2.9.37
bootstrap: ^5.2.3
bootstrap-icons: ^1.10.5
chart.js: ^3.9.1
ng2-charts: ^3.1.2
rxjs: ~7.8.0
tslib: ^2.3.0
zone.js: ~0.13.0
```


## - Les intercepteurs

digital-banking documentation

Type to search

- Getting started
- Overview
- README
- Dependencies
- Properties
- Modules
- Injectables
- Interceptors
- Guards
- Interfaces
- Miscellaneous
- Routes
- Documentation coverage

Documentation generated using



## Interceptors / TokenInterceptor

Info

Source

```
1 import { Injectable } from '@angular/core';
2 import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor, HTTP_INTERCEPTORS } from '@angular/common/http';
3 import { Observable, catchError, throwError } from 'rxjs';
4 import { TokenService } from '../services/token.service';
5
6 @Injectable()
7 export class TokenInterceptor implements HttpInterceptor {
8
9   constructor(private tokenService: TokenService) {}
10
11   intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
12     const token = this.tokenService.getToken()
13     if (token !== null) {
14       const cloned = request.clone({
15         setHeaders: {
16           Authorization: `Bearer ${token}`
17         }
18       });
19       return next.handle(cloned).pipe(
20         catchError(error => {
21           console.log(error)
22           if (error.status === 401) {
23             this.tokenService.clearToken()
24           }
25           return throwError('Session expired')
26         })
27       )
28     }
29     return next.handle(request);
30   }
31 }
32
33 export const TokenInterceptorProvider = {
34   provide: HTTP_INTERCEPTORS,
35   useClass: TokenInterceptor,
36   multi: true
37 };
38
```



## IV. Demonstration

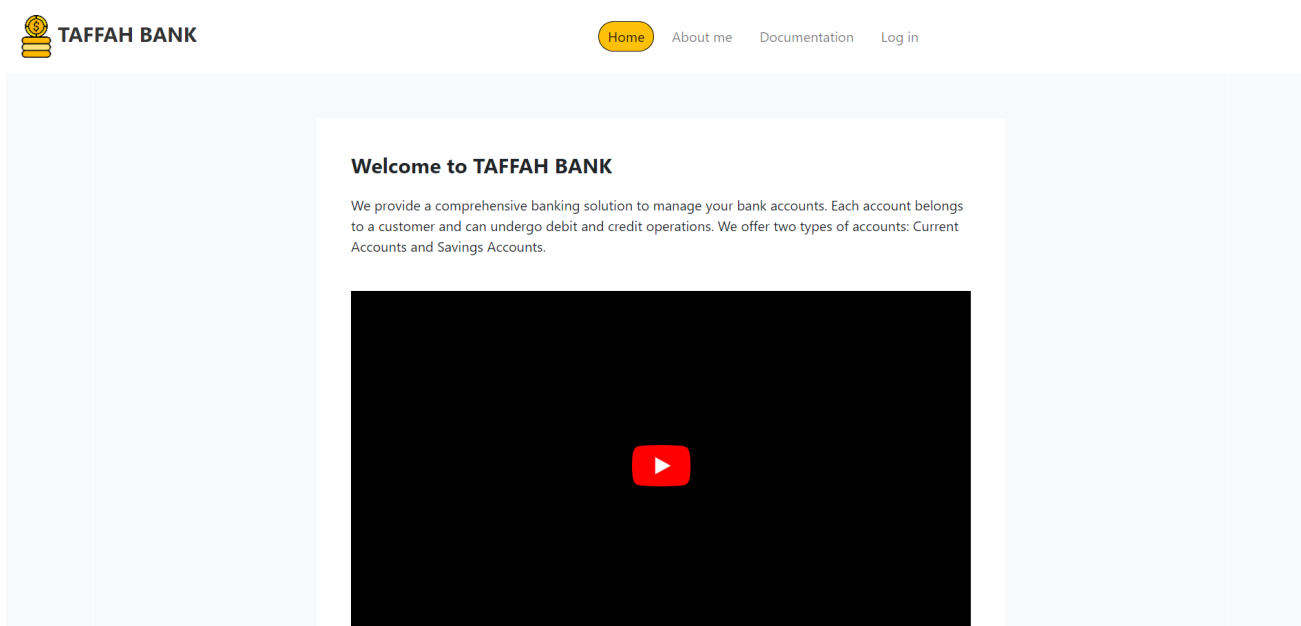
L'application est divisée en deux parties distinctes : une partie publique qui ne nécessite pas d'authentification et une partie admin qui requiert une authentification sécurisée en utilisant un jeton JWT (JSON Web Token).

La partie publique de l'application permet aux utilisateurs d'accéder à certaines fonctionnalités sans avoir à s'authentifier. Cela peut inclure la consultation des informations générales sur les services offerts par la banque, la visualisation des taux d'intérêt, ou encore l'accès à des pages d'aide et de contact. Les utilisateurs peuvent explorer ces fonctionnalités sans avoir besoin de fournir des informations d'identification.

D'autre part, la partie admin de l'application est réservée aux administrateurs ou au personnel autorisé. Pour accéder à cette partie, une authentification est requise. L'authentification est réalisée de manière sécurisée en utilisant un jeton JWT. Lorsqu'un administrateur se connecte avec ses informations d'identification, l'application génère un jeton JWT qui est ensuite utilisé pour authentifier et autoriser les requêtes vers les fonctionnalités administratives. Ce mécanisme de sécurité garantit que seuls les utilisateurs authentifiés et autorisés peuvent accéder et effectuer des opérations dans la partie admin de l'application.

En divisant l'application en deux parties avec des exigences d'authentification différentes, nous assurons une expérience utilisateur conviviale pour les utilisateurs publics, tout en garantissant la sécurité et la confidentialité des fonctionnalités administratives.

### 1) Interface public






I am Achraf TAFFAH a second-year engineering student at ENSET Mohammedia, specializing in software engineering and distributed computer systems.

digital-banking documentation

Type to search

- Getting started
  - Overview
  - README
  - Dependencies
  - Properties
- Modules
  - Injectables
  - Interceptors
  - Guards
  - Interfaces
  - Miscellaneous
  - Routes
  - Documentation coverage

Documentation generated using



## Examen

This project was generated with [Angular CLI](#) version 16.0.2.

### Development server

Run `ng serve` for a dev server. Navigate to `http://localhost:4200/`. The application will automatically reload if you change any of the source files.

### Code scaffolding

Run `ng generate component component-name` to generate a new component. You can also use `ng generate directive|pipe|service|class|guard|interface|enum|module`.

### Build

Run `ng build` to build the project. The build artifacts will be stored in the `dist/` directory.

### Running unit tests

Run `ng test` to execute the unit tests via [Karma](#).

### Running end-to-end tests

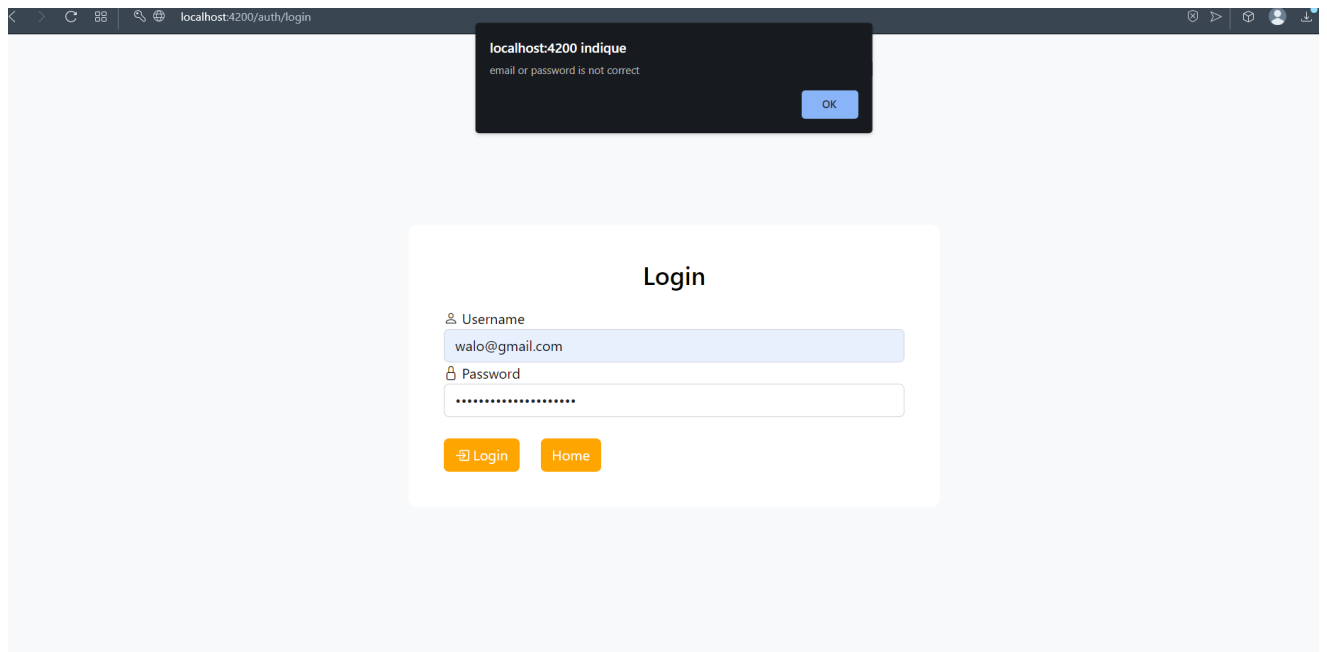
Run `ng e2e` to execute the end-to-end tests via a platform of your choice. To use this command, you need to first add a package that implements end-to-end testing capabilities.

### Further help

To get more help on the Angular CLI use `ng help` or go check out the [Angular CLI Overview and Command Reference](#) page.

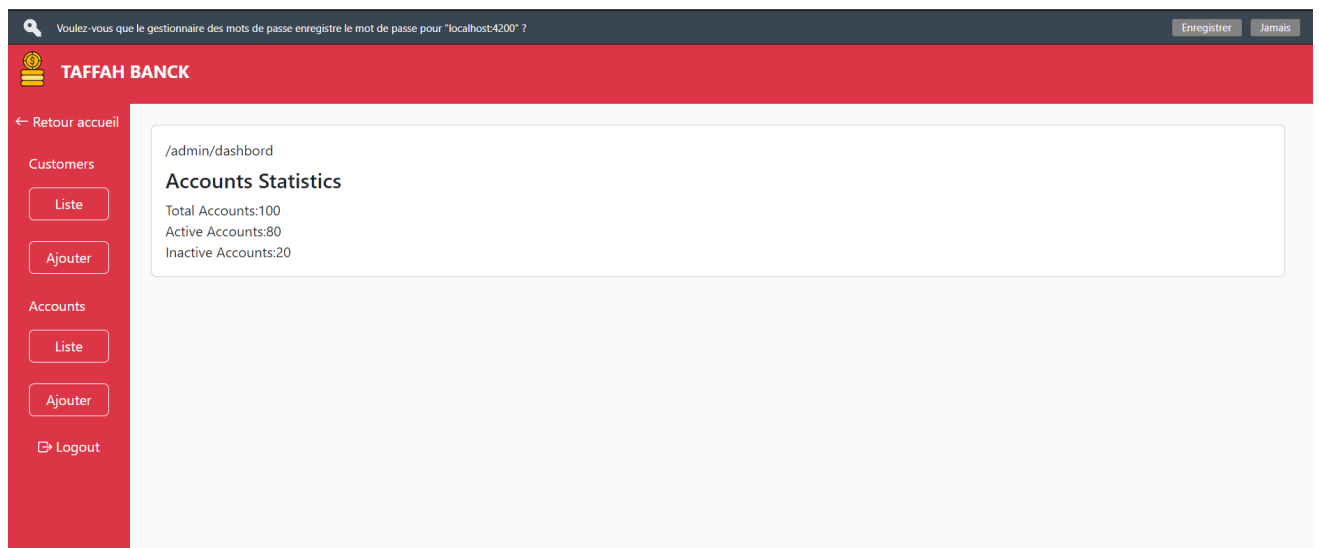
## 2) Interface d'authentification

- L'application est sécuriser on utilisant le token JWT.

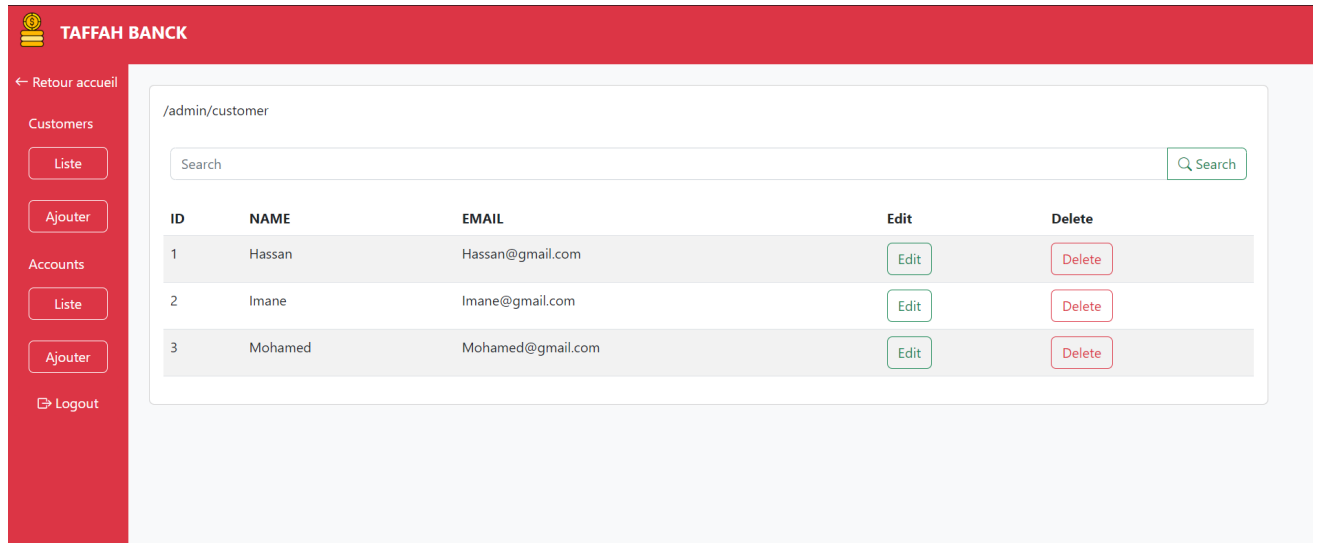


### 3) Interface Administrateur

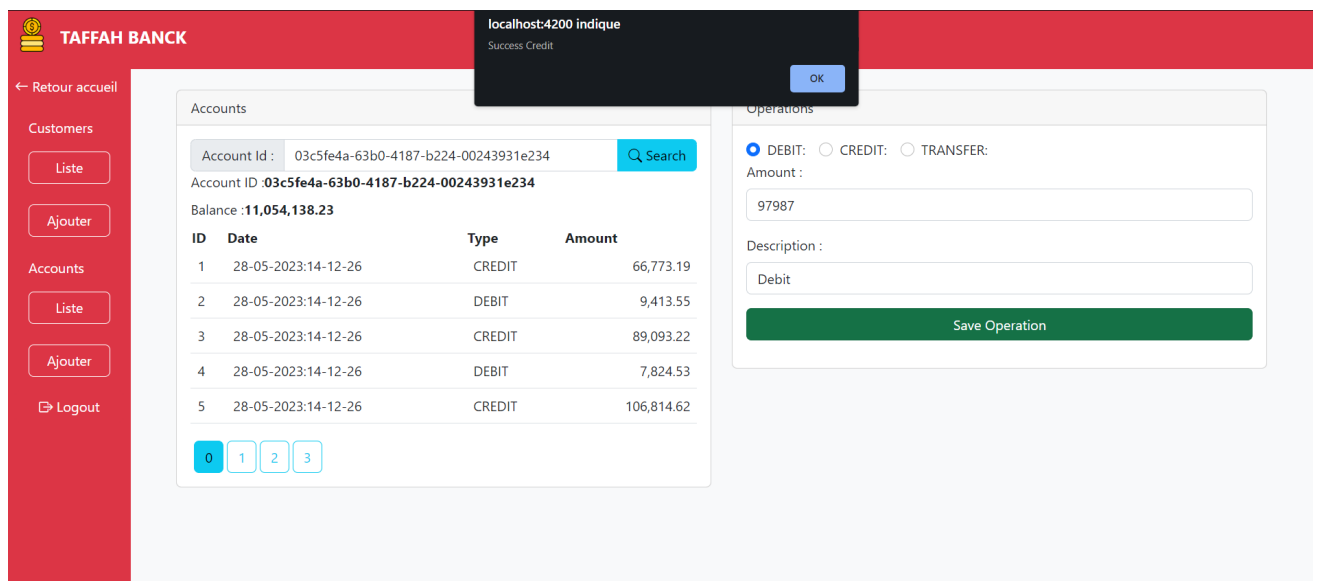
- la page dashbord



- la page customer



- la page account



## Conclusion

En conclusion, ce projet de digital banking basé sur l'architecture JEE et les middlewares a été réalisé avec succès en utilisant Spring Boot pour le backend et Angular pour le frontend. Nous avons pu mettre en œuvre les fonctionnalités de base pour la gestion des comptes bancaires, en mettant l'accent sur la séparation entre la partie publique et la partie administration.

Cependant, pour rendre notre application encore plus performante, évolutive et capable de gérer une charge croissante, nous pourrions envisager d'adopter une approche de microservices. Les microservices sont une architecture logicielle dans laquelle une application est décomposée en services autonomes, qui peuvent être développés, déployés et mis à l'échelle indépendamment les uns des autres.

En utilisant une approche de microservices, nous pourrions découpler les différentes fonctionnalités de notre application, telles que la gestion des clients, la gestion des comptes, et les opérations bancaires, en des services distincts. Chaque service pourrait être développé et déployé indépendamment, ce qui faciliterait la maintenance, la mise à l'échelle et la résilience de l'application.

De plus, en adoptant une approche de microservices, nous pourrions utiliser des technologies telles que Kubernetes pour orchestrer et gérer les conteneurs dans lesquels les microservices s'exécutent. Cela nous permettrait de bénéficier d'une mise à l'échelle automatique, d'une résilience améliorée et d'une gestion simplifiée des déploiements.

En conclusion, l'adoption d'une approche de microservices pourrait contribuer à rendre notre application de digital banking plus flexible, évolutive et capable de faire face à des charges de travail élevées. Cela nous permettrait de résoudre les problèmes potentiels de montée en charge et d'optimiser les performances de notre application tout en facilitant la maintenance et le développement continu.