<h1 align="center"> Digital Banking Application </h1>

<br>

This web application allows you to manage bank accounts. It is developed using Spring Boot for the backend and Angular for the frontend.

[![View Backend](https://img.shields.io/badge/View\_Backend-GitHub-blue.svg)](https://github.com/oussama-tahri/Digital-Banking/tree/master/Digital%20Banking%20Backend) [![View Frontend](https://img.shields.io/badge/View\_Frontend-GitHub-green.svg)](https://github.com/oussama-tahri/Digital-Banking/tree/master/Digital-Banking-Web)

**## Description**

The Digital Banking Application provides functionality to manage different types of bank accounts. There are two types of accounts: Current Account and Saving Account. Each account has a unique ID and specific attributes based on its type. Current Accounts have an Overdraft attribute, while Saving Accounts have an Interest Rate attribute.

In addition to the account details, each bank account maintains a list of operations performed on it. An operation includes an ID, operation date, amount, description, and type (debit, credit, or transfer). The account history keeps track of the account balance after each operation.

Customers are registered in the system and have the following attributes: ID, name, and email address. Customers can have multiple bank accounts associated with their profile.

**## Backend**

The backend of the Digital Banking Application is built with Spring Boot. It provides RESTful APIs to perform various operations on bank accounts and customers. The backend architecture follows a layered approach, consisting of the following components:

**### Security**

The backend implements security using JWT (JSON Web Token) to ensure secure authentication and authorization. When a user logs in successfully, the backend generates a JWT token containing the user's roles (user or admin) as claims. This token is then returned to the frontend and stored securely in the browser.

**### Controllers**

The controllers handle incoming requests and delegate the processing to the appropriate services. They are responsible for mapping API endpoints and validating request parameters.

**### Services**

The services contain the business logic of the application. They handle the core functionality, such as creating bank accounts, performing operations, retrieving account details, and managing customer information.

**### Repositories**

The repositories interact with the database to perform CRUD (Create, Read, Update, Delete) operations. They provide an abstraction layer between the application and the underlying data storage.

**### Entities**

The entities represent the domain objects of the application. They are mapped to database tables and define the structure of the data.

**### Data Transfer Objects (DTOs)**

DTOs are used to transfer data between the backend and frontend. They encapsulate the necessary information and provide a structured representation of the data.

**## Frontend**

The frontend of the Digital Banking Application is developed using Angular. It provides a user-friendly interface for interacting with the application.

**### Security**

In the frontend, the JWT token received from the backend is securely stored in the browser, either in local storage or session storage. Angular route guards are implemented to control access to different routes based on user roles. The route guards check for the presence of a valid JWT token and the required role before allowing access to specific routes. An HTTP interceptor is set up to automatically attach the JWT token to outgoing API requests, ensuring that only authenticated and authorized users can access protected backend routes.

**### Components**

Components are responsible for rendering the user interface and handling user interactions. They encapsulate specific functionalities and can be reused across different pages.

**### Services**

Services in the frontend handle communication with the backend APIs. They make HTTP requests to fetch data and update the user interface accordingly.

**### Models**

Models represent the data structures used in the frontend. They mirror the backend DTOs and provide a consistent data format for the application.

**## Getting Started**

To run the Digital Banking Application locally, follow the instructions below:

1. Clone the repository from [GitHub](https://github.com/oussama-tahri/Digital-Banking.git).

2. Configure the backend by updating the necessary configuration files, such as database settings and API endpoints.

3. Build and run the backend application using your preferred IDE or by running the appropriate Maven commands.

4. Configure the frontend by updating the API endpoint URLs to match your backend setup.

5. Install the required dependencies for the frontend using `npm install`.

6. Build and run the frontend application using `ng serve`.

7. Access the application in your browser at `http://localhost:4200`.

**## Conclusion**

The Digital Banking Application provides a comprehensive solution for managing bank accounts and performing banking operations. With its Spring Boot backend and Angular frontend, it offers a robust and user-friendly interface for both customers and bank administrators. The application supports various operations such as creating customer profiles, opening different types of bank accounts, performing debit and credit transactions, transferring funds between accounts, and viewing transaction history.

By leveraging the power of Spring Boot and Angular, the application ensures scalability, security, and a seamless user experience. The modular architecture allows for easy extension and customization, making it suitable for future enhancements and integration with other systems.

To get started with the Digital Banking Application, follow the instructions provided in the "Getting Started" section above. Feel free to explore the backend and frontend repositories to understand the implementation details and contribute to the project.

For any questions or assistance, please refer to the respective repository's documentation and issue tracker. We appreciate your interest and contribution to the Digital Banking Application.

Thank you for choosing our application. Happy banking!

Ce rapport présente la partie backend d'une application web de gestion des comptes bancaires appelée "Digital Banking". L'application est développée en utilisant le framework Spring Boot, qui offre une structure robuste et facilite le développement d'applications Java.

L'objectif principal de l'application est de permettre la gestion des comptes bancaires pour les clients. Chaque compte bancaire appartient à un client et peut subir plusieurs opérations de type "Débit" ou "Crédit". Il existe deux types de comptes : les comptes courants et les comptes épargne.

**## Architecture Backend**

La partie backend de l'application Digital Banking suit une architecture en couches, ce qui permet une séparation claire des responsabilités et facilite la maintenance et l'évolutivité de l'application. Les principales couches de l'application backend sont les suivantes :

1. Couche DAO : Cette couche gère l'accès aux données et comprend les interfaces JPA Repository basées sur Spring Data. Ces interfaces fournissent les opérations de base pour interagir avec les entités persistantes.

2. Entités JPA : Les entités JPA représentent les objets métier de l'application et sont mappées aux tables de la base de données. Les entités de l'application Digital Banking sont les suivantes :

   - Customer : Représente un client de la banque.

   - BankAccount : Classe abstraite représentant un compte bancaire. Les sous-classes de BankAccount sont CurrentAccount et SavingAccount.

   - CurrentAccount : Représente un compte courant avec une autorisation de découvert.

   - SavingAccount : Représente un compte épargne avec un taux d'intérêt.

   - AccountOperation : Représente une opération effectuée sur un compte bancaire.

3. Couche service : Cette couche contient la logique métier de l'application et offre des services pour effectuer des opérations sur les comptes bancaires. Elle utilise les interfaces DAO pour accéder aux données et les mappers pour convertir les entités en DTO (Data Transfer Object) et vice versa.

4. DTOs : Les DTOs sont des objets utilisés pour transférer des données entre les différentes couches de l'application. Ils permettent de définir les informations à envoyer ou à recevoir lors des appels API.

5. RestController : Les RestControllers sont des composants qui exposent les services de l'application via des API REST. Ils reçoivent les requêtes HTTP, appellent les services appropriés et renvoient les réponses aux clients.

6. Exceptions métier : Ces exceptions sont utilisées pour gérer les erreurs spécifiques à l'application. Dans l'application Digital Banking, nous avons les exceptions suivantes :

   - CustomerNotFoundException : Lancée lorsque le client n'est pas trouvé.

   - BankAccountNotFoundException : Lancée lorsque le compte bancaire n'est pas trouvé.

   - BalanceNotSufficientException : Lancée lorsque le solde du compte n'est pas suffisant pour effectuer une opération.

7. Sécurité JWT avec Spring Security : L'application utilise Spring Security avec l'authentification basée sur JWT (JSON Web Token). Deux utilisateurs sont configurés en mémoire (user et admin) avec des mots de passe encodés de type BCrypt. Les utilisateurs authentifiés reçoivent un token JWT qui doit être inclus dans les en-têtes des requêtes pour accéder aux API sécurisées. Le serveur vérifie la validité du token avant de permettre l'accès aux ressources protégées.

**# - les entités et les interfaces JPA**

**## Détails techniques**

Voici quelques détails techniques importants concernant l'implémentation de l'application Digital Banking :

**### @Entity JPA Entities**

Les entités JPA de l'application sont annotées avec `@Entity` pour les identifier comme entités persistantes. La classe abstraite `BankAccount` utilise l'annotation `@Inheritance(strategy = InheritanceType.JOINED)` pour spécifier une stratégie d'héritage de type "JOINED". Cela signifie que les sous-classes `CurrentAccount` et `SavingAccount` auront leurs propres tables distinctes, mais seront liées à la table de la classe `BankAccount` par une clé étrangère.

**### @Repository**

Les interfaces DAO de l'application utilisent l'annotation `@Repository` pour indiquer à Spring qu'il s'agit de composants de persistance qui gèrent l'accès aux données. Ces interfaces étendent les interfaces JPA Repository de Spring Data, telles que `JpaRepository` ou `CrudRepository`, et fournissent des méthodes pour effectuer des opérations de base sur les entités.

**# - Les couches (dtos, mappers, services, web)**

**### @Service**

Les classes de service de l'application utilisent l'annotation `@Service` pour indiquer qu'elles contiennent la logique métier de l'application. Ces classes utilisent les interfaces DAO pour accéder aux données et effectuer des opérations sur les entités. Elles peuvent également utiliser des mappers pour convertir les entités en DTOs et vice versa.

**### @RestController**

Les RestControllers de l'application utilisent l'annotation `@RestController` pour indiquer qu'ils sont des composants qui exposent les services de l'application via des API REST. Les méthodes de ces classes sont annotées avec des annotations telles que `@GetMapping`, `@PostMapping`, `@PutMapping`, etc., pour spécifier les points de terminaison des API et les opérations HTTP correspondantes.

**### @ExceptionHandler**

Les RestControllers utilisent également l'annotation `@ExceptionHandler` pour gérer les exceptions spécifiques à l'application. Ces méthodes sont annotées avec des types d'exception spécifiques, tels que `@ExceptionHandler(CustomerNotFoundException.class)`, et renvoient les réponses d'erreur appropriées aux clients.

**## 1- La configuration**

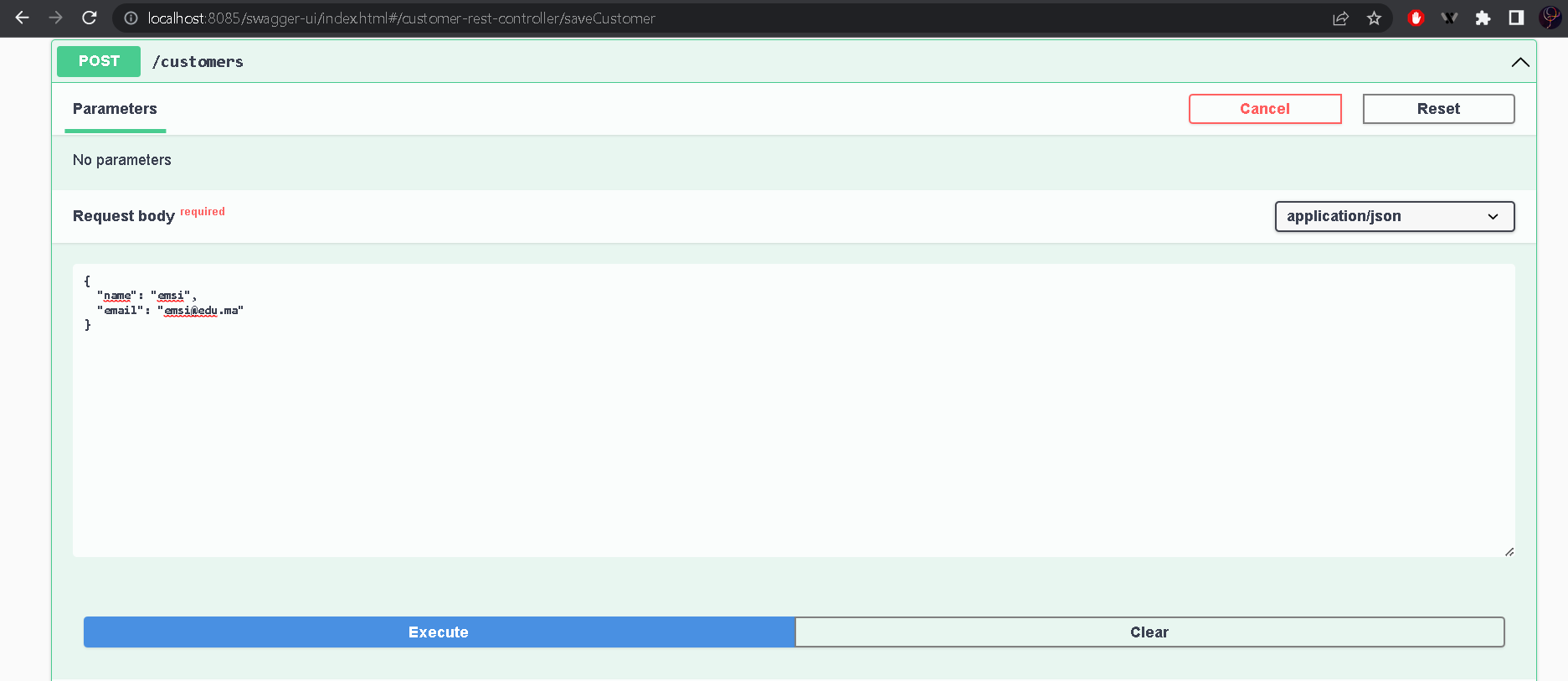
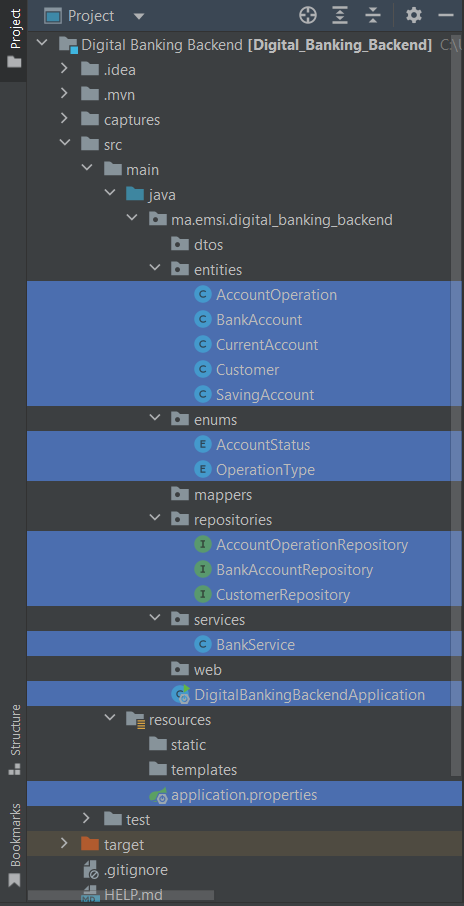
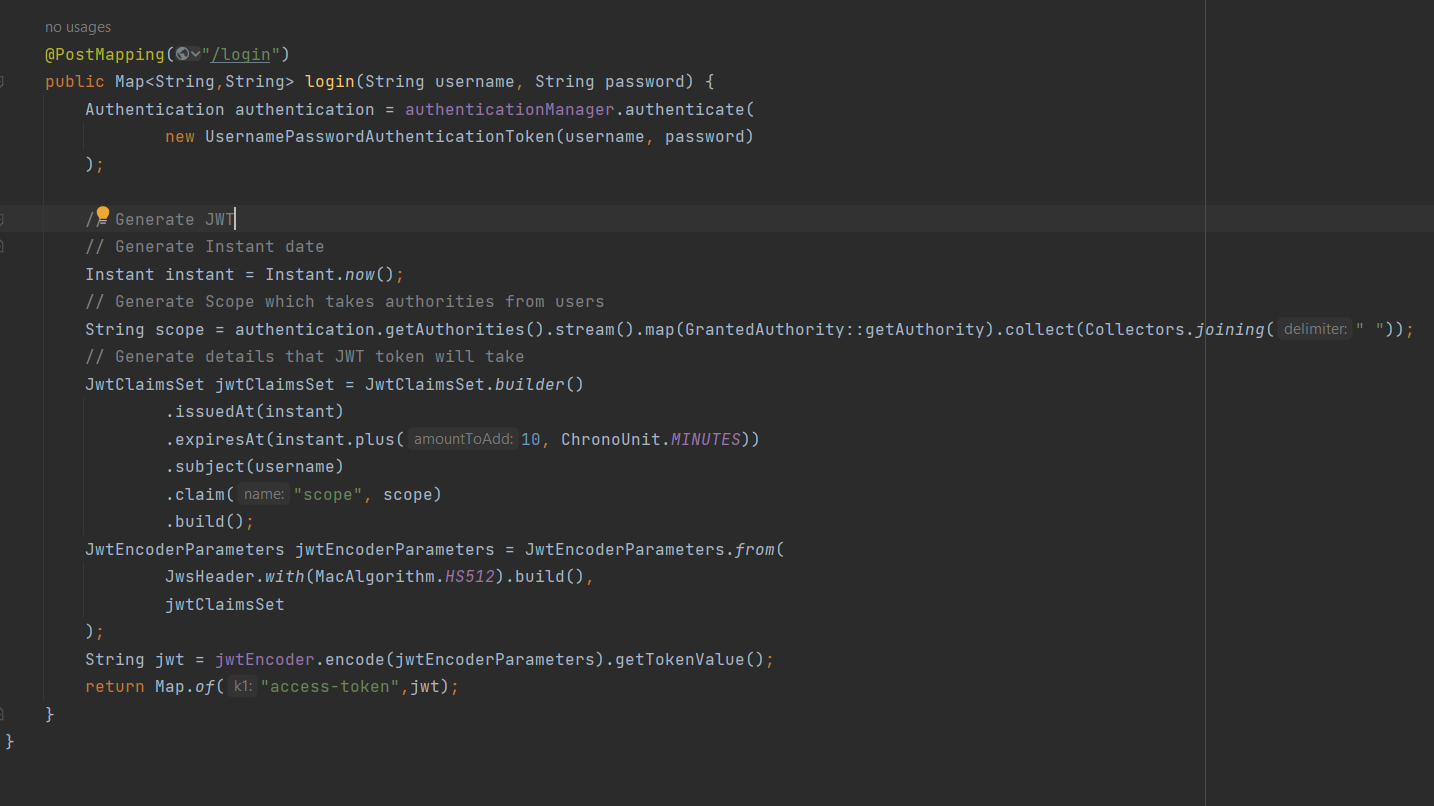
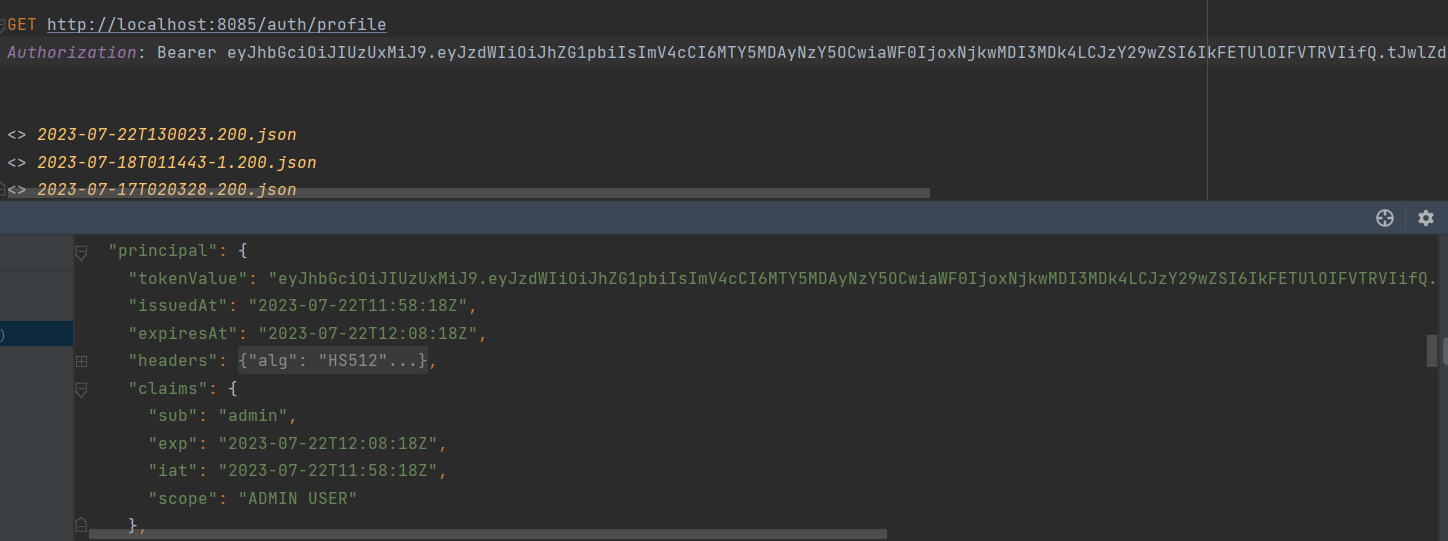
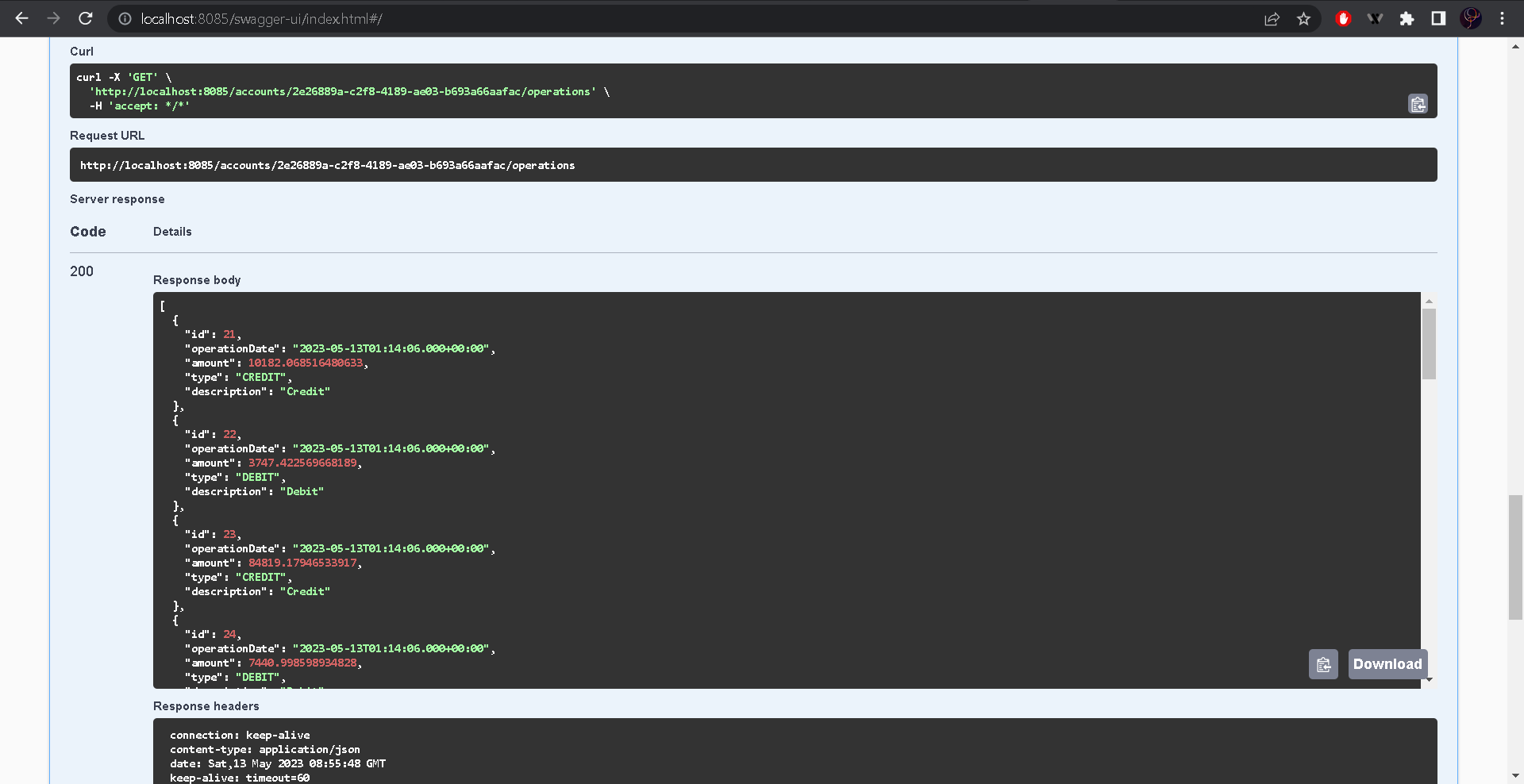
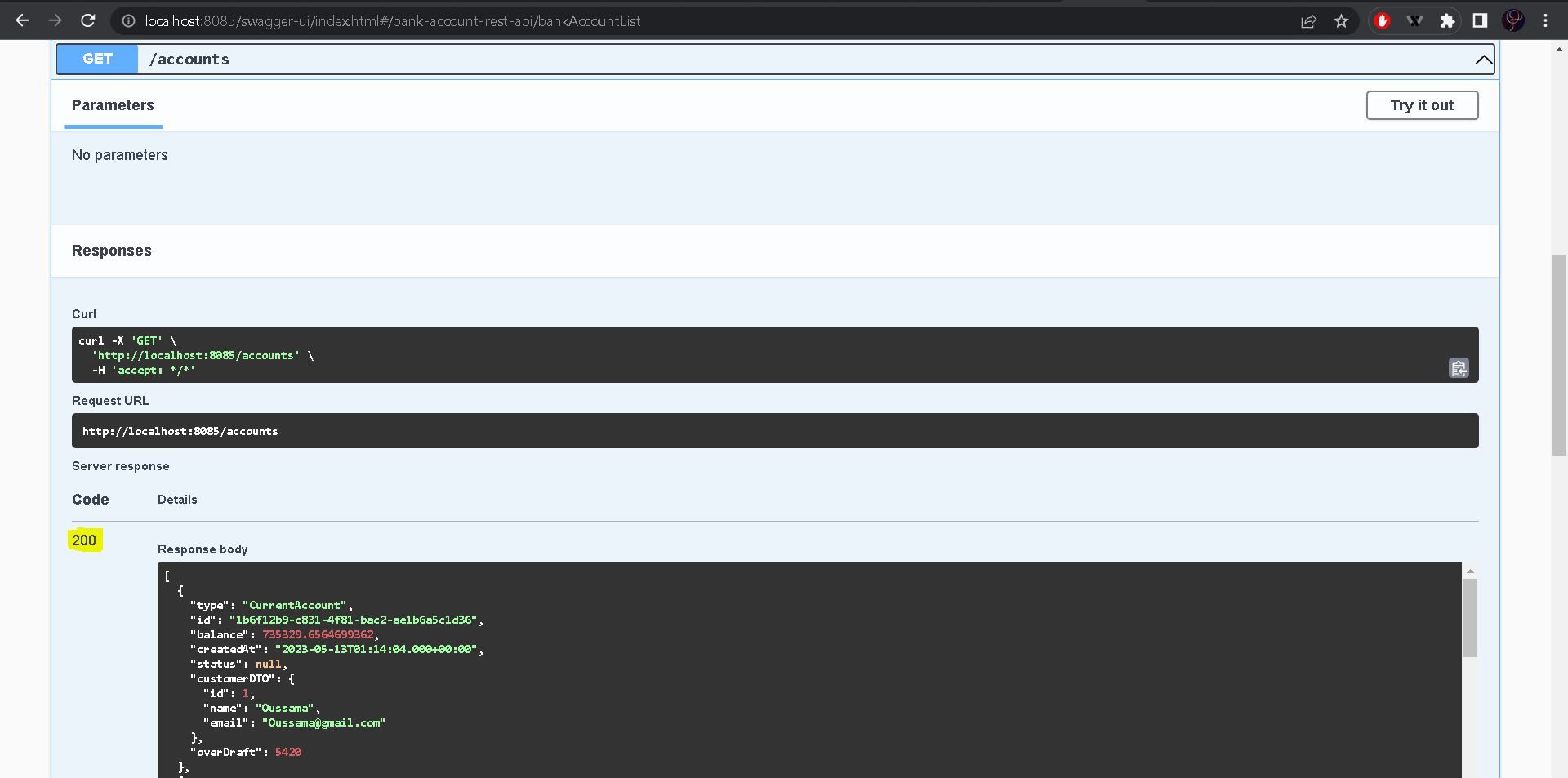
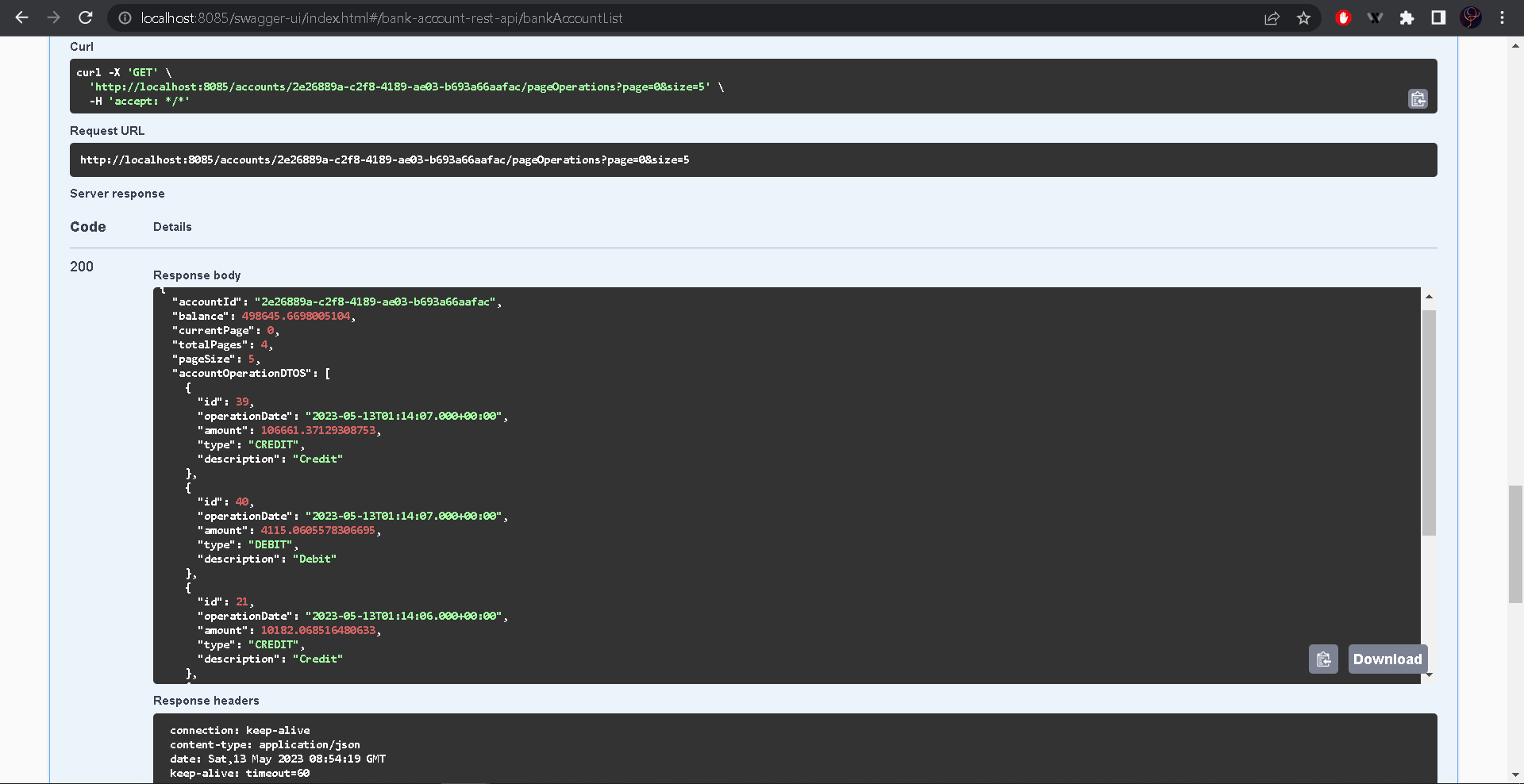
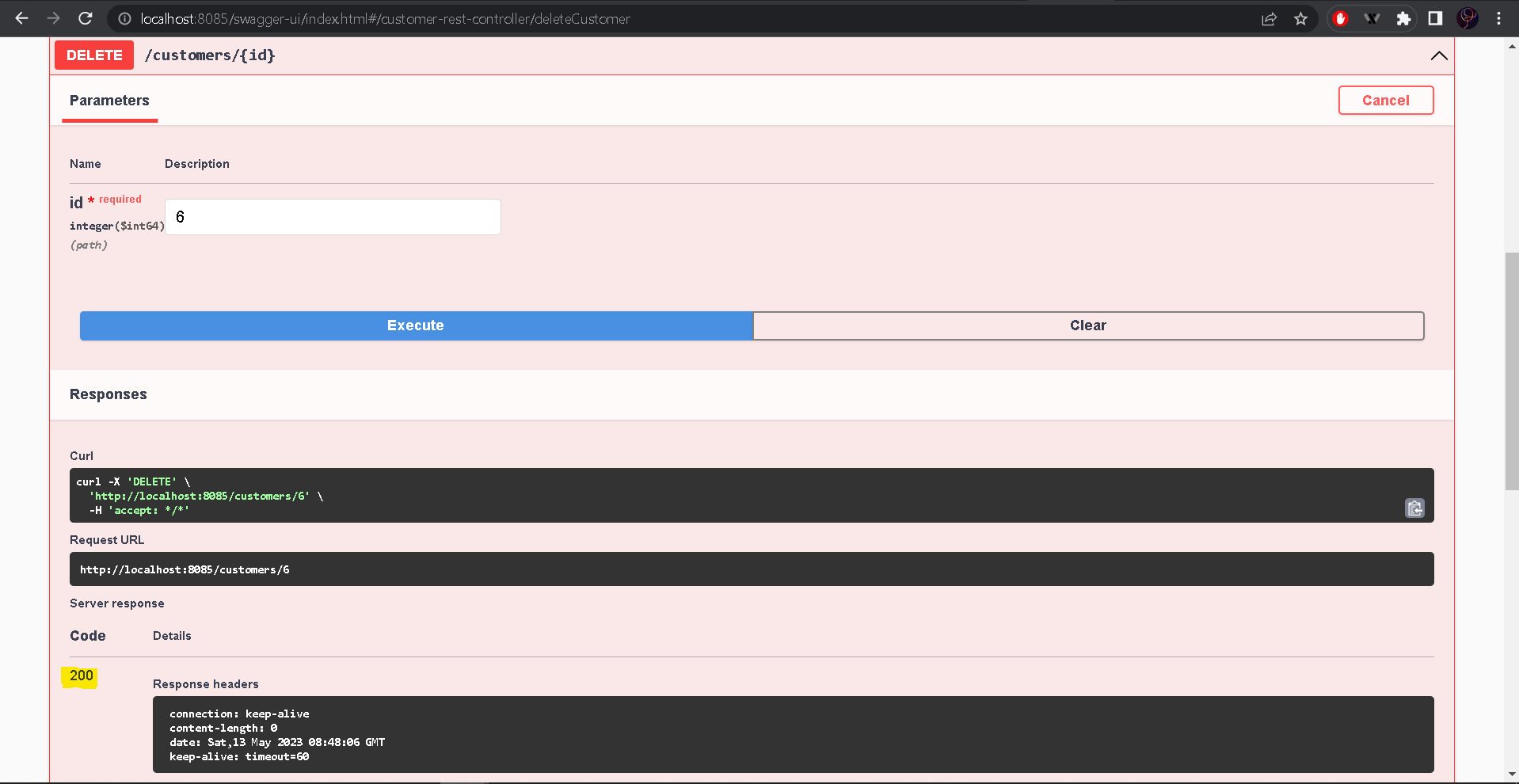
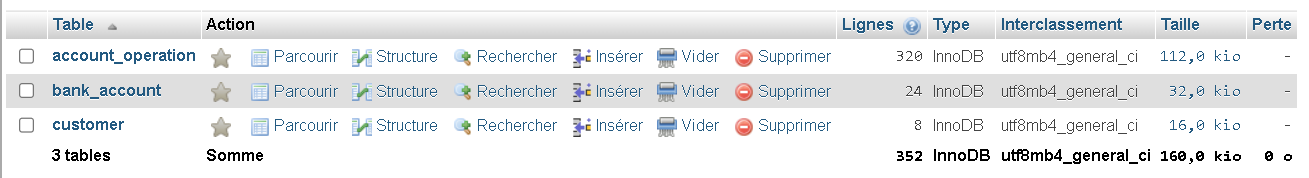
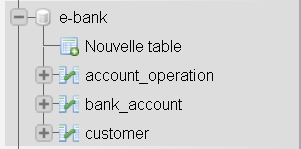
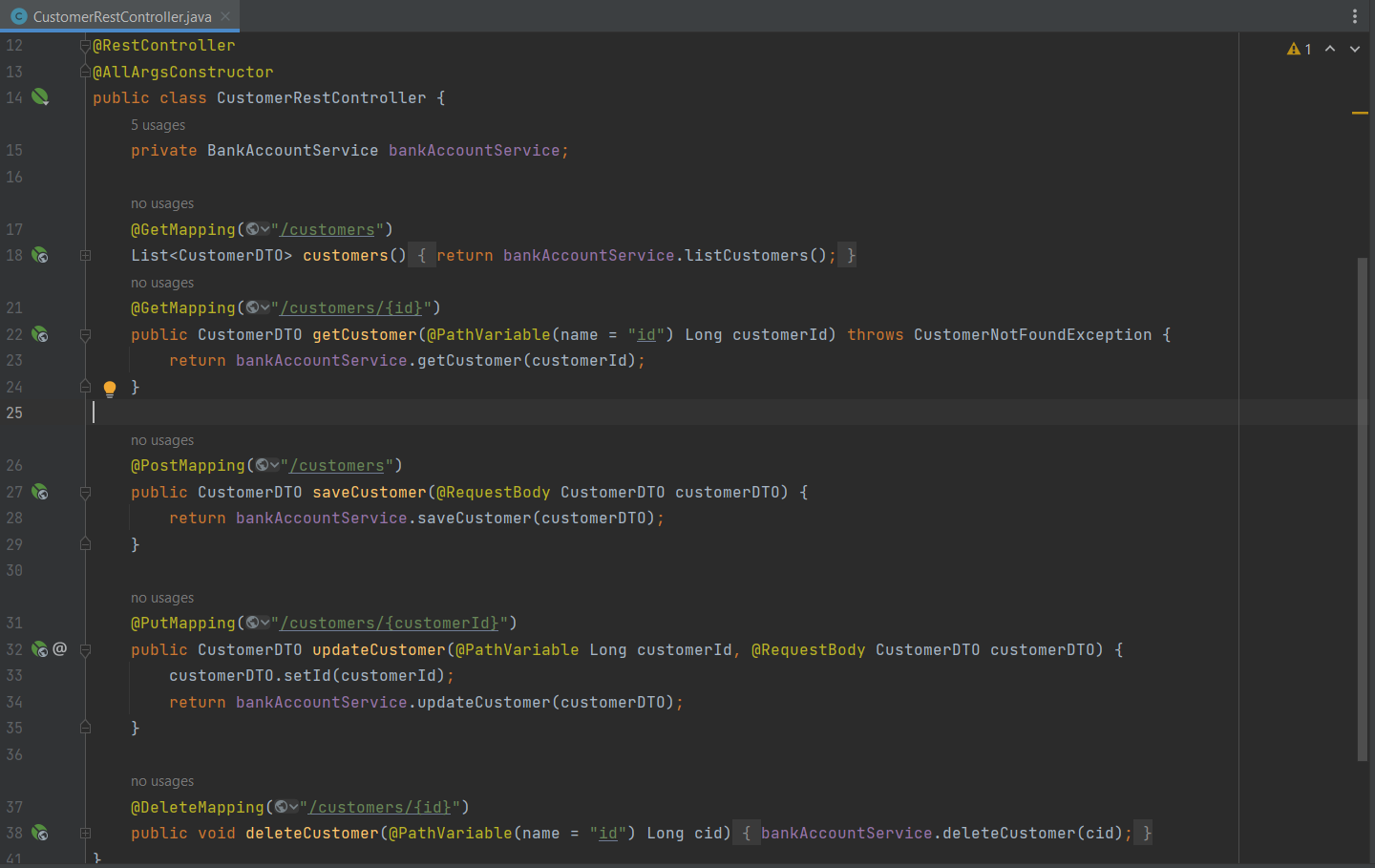
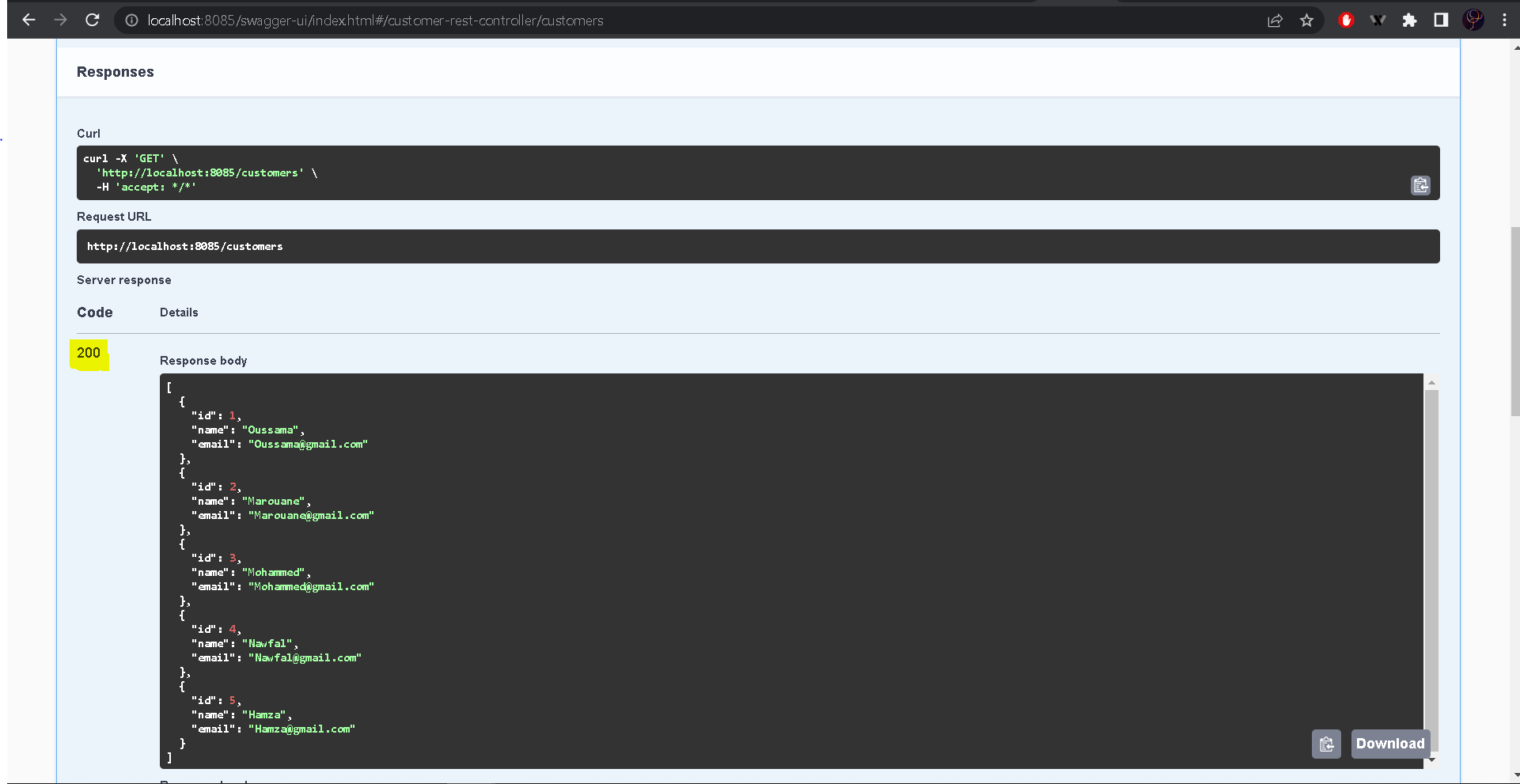
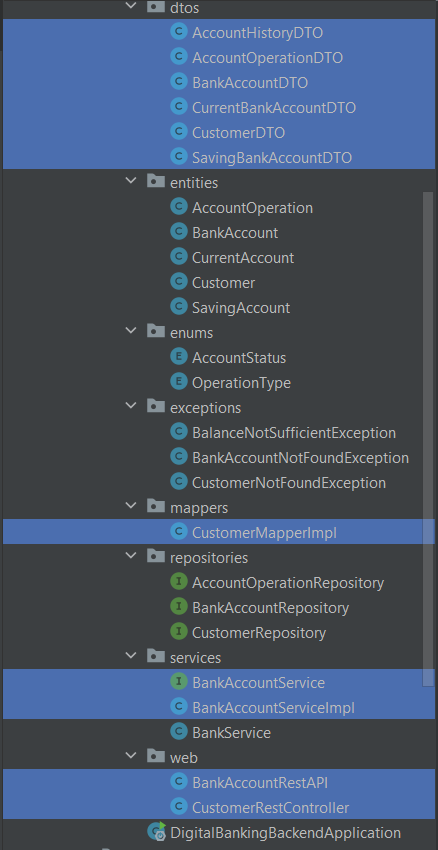
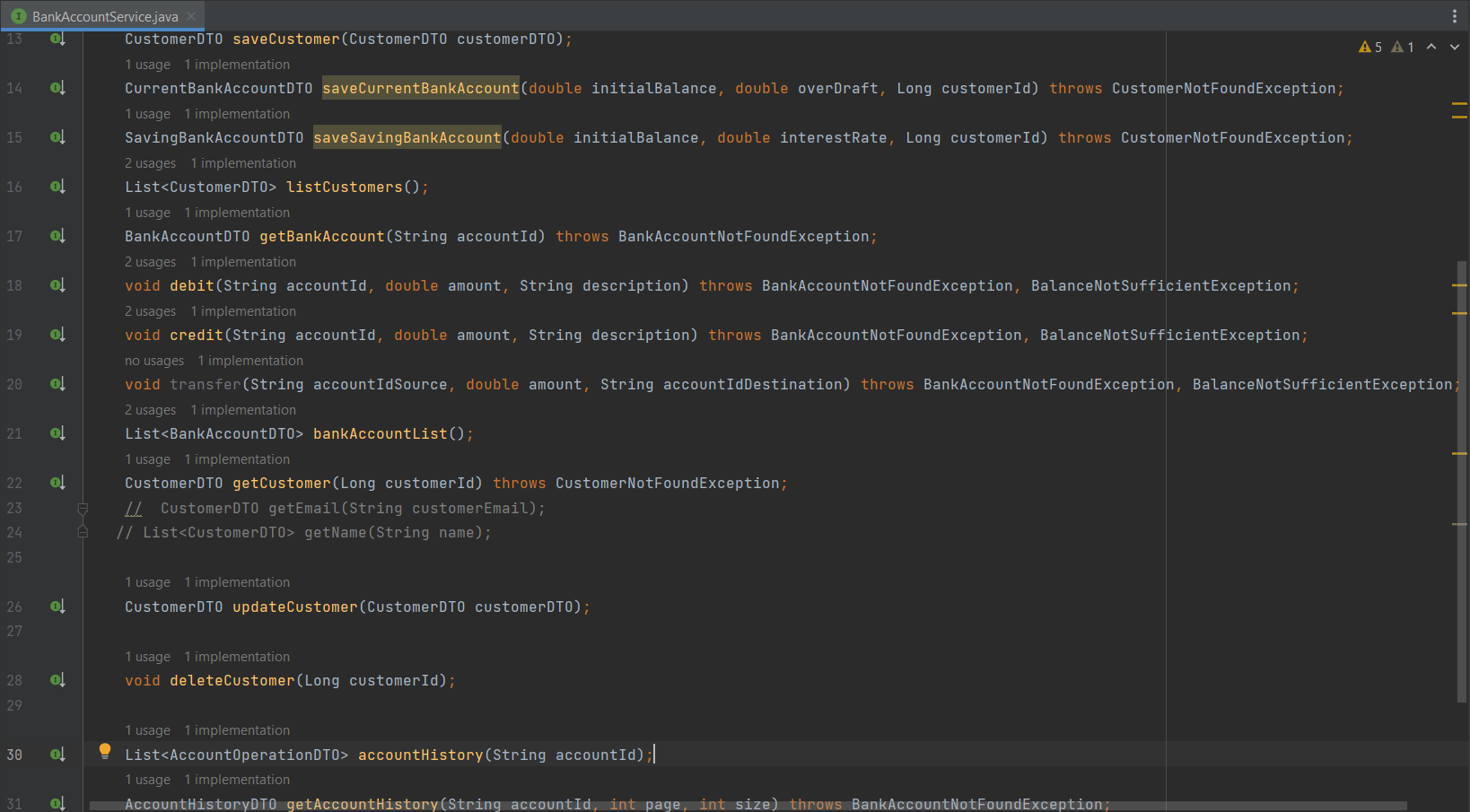
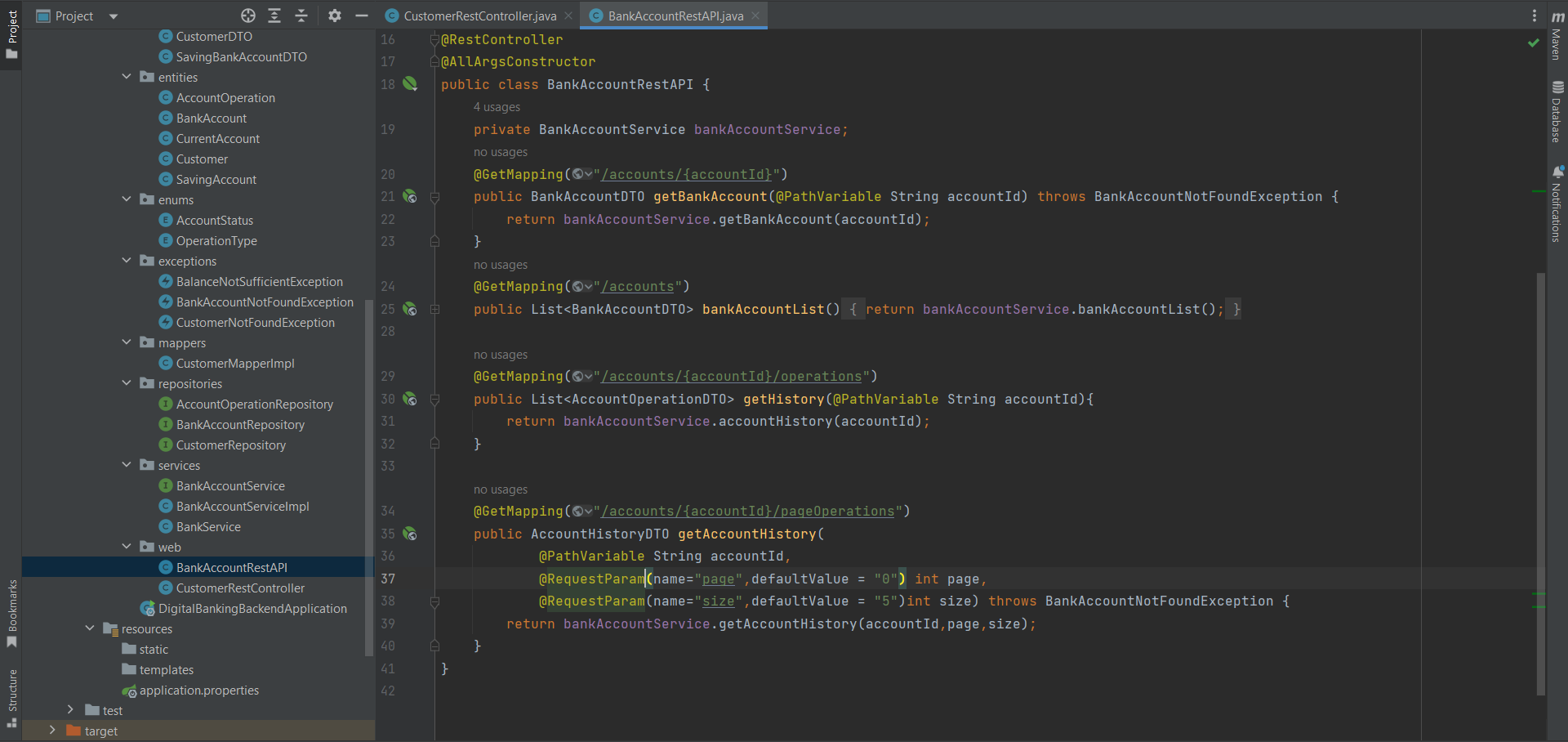
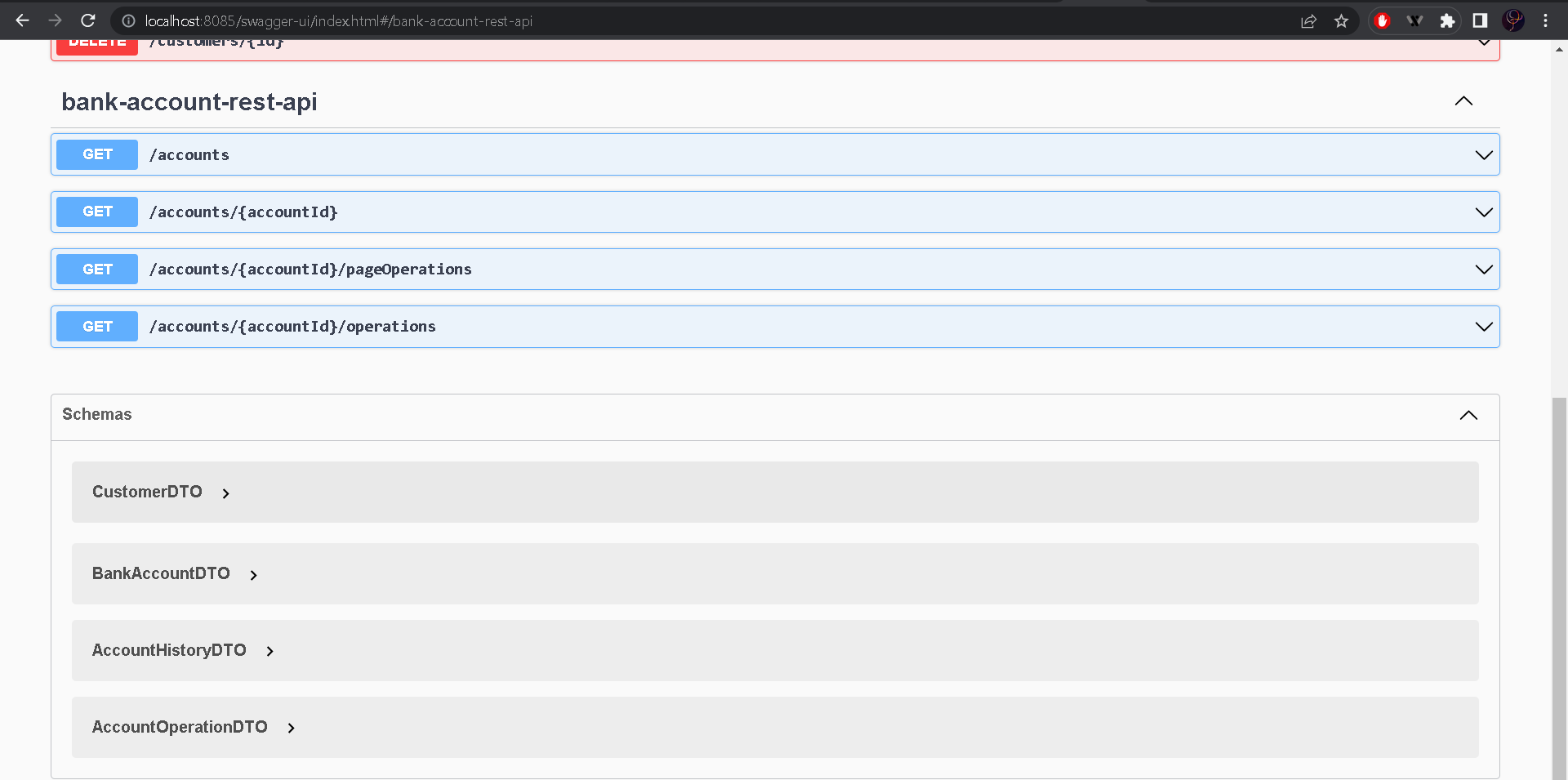
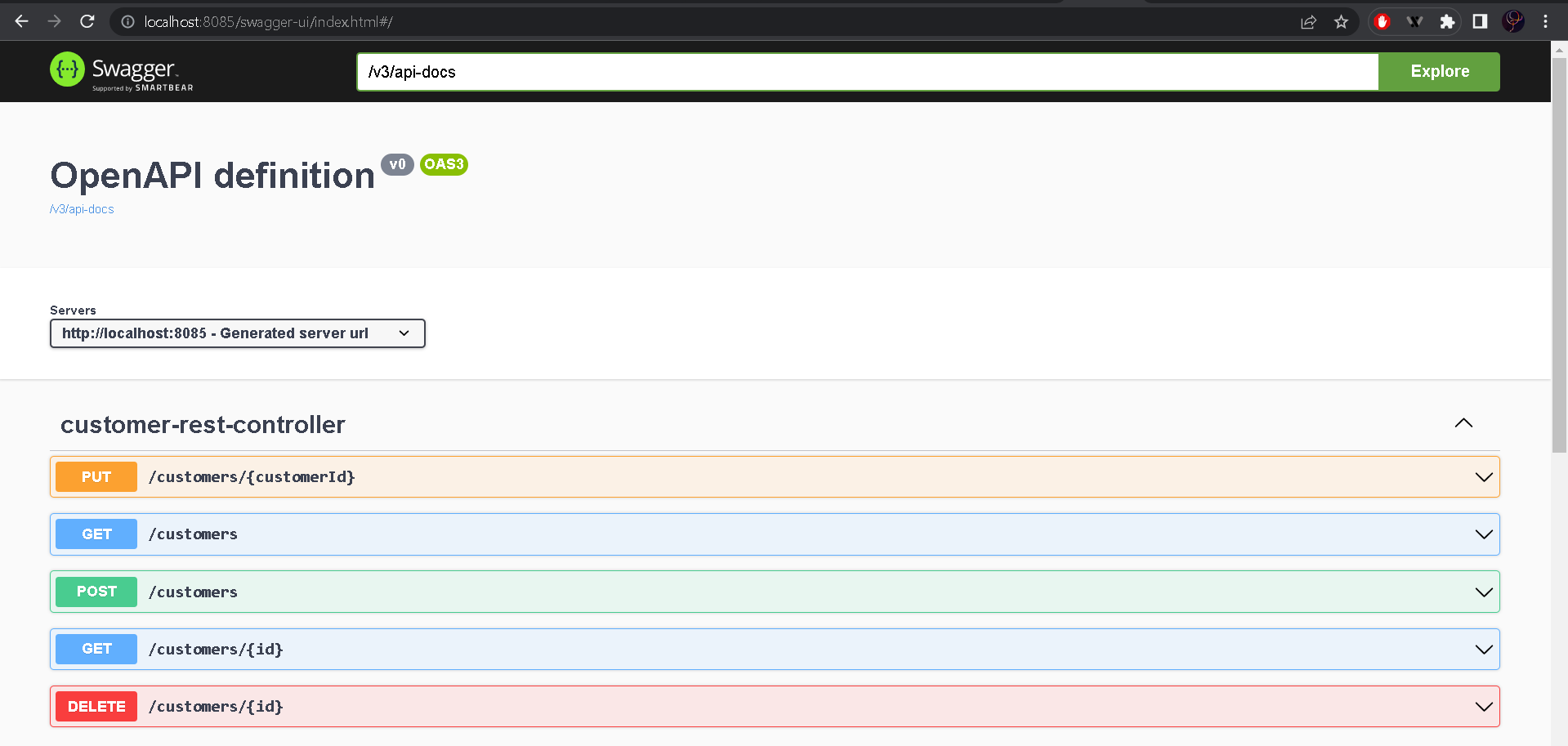
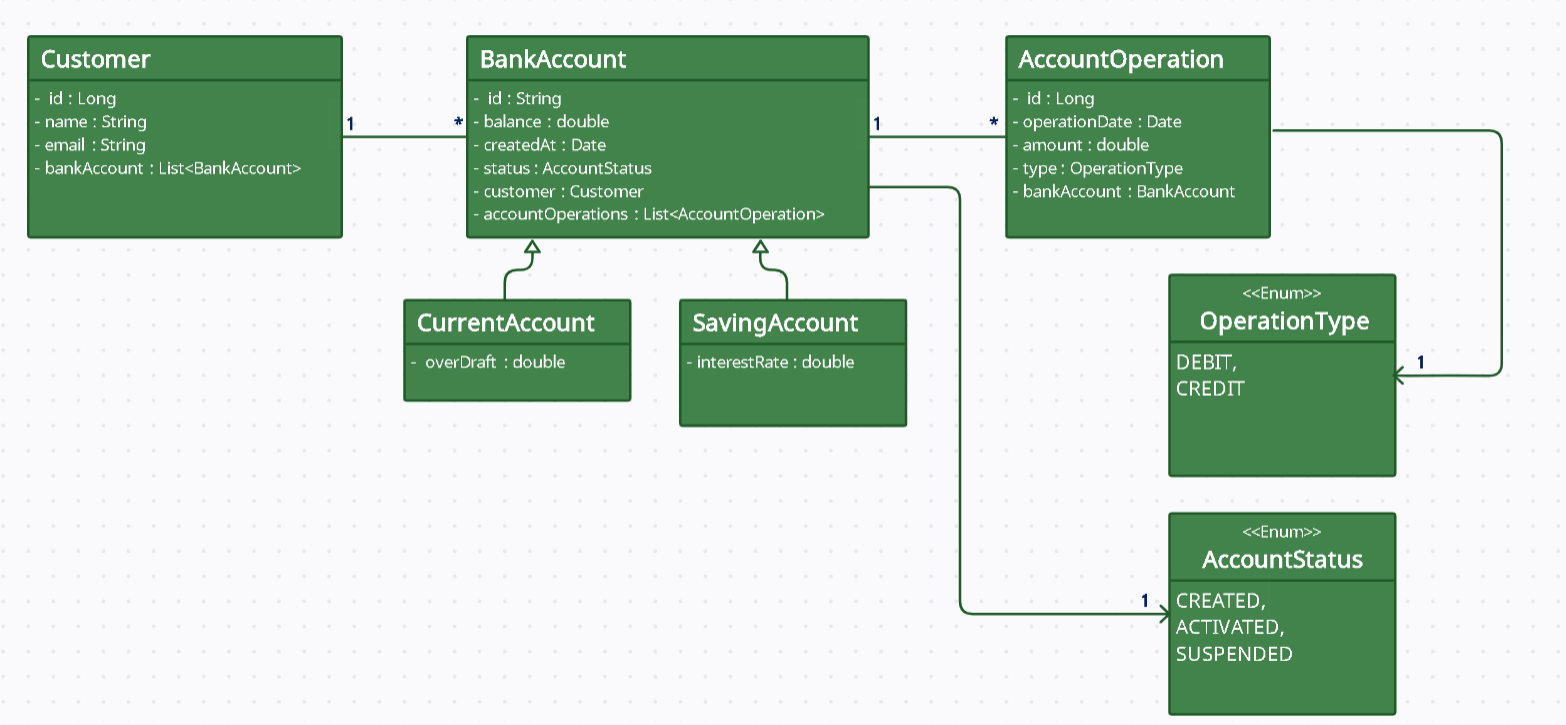
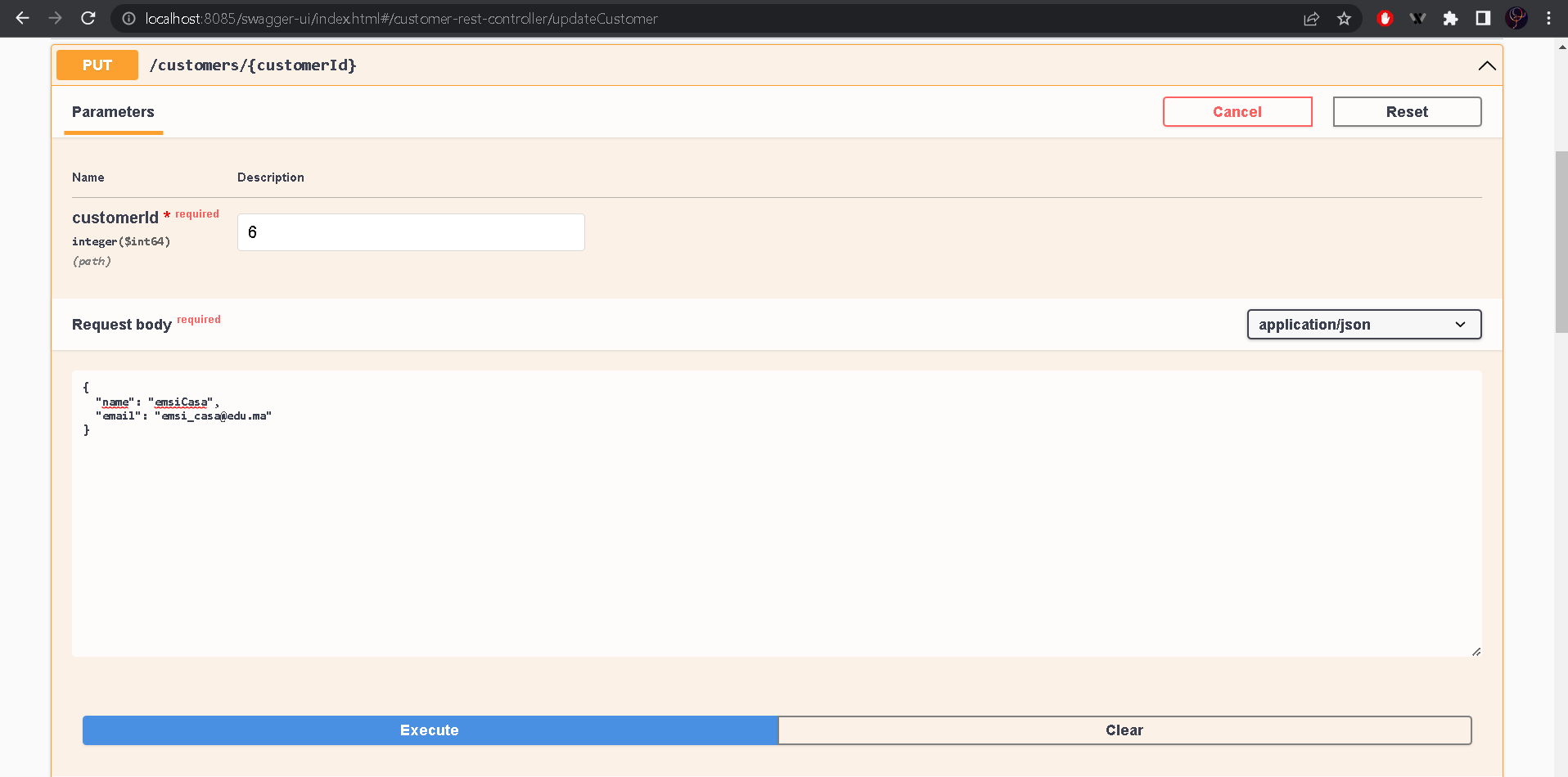
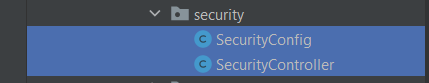
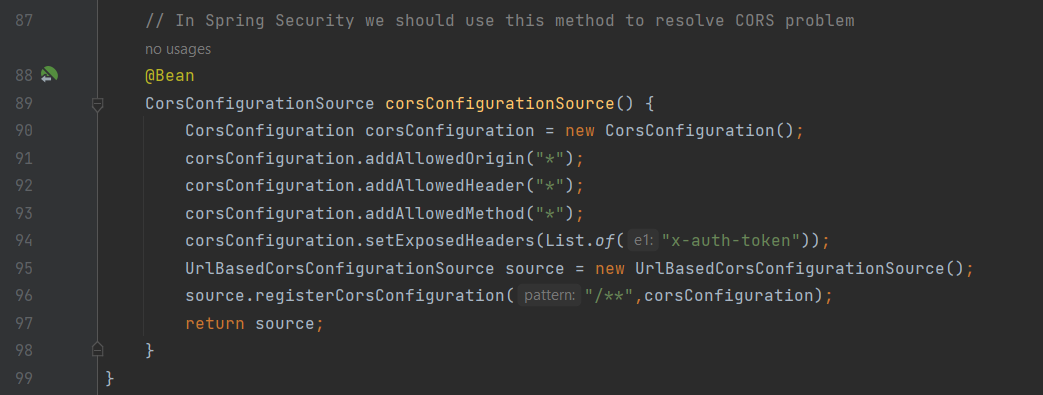
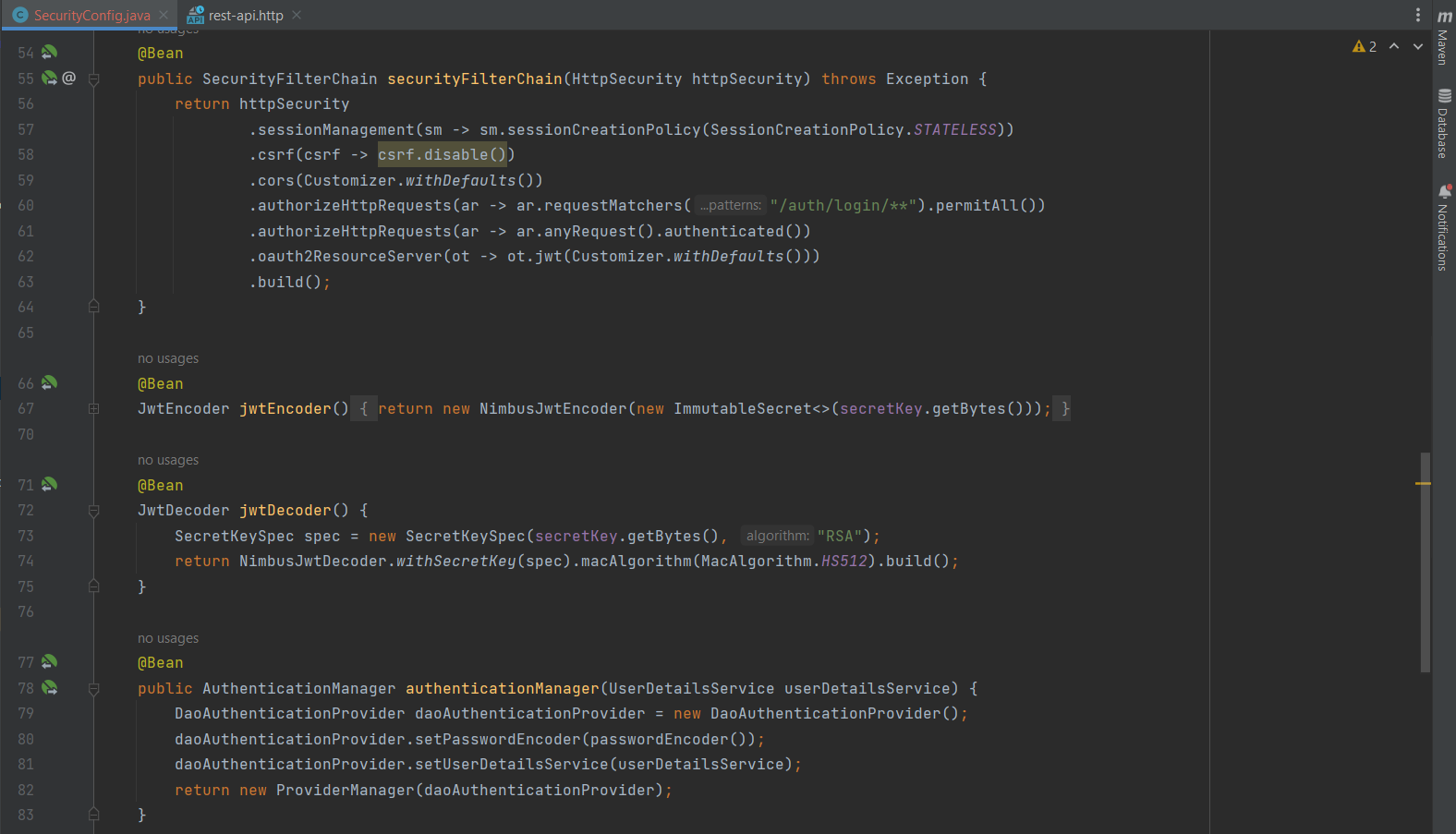
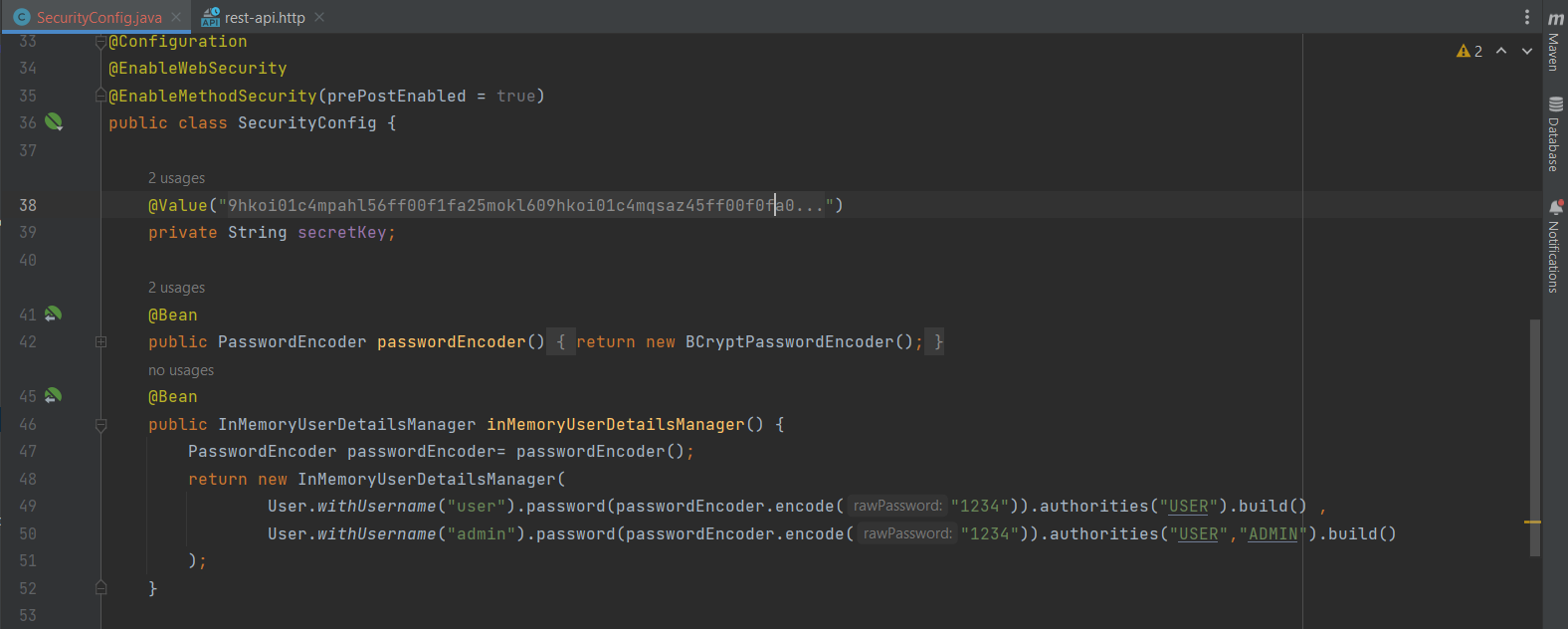
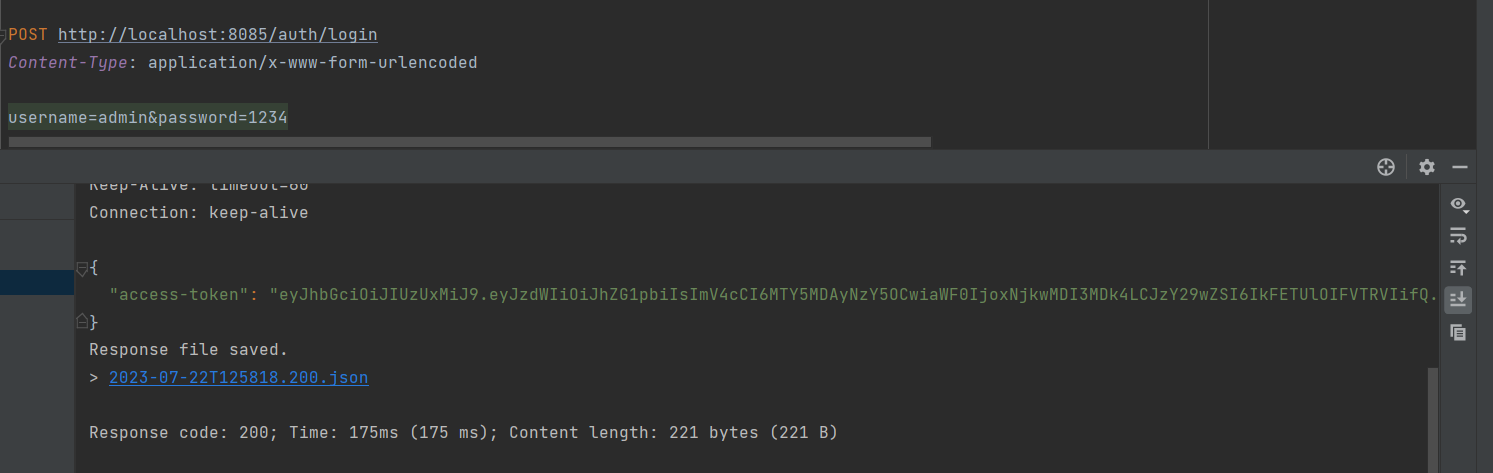
**### Configuration de la sécurité :**

Dans cette partie on configure un système d'authentification en mémoire avec deux utilisateurs (un utilisateur normal et un administrateur) ayant des mots de passe encodés en BCrypt.</h4>

Ce code configure la sécurité d'une application Spring avec une gestion de sessions sans état, la désactivation de la protection CSRF, la configuration CORS par défaut, l'autorisation de l'URL "/auth/login/\*\*" pour tous, et l'authentification via JWT en utilisant les beans JwtEncoder et JwtDecoder. Il crée également un gestionnaire d'authentification personnalisé avec l'objet DaoAuthenticationProvider et un gestionnaire d'authentification global avec le ProviderManager.</h4>

**### Genérer le Token**

**## Conclusion**



L'architecture backend de l'application Digital Banking repose sur Spring Boot et suit une approche en couches pour une meilleure séparation des responsabilités. Les couches DAO, service et RestController travaillent ensemble pour permettre la gestion des comptes bancaires et la manipulation des données. Les entités JPA représentent les objets métier et sont mappées aux tables de la base de données. Les DTOs sont utilisés pour transférer les données entre les différentes couches de l'application.

L'utilisation de Spring Boot facilite le développement, la configuration et la gestion de l'application. De plus, l'utilisation des annotations telles que `@Entity`, `@Repository`, `@Service` et `@RestController` permet de définir clairement le rôle des différents composants de l'application.

De plus, la sécurité de l'application est assurée grâce à l'utilisation de JSON Web Tokens (JWT). Les tokens JWT sont utilisés pour l'authentification et l'autorisation des utilisateurs, permettant ainsi de sécuriser les ressources sensibles de l'application. Le code inclut la configuration d'un filtre de sécurité qui utilise un encodeur et un décodeur JWT pour gérer les tokens. L'authentification est également prise en charge en utilisant l'interface UserDetailsService pour récupérer les informations d'utilisateur à partir de la couche DAO. Grâce à cette approche de sécurité, l'application garantit l'accès sécurisé aux données bancaires des utilisateurs. En somme, l'architecture backend du projet Digital Banking allie à la fois la flexibilité, la sécurité et la modularité pour offrir une expérience bancaire fluide et sécurisée aux utilisateurs.

En utilisant cette architecture, il est possible d'étendre l'application Digital Banking avec de nouvelles fonctionnalités et de maintenir un code clair et modulaire.

Partie Frontend de l'application web Digital Banking

Ce rapport décrit la partie frontend de l'application web Digital Banking développée en utilisant le framework Angular. L'application propose différentes fonctionnalités pour gérer les opérations bancaires en ligne, y compris la consultation des comptes, la gestion des clients et les opérations financières telles que le débit, le crédit et le transfert.

<br>Cette architecture frontend avec une couche de sécurité JWT renforcée contribue à offrir une expérience utilisateur sécurisée et fiable

<hr>

**# L'authentification**

Cette section représente un formulaire de connexion stylisé dans une carte avec des champs pour saisir le nom d'utilisateur et le mot de passe, ainsi qu'un bouton de connexion. Selon le type d'utilisateur authentifié (utilisateur "user" ou "admin"), des fonctionnalités spécifiques sont autorisées : l'utilisateur "user" peut consulter les clients et leurs opérations pour son compte, tandis que l'utilisateur "admin" a des droits étendus pour consulter, ajouter, modifier et supprimer des clients, ainsi que pour effectuer des transactions (débit, crédit, transfert).

**## Interface Home**

Cette section d'accueil présente un titre de bienvenue et deux boutons. Le premier bouton redirige vers la page de consultation des comptes bancaires, tandis que le deuxième bouton permet de vérifier les clients enregistrés.

L'interface principale de l'application est représentée par une section avec la classe CSS "home". Elle comprend les éléments suivants :

```

**# Partie Customers**

La partie "Customers" est divisée en deux sections : la consultation des clients existants et l'ajout de nouveaux clients.

**## 1.1- Consultation des clients**

Cette section affiche la liste des clients existants. Elle comporte un champ de recherche permettant de filtrer les clients par mot clé. Chaque client est affiché dans un tableau avec les informations telles que l'ID, le nom et l'e-mail. Deux boutons sont disponibles pour supprimer le client ou accéder à ses comptes.

**## 1.5- Ajout de nouveaux clients**

Cette section permet d'ajouter de nouveaux clients à l'application. Un formulaire est affiché avec les champs requis tels que le nom et l'e-mail. Des validations sont effectuées sur les champs pour s'assurer que les données saisies sont correctes. L'utilisateur peut enregistrer le nouveau client en cliquant sur le bouton "Save".

**# Comptes bancaires et opérations**

La section des comptes bancaires permet de consulter les informations d'un compte spécifique et d'effectuer des opérations financières.

Cette section est divisée en deux parties :

- La première partie affiche les détails du compte bancaire spécifié, y compris l'ID du compte et le solde actuel. Elle liste également les opérations effectuées sur le compte, affichant leur ID, la date, le type et le montant.

- La deuxième partie permet à l'utilisateur d'effectuer des opérations financières sur le compte sélectionné. Il peut choisir entre les options de débit, de crédit et de transfert. Si l'option de transfert est sélectionnée, un champ supplémentaire pour spécifier le compte de destination s'affiche. L'utilisateur doit également saisir le montant et une description de l'opération. Ensuite, il peut enregistrer l'opération en cliquant sur le bouton correspondant.

**## Interface Home**

Cette section d'accueil présente un titre de bienvenue et deux boutons. En tant qu'utilisateur, vous pouvez consulter vos comptes bancaires, mais l'accès aux informations détaillées des clients et aux actions avancées est restreint.

**# Partie Customers**

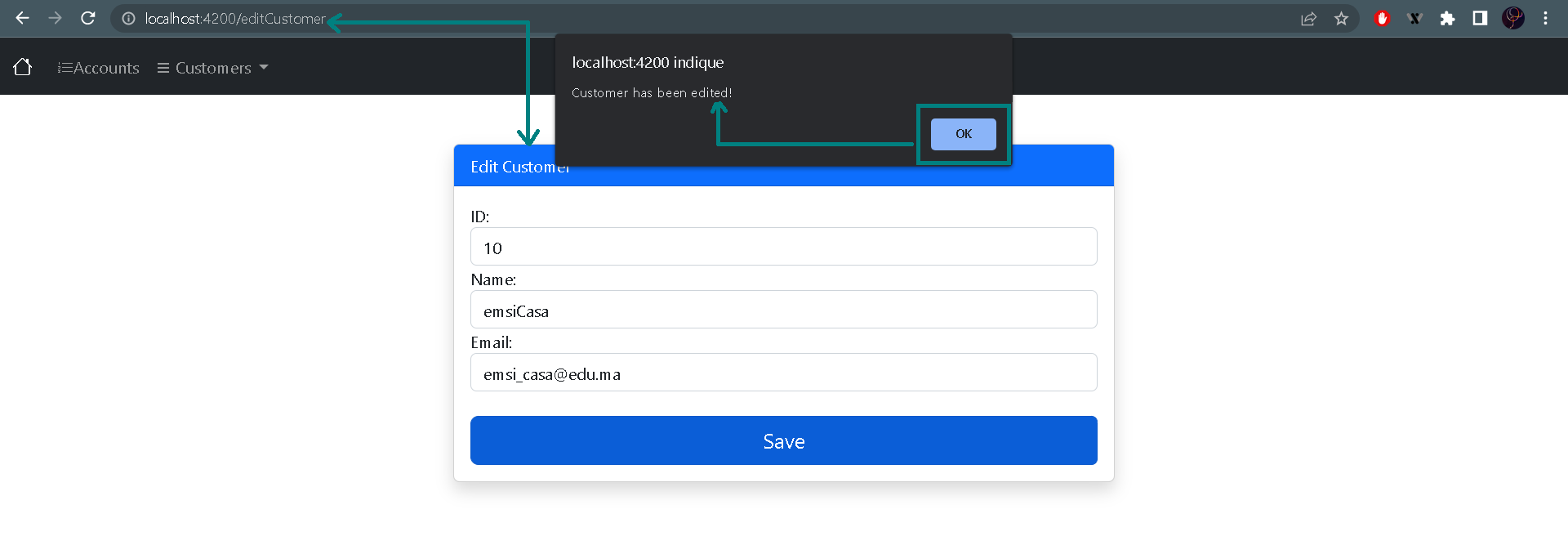
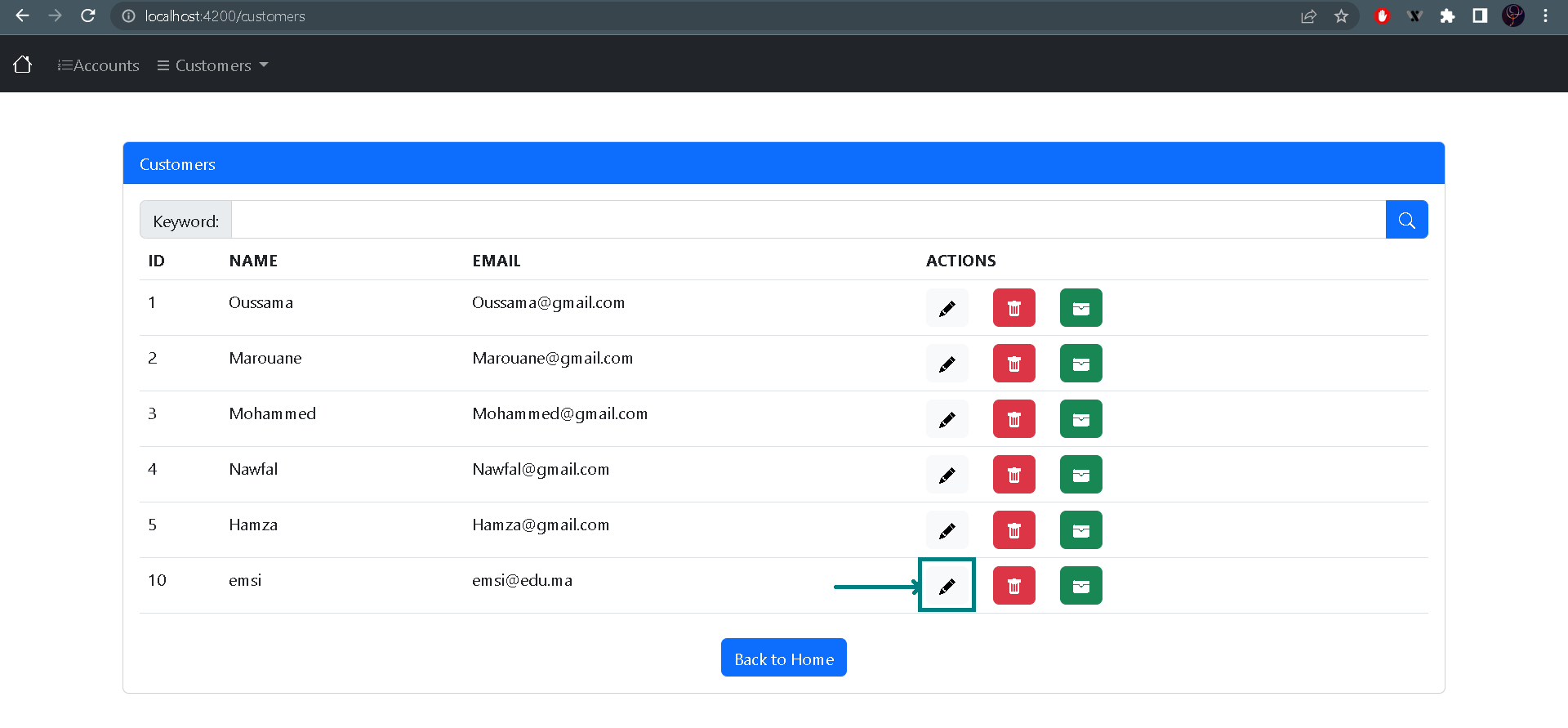
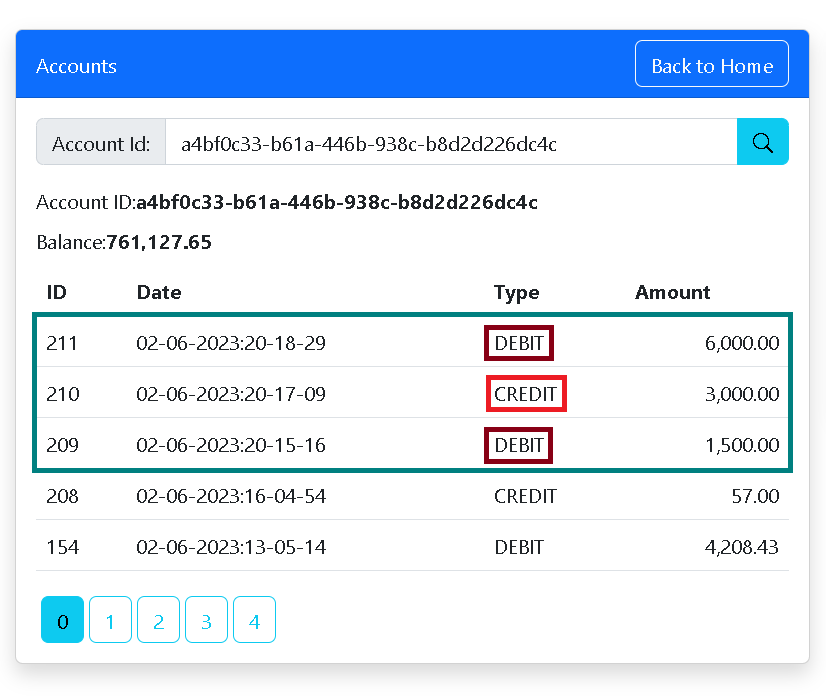
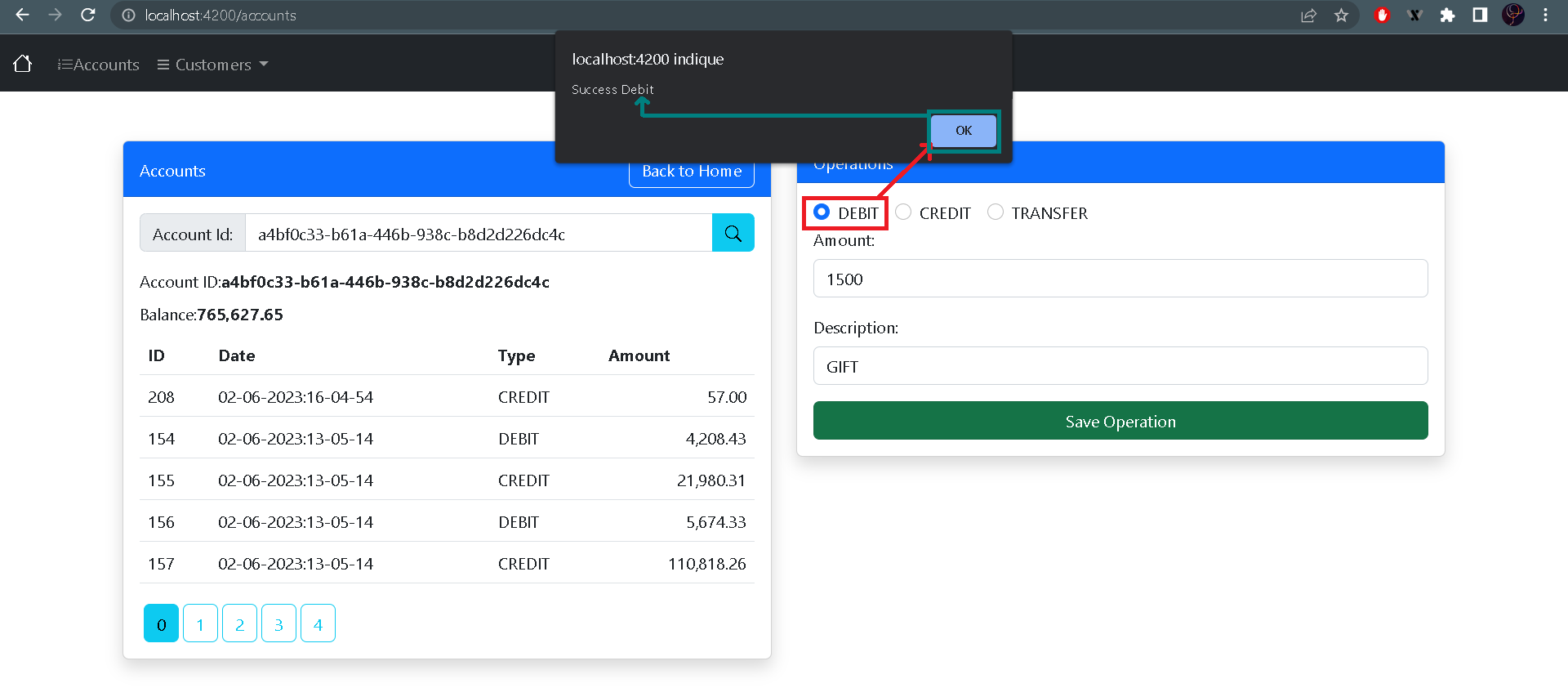
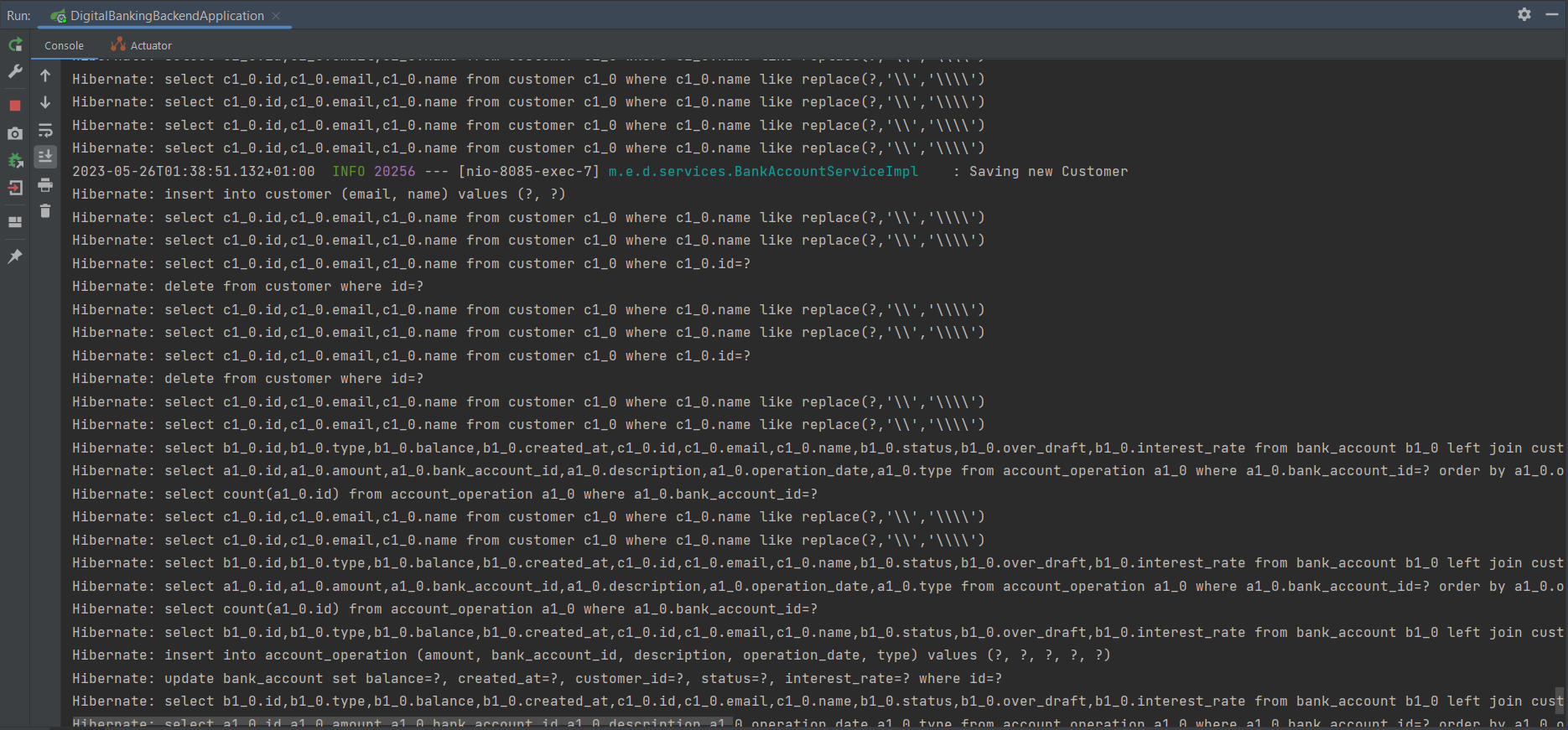
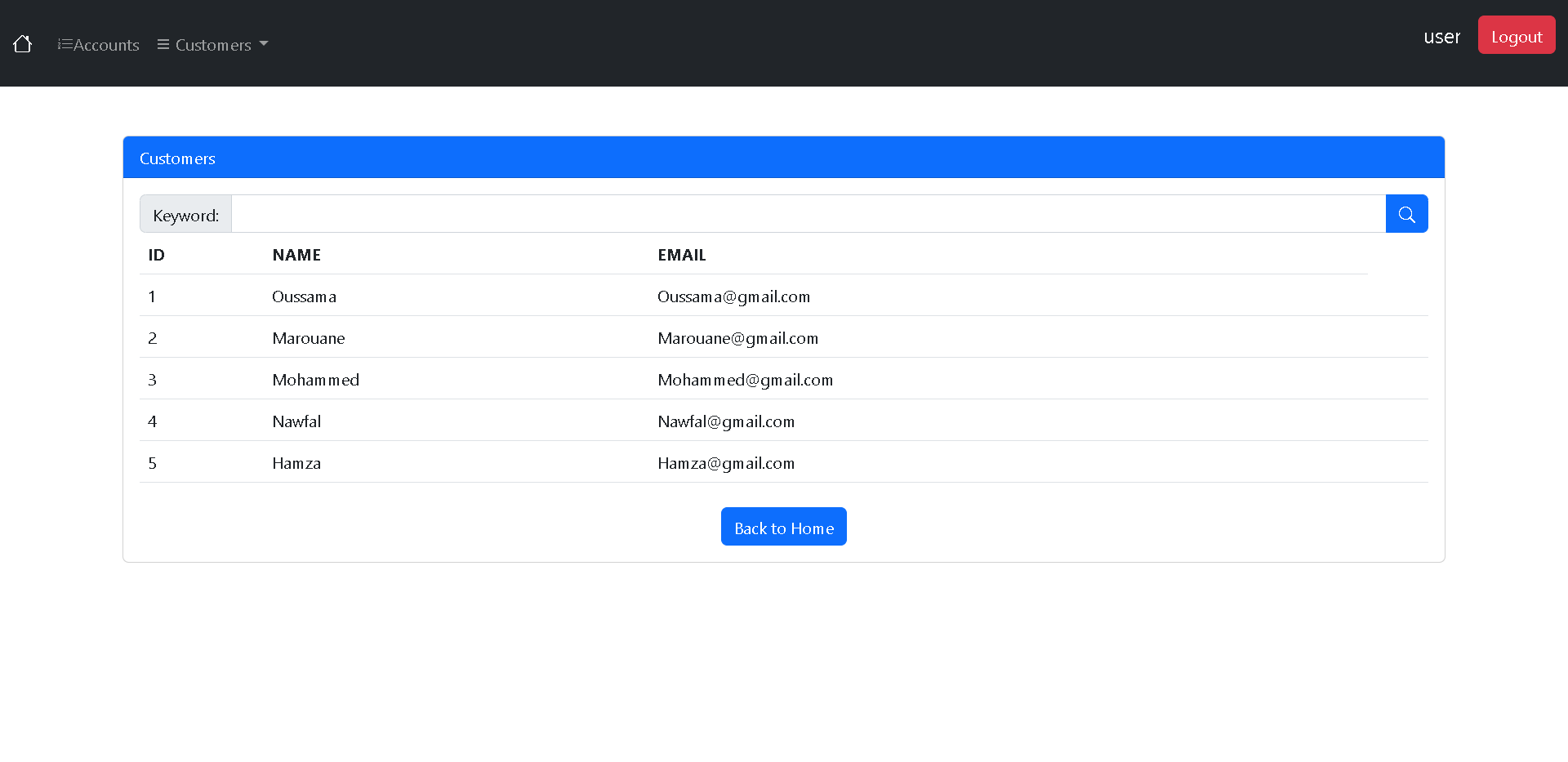
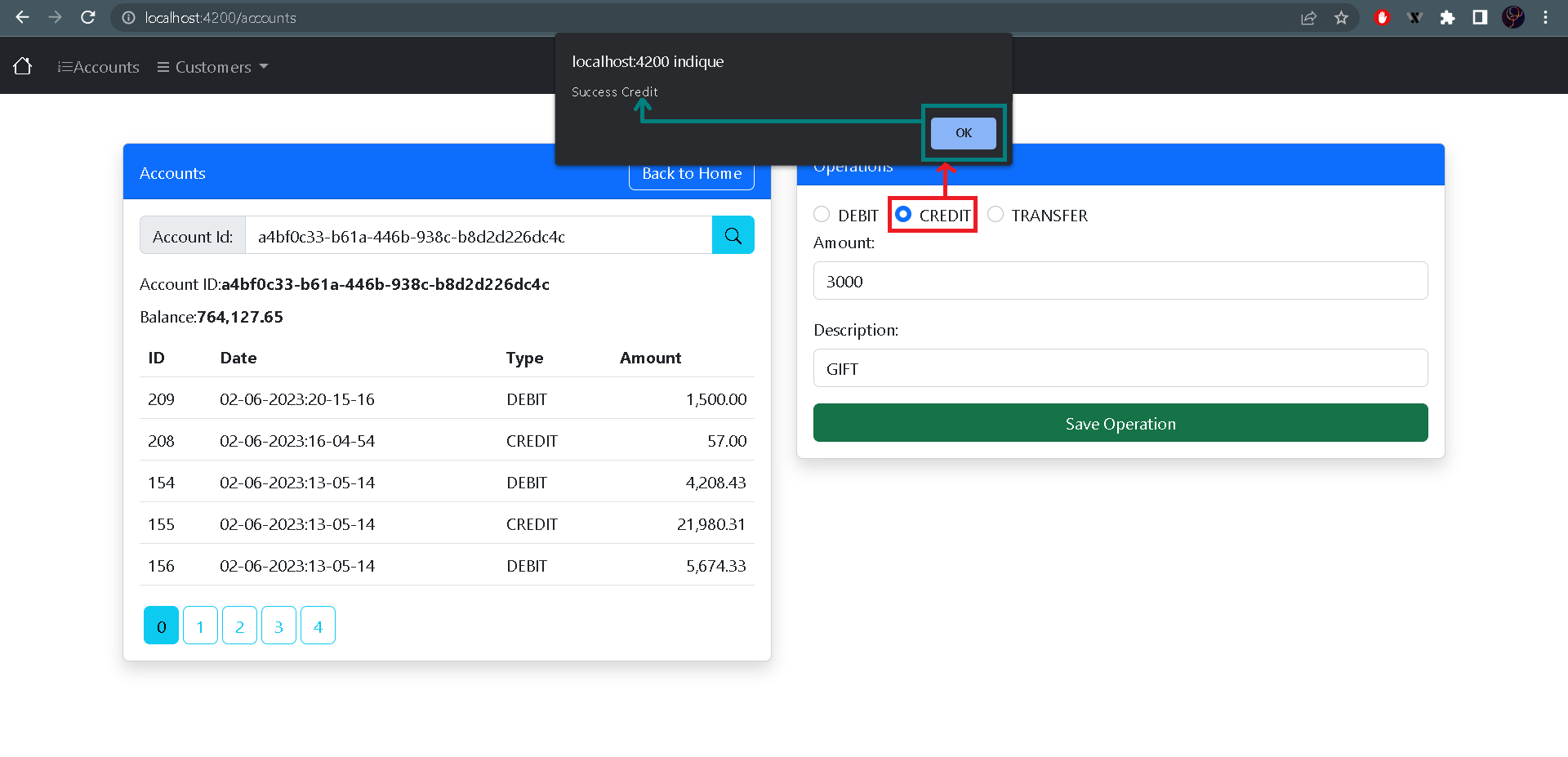
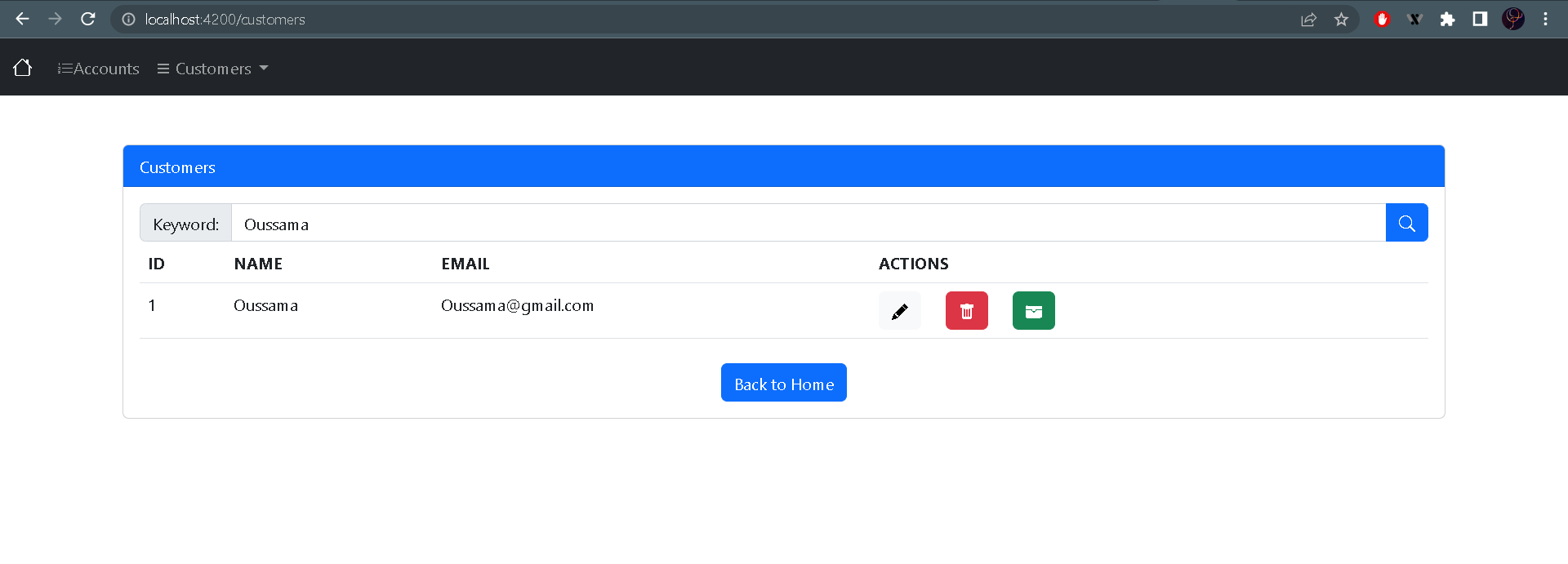
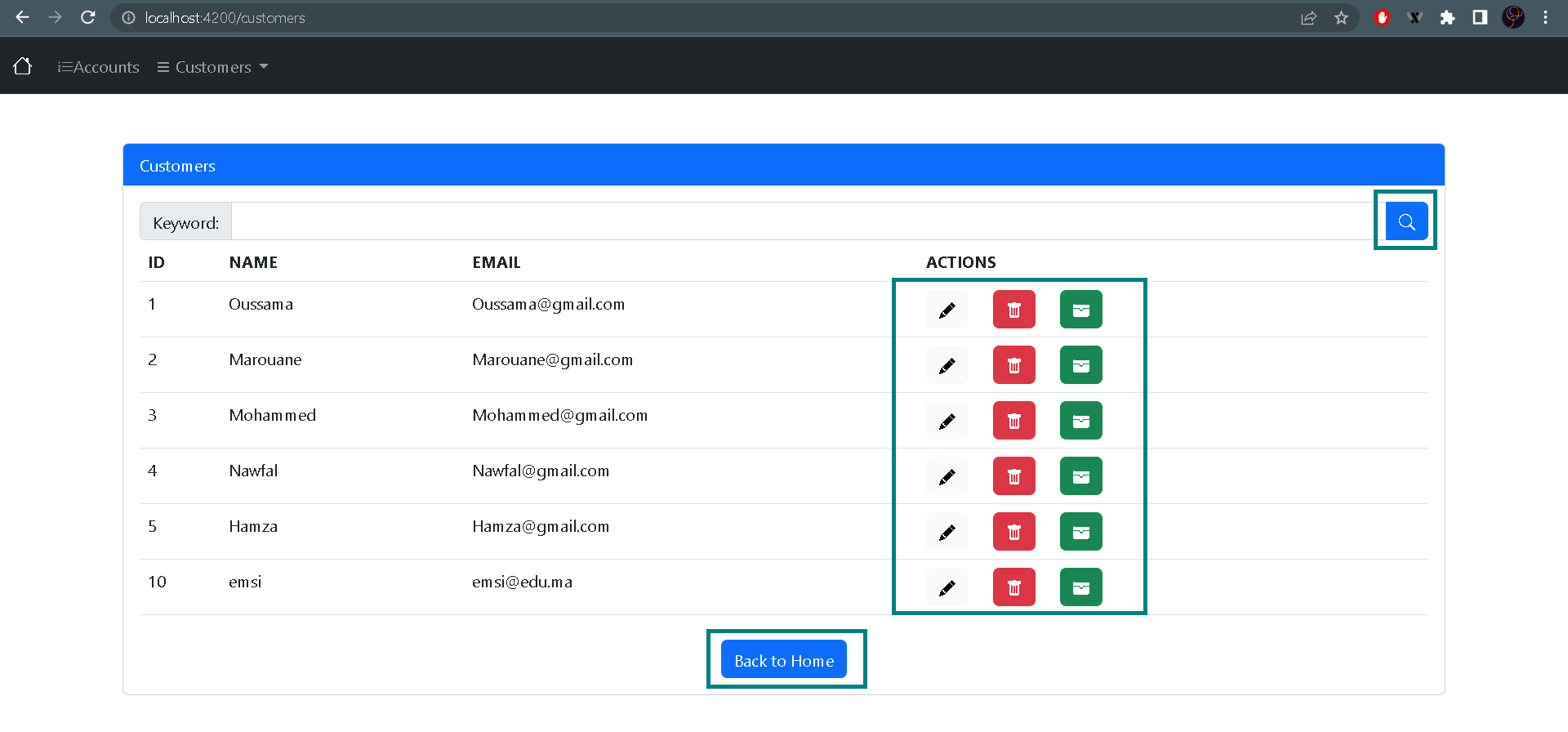
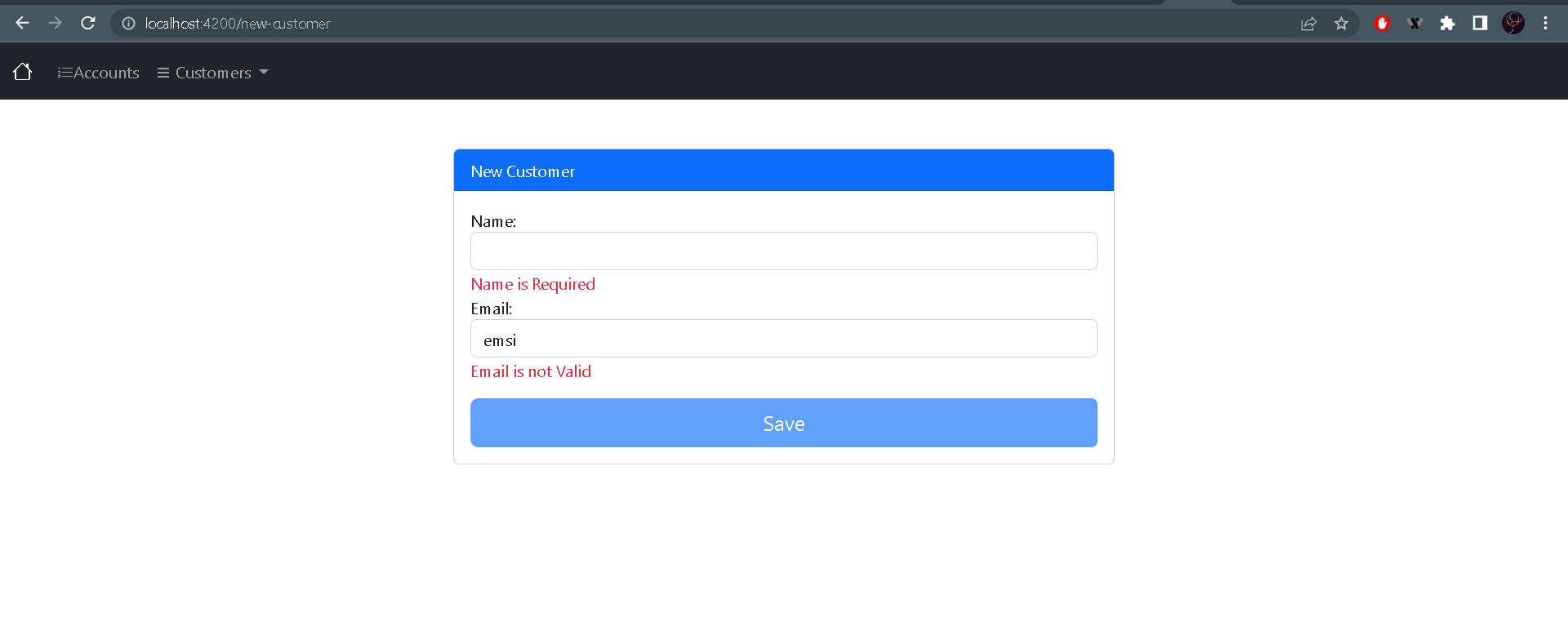
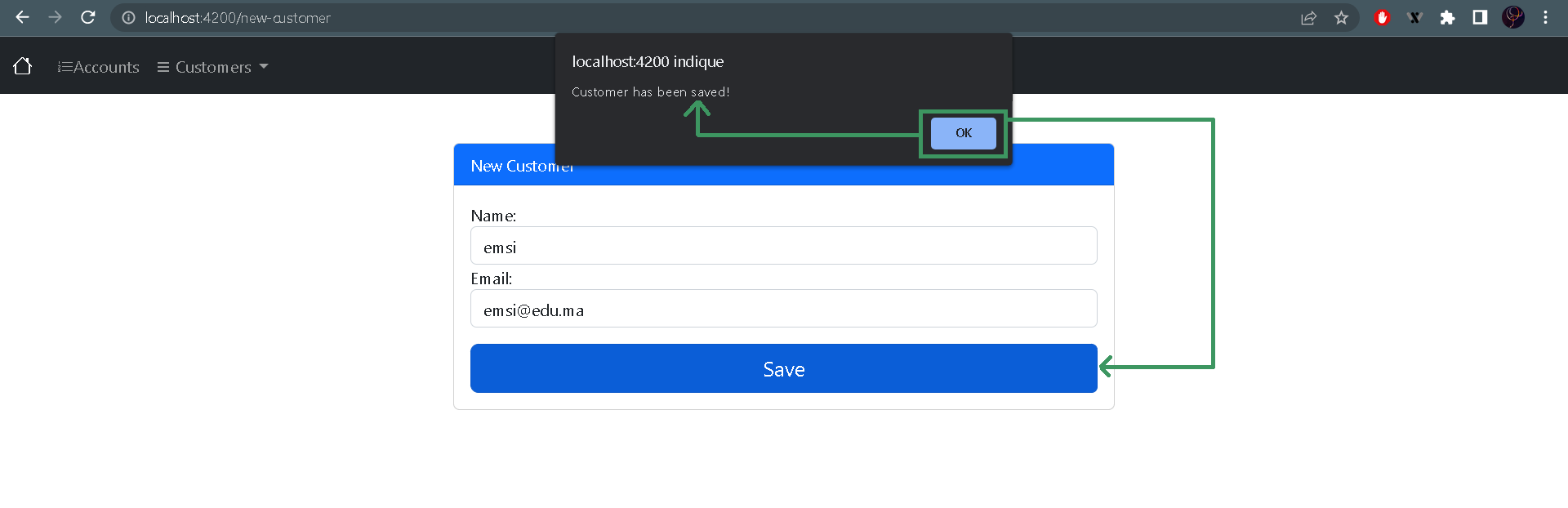
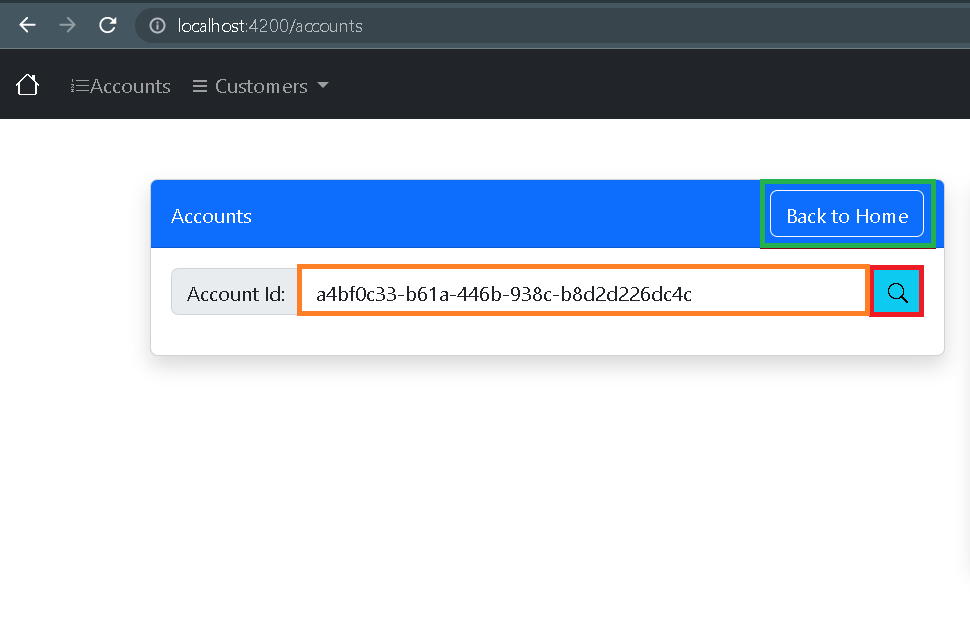
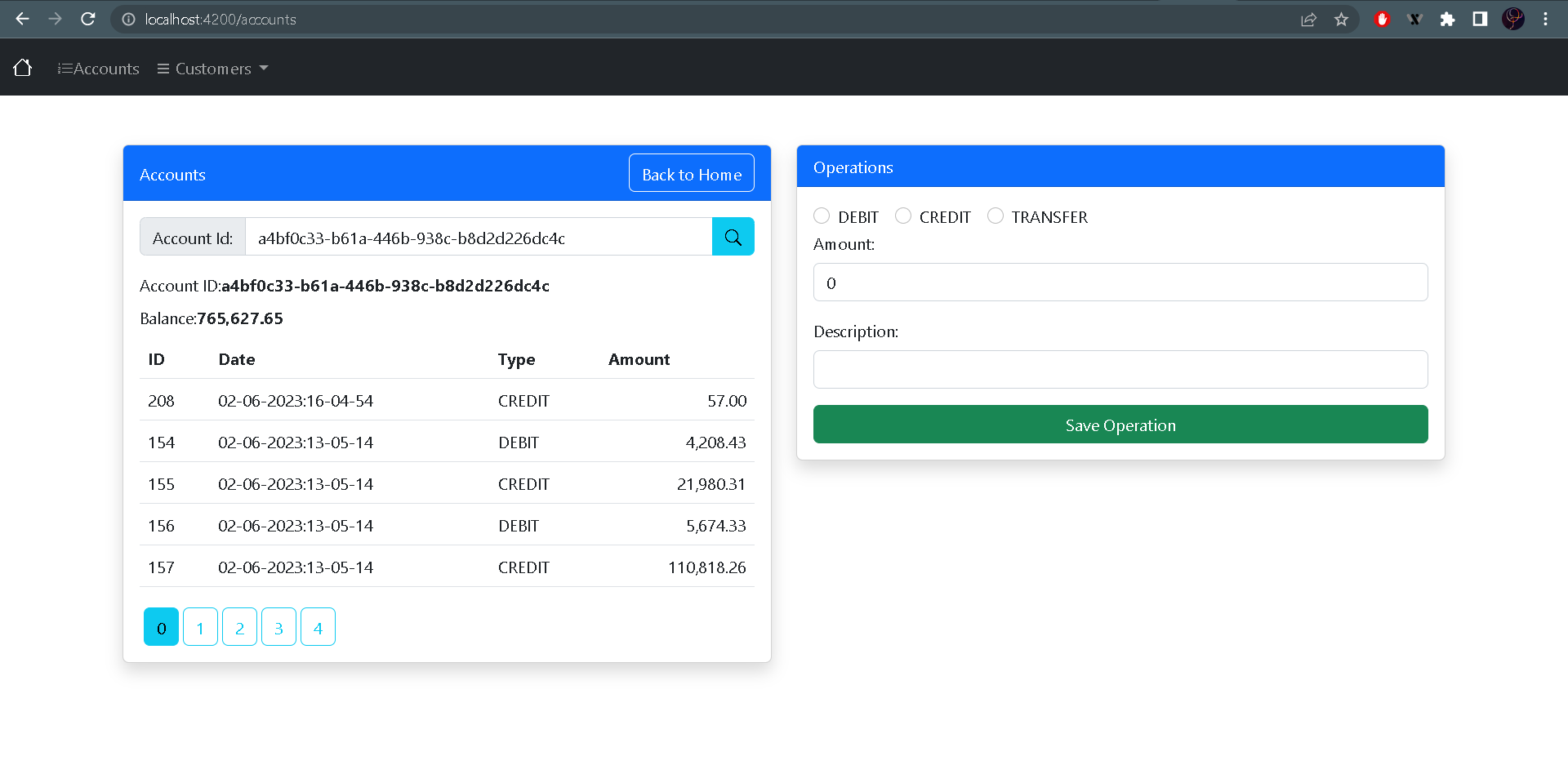
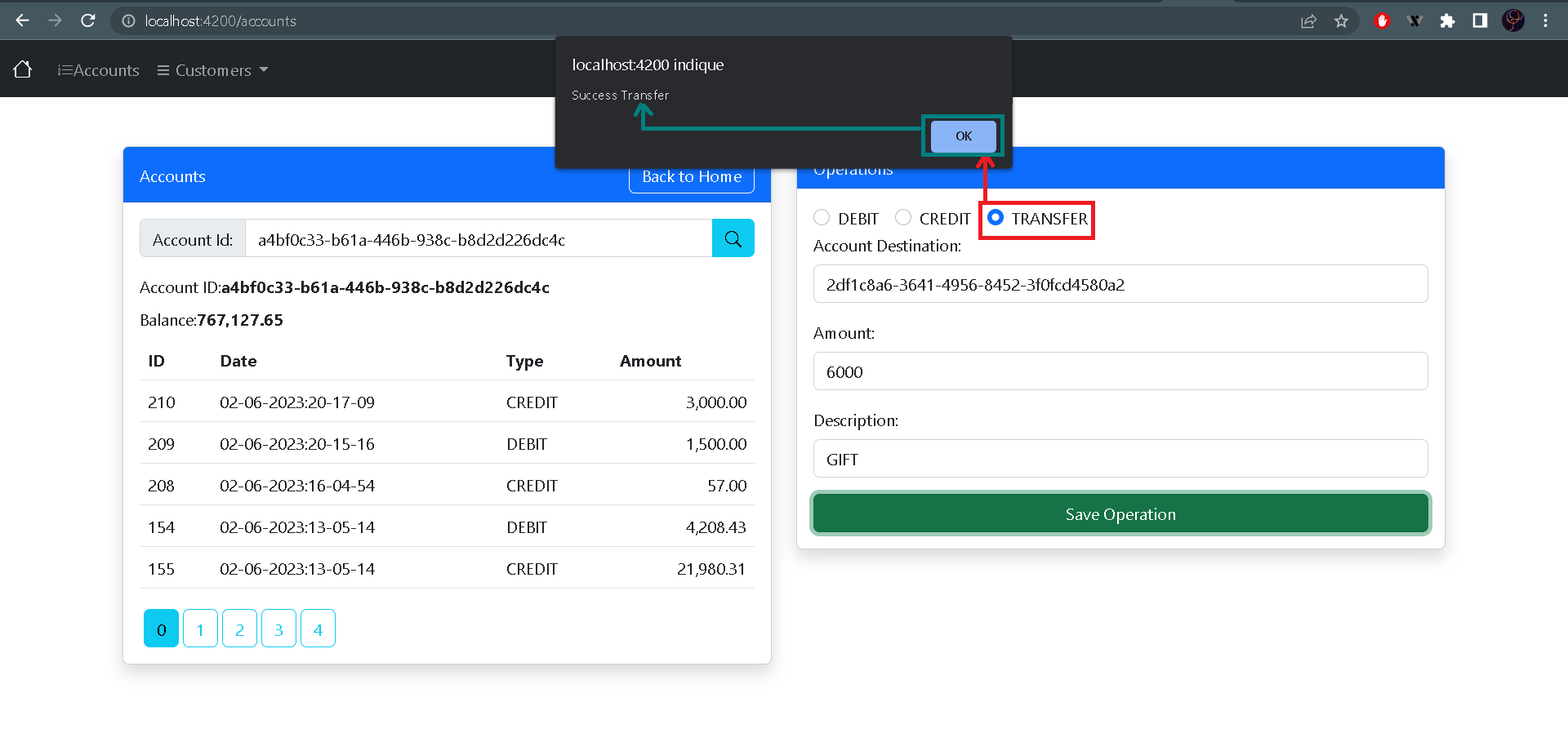
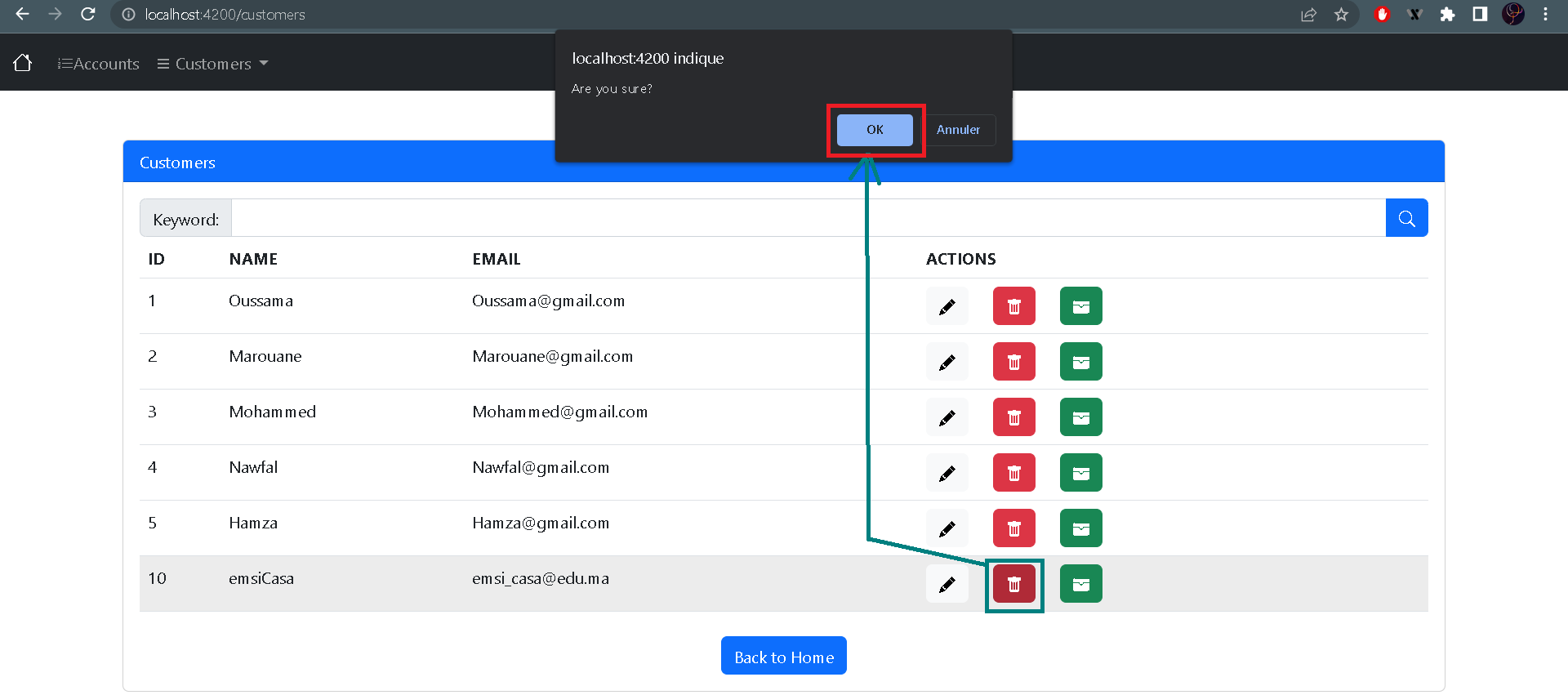
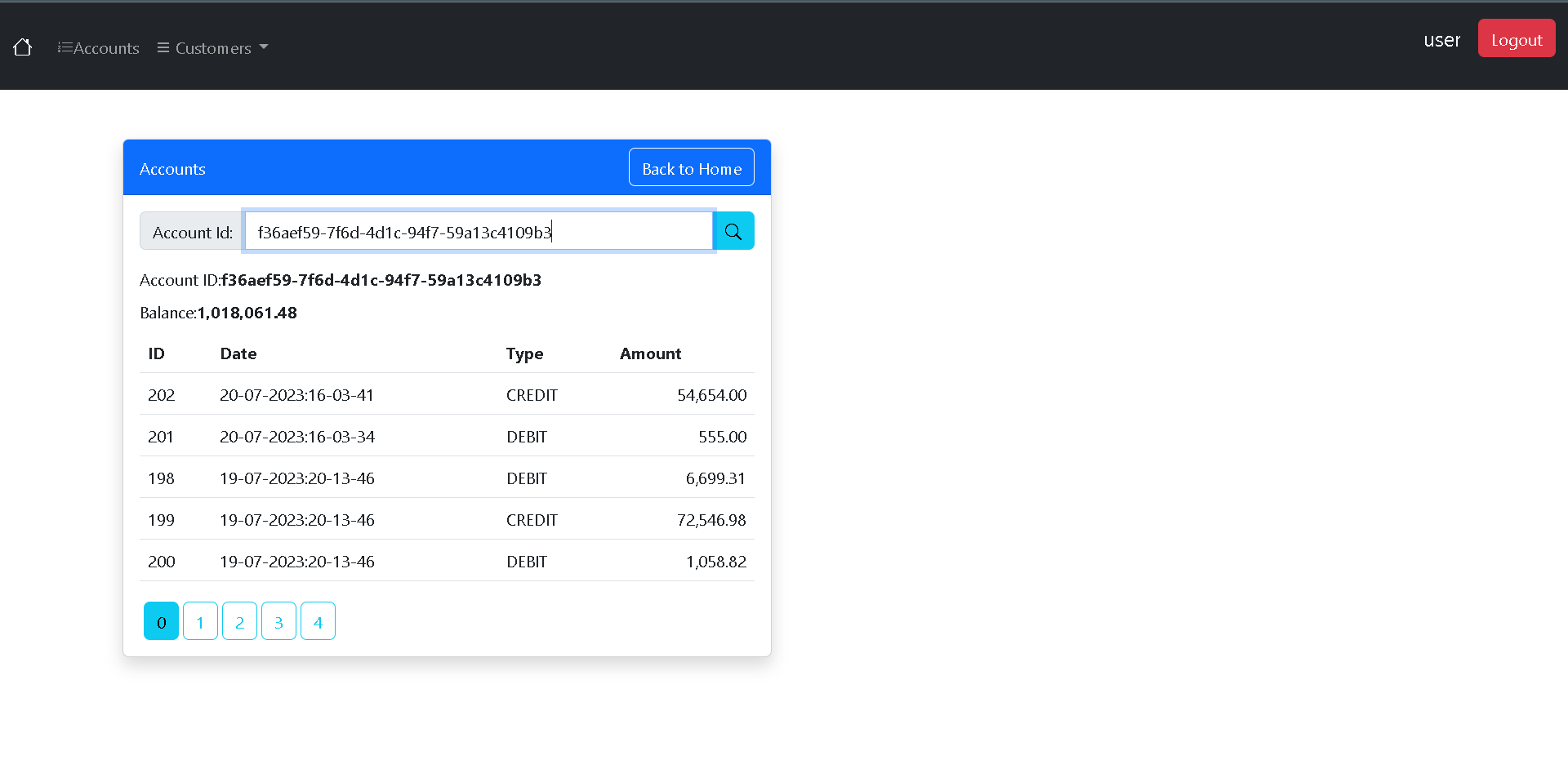
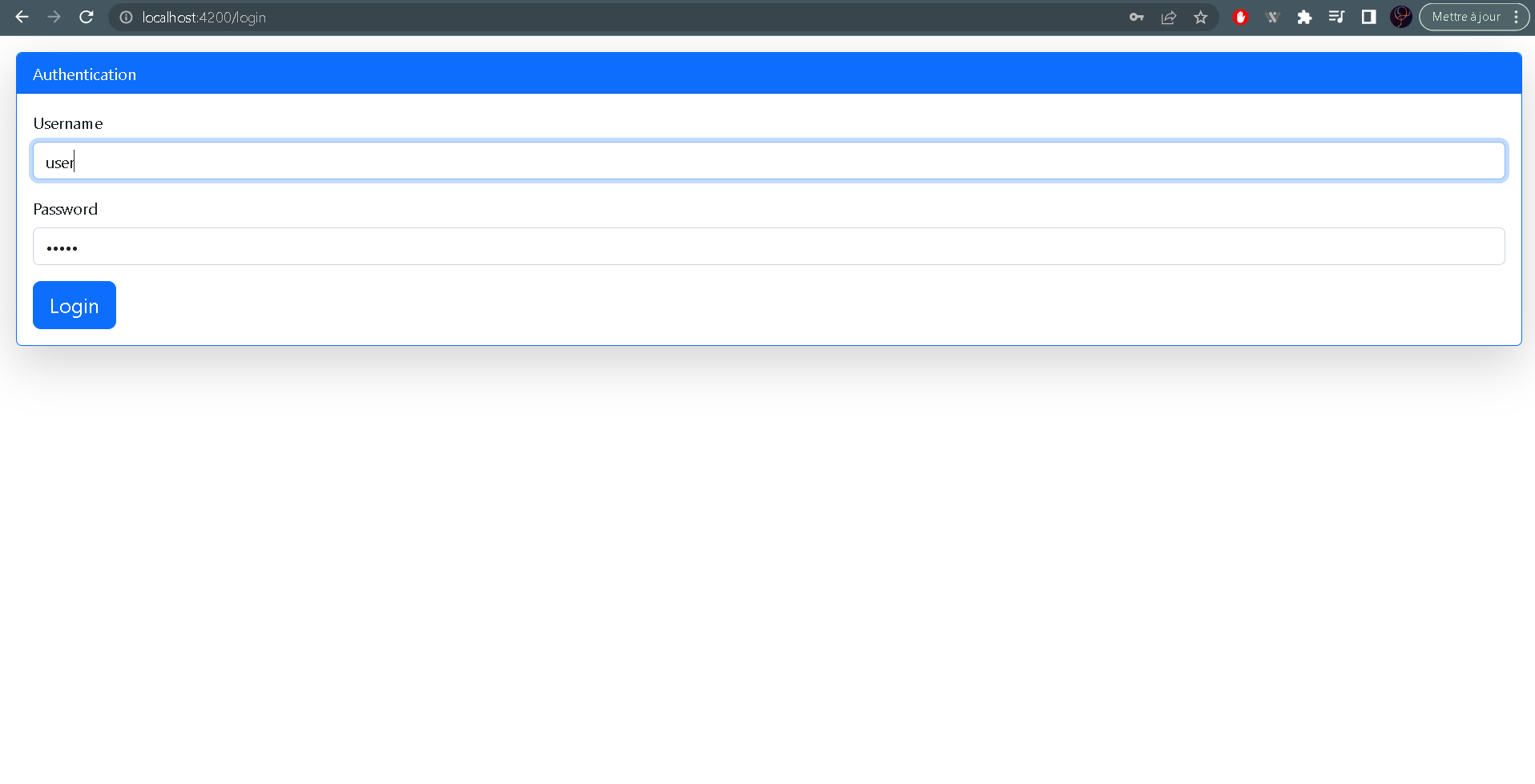
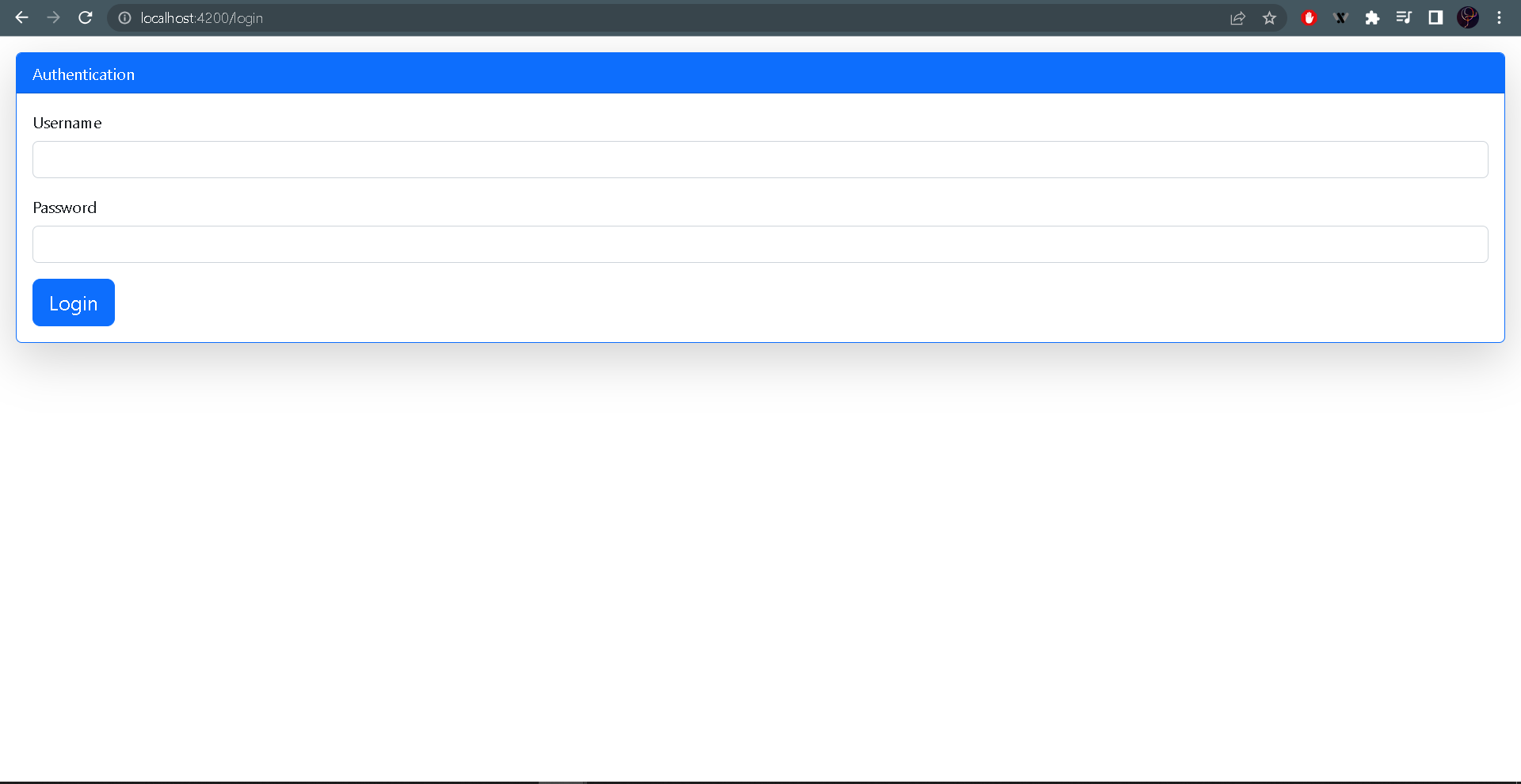
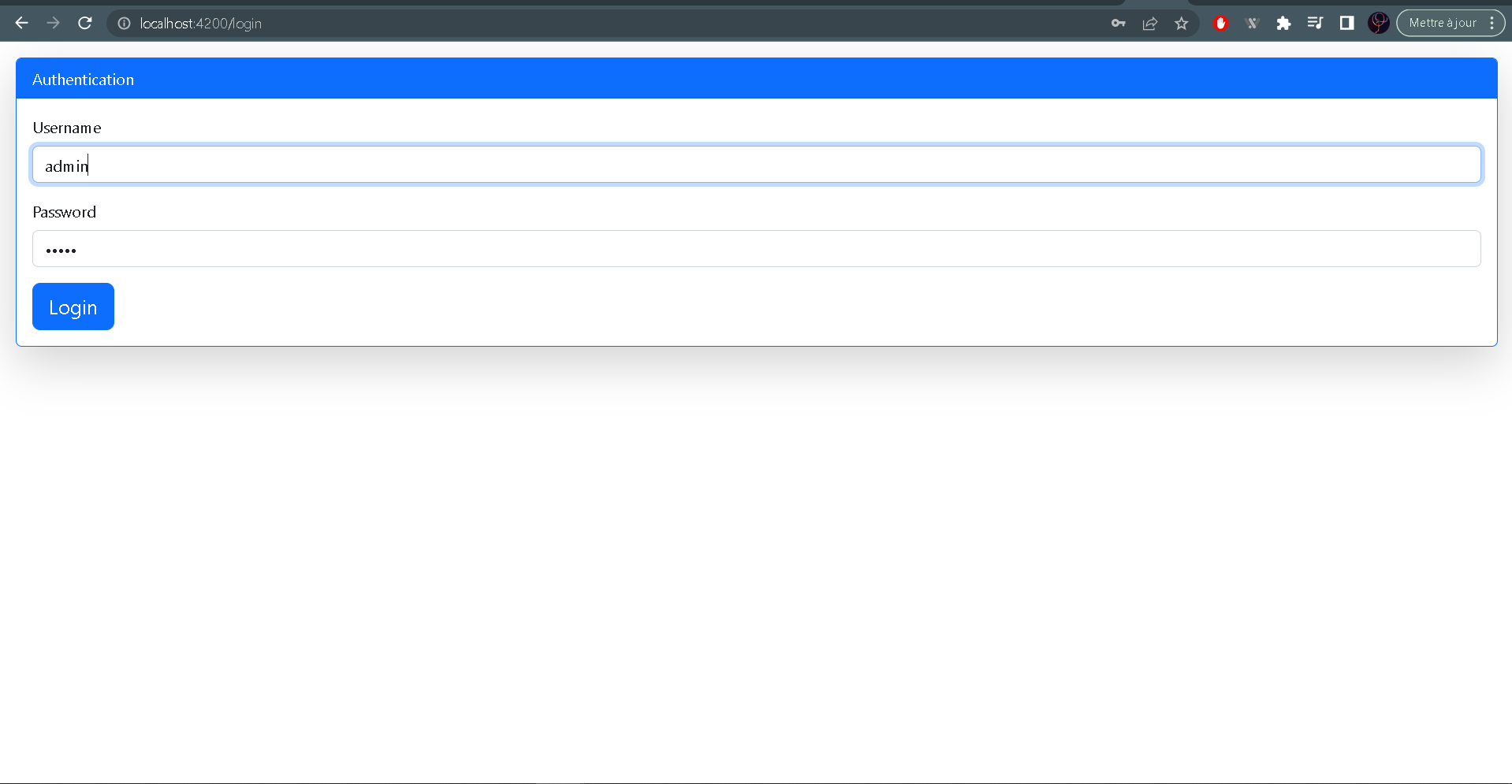
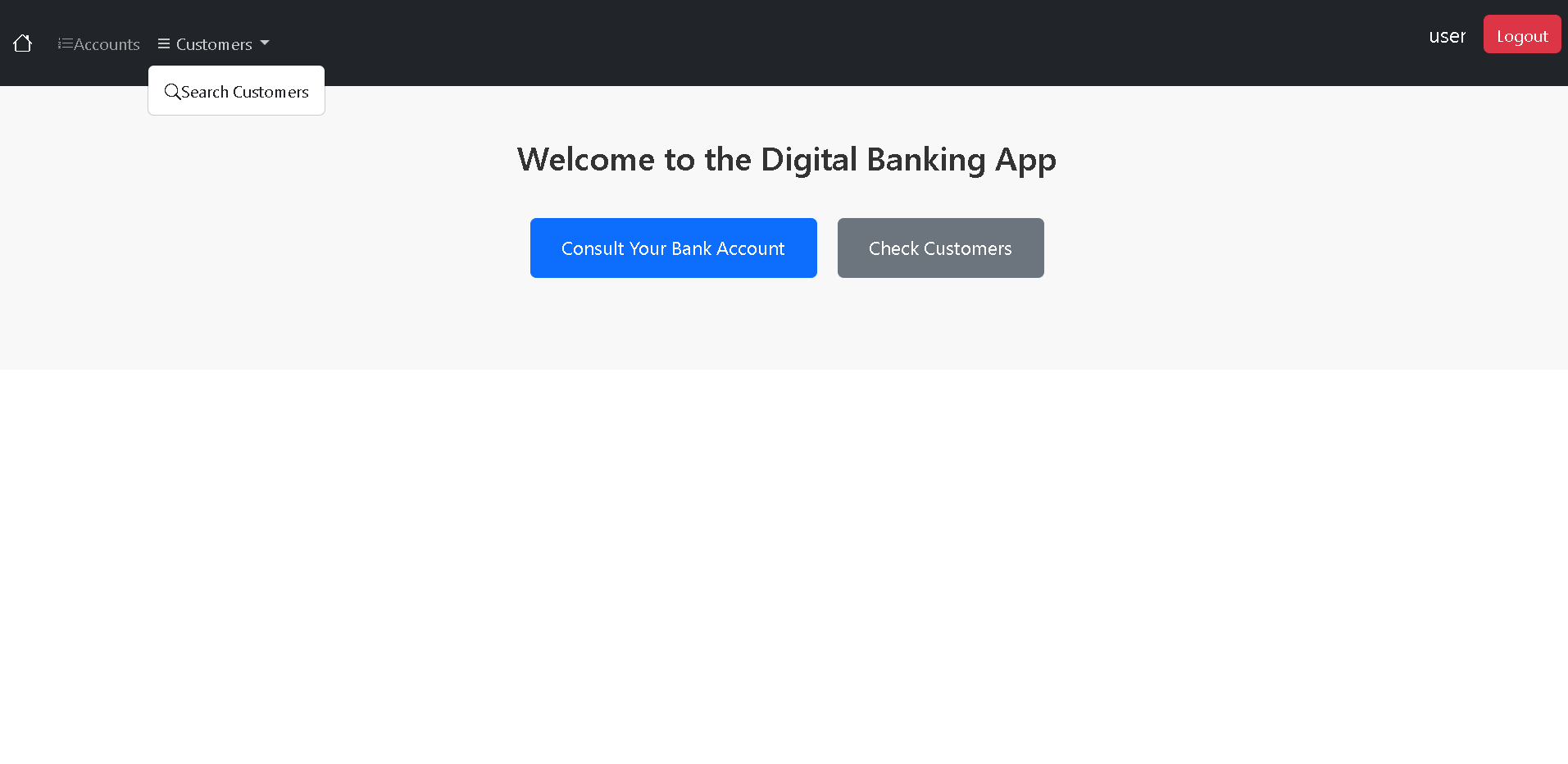
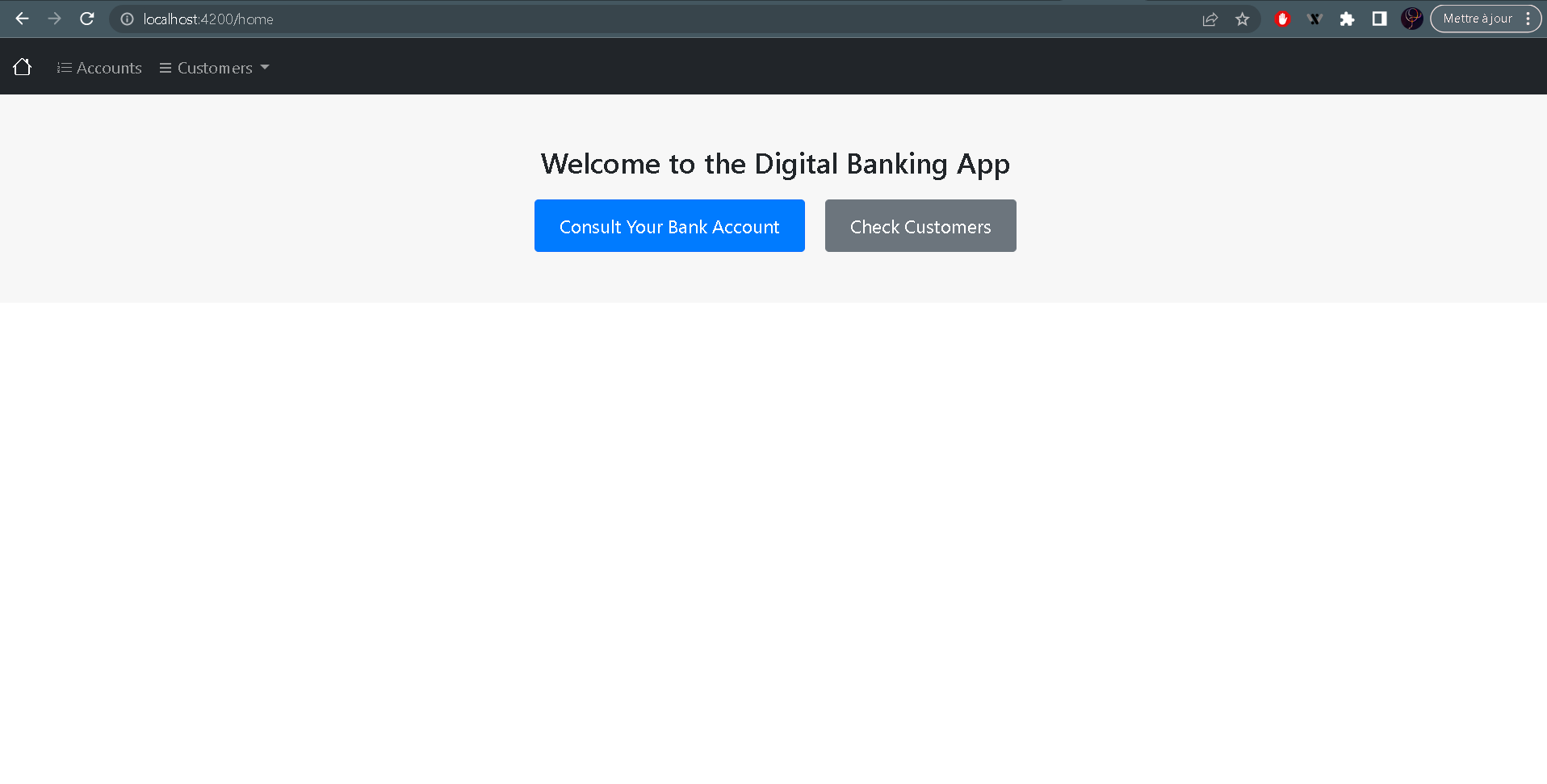
**## 1.1- Consultation des clients**

Dans cette section on affiche une liste de clients dans un tableau. Si l'utilisateur est un "user", il peut consulter la liste des clients, mais les actions avancées telles que l'édition, la suppression et l'accès aux comptes des clients ne sont pas disponibles pour lui.

**# Comptes bancaires**

Pour un utilisateur, cette section affiche un formulaire permettant de rechercher des comptes bancaires en utilisant leur ID. Les détails du compte bancaire sont affichés, y compris le solde et les opérations effectuées sur le compte. L'utilisateur peut également naviguer entre les pages des opérations et voir les détails de chaque opération.

Cependant, certaines fonctionnalités sont restreintes pour un utilisateur. Par exemple, la partie "Operations" qui permet d'effectuer des opérations (débit, crédit, transfert) et d'ajouter des opérations est réservée à l'administrateur.



**# Conclusion**

Ce rapport a décrit la partie frontend de l'application web Digital Banking développée en utilisant le framework Angular. L'interface utilisateur permet aux utilisateurs de consulter les clients enregistrés, d'ajouter de nouveaux clients, de consulter les détails des comptes bancaires, d'afficher les opérations effectuées et d'effectuer de nouvelles opérations financières telles que le débit, le crédit et le transfert. L'application offre une expérience conviviale pour gérer les opérations bancaires en ligne de manière efficace et sécurisée.