ECOLE NORMALE SUPÉRIEURE DE
L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA

UNIVERSITÉ HASSAN II DE CASABLANCA

ENSET

المدرسة العليا لأساتذة التعليم التقني
المحمدية
جامعة الحسن الثاني بالدار البيضاء

DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

Projet JEE Spring Angular

Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID

Digital Banking

Réalisé par :

Najat ES-SAYYAD

Année Universitaire : 2022-2023

ENSET, Avenue Hassan II - B.P. 159 - Mohammedia - Maroc
☎ 05 23 32 22 20 / 05 23 32 35 30 – Fax : 05 23 32 25 46 - Site Web: www.enset-media.ac.ma
E-Mail : contact@enset-media.ac.ma

Introduction

Dans cet examen, nous allons explorer une application qui suit une architecture en trois couches. Cette architecture est un modèle couramment utilisé dans le développement d'applications JEE, qui offre une structure modulaire et une séparation claire des responsabilités.

La première couche de notre application est la couche DAO (Data Access Object). Cette couche repose sur des technologies telles que Spring Data, JPA, Hibernate et JDBC. Elle est responsable de l'accès aux données de l'application. Grâce à Spring Data, le développement des opérations d'accès aux données est simplifié, tandis que JPA facilite le mappage des objets Java aux entités de base de données relationnelles. Hibernate, quant à lui, est un framework de persistance populaire qui implémente JPA et facilite l'interaction avec la base de données. Enfin, JDBC est une API Java standard pour interagir avec les bases de données relationnelles.

La deuxième couche de notre architecture est la couche métier. Cette couche est responsable de la logique métier de notre application. Elle contient des interfaces définissant les contrats que les services métier doivent suivre, ainsi que leurs implémentations correspondantes. La couche métier traite les règles de gestion, les calculs et les opérations spécifiques à notre domaine d'application. Elle encapsule la logique métier pour assurer la cohérence et la réutilisabilité du code.

La troisième couche de notre architecture est la couche DTO (Data Transfer Object) et les mappeurs DTO/Entités JPA. Les DTO sont utilisés pour transférer des données entre les différentes couches de l'application. Ils sont généralement des objets simples et légers qui contiennent les données nécessaires pour une opération spécifique. Les mappeurs DTO/Entités JPA facilitent la conversion des données entre les objets DTO et les entités JPA. Ces composants permettent de séparer les modèles de données internes de l'application des modèles de données exposés via l'API.

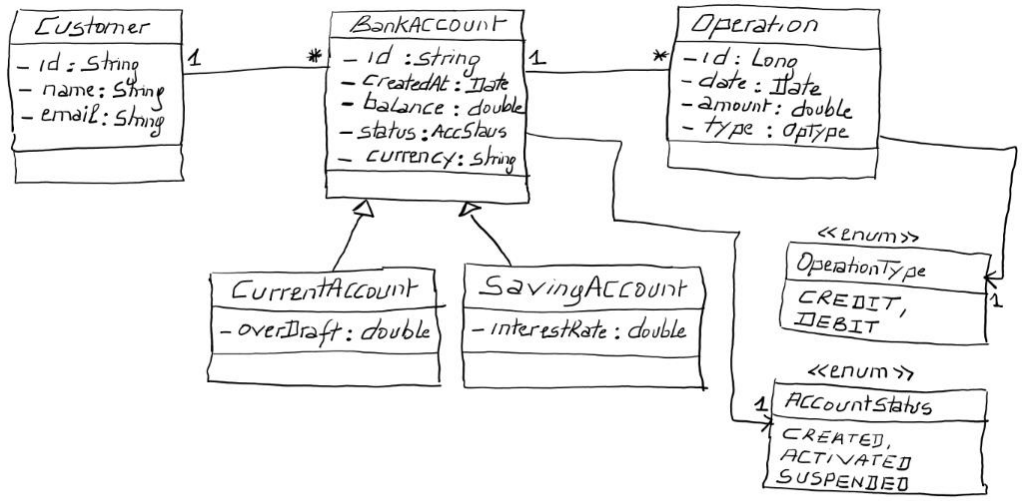
Enfin, nous avons la couche Web, qui repose sur le framework Spring MVC. Cette couche est responsable de la gestion des requêtes HTTP, de l'interaction avec les services métier et de la fourniture des réponses appropriées aux clients. Le Frontend de notre application peut être basé sur Angular ou Thymeleaf, deux frameworks populaires pour le développement d'interfaces utilisateur Web.

En résumé, l'architecture en trois couches que nous allons étudier dans cet examen nous permet de séparer les préoccupations et de créer une application modulaire et bien structurée. La couche DAO gère l'accès aux données, la couche métier gère la logique métier, la couche DTO facilite le transfert de données, et la couche Web gère les interactions avec les clients. Cette architecture favorise la réutilisabilité du code, la maintenabilité et la scalabilité de notre application.

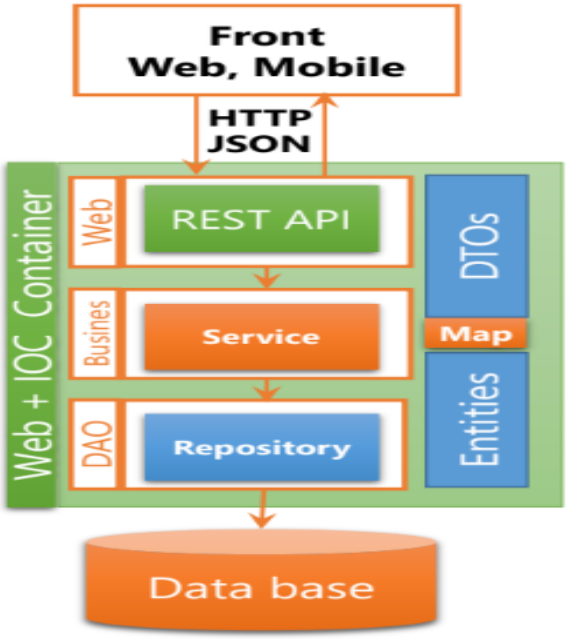
Énoncé :

On souhaite créer une application qui permet de gérer des comptes bancaires. Chaque compte appartient à un client. Un compte peut subir plusieurs opérations de type DEBIT ou CREDIT. Il existe deux types de comptes : Comptes courants et comptes épargnes.

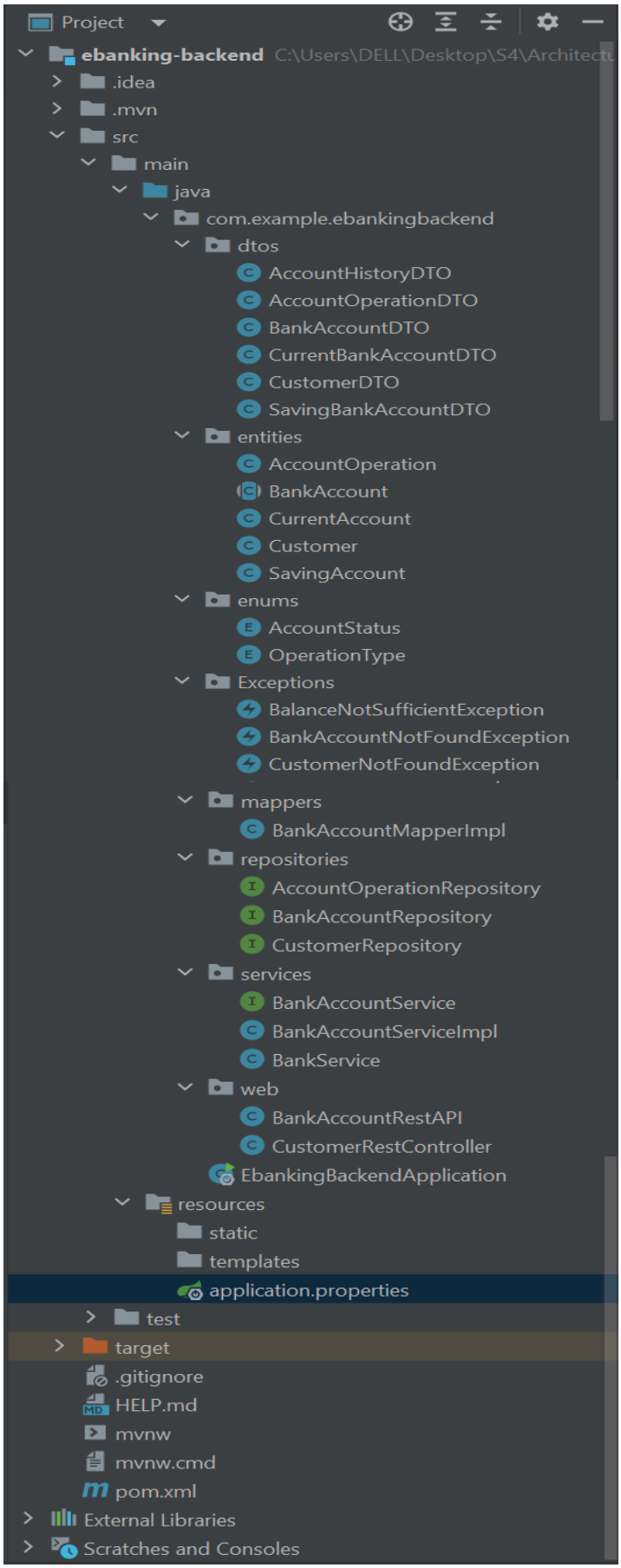
Diagramme de classe :



Architecture du projet :



Structure du projet :



Partie 1 : Couche DAO :

- Créer les entités JPA : Customer, BankAccount, Saving Account, CurrentAccount, AccountOperation

Customer :

```
package com.example.ebankingbackend.entities;

import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Customer {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @OneToMany(mappedBy = "customer")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<BankAccount> bankAccounts;
}
```

BankAccount :

```
package com.example.ebankingbackend.entities;

import com.example.ebankingbackend.enums.AccountStatus;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;
import java.util.List;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TYPE",length = 4)
@Data @NoArgsConstructor @AllArgsConstructor
public abstract class BankAccount {
    @Id
    private String id;
    private double balance; //solde
    private Date createdAt;
    @Enumerated(EnumType.STRING)
    private AccountStatus status;
    @ManyToOne
    private Customer customer;
    @OneToMany(mappedBy="bankAccount",fetch = FetchType.LAZY)
    private List<AccountOperation> accountOperations;

}
```

Saving Account :

```
package com.example.ebankingbackend.entities;


import jakarta.persistence.DiscriminatorValue;
import jakarta.persistence.Entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@DiscriminatorValue("SA")
@Data @NoArgsConstructor @AllArgsConstructor
public class SavingAccount extends BankAccount{
    private double interestRate;
}
```

CurrentAccount :

```
package com.example.ebankingbackend.entities;

import jakarta.persistence.DiscriminatorValue;
import jakarta.persistence.Entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@DiscriminatorValue("CA")
@Data
@NoArgsConstructor
@AllArgsConstructor

public class CurrentAccount extends  BankAccount{
    private double overDraft;
}
```

AccountOperation :

```
package com.example.ebankingbackend.entities;

import com.example.ebankingbackend.enums.OperationType;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor

public class AccountOperation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date operationDate;
```

```
    private double amount;
    @Enumerated(EnumType.STRING)
    private OperationType type;
    @ManyToOne
    private BankAccount bankAccount;

    private String description;
}
```

enums :

AccountStatus :

```
package com.example.ebankingbackend.enums;

public enum AccountStatus {
    CREATED,ACTIVATED,SUSPENDED
}
```

OperationType :

```
package com.example.ebankingbackend.enums;

public enum OperationType {
    DEBIT, CREDIT
}
```

- Créer les interfaces JPA Repository basées sur Spring Data

AccountOperationRepository :

```
package com.example.ebankingbackend.repositories;

import com.example.ebankingbackend.entities.AccountOperation;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface AccountOperationRepository extends
JpaRepository<AccountOperation,Long> {
    List<AccountOperation> findByBankAccountId(String accountId);
    Page<AccountOperation> findByBankAccountId(String accountId, Pageable
pageable);

}
```

BankAccountRepository :

```
package com.example.ebankingbackend.repositories;

import com.example.ebankingbackend.entities.BankAccount;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BankAccountRepository extends
JpaRepository<BankAccount,String> {
}
```

CustomerRepository :

```
package com.example.ebankingbackend.repositories;

import com.example.ebankingbackend.entities.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface CustomerRepository extends JpaRepository<Customer,Long>
{
   @Query("select c from Customer c where c.name like :kw")
    List<Customer> searchCustomer(@Param(value="kw") String keyword);
}
```

- Tester la couche DAO

EbankingBackendAppliaction :

```
package com.example.ebankingbackend;

import com.example.ebankingbackend.Exceptions.CustomerNotFoundException;
import com.example.ebankingbackend.dtos.BankAccountDTO;
import com.example.ebankingbackend.dtos.CurrentBankAccountDTO;
import com.example.ebankingbackend.dtos.CustomerDTO;
import com.example.ebankingbackend.dtos.SavingBankAccountDTO;
import com.example.ebankingbackend.entities.*;
import com.example.ebankingbackend.enums.AccountStatus;
import com.example.ebankingbackend.enums.OperationType;
import
com.example.ebankingbackend.repositories.AccountOperationRepository;
import com.example.ebankingbackend.repositories.BankAccountRepository;
import com.example.ebankingbackend.repositories.CustomerRepository;
import com.example.ebankingbackend.services.BankAccountService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Date;
import java.util.List;
import java.util.UUID;
import java.util.stream.Stream;

@SpringBootApplication
public class EbankingBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(EbankingBackendApplication.class, args);
    }

@Bean
CommandLineRunner start(CustomerRepository customerRepository,
                        BankAccountRepository bankAccountRepository,
                        AccountOperationRepository
accountOperationRepository) {

    return args -> {
```

```
        Stream.of("Hassan", "Yassine", "Aicha").forEach(name -> {
            Customer customer = new Customer();
            customer.setName(name);
            customer.setEmail(name + "@gmail.com");
            customerRepository.save(customer);
        });
        customerRepository.findAll().forEach(customer -> {
            CurrentAccount currentAccount = new CurrentAccount();
            currentAccount.setId(UUID.randomUUID().toString());
            currentAccount.setBalance(Math.random() * 90000);
            currentAccount.setCreatedAt(new Date());
            currentAccount.setStatus(AccountStatus.CREATED);
            currentAccount.setCustomer(customer);
            currentAccount.setOverDraft(9000);
            bankAccountRepository.save(currentAccount);

            SavingAccount savingAccount = new SavingAccount();
            savingAccount.setId(UUID.randomUUID().toString());
            savingAccount.setBalance(Math.random() * 90000);
            savingAccount.setCreatedAt(new Date());
            savingAccount.setStatus(AccountStatus.CREATED);
            savingAccount.setCustomer(customer);
            savingAccount.setInterestRate((5.5));
            bankAccountRepository.save(savingAccount);
        });
        bankAccountRepository.findAll().forEach(bankAccount -> {

            for (int i = 0; i < 10; i++) {
                AccountOperation accountOperation = new
AccountOperation();
                accountOperation.setOperationDate(new Date());
                accountOperation.setAmount(Math.random() * 12000);
                accountOperation.setType(Math.random() > 0.5 ?
OperationType.DEBIT : OperationType.CREDIT);
                accountOperation.setBankAccount(bankAccount);
                accountOperationRepository.save(accountOperation);
            }
        });
    };

}}
```

Application.properties :

```
server.port=8085
spring.datasource.url=jdbc:mysql://localhost:3306/E-
BANK?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDial
ect
spring.jpa.show-sql=true
```

Contenant le mot :

Table	Action	Lignes	Type	Interclassement	Taille
account_operation	Parcourir Structure Rechercher Insérer Vider Supprimer	60	InnoDB	utf8mb4_general_ci	32,0 kio
bank_account	Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8mb4_general_ci	32,0 kio
customer	Parcourir Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8mb4_general_ci	16,0 kio
3 tables	Somme	69	InnoDB	utf8mb4_general_ci	80,0 kio

Tout cocher Avec la sélection :

```
SELECT * FROM `customer`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette table Trier par clé : Aucun(e)

Options supplémentaires

	id	email	name
Éditer Copier Supprimer	1	Hassan@gmail.com	Hassan
Éditer Copier Supprimer	2	Yassine@gmail.com	Yassine
Éditer Copier Supprimer	3	Aicha@gmail.com	Aicha

```
SELECT * FROM `account_operation`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

1 > >> | Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette table Trier par clé : Aucun(e)

Options supplémentaires

	amount	id	operation_date	bank_account_id	description	type
Éditer Copier Supprimer	11844.520437274627	1	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	4727.647678227155	2	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	9504.291791364729	3	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	DEBIT
Éditer Copier Supprimer	11786.0548290271	4	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	4929.620275250056	5	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	5743.238983545548	6	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	DEBIT
Éditer Copier Supprimer	5828.617509994608	7	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	2691.3716321548236	8	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	11738.613402685925	9	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	CREDIT
Éditer Copier Supprimer	9897.265925199126	10	2023-05-29 19:54:53.000000	08553a8f-d0e0-4e20-8141-55350b4e2111	NULL	DEBIT
Éditer Copier Supprimer	6718.1044958432885	11	2023-05-29 19:54:53.000000	2fa020d9-234c-4ff3-99be-7a2e8b80c3fb	NULL	CREDIT

```
SELECT * FROM `bank_account`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette table Trier par clé : Aucun(e)

Options supplémentaires

←T→	balance	interest_rate	over_draft	type	created_at	customer_id	id	status
Éditer Copier Supprimer	6445.868319342814	5.5	NULL	SA	2023-05-29 19:54:53.000000	2	08553a8f-d0e0-4e20-8141-55350b4e2111	CREATED
Éditer Copier Supprimer	62914.84857786634	NULL	9000	CA	2023-05-29 19:54:53.000000	1	2fa020d9-234c-4ff3-99be-7a2e8b80c3fb	CREATED
Éditer Copier Supprimer	88793.35772806664	NULL	9000	CA	2023-05-29 19:54:53.000000	3	30f3e064-c1b7-4b80-a503-88cba0f8b344	CREATED
Éditer Copier Supprimer	65578.57001250166	5.5	NULL	SA	2023-05-29 19:54:53.000000	3	5700b415-3807-416c-aaea-c20adb969405	CREATED
Éditer Copier Supprimer	54430.989538263384	NULL	9000	CA	2023-05-29 19:54:53.000000	2	8dfe2b11-652b-4105-a071-1cbf95036cb7	CREATED
Éditer Copier Supprimer	83833.27201599532	5.5	NULL	SA	2023-05-29 19:54:53.000000	1	fbe8742f-22a4-4a33-ac1d-fd044ffbf1e3	CREATED

Partie 2 : Couche services, DTO et mappers

Couche services :

Interface BankAccountService :

```java
package com.example.ebankingbackend.services;

import
com.example.ebankingbackend.Exceptions.BalanceNotSufficientException;
import
com.example.ebankingbackend.Exceptions.BankAccountNotFoundException;
import com.example.ebankingbackend.Exceptions.CustomerNotFoundException;
import com.example.ebankingbackend.dtos.*;


import java.util.List;

public interface BankAccountService {
    CustomerDTO saveCustomer(CustomerDTO customerDTO);
    CurrentBankAccountDTO saveCurrentBankAccount(double initialBalance,
double overDraft, Long customerId) throws CustomerNotFoundException;
    SavingBankAccountDTO saveSavingBankAccount(double initialBalance,
double interestRate, Long customerId) throws CustomerNotFoundException;
    List<CustomerDTO> listCustomers();
    BankAccountDTO getBankAccount(String accountId) throws
BankAccountNotFoundException;
    void debit(String accountId,double amount,String description) throws
BankAccountNotFoundException, BalanceNotSufficientException;
    void credit(String accountId, double amount,String description)
throws BankAccountNotFoundException;
    void transfer(String accountIdSource,String
accountIdDestination,double amount) throws BankAccountNotFoundException,
BalanceNotSufficientException;

    List<BankAccountDTO> bankAccountList();

    CustomerDTO getCustomer(Long customerId) throws
CustomerNotFoundException;

    CustomerDTO updateCustomer(CustomerDTO customerDTO);
```

```
    void deleteCustomer(Long customerId);

    List<AccountOperationDTO> accountHistory(String accountId);

    AccountHistoryDTO getAccountHistory(String accountId, int page, int
size) throws BankAccountNotFoundException;

    List<CustomerDTO> searchCustomers(String keyword);
}
```

Implémentation de l'interface BankAccountServiceImpl :

```
package com.example.ebankingbackend.services;

import
com.example.ebankingbackend.Exceptions.BalanceNotSufficientException;
import
com.example.ebankingbackend.Exceptions.BankAccountNotFoundException;
import com.example.ebankingbackend.Exceptions.CustomerNotFoundException;
import com.example.ebankingbackend.dtos.*;
import com.example.ebankingbackend.entities.*;
import com.example.ebankingbackend.enums.OperationType;
import com.example.ebankingbackend.mappers.BankAccountMapperImpl;
import
com.example.ebankingbackend.repositories.AccountOperationRepository;
import com.example.ebankingbackend.repositories.BankAccountRepository;
import com.example.ebankingbackend.repositories.CustomerRepository;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Date;
import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;


@Service
@Transactional
@AllArgsConstructor
@Slf4j
public class BankAccountServiceImpl implements BankAccountService {

    private CustomerRepository customerRepository;
    private BankAccountRepository bankAccountRepository;
    private AccountOperationRepository accountOperationRepository;

    private BankAccountMapperImpl dtoMapper;



   // Logger log= LoggerFactory.getLogger(this.getClass().getName());

    @Override
    public CustomerDTO saveCustomer(CustomerDTO customerDTO) {
        log.info("Saving new Customer");
```

```
        Customer customer=dtoMapper.fromcustomerDTO(customerDTO);
        Customer savedCustomer=customerRepository.save(customer);
        return dtoMapper.fromcustomer(savedCustomer);
    }

    @Override
    public CurrentBankAccountDTO saveCurrentBankAccount(double
initialBalance, double overDraft, Long customerId) throws
CustomerNotFoundException {
        Customer
customer=customerRepository.findById(customerId).orElse(null);
        if(customer==null)
            throw new CustomerNotFoundException("Customer not found");
        CurrentAccount currentAccount=new CurrentAccount();
        currentAccount.setId(UUID.randomUUID().toString());
        currentAccount.setCreatedAt(new Date());
        currentAccount.setBalance(initialBalance);
        currentAccount.setOverDraft(overDraft);
        currentAccount.setCustomer(customer);
        CurrentAccount
savedBankAccount=bankAccountRepository.save(currentAccount);
        return dtoMapper.fromCurrentBankAccount(savedBankAccount);
    }

    @Override
    public SavingBankAccountDTO saveSavingBankAccount(double
initialBalance, double interestRate, Long customerId) throws
CustomerNotFoundException {
        Customer
customer=customerRepository.findById(customerId).orElse(null);
        if(customer==null)
            throw new CustomerNotFoundException("Customer not found");
        SavingAccount savingAccount=new SavingAccount();
        savingAccount.setId(UUID.randomUUID().toString());
        savingAccount.setCreatedAt(new Date());
        savingAccount.setBalance(initialBalance);
        savingAccount.setInterestRate(interestRate);
        savingAccount.setCustomer(customer);
        SavingAccount
savedBankAccount=bankAccountRepository.save(savingAccount);
        return dtoMapper.fromSavingBankAccount(savedBankAccount);

    }


    @Override
    public List<CustomerDTO> listCustomers() {
        List<Customer> customers=customerRepository.findAll();
        List<CustomerDTO> customerDTOS= customers.stream().map(customer -
> dtoMapper.fromcustomer(customer))
                .collect((Collectors.toList()));
        return customerDTOS;
    }

    @Override
    public BankAccountDTO getBankAccount(String accountId) throws
BankAccountNotFoundException {
        BankAccount bankAccount=bankAccountRepository.findById(accountId)
                .orElseThrow(()->new
BankAccountNotFoundException("BankAccount not found"));
            if(bankAccount instanceof SavingAccount){
```

```
                SavingAccount savingAccount=(SavingAccount) bankAccount;
                return dtoMapper.fromSavingBankAccount(savingAccount);
            }else{
                CurrentAccount currentAccount=(CurrentAccount)
bankAccount;
                return dtoMapper.fromCurrentBankAccount(currentAccount);
            }
    }

    @Override
    public void debit(String accountId, double amount, String
description) throws BankAccountNotFoundException,
BalanceNotSufficientException {
        BankAccount bankAccount=bankAccountRepository.findById(accountId)
                .orElseThrow(()->new
BankAccountNotFoundException("BankAccount not found"));
         if (bankAccount.getBalance()<amount)
             throw new BalanceNotSufficientException("Balance not
sufficient");
        AccountOperation accountOperation=new AccountOperation();
        accountOperation.setType(OperationType.DEBIT);
        accountOperation.setAmount(amount);
        accountOperation.setDescription(description);
        accountOperation.setOperationDate(new Date());
        accountOperation.setBankAccount(bankAccount);
        accountOperationRepository.save(accountOperation);
        bankAccount.setBalance(bankAccount.getBalance()-amount);
        bankAccountRepository.save(bankAccount);
    }

    @Override
    public void credit(String accountId, double amount, String
description) throws BankAccountNotFoundException {
        BankAccount bankAccount=bankAccountRepository.findById(accountId)
                .orElseThrow(()->new
BankAccountNotFoundException("BankAccount not found"));
AccountOperation accountOperation=new AccountOperation();
        accountOperation.setType(OperationType.CREDIT);
        accountOperation.setAmount(amount);
        accountOperation.setDescription(description);
        accountOperation.setOperationDate(new Date());
        accountOperation.setBankAccount(bankAccount);
        accountOperationRepository.save(accountOperation);
        bankAccount.setBalance(bankAccount.getBalance()+amount);
        bankAccountRepository.save(bankAccount);
    }

    @Override
    public void transfer(String accountIdSource, String
accountIdDestination, double amount) throws BankAccountNotFoundException,
BalanceNotSufficientException {
        debit(accountIdSource,amount,"Transfer to
"+accountIdDestination);
        credit(accountIdDestination,amount,"Transfer from
"+accountIdSource);
    }

    @Override
    public List<BankAccountDTO> bankAccountList() {
        List<BankAccount> bankAccounts=bankAccountRepository.findAll();
        List<BankAccountDTO> bankAccountDTOS=
```

```
bankAccounts.stream().map(bankAccount -> {
            if(bankAccount instanceof SavingAccount){
                SavingAccount savingAccount=(SavingAccount) bankAccount;
                return dtoMapper.fromSavingBankAccount(savingAccount);
            }else{
                CurrentAccount currentAccount=(CurrentAccount)
bankAccount;
                return dtoMapper.fromCurrentBankAccount(currentAccount);
        }
        }).collect(Collectors.toList());
        return bankAccountDTOS;
    }

    @Override
    public CustomerDTO getCustomer(Long customerId) throws
CustomerNotFoundException {
        Customer customer=customerRepository.findById(customerId)
                .orElseThrow(()->new CustomerNotFoundException("Customer
Not found"));
        return dtoMapper.fromcustomer(customer);
    }

    @Override
    public CustomerDTO updateCustomer(CustomerDTO customerDTO) {
        log.info("Saving new Customer");
        Customer customer=dtoMapper.fromcustomerDTO(customerDTO);
        Customer savedCustomer=customerRepository.save(customer);
        return dtoMapper.fromcustomer(savedCustomer);
    }

    @Override
    public void deleteCustomer(Long customerId){
        customerRepository.deleteById(customerId);
    }

   @Override
    public List<AccountOperationDTO> accountHistory(String accountId){
        List<AccountOperation> accountOperations=
accountOperationRepository.findByBankAccountId(accountId);
        return accountOperations.stream().map(op-
>dtoMapper.fromAccountOpeartion(op)).collect(Collectors.toList());
    }

    @Override
    public AccountHistoryDTO getAccountHistory(String accountId, int
page, int size) throws BankAccountNotFoundException {
        BankAccount
bankAccount=bankAccountRepository.findById(accountId).orElse(null);
        if(bankAccount==null) throw new
BankAccountNotFoundException("Account not Found");
        Page<AccountOperation> accountOperations=
accountOperationRepository.findByBankAccountId(accountId,
PageRequest.of(page,size));
        AccountHistoryDTO accountHistoryDTO=new AccountHistoryDTO();
        List<AccountOperationDTO> accountOperationDTOS=
accountOperations.getContent().stream().map(op-
>dtoMapper.fromAccountOpeartion(op)).collect(Collectors.toList());
        accountHistoryDTO.setAccountOperationDTOS(accountOperationDTOS);
        accountHistoryDTO.setAccountId(bankAccount.getId());
        accountHistoryDTO.setBalance(bankAccount.getBalance());
        accountHistoryDTO.setCurrentPage(page);
```

```
        accountHistoryDTO.setPageSize(size);

accountHistoryDTO.setTotalPages(accountOperations.getTotalPages());

        return accountHistoryDTO;
    }

    @Override
    public List<CustomerDTO> searchCustomers(String keyword) {
        List<Customer>
customers=customerRepository.searchCustomer(keyword);
        List<CustomerDTO> customerDTOS=customers.stream().map(customer ->
dtoMapper.fromcustomer(customer)).collect(Collectors.toList());


        return customerDTOS;
    }

}
```

DTO :

CustomerDTO :

```
package com.example.ebankingbackend.dtos;


import lombok.Data;


@Data
public class CustomerDTO {
    private Long id;
    private String name;
    private String email;

}
```

BankAccountDTO :

```
package com.example.ebankingbackend.dtos;

import lombok.Data;

@Data
public class BankAccountDTO {
    private String type;
}
```

CurrentBankAccountDTO :

```
package com.example.ebankingbackend.dtos;


import com.example.ebankingbackend.enums.AccountStatus;
import lombok.Data;

import java.util.Date;
```

```
@Data
public class CurrentBankAccountDTO extends BankAccountDTO{

        private String id;
        private double balance; //solde
        private Date createdAt;
        private AccountStatus status;
        private CustomerDTO customerDTO;
        private double overDraft;

}
```

SavingBankAccountDTO :

```
package com.example.ebankingbackend.dtos;


import com.example.ebankingbackend.enums.AccountStatus;
import lombok.Data;
import java.util.Date;

@Data
public class SavingBankAccountDTO extends BankAccountDTO{

        private String id;
        private double balance; //solde
        private Date createdAt;
        private AccountStatus status;
        private CustomerDTO customerDTO;
        private double interestRate;

}
```

AccountOperationDTO :

```
package com.example.ebankingbackend.dtos;


import com.example.ebankingbackend.enums.OperationType;

import lombok.Data;


import java.util.Date;


@Data

public class AccountOperationDTO {
    private Long id;
    private Date operationDate;
    private double amount;
    private OperationType type;
    private String description;
}
```

AccountHistoryDTO :

```
package com.example.ebankingbackend.dtos;

import lombok.Data;

import java.util.List;

@Data
public class AccountHistoryDTO {
    private String accountId;
    private double balance;

    private int currentPage;
    private int totalPages;
    private int pageSize;
    private List<AccountOperationDTO> accountOperationDTOS;
}
```

Mappers :

BankAccountMapperImpl :

```
package com.example.ebankingbackend.mappers;

import com.example.ebankingbackend.dtos.AccountOperationDTO;
import com.example.ebankingbackend.dtos.CurrentBankAccountDTO;
import com.example.ebankingbackend.dtos.CustomerDTO;
import com.example.ebankingbackend.dtos.SavingBankAccountDTO;
import com.example.ebankingbackend.entities.AccountOperation;
import com.example.ebankingbackend.entities.CurrentAccount;
import com.example.ebankingbackend.entities.Customer;
import com.example.ebankingbackend.entities.SavingAccount;
import org.springframework.beans.BeanUtils;
import org.springframework.stereotype.Service;

@Service
public class BankAccountMapperImpl {
    public CustomerDTO fromcustomer(Customer customer){
        CustomerDTO customerDTO=new CustomerDTO();
        customerDTO.setId(customer.getId());
        customerDTO.setName(customer.getName());
        customerDTO.setEmail(customer.getEmail());
        return customerDTO;
    }
    public Customer fromcustomerDTO(CustomerDTO customerDTO){
        Customer customer=new Customer();
        BeanUtils.copyProperties(customerDTO,customer);
        return customer;
    }

    public SavingBankAccountDTO fromSavingBankAccount(SavingAccount
savingAccount){

        SavingBankAccountDTO savingBankAccountDTO=new
SavingBankAccountDTO();
        BeanUtils.copyProperties(savingAccount,savingBankAccountDTO);

savingBankAccountDTO.setCustomerDTO(fromcustomer(savingAccount.getCustome
r()));
```

```
savingBankAccountDTO.setType(savingAccount.getClass().getSimpleName());
        return savingBankAccountDTO;
    }
    public SavingAccount fromSavingBankAccountDTO(SavingBankAccountDTO
savingBankAccountDTO){
        SavingAccount savingAccount=new SavingAccount();
        BeanUtils.copyProperties(savingBankAccountDTO,savingAccount);

savingAccount.setCustomer(fromcustomerDTO(savingBankAccountDTO.getCustome
rDTO()));
        return savingAccount;
    }

    public CurrentBankAccountDTO fromCurrentBankAccount(CurrentAccount
currentAccount){
        CurrentBankAccountDTO currentBankAccountDTO=new
CurrentBankAccountDTO();
        BeanUtils.copyProperties(currentAccount,currentBankAccountDTO);

currentBankAccountDTO.setCustomerDTO(fromcustomer(currentAccount.getCusto
mer()));

currentBankAccountDTO.setType(currentAccount.getClass().getSimpleName());
        return currentBankAccountDTO;
    }
    public CurrentAccount fromCurrentBankAccountDTO(CurrentBankAccountDTO
currentBankAccountDTO){
        CurrentAccount currentAccount=new CurrentAccount();
        BeanUtils.copyProperties(currentBankAccountDTO,currentAccount);

currentAccount.setCustomer(fromcustomerDTO(currentBankAccountDTO.getCusto
merDTO()));
        return currentAccount;
    }

    public AccountOperationDTO fromAccountOpeartion(AccountOperation
accountOperation){
        AccountOperationDTO accountOperationDTO=new
AccountOperationDTO();
        BeanUtils.copyProperties(accountOperation,accountOperationDTO);
        return accountOperationDTO;
    }

}
```

Exception :

BalanceNotSufficientException :

```
package com.example.ebankingbackend.Exceptions;

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

BankAccountNotFoundException :

```
package com.example.ebankingbackend.Exceptions;

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

CustomerNotFoundException :

```
package com.example.ebankingbackend.Exceptions;

public class CustomerNotFoundException extends Exception {
    public CustomerNotFoundException(String message) {

        super(message);
    }
}
```

Partie 3 : Couche Web (RestControllers)

BankAccountRestAPI :

```
package com.example.ebankingbackend.web;


import
com.example.ebankingbackend.Exceptions.BankAccountNotFoundException;
import com.example.ebankingbackend.dtos.AccountHistoryDTO;
import com.example.ebankingbackend.dtos.AccountOperationDTO;
import com.example.ebankingbackend.dtos.BankAccountDTO;
import com.example.ebankingbackend.services.BankAccountService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@CrossOrigin("*")

public class BankAccountRestAPI {
   private BankAccountService bankAccountService;

   public BankAccountRestAPI(BankAccountService bankAccountService){
       this.bankAccountService=bankAccountService;
   }

   @GetMapping("/accounts/{accountId}")
   public BankAccountDTO getBankAccount(@PathVariable String accountId)
throws BankAccountNotFoundException{
       return bankAccountService.getBankAccount(accountId);
   }

   @GetMapping("/accounts")
    public List<BankAccountDTO> listAccounts(){
       return bankAccountService.bankAccountList();
   }

    @GetMapping("/accounts/{accountId}/operations")
    public List<AccountOperationDTO> getHistory(@PathVariable String
```

```
accountId){
        return bankAccountService.accountHistory(accountId);
    }

    @GetMapping("/accounts/{accountId}/pageOperations")
    public AccountHistoryDTO getAccountHistory(
            @PathVariable String accountId,
            @RequestParam(name="page",defaultValue = "0") int page,
            @RequestParam(name="size",defaultValue = "5") int size)
throws BankAccountNotFoundException {
        return bankAccountService.getAccountHistory(accountId,page,size);
    }


}
```

CustomerRestController :

```
package com.example.ebankingbackend.web;


import com.example.ebankingbackend.Exceptions.CustomerNotFoundException;
import com.example.ebankingbackend.dtos.CustomerDTO;
import com.example.ebankingbackend.services.BankAccountService;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@AllArgsConstructor
@Slf4j
@CrossOrigin("*")
public class CustomerRestController {
    private BankAccountService bankAccountService;

    @GetMapping("/customers")
    public List<CustomerDTO> customers(){
        return bankAccountService.listCustomers();
    }

    @GetMapping("/customers/search")
    public List<CustomerDTO>
searchCustomers(@RequestParam(name="keyword", defaultValue = "")String
keyword){
        return bankAccountService.searchCustomers("%"+keyword+"%");
    }
    @GetMapping("/customers/{id}")
    public CustomerDTO getCustomer(@PathVariable(name="id") Long
customerId) throws CustomerNotFoundException {
        return bankAccountService.getCustomer(customerId);
    }

    @PutMapping("/customers")
    public CustomerDTO saveCustomer(@RequestBody CustomerDTO
customerDTO){
        return bankAccountService.saveCustomer(customerDTO);
    }

    @PutMapping("/customers/{customerId}")
```

```
    public CustomerDTO updateCustomer(@PathVariable Long
customerId,@RequestBody CustomerDTO customerDTO){
            customerDTO.setId(customerId);
            return bankAccountService.updateCustomer(customerDTO);
    }

    @DeleteMapping("/customers/{id}")
    public void deleteCustomer(@PathVariable Long id){
        bankAccountService.deleteCustomer(id);
    }
}
```

J'ai changé dans EbankingBanckendApplication :

```
package com.example.ebankingbackend;

import com.example.ebankingbackend.Exceptions.CustomerNotFoundException;
import com.example.ebankingbackend.dtos.BankAccountDTO;
import com.example.ebankingbackend.dtos.CurrentBankAccountDTO;
import com.example.ebankingbackend.dtos.CustomerDTO;
import com.example.ebankingbackend.dtos.SavingBankAccountDTO;
import com.example.ebankingbackend.entities.*;
import com.example.ebankingbackend.enums.AccountStatus;
import com.example.ebankingbackend.enums.OperationType;
import
com.example.ebankingbackend.repositories.AccountOperationRepository;
import com.example.ebankingbackend.repositories.BankAccountRepository;
import com.example.ebankingbackend.repositories.CustomerRepository;
import com.example.ebankingbackend.services.BankAccountService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Date;
import java.util.List;
import java.util.UUID;
import java.util.stream.Stream;

@SpringBootApplication
public class EbankingBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(EbankingBackendApplication.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(BankAccountService
bankAccountService) {

        return args -> {
            Stream.of("Hassan", "Yassine", "Aicha").forEach(name -> {
                CustomerDTO customer = new CustomerDTO();
                customer.setName(name);
                customer.setEmail(name + "@gmail.com");
                bankAccountService.saveCustomer(customer);
            });
            bankAccountService.listCustomers().forEach(customer -> {
                try {

bankAccountService.saveCurrentBankAccount(Math.random() * 90000, 9000,
```

```
customer.getId());

bankAccountService.saveSavingBankAccount(Math.random()*120000,5.5,
customer.getId());


                } catch (CustomerNotFoundException e) {
                    e.printStackTrace();
                }
            });
            List<BankAccountDTO> bankAccounts=
bankAccountService.bankAccountList();
            for (BankAccountDTO bankAccount:bankAccounts){
                for(int i=0;i<10;i++){
                    String accountId;
                    if(bankAccount instanceof SavingBankAccountDTO)
                    {
                        accountId=((SavingBankAccountDTO)
bankAccount).getId();
                    }else{
                        accountId=((CurrentBankAccountDTO)
bankAccount).getId();

                    }

bankAccountService.credit(accountId,1000+Math.random()*120000,"Credit");

bankAccountService.debit(accountId,1000+Math.random()*9000,"Debit");

                }}
        };


}
```

Résultats :

Swagger :

Swagger UI

localhost:8085/swagger-ui/index.html

Swagger

/v3/api-docs

Explore

OpenAPI definition v0 OAS3

/v3/api-docs

Servers

http://localhost:8085 - Generated server url

customer-rest-controller

- GET /customers
- PUT /customers
- PUT /customers/{customerId}
- GET /customers/{id}
- DELETE /customers/{id}
- GET /customers/search

bank-account-rest-api

- GET /accounts
- GET /accounts/{accountId}
- GET /accounts/{accountId}/pageOperations
- GET /accounts/{accountId}/operations

Schemas

- CustomerDTO
- BankAccountDTO
- AccountHistoryDTO
- AccountOperationDTO

customer-rest-controller

GET /customers

Parameters

Cancel

No parameters

Execute | Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8085/customers' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8085/customers
```

Server response

Code	Details
200	Response body


```
[
  {
    "id": 1,
    "name": "Hassan",
    "email": "Hassan@gmail.com"
  },
  {
    "id": 2,
    "name": "Yassine",
    "email": "Yassine@gmail.com"
  },
  {
    "id": 3,
    "name": "Aicha",
    "email": "Aicha@gmail.com"
  }
]
```


Download

Response headers

GET /accounts/{accountId}

Parameters

Cancel

Name	Description
accountId * required string (path)	08553a8f-d0e0-4e20-8141-55350b4e2111

Execute | Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8085/accounts/08553a8f-d0e0-4e20-8141-55350b4e2111' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8085/accounts/08553a8f-d0e0-4e20-8141-55350b4e2111
```

Server response

Code	Details
200	Response body

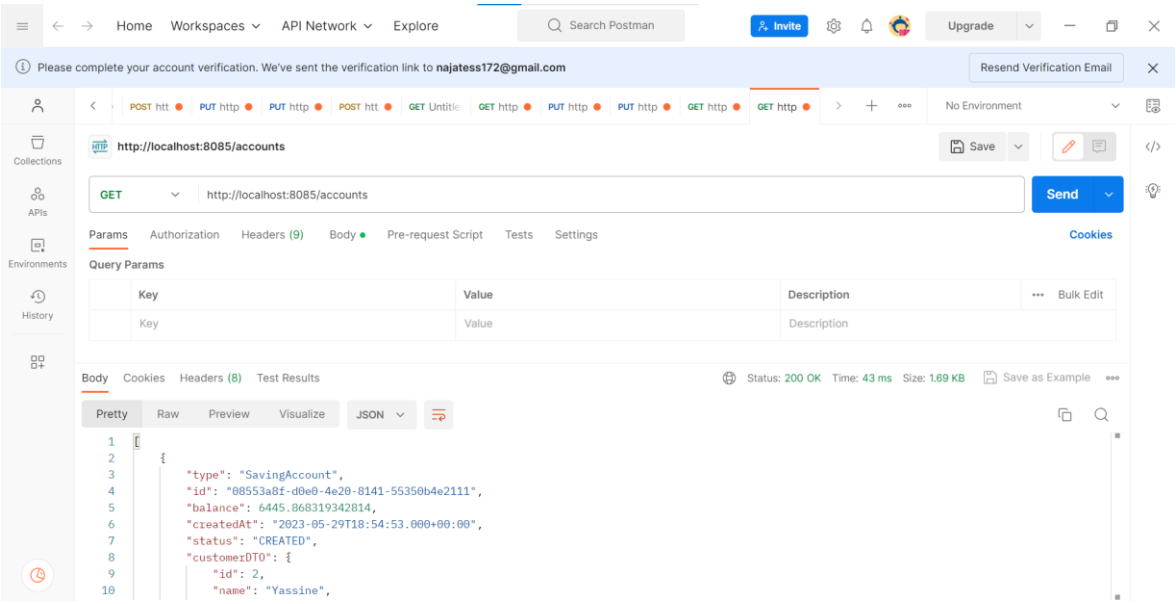

```
{
  "type": "SavingAccount",
  "id": "08553a8f-d0e0-4e20-8141-55350b4e2111",
  "balance": 6445.868319342814,
  "createdAt": "2023-05-29T18:54:53.000+00:00",
  "status": "CREATED",
  "customerDTO": {
    "id": 2,
    "name": "Yassine",
    "email": "Yassine@gmail.com"
  },
  "interestRate": 5.5
}
```

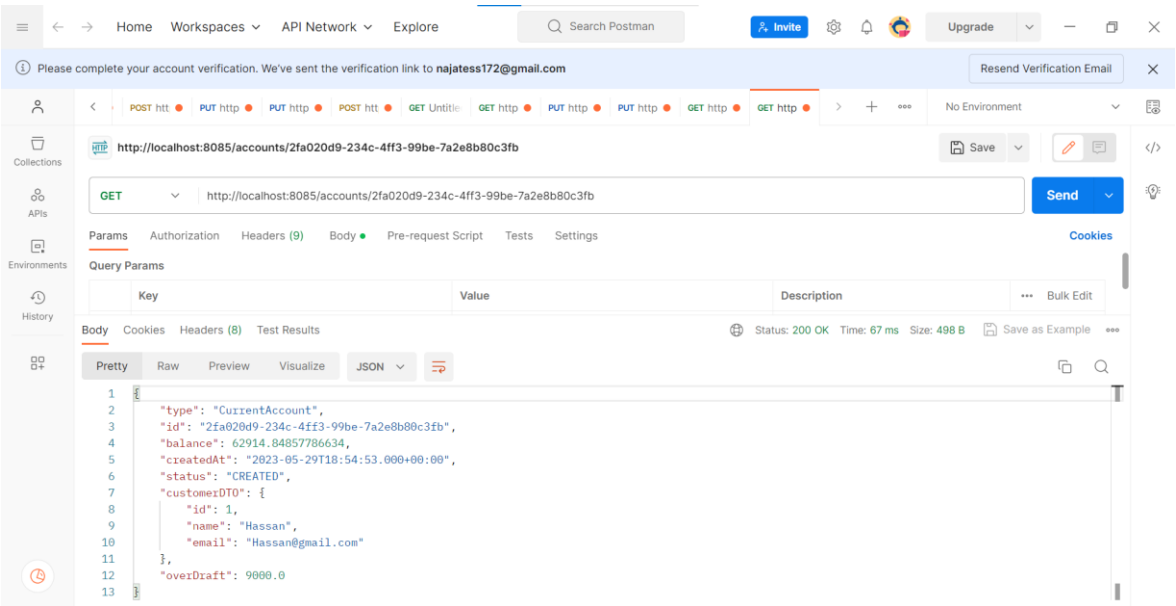

Download

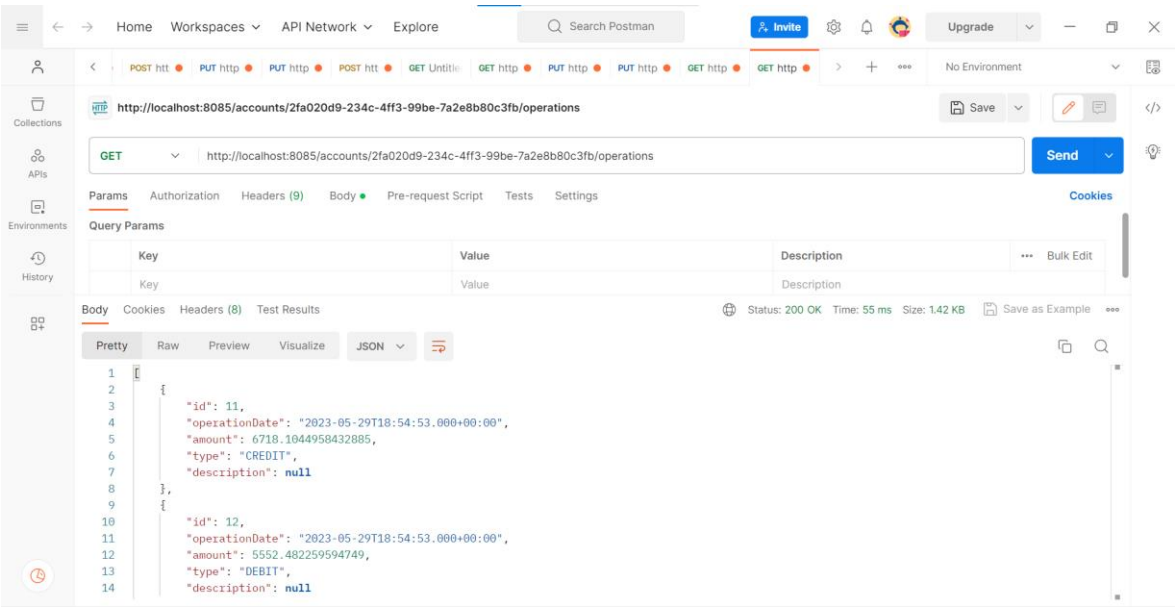
Response headers

Postman :

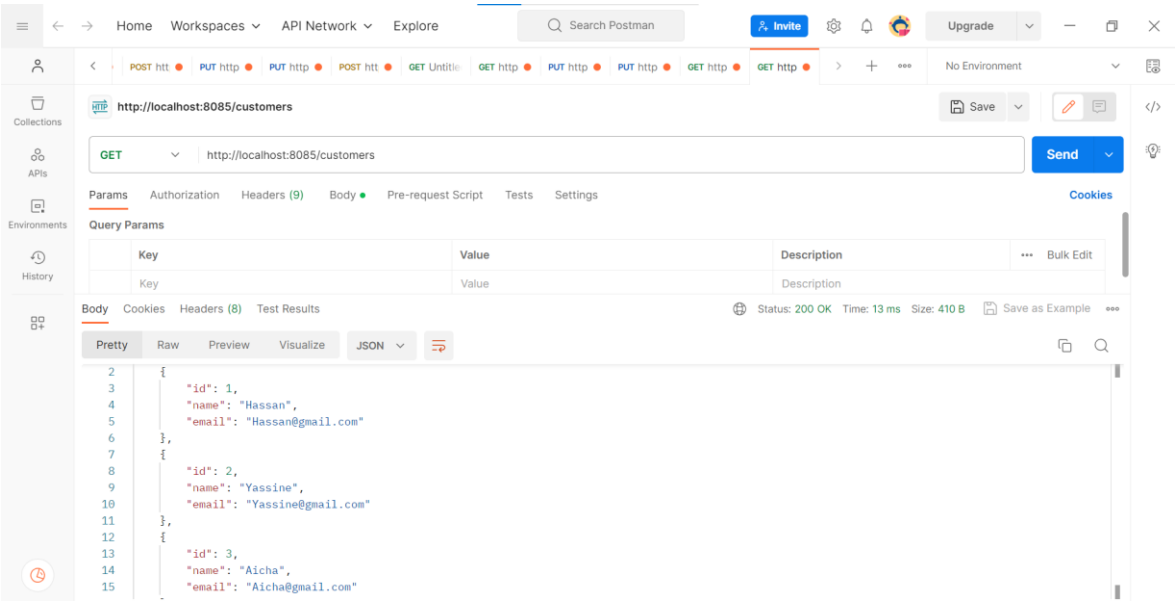
Bank Account

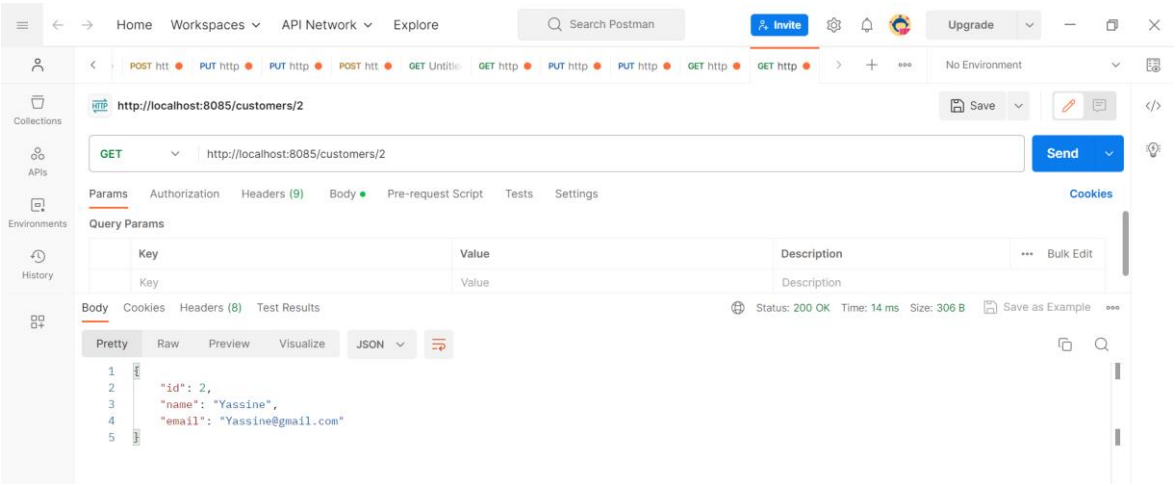


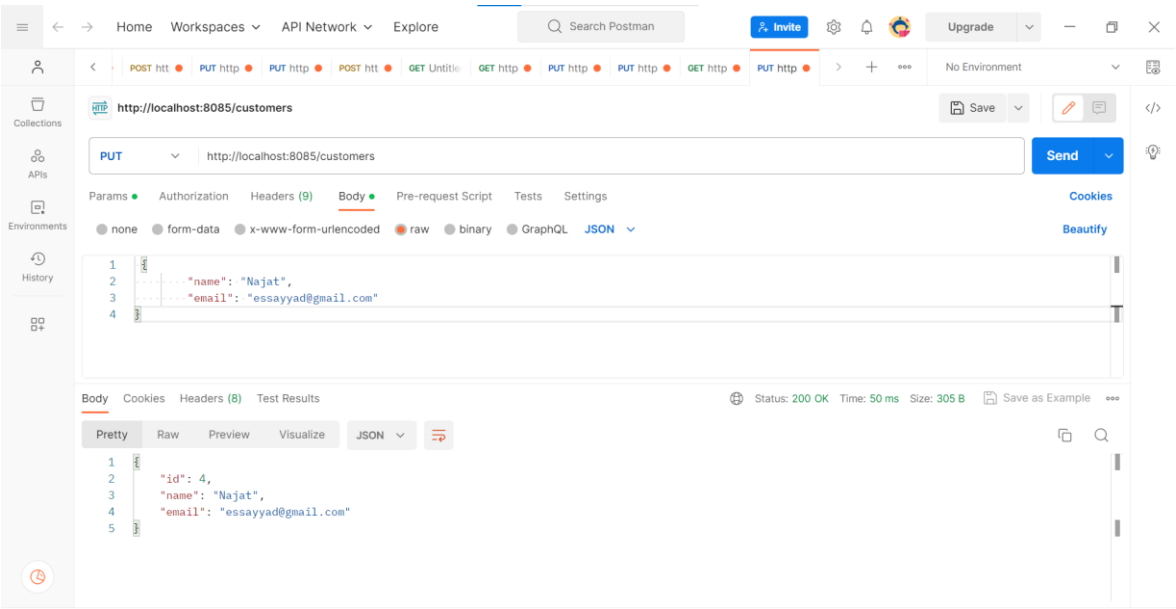


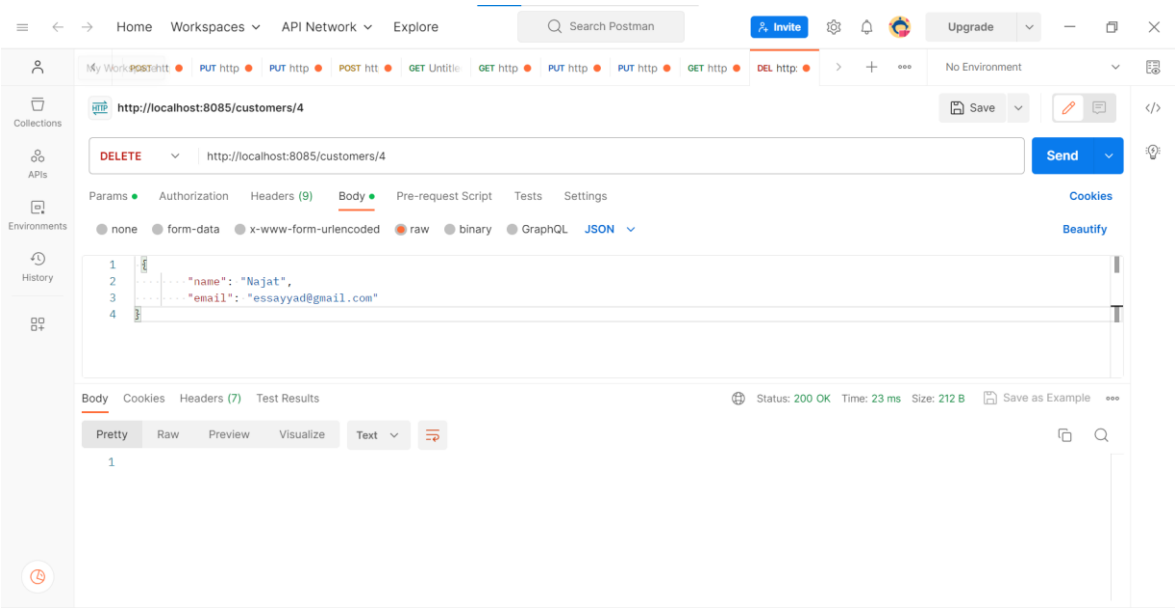

Home Workspaces API Network Explore
Search Postman
Invite
Upgrade
No Environment
Collections
APIs
Environments
History
http://localhost:8085/accounts/2fa020d9-234c-4ff3-99be-7a2e8b80c3fb/operations
Save
GET
http://localhost:8085/accounts/2fa020d9-234c-4ff3-99be-7a2e8b80c3fb/operations
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies
Query Params
Key Value Description Bulk Edit
Body Cookies Headers (8) Test Results
Status: 200 OK Time: 55 ms Size: 1.42 KB
Save as Example
Pretty Raw Preview Visualize JSON
[
{
"id": 11,
"operationDate": "2023-05-29T18:54:53.000+00:00",
"amount": 6718.1044958432885,
"type": "CREDIT",
"description": null
},
{
"id": 12,
"operationDate": "2023-05-29T18:54:53.000+00:00",
"amount": 5552.482259594749,
"type": "DEBIT",
"description": null

Customers :


Home Workspaces API Network Explore
Search Postman
Invite
Upgrade
No Environment
Collections
APIs
Environments
History
http://localhost:8085/customers
Save
GET
http://localhost:8085/customers
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies
Query Params
Key Value Description Bulk Edit
Body Cookies Headers (8) Test Results
Status: 200 OK Time: 13 ms Size: 410 B
Save as Example
Pretty Raw Preview Visualize JSON
{
"id": 1,
"name": "Hassan",
"email": "Hassan@gmail.com"
},
{
"id": 2,
"name": "Yassine",
"email": "Yassine@gmail.com"
},
{
"id": 3,
"name": "Aicha",
"email": "Aicha@gmail.com"

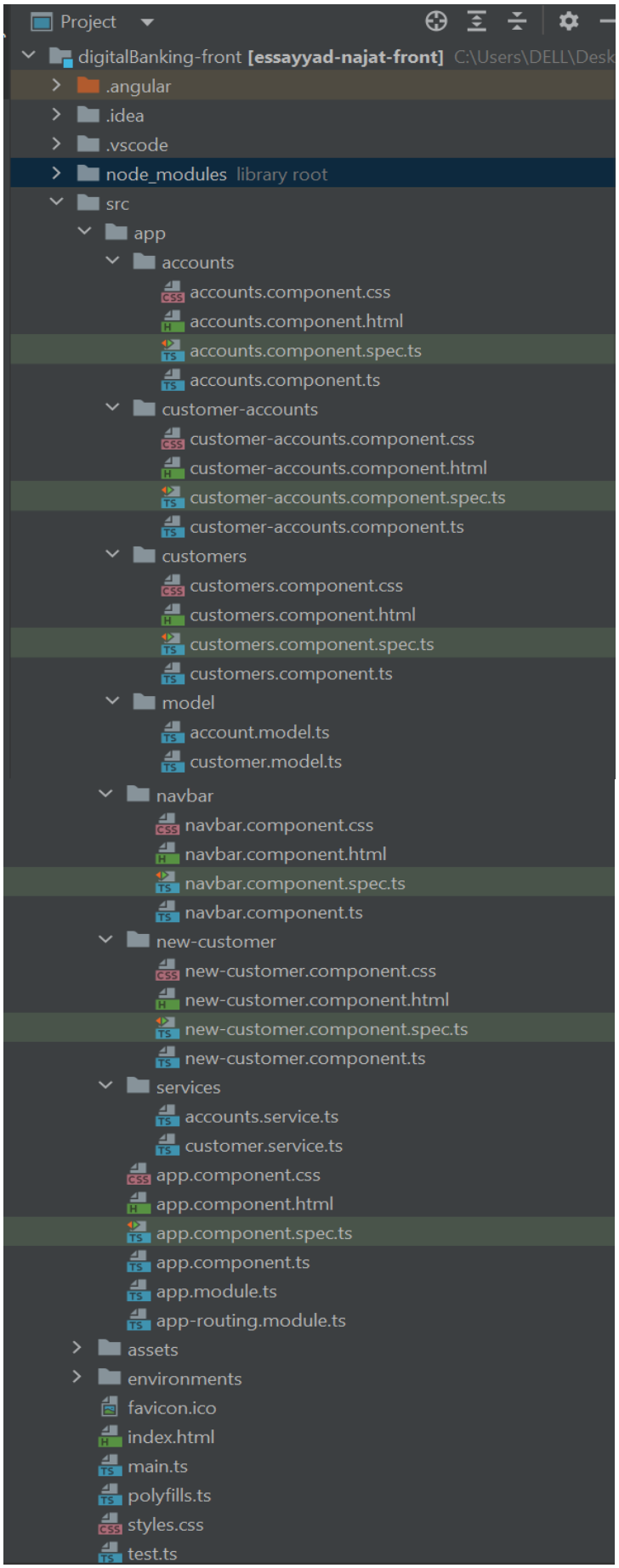

Home Workspaces API Network Explore
Search Postman
Invite
Upgrade
No Environment
Collections
APIs
Environments
History
http://localhost:8085/customers/2
Save
GET
http://localhost:8085/customers/2
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies
Query Params
Key Value Description Bulk Edit
Body Cookies Headers (8) Test Results
Status: 200 OK Time: 14 ms Size: 306 B
Save as Example
Pretty Raw Preview Visualize JSON
"id": 2,
"name": "Yassine",
"email": "Yassine@gmail.com"





Partie 4 : Frontend Angular

La structure du projet :


Project
digitalBanking-front [essayyad-najat-front] C:\Users\DELL\Desk
.angular
.idea
.vscode
node_modules library root
src
app
accounts
accounts.component.css
accounts.component.html
accounts.component.spec.ts
accounts.component.ts
customer-accounts
customer-accounts.component.css
customer-accounts.component.html
customer-accounts.component.spec.ts
customer-accounts.component.ts
customers
customers.component.css
customers.component.html
customers.component.spec.ts
customers.component.ts
model
account.model.ts
customer.model.ts
navbar
navbar.component.css
navbar.component.html
navbar.component.spec.ts
navbar.component.ts
new-customer
new-customer.component.css
new-customer.component.html
new-customer.component.spec.ts
new-customer.component.ts
services
accounts.service.ts
customer.service.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
app-routing.module.ts
assets
environments
favicon.ico
index.html
main.ts
polyfills.ts
styles.css
test.ts

accounts.component.html :

```
<div class="container mt-2">
  <div class="row">
    <div class="col col-md-6">
      <div class="card">
        <div class="card-header">Accounts</div>
        <div class="card-body">
          <form [formGroup]="accountFormGroup"
(ngSubmit)="handleSearchAccount()">
            <div class="input-group">
              <label class="input-group-text">Account Id :</label>
              <input type="text" formControlName="accountId" class="form-
control">
              <button class="btn btn-info">
                <i class="bi bi-search"></i>
                Search
              </button>
            </div>
          </form>
          <ng-template #errorTemplate>
            <ng-container *ngIf="errorMessage ; else loading">
              <div class="text-danger">{{errorMessage}}</div>
            </ng-container>
            <ng-template #loading>
              Loading ...
            </ng-template>
          </ng-template>
          <ng-container *ngIf="accountObservable | async as
accountDetails">
            <div class="mb-2">
              <label>Account ID :</label>

<label><strong>{{accountDetails.accountId}}</strong></label>
            </div>
            <div class="mb-2">
              <label>Balance :</label>
              <label><strong>{{accountDetails.balance | number : '1.2-
2'}}</strong></label>
            </div>
            <table class="table">
              <thead>
                <th>ID</th><th>Date</th><th>Type</th><th>Amount</th>
              </thead>
              <tbody>
                <tr *ngFor="let op of
accountDetails.accountOperationDTOS">
                  <td>{{op.id}}</td>
                  <td>{{op.operationDate | date : 'dd-MM-yyyy:HH-mm-
ss'}}</td>
                  <td>{{op.type}}</td>
                  <td class="text-end">{{op.amount | number : '1.2-
2'}}</td>
                </tr>
              </tbody>
            </table>
            <ul class="nav nav-pills">
              <li *ngFor="let item of
[].constructor(accountDetails.totalPages);let page=index">
                <a [ngClass]="page==currentPage?'btn-info':'btn-outline-
info'" (click)="gotoPage(page)" class="btn  ms-1 mt-1">{{page}}</a>
```

```
              </li>
            </ul>
          </ng-container>
        </div>
      </div>
    </div>
    <div class="col col-md-6">
      <div class="card" *ngIf="accountObservable">
        <div class="card-header">Operations</div>
        <div class="card-body">

          <form [formGroup]="operationFromGroup"
(ngSubmit)="handleAccountOperation()" method="post">
            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio"
formControlName="operationType" value="DEBIT">
              <label class="form-check-label">DEBIT:</label>
            </div>
            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio"
formControlName="operationType" value="CREDIT">
              <label class="form-check-label">CREDIT:</label>
            </div>
            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio"
formControlName="operationType" value="TRANSFER">
              <label class="form-check-label">TRANSFER:</label>
            </div>
            <div class="mb-3"
*ngIf="operationFromGroup.value.operationType=='TRANSFER'">
              <label class="form-label">Account Destination :</label>
              <input type="text" formControlName="accountDestination"
class="form-control">
            </div>
            <div class="mb-3">
              <label class="form-label">Amount :</label>
              <input type="text" formControlName="amount" class="form-
control">
            </div>
            <div class="mb-3">
              <label class="form-label">Description :</label>
              <input type="text" formControlName="description"
class="form-control">
            </div>
            <div class="d-grid mb-3">
              <button class="btn btn-success">Save Operation</button>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

customer-accounts.components.html :

```
<div class="container">
  <div>{{customerId}}</div>
  <div>{{customer | json}}</div>
</div>
```

customers.component.html :

```
<div class="container mt-2">
  <ng-container *ngIf="customers | async as listCustomers; else
failureOrLading">
    <div class="card">
      <div class="card-header">Customers</div>
      <div class="card-body">
        <div *ngIf="searchFormGroup">
          <form [formGroup]="searchFormGroup"
(ngSubmit)="handleSearchCustomers()">
            <div class="input-group">
              <label class="input-group-text">Keyword :</label>
              <input type="text" formControlName="keyword" class="form-
control">
              <button class="btn btn-info">
                <i class="bi bi-search"></i>
              </button>
            </div>
          </form>
        </div>
        <table class="table">
          <thead>
          <tr>
            <th>ID</th><th>Name</th><th>Email</th>
          </tr>
          </thead>
          <tbody>
          <tr *ngFor="let c of customers | async">
            <td>{{c.id}}</td>
            <td>{{c.name}}</td>
            <td>{{c.email}}</td>
            <td>
              <button (click)="handleDeleteCustomer(c)" class="btn btn-
danger">
                <i class="bi bi-trash"></i>
              </button>
            </td>
            <td>
              <button (click)="handleCustomerAccounts(c)" class="btn btn-
success">
                Accounts
              </button>
            </td>
          </tr>
          </tbody>
        </table>
      </div>
    </div>
  </ng-container>
  <ng-template #failureOrLading>
    <ng-container *ngIf="errorMessage; else loading">
      <div class="text-danger">
        {{errorMessage}}
      </div>
    </ng-container>
    <ng-template #loading>
      Loading .....
    </ng-template>
  </ng-template>
</div>
```

Services :

accounts.service.ts :

```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {environment} from "../../environments/environment";
import {Observable} from "rxjs";
import {AccountDetails} from "../model/account.model";

@Injectable({
  providedIn: 'root'
})
export class AccountsService {

  constructor(private http : HttpClient) { }

  public getAccount(accountId : string, page : number, size :
number):Observable<AccountDetails>{
    return
this.http.get<AccountDetails>(environment.backendHost+"/accounts/"+accoun
tId+"/pageOperations?page="+page+"&size="+size);
  }
  public debit(accountId : string, amount : number, description:string){
    let data={accountId : accountId, amount : amount, description :
description}
    return
this.http.post(environment.backendHost+"/accounts/debit",data);
  }
  public credit(accountId : string, amount : number, description:string){
    let data={accountId : accountId, amount : amount, description :
description}
    return
this.http.post(environment.backendHost+"/accounts/credit",data);
  }
  public transfer(accountSource: string,accountDestination: string,
amount : number, description:string){
    let data={accountSource, accountDestination, amount, description }
    return
this.http.post(environment.backendHost+"/accounts/transfer",data);
  }
}
```

customer.service.ts :

```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {Observable} from "rxjs";
import {Customer} from "../model/customer.model";
import {environment} from "../../environments/environment";

@Injectable({
  providedIn: 'root'
})
export class CustomerService {
  constructor(private http:HttpClient) { }

  public getCustomers():Observable<Array<Customer>>{
    return
this.http.get<Array<Customer>>(environment.backendHost+"/customers")
```

```
  }
  public searchCustomers(keyword : string):Observable<Array<Customer>>{
    return
this.http.get<Array<Customer>>(environment.backendHost+"/customers/search
?keyword="+keyword)
  }
  public saveCustomer(customer: Customer):Observable<Customer>{
    return
this.http.post<Customer>(environment.backendHost+"/customers",customer);
  }
  public deleteCustomer(id: number){
    return this.http.delete(environment.backendHost+"/customers/"+id);
  }
}
```

environment : pour lier entre les deux applications fontend et backend :

```
environment.ts

1
    5+ usages
2   export const environment = {
3     production: false,
4     backendHost : "http://localhost:8085"
5   };
6
7
```

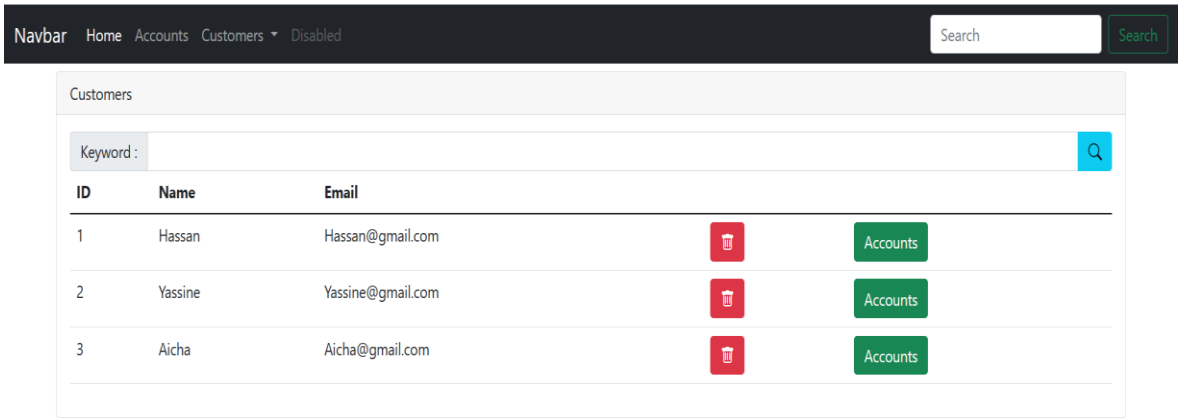
Et j'ai ajouté :

```
@CrossOrigin("*")
```

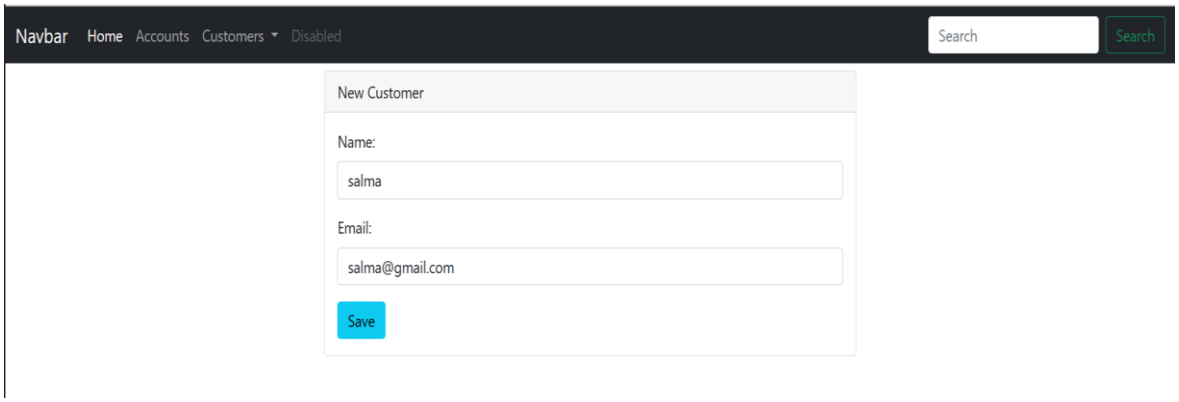
Dans les classes de la couche Web pour donner la permission

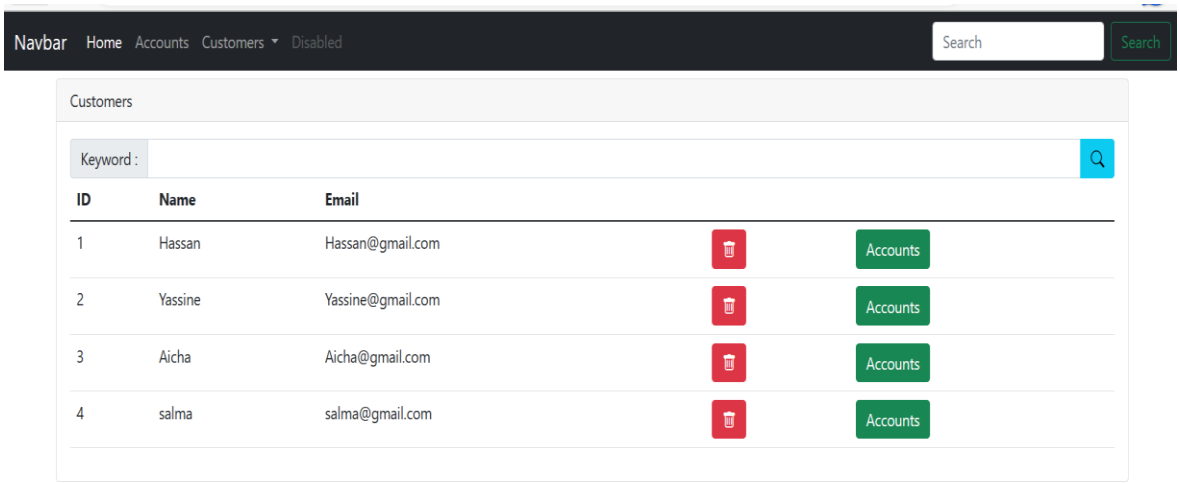
Résultats :

Affichage :

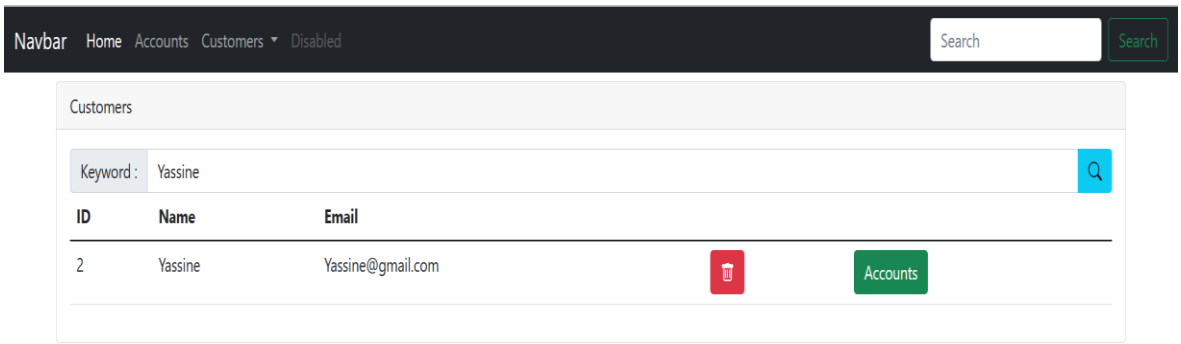


Ajouter :





Chercher :



Chercher un compte :

Navbar Home Accounts Customers Disabled Search Search

Accounts

Account Id : 2ea2ef73-bdac-4550-b588-a88b446294a0 Search

Account ID :**2ea2ef73-bdac-4550-b588-a88b446294a0**

Balance :**491,277.19**

ID	Date	Type	Amount
1	29-05-2023:21-03-02	CREDIT	87,296.39
2	29-05-2023:21-03-02	DEBIT	2,822.27
3	29-05-2023:21-03-02	CREDIT	67,127.37
4	29-05-2023:21-03-02	DEBIT	7,766.62
5	29-05-2023:21-03-02	CREDIT	19,593.00

0 1 2 3

Operations

DEBIT: CREDIT: TRANSFER:

Amount :

0

Description :

Save Operation

Navbar Home Accounts Customers Disabled Search Search

Accounts

Account Id : 2ea2ef73-bdac-4550-b588-a88b446294a0 Search

Account ID :**2ea2ef73-bdac-4550-b588-a88b446294a0**

Balance :**491,277.19**

ID	Date	Type	Amount
11	29-05-2023:21-03-02	CREDIT	49,197.10
12	29-05-2023:21-03-02	DEBIT	8,714.09
13	29-05-2023:21-03-02	CREDIT	26,037.75
14	29-05-2023:21-03-02	DEBIT	3,089.45
15	29-05-2023:21-03-02	CREDIT	35,175.96

0 1 2 3

Operations

DEBIT: CREDIT: TRANSFER:

Amount :

0

Description :

Save Operation