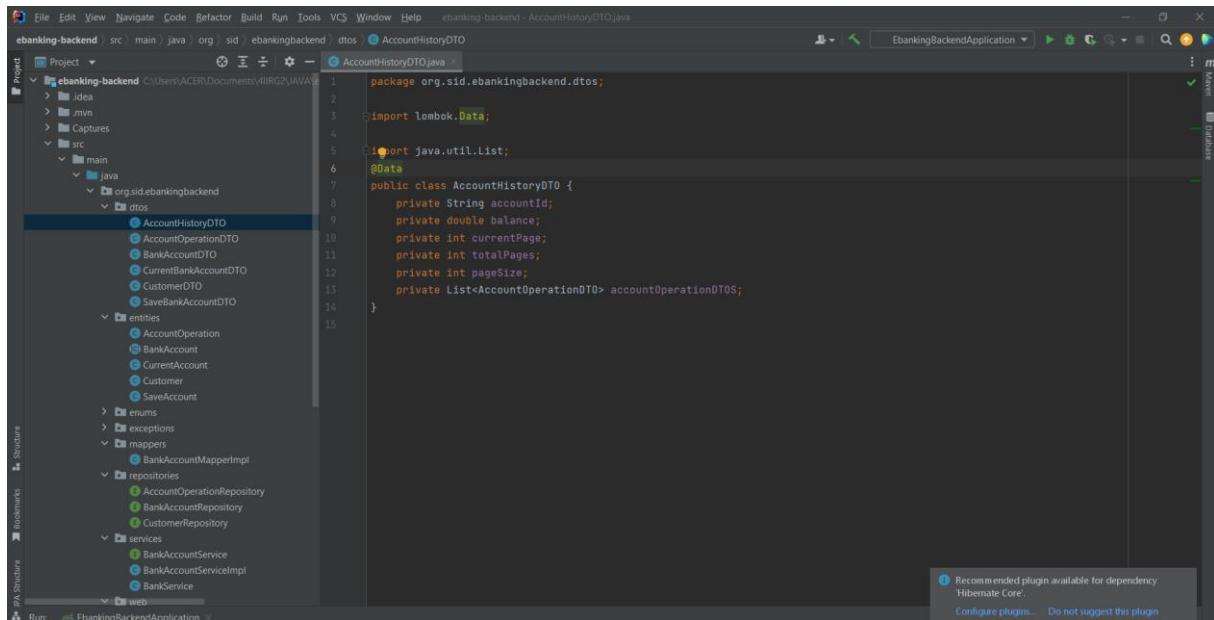


# Spring Angular Use case Digital Banking

## Package dtos

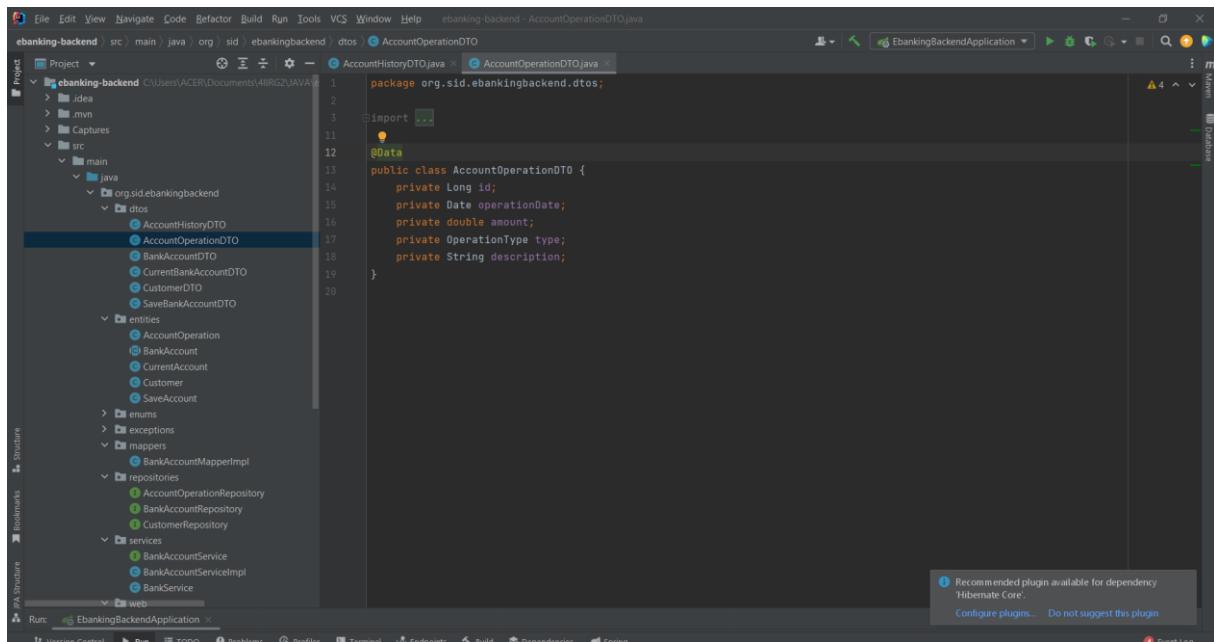
### AccountHistoryDTO.java



```
package org.sid.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;
}
```

### AccountOperationDTO.java



```
package org.sid.ebankingbackend.dtos;

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

### BankAccountDTO.java

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - BankAccountDTO.java
ebanking-backend > src > main > java > org > sid > ebankingbackend > dtos > BankAccountDTO.java
Project AccountHistoryDTO.java AccountOperationDTO.java BankAccountDTO.java
1 package org.sid.ebankingbackend.dtos;
2
3 import lombok.Data;
4
5 @Data
6 public class BankAccountDTO {
7     private String type;
8 }

```

Recommmended plugin available for dependency 'Hibernate Core'.  
Configure plugins... Do not suggest this plugin

## CurrentBankAccountDTO.java

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - CurrentBankAccountDTO.java
ebanking-backend > src > main > java > org > sid > ebankingbackend > dtos > CurrentBankAccountDTO.java > EbankingBackendApplication
Project AccountHistoryDTO.java AccountOperationDTO.java BankAccountDTO.java CurrentBankAccountDTO.java
1 package org.sid.ebankingbackend.dtos;
2
3 import ...
4
5 @Data
6 public class CurrentBankAccountDTO extends BankAccountDTO {
7     private String id;
8     private double balance;
9     private Date createdAt;
10    private AccountStatus status;
11    private CustomerDTO customerDTO;
12    private double overDraft;
13 }

```

Recommmended plugin available for dependency 'Hibernate Core'.  
Configure plugins... Do not suggest this plugin

## CustomerDTO.java

```
package org.sid.ebankingbackend.dtos;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import javax.persistence.*;
import java.util.List;
@Data
public class CustomerDTO {
    private Long id;
    private String name;
    private String email;
}
```

## SaveBankAccountDTO.java

```
package org.sid.ebankingbackend.dtos;
import lombok.Data;
import org.sid.ebankingbackend.enums.AccountStatus;
import java.util.Date;
@Data
public class SaveBankAccountDTO extends BankAccountDTO {
    private String id;
    private double balance;
    private Date createdDate;
    private AccountStatus status;
    private CustomerDTO customerDTO;
    private double interestRate;
}
```

## Package entities

### AccountOperation.java

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - AccountOperation.java

ebanking-backend > src > main > java > org > sid > ebankingbackend > entities > AccountOperation > description

```

package org.sid.ebankingbackend.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.sid.ebankingbackend.enums.OperationType;

import javax.persistence.*;
import java.util.Date;

@Date @NoArgsConstructor @AllArgsConstructor
@Entity
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;
}

```

Project Structure

Run: EbankingBackendApplication

Recommen... Recommended plugin available for dependency 'Hibernate Core'. Configure plugins... Do not suggest this plugin

## BankAccount.java

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - BankAccount.java

ebanking-backend > src > main > java > org > sid > ebankingbackend > entities > BankAccount > accountOperations

```

package org.sid.ebankingbackend.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.sid.ebankingbackend.enums.AccountStatus;

import javax.persistence.*;
import java.util.Date;
import java.util.List;

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

```

Project Structure

Run: EbankingBackendApplication

Recommen... Recommended plugin available for dependency 'Hibernate Core'. Configure plugins... Do not suggest this plugin

## CurrentAccount.java

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - CurrentAccount.java

ebanking-backend > src > main > java > org > sid > ebankingbackend > entities > CurrentAccount.java

```

package org.sid.ebankingbackend.entities;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Id;
@DiscriminatorValue("CA")//Dans la colonne type, la valeur sera CA pour dire current
public class CurrentAccount extends BankAccount{
    private double overDraft;
}

```

Project > ebanking-backend > src > main > java > org > sid > ebankingbackend > entities > CurrentAccount.java

Structure Bookmarks Java Structure Run: EbankingBackendApplication

Java Version Control Run TODO Problems Profiler Terminal Endpoints Build Dependencies Spring Event Log

Configure plugins... Do not suggest this plugin

## Customer.java

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - Customer.java

ebanking-backend > src > main > java > org > sid > ebankingbackend > entities > Customer.java

```

package org.sid.ebankingbackend.entities;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.Fetch;
import javax.persistence.*;
import java.util.List;
@Data
@NoArgsConstructor @AllArgsConstructor
@Entity
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;
}

```

Project > ebanking-backend > src > main > java > org > sid > ebankingbackend > entities > Customer.java

Structure Bookmarks Java Structure Run: EbankingBackendApplication

Java Version Control Run TODO Problems Profiler Terminal Endpoints Build Dependencies Spring Event Log

Configure plugins... Do not suggest this plugin

## SaveAccount.java

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "ebanking-backend". It contains a "src" directory with "main" and "test" packages. "main" contains "java" and "resources" sub-directories. "java" has sub-packages "org.sid.ebankingbackend" and "dtos". "org.sid.ebankingbackend" contains "CustomerHistoryDTO", "AccountOperationDTO", "BankAccountDTO", "CurrentBankAccountDTO", "CustomerDTO", "SaveBankAccountDTO", and "SaveAccount". "resources" contains "enums", "exceptions", and "mappers". "mappers" contains "BankAccountMapperImpl". "test" contains "repositories" (with "AccountOperationRepository", "BankAccountRepository", "CustomerRepository") and "services" (with "BankAccountService", "BankAccountServiceImpl", "BankService").
- Code Editor:** The "SaveAccount.java" file is open. The code defines a class "SaveAccount" that extends "BankAccount". It includes annotations like @Data, @NoArgsConstructor, @AllArgsConstructor, and @DiscriminatorValue("SA"). The class has a private field "interestRate".
- Toolbars and Status Bar:** The top bar shows tabs for various files like "CustomerDTO.java", "BankAccount.java", etc. The bottom status bar indicates "Recommended plugin available for dependency 'Hibernate Core'".

## Package enums

## AccountStatus.java

```
ebanking-backend > src > main > java > org > sid > ebankingbackend > enums > AccountStatus.java
```

```
ebanking-backend C:\Users\ACER\Documents\4IRG2\JAVA\1 package org.sid.ebankingbackend.enums;
public enum AccountStatus {
    CREATED, ACTIVATED, SUSPENDED
}
```

## OperationType.java

The screenshot shows the IntelliJ IDEA interface with the code editor open to `OperationType.java`. The code defines an enum `OperationType` with values `DEBIT` and `CREDIT`. The project structure on the left shows packages like `org.sid.ebankingbackend` containing `enums`, `entities`, `exceptions`, `mappers`, `repositories`, and `services`. A tooltip at the bottom right suggests a plugin for Hibernate Core.

```
package org.sid.ebankingbackend.enums;

public enum OperationType {
    DEBIT, CREDIT
}
```

## Package exceptions

### BalanceNotSufficientException.java

The screenshot shows the IntelliJ IDEA interface with the code editor open to `BalanceNotSufficientException.java`. This class extends `Exception` and has a constructor that takes a `String message`. The project structure on the left shows the same package structure as the previous screenshot, including the `exceptions` folder where this file is located. A tooltip at the bottom right suggests a plugin for Hibernate Core.

```
package org.sid.ebankingbackend.exceptions;

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

### BankAccountNotFoundException.java

```
package org.sid.ebankingbackend.exceptions;

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

## CustomerNotFoundException.java

```
package org.sid.ebankingbackend.exceptions;

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

## Package mappers

### BankAccountMapperImpl.java

The screenshot shows the Java code for `BankAccountMapperImpl`. The code is annotated with `@Service` and `@Repository`. It contains methods for mapping `Customer` to `CustomerDTO` and `SaveAccount` to `SaveBankAccountDTO`. The code uses `BeanUtils` for copying properties between entities and DTOs.

```
package org.sid.ebankingbackend.mappers;

import com.fasterxml.jackson.databind.util.BeanUtil;
import org.sid.ebankingbackend.dtos.AccountOperationDTO;
import org.sid.ebankingbackend.dtos.CurrentBankAccountDTO;
import org.sid.ebankingbackend.dtos.CustomerDTO;
import org.sid.ebankingbackend.dtos.SaveBankAccountDTO;
import org.sid.ebankingbackend.entities.AccountOperation;
import org.sid.ebankingbackend.entities.CurrentAccount;
import org.sid.ebankingbackend.entities.Customer;
import org.sid.ebankingbackend.entities.SaveAccount;
import org.sid.ebankingbackend.beans.BeansUtils;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Service;

@Service
public class BankAccountMapperImpl {

    public CustomerDTO fromCustomer(Customer customer) {
        CustomerDTO customerDTO=new CustomerDTO();
        BeansUtils.copyProperties(customer,customerDTO);
        /*customerDTO.setid(customer.getid());
        customerDTO.setName(customer.getName());
        customerDTO.setEmail(customer.getEmail());*/
        return customerDTO;
    }

    public Customer fromCustomerDTO(CustomerDTO customerDTO) {
        Customer customer=new Customer();
        BeansUtils.copyProperties(customerDTO,customer);
        return customer;
    }
}
```

The screenshot shows the Java code for `BankAccountMapperImpl`. The code is annotated with `@Service` and `@Repository`. It contains methods for mapping `SaveAccount` to `SaveBankAccount`, `CurrentAccount` to `CurrentBankAccount`, and `AccountOperation` to `AccountOperationDTO`. The code uses `BeanUtils` for copying properties between entities and DTOs.

```
public SaveBankAccount fromSaveBankAccount(SaveAccount saveAccount) {
    SaveBankAccountDTO saveBankAccountDTO=new SaveBankAccountDTO();
    BeanUtils.copyProperties(saveAccount,saveBankAccountDTO);
    saveBankAccountDTO.setCustomerDTO(fromCustomer(saveAccount.getCustomer()));
    saveBankAccountDTO.setType(saveAccount.getClass().getSimpleName());
    return saveBankAccountDTO;
}

public SaveAccount fromSaveBankAccountDTO(SaveBankAccountDTO saveBankAccountDTO) {
    SaveAccount saveAccount=new SaveAccount();
    BeanUtils.copyProperties(saveBankAccountDTO,saveAccount);
    saveAccount.setCustomer(fromCustomer(saveBankAccountDTO.getCustomerDTO()));
    return saveAccount;
}

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

public CurrentAccount fromCurrentBankAccountDTO(CurrentBankAccountDTO currentBankAccountDTO) {
    CurrentAccount currentAccount=new CurrentAccount();
    BeanUtils.copyProperties(currentBankAccountDTO,currentAccount);
    currentAccount.setCustomer(fromCustomer(currentBankAccountDTO.getCustomerDTO()));
    return currentAccount;
}

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

```
public class BankAccountMapperImpl implements BankAccountMapper {
    public SaveAccount saveAccount(SaveAccount saveAccount) {
        saveAccount.setCustomer(fromCustomerDTO(saveBankAccountDTO.getCustomerDTO()));
        return saveAccount;
    }

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

    public CurrentAccount fromCurrentBankAccountDTO(CurrentBankAccountDTO currentBankAccountDTO) {
        CurrentAccount currentAccount=new CurrentAccount();
        BeanUtils.copyProperties(currentBankAccountDTO,currentAccount);
        currentAccount.setCustomer(fromCustomerDTO(currentBankAccountDTO.getCustomerDTO()));
        return currentAccount;
    }

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

## Package repositories

### AccountOperationRepository.java

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

### BankAccountRepository.java

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - BankAccountRepository.java
ebanking-backend > src > main > java > org > sid > ebankingbackend > repositories > BankAccountRepository.java
Project Structure
1 package org.sid.ebankingbackend.repositories;
2
3 import ...
4
5 public interface BankAccountRepository extends JpaRepository<BankAccount, String> {
6
7     void save(BankAccount bankAccount);
8 }
9
```

Recommending plugin available for dependency 'Hibernate Core'.  
Configure plugins... Do not suggest this plugin

## CustomerRepository.java

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - CustomerRepository.java
ebanking-backend > src > main > java > org > sid > ebankingbackend > repositories > CustomerRepository.java
Project Structure
1 package org.sid.ebankingbackend.repositories;
2
3 import org.sid.ebankingbackend.entities.Customer;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface CustomerRepository extends JpaRepository<Customer, Long> {
7
8     void save(Customer customer);
9 }
```

Recommending plugin available for dependency 'Hibernate Core'.  
Configure plugins... Do not suggest this plugin

## Package services

### BankAccountService.java

The screenshot shows the IntelliJ IDEA interface with the file `BankAccountService.java` open. The code defines a service interface for managing bank accounts. It includes methods for saving customers, getting current bank accounts, and performing credit, debit, and transfer operations. The code uses annotations like `@Service`, `@Transactional`, and `@AllArgsConstructor`. A tooltip at the bottom right suggests a plugin for Hibernate Core.

```
package org.sid.ebankingbackend.services;

import ...

public interface BankAccountService {
    Customer saveCustomer(Customer customer);

    CustomerDTO saveCustomer(CustomerDTO customerDTO);

    CurrentBankAccountDTO saveCurrentBankAccount(double initialBalance, double overDraft, Long customerId) throws CustomerNotFoundException;
    SaveBankAccountDTO saveSavingBankAccount(double initialBalance, double overDraft, Long customerId) throws CustomerNotFoundException;
    List<CustomerDTO> listCustomers();
    BankAccountDTO getBankAccount(String accountId) throws BankAccountNotFoundException;
    void credit(String accountId, double amount, String description) throws BankAccountNotFoundException;
    void debit(String accountId, double amount, String description) throws BankAccountNotFoundException, BalanceNotSufficientException;
    void transfer(String accountIdSource, String accountIdDestination, double amount) throws BankAccountNotFoundException, BalanceNotSufficientException;

    List<BankAccountDTO> bankAccountList();

    CustomerDTO getCustomer(@PathVariable(name = "id") 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;
```

## BankAccountServiceImpl.java

The screenshot shows the IntelliJ IDEA interface with the file `BankAccountServiceImpl.java` open. This implementation class for the `BankAccountService` interface overrides the `saveCustomer` method. It retrieves a customer from the repository, logs a message, maps the DTO to a domain object using a mapper, and saves it back to the repository. A tooltip at the bottom right suggests a plugin for Hibernate Core.

```
@Service
@Transactional
@AllArgsConstructor
@Slf4j
public class BankAccountServiceImpl implements BankAccountService{
    private BankAccountRepository bankAccountRepository;
    private CustomerRepository customerRepository;
    private AccountOperationRepository accountOperationRepository;
    private BankAccountMapperImpl dtoMapper;

    @Override
    public Customer saveCustomer(Customer customer) { return null; }

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

    @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");
        }
    }
}
```

The screenshot shows the Java code for the `BankAccountServiceImpl` class. The code handles saving a bank account, listing customers, and performing credit operations.

```
public void saveBankAccount(BankAccount bankAccount) {
    CurrentAccount currentAccount = new CurrentAccount();
    currentAccount.setId(UUID.randomUUID().toString());
    currentAccount.setCreatedAt(new Date());
    currentAccount.setBalance(initialBalance);
    currentAccount.setOverDraft(overDraft);
    currentAccount.setCustomer(customer);
    currentAccount = bankAccountRepository.save(currentAccount);
    return dtoMapper.fromCurrentBankAccount(currentAccount);
}

@Override
public SaveBankAccountDTO 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");
    }

    SaveAccount saveAccount = new SaveAccount();
    saveAccount.setId(UUID.randomUUID().toString());
    saveAccount.setCreatedAt(new Date());
    saveAccount.setBalance(initialBalance);
    saveAccount.setInterestRate(interestRate);
    saveAccount.setCustomer(customer);
    SaveAccount savedBankAccount = bankAccountRepository.save(saveAccount);
    return dtoMapper.fromSaveBankAccount(saveAccount);
}

@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);
    if (bankAccount instanceof CurrentAccount) {
        CurrentAccount currentAccount = (CurrentAccount) bankAccount;
        return dtoMapper.fromCurrentBankAccount(currentAccount);
    } else {
        SaveAccount saveAccount = (SaveAccount) bankAccount;
        return dtoMapper.fromSaveBankAccount(saveAccount);
    }
}

@Override
public void credit(String accountId, double amount, String description) throws BankAccountNotFoundException {
    BankAccount bankAccount = bankAccountRepository.findById(accountId);
    if (bankAccount == null) {
        throw new BankAccountNotFoundException("Bank Account Not Found");
    }
    AccountOperation accountOperation = new AccountOperation();
    accountOperation.setType(OperationType.CREDIT);
    accountOperation.setAmount(amount);
    accountOperation.setDescription(description);
    bankAccount.setOperations(accountOperation);
    bankAccountRepository.save(bankAccount);
}
```

The screenshot shows the Java code for the `BankAccountServiceImpl` class. The code handles saving a bank account, listing customers, and performing credit operations.

```
public void saveBankAccount(BankAccount bankAccount) {
    CurrentAccount currentAccount = new CurrentAccount();
    currentAccount.setId(UUID.randomUUID().toString());
    currentAccount.setCreatedAt(new Date());
    currentAccount.setBalance(initialBalance);
    currentAccount.setOverDraft(overDraft);
    currentAccount.setCustomer(customer);
    currentAccount = bankAccountRepository.save(currentAccount);
    return dtoMapper.fromCurrentBankAccount(currentAccount);
}

@Override
public SaveBankAccountDTO 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");
    }

    SaveAccount saveAccount = new SaveAccount();
    saveAccount.setId(UUID.randomUUID().toString());
    saveAccount.setCreatedAt(new Date());
    saveAccount.setBalance(initialBalance);
    saveAccount.setInterestRate(interestRate);
    saveAccount.setCustomer(customer);
    SaveAccount savedBankAccount = bankAccountRepository.save(saveAccount);
    return dtoMapper.fromSaveBankAccount(saveAccount);
}

@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);
    if (bankAccount instanceof CurrentAccount) {
        CurrentAccount currentAccount = (CurrentAccount) bankAccount;
        return dtoMapper.fromCurrentBankAccount(currentAccount);
    } else {
        SaveAccount saveAccount = (SaveAccount) bankAccount;
        return dtoMapper.fromSaveBankAccount(saveAccount);
    }
}

@Override
public void credit(String accountId, double amount, String description) throws BankAccountNotFoundException {
    BankAccount bankAccount = bankAccountRepository.findById(accountId);
    if (bankAccount == null) {
        throw new BankAccountNotFoundException("Bank Account Not Found");
    }
    AccountOperation accountOperation = new AccountOperation();
    accountOperation.setType(OperationType.CREDIT);
    accountOperation.setAmount(amount);
    accountOperation.setDescription(description);
    bankAccount.setOperations(accountOperation);
    bankAccountRepository.save(bankAccount);
}
```

```
banking-backend / src / main / java / org / sid / ebankingbackend / services / BankAccountServiceImpl.java
...
@Override
public void debit(String accountId, double amount, String description) throws BankAccountNotFoundException, BalanceNotSufficientException {
    BankAccount bankAccount=bankAccountRepository.findById(accountId);
    if(bankAccount.getBalance()<amount){
        throw new BalanceNotSufficientException("Solde insuffisant");
    }
    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 transfer(String accountIdSource, String accountIdDestination, double amount) {
    debit(accountIdSource,amount,description:"Transfert to "+accountIdDestination);
    debit(accountIdSource,-amount,description:"Transfert from "+accountIdSource);
    credit(accountIdDestination,amount,description:"Transfert from "+accountIdSource);
}

@Override
public void balance(String accountId) {
    BankAccount bankAccount=bankAccountRepository.findById(accountId);
    if(bankAccount==null){
        throw new BankAccountNotFoundException("Bank Account Not Found");
    }
    System.out.println("Le solde de l'account "+bankAccount.getId()+" est "+bankAccount.getBalance());
}
```

```
banking-backend / src / main / java / org / sid / ebankingbackend / services / BankAccountServiceImpl.java
...
@Override
public List<BankAccountDTO> bankAccountList(){
    List<BankAccount> bankAccounts= bankAccountRepository.findAll();
    List<BankAccountDTO> bankAccountDTOs= bankAccounts.stream().map(bankAccount -> {
        if(bankAccount instanceof SaveAccount){
            SaveAccount saveAccount=(SaveAccount) bankAccount;
            return dtoMapper.fromSaveBankAccount(saveAccount);
        }
        else{
            CurrentAccount currentAccount=(CurrentAccount) bankAccount;
            return dtoMapper.fromCurrentBankAccount(currentAccount);
        }
    }).collect(Collectors.toList());
    return bankAccountDTOs;
}

@Override
public CustomerDTO getCustomer(@PathVariable(name = "id") 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) {
    Customer customer = customerRepository.findById(customerDTO.getId())
        .orElseThrow(() -> new CustomerNotFoundException("Customer Not Found"));
    customer.setName(customerDTO.getName());
    customer.setAddress(customerDTO.getAddress());
    customer.setPhone(customerDTO.getPhone());
    customer.setEmail(customerDTO.getEmail());
    customer.setCustomerId(customerDTO.getCustomerId());
    customerRepository.save(customer);
    return dtoMapper.fromCustomer(customer);
}
```

```
ebanking-backend - BankAccountServiceImpl.java
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - BankAccountServiceImpl.java
ebanking-backend > src > main > java > org > sid > ebankingbackend > services > BankAccountServiceImpl.java
Project Structure
  AccountOperation
    - BankAccount
    - CurrentAccount
    - Customer
    - SaveAccount
  enums
    - AccountStatus
    - OperationType
  exceptions
    - BalanceNotSufficientException
    - BankAccountNotFoundException
    - CustomerNotFoundException
  mappers
    - BankAccountMapperImpl
  repositories
    - AccountOperationRepository
    - BankAccountRepository
    - CustomerRepository
  services
    - BankAccountService
      - BankAccountServiceImpl
    - BankService
  web
    - BankAccountRestAPI
    - CustomerRestController
  EbankingBackendApplication
  resources
    static
    templates
    application.properties
  test
  target
  .gitignore
  ebanking-backend.iml
  Run: EbankingBackendApplication
```

```
1 AccountOperationRepository.java x 1 BankAccountRepository.java x 1 CustomerRepository.java x 1 BankAccountService.java x 1 BankAccountServiceImpl.java x
  179 log.info("Saving new Customer");
  180     Customer customer= dtoMapper.fromCustomerDTO(customerDTO);
  181     Customer savedCustomer=customerRepository.save(customer);
  182     return dtoMapper.fromCustomer(customer);
  183 }
  184 @Override
  185 public void deleteCustomer(Long customerId){
  186     customerRepository.deleteById(customerId);
  187 }
  188 @Override
  189 public List<AccountOperationDTO> accountHistory(String accountId){
  190     List<AccountOperation> accountOperations = accountOperationRepository.findByIdByBankAccountId(accountId);
  191     return accountOperations.stream().map(op-> dtoMapper.fromAccountOperation(op)).collect(Collectors.toList());
  192 }
  193
  194 @Override
  195 public AccountHistoryDTO getAccountHistory(String accountId, int page, int size) throws BankAccountNotFoundException {
  196     BankAccount bankAccount=bankAccountRepository.findById(accountId).orElse( other: null);
  197     if(bankAccount==null){
  198         throw new BankAccountNotFoundException("Bank Account Not Found");
  199     }
  200     Page<AccountOperation> accountOperations = accountOperationRepository.findByIdByBankAccountId(accountId, PageRequest.of(page, size));
  201     AccountHistoryDTO accountHistoryDTO=new AccountHistoryDTO();
  202     List<AccountOperationDTO> accountOperationDTOs = accountOperations.getContent().stream().map(op -> dtoMapper.fromAccountOperation(op));
  203     accountHistoryDTO.setAccountId(bankAccount.getId());
  204     accountHistoryDTO.setCurrentPage(page);
  205     accountHistoryDTO.setPageSize(size);
  206     accountHistoryDTO.setTotalPages(accountOperations.getTotalPages());
  207     return accountHistoryDTO;
  208 }
```

Recommende... Recommended plugin available for dependency 'Hibernate Core'.  
Configure plugins... Do not suggest this plugin

```
ebanking-backend - BankAccountServiceImpl.java
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ebanking-backend - BankAccountServiceImpl.java
ebanking-backend > src > main > java > org > sid > ebankingbackend > services > BankAccountServiceImpl.java
Project Structure
  AccountOperation
    - BankAccount
    - CurrentAccount
    - Customer
    - SaveAccount
  enums
    - AccountStatus
    - OperationType
  exceptions
    - BalanceNotSufficientException
    - BankAccountNotFoundException
    - CustomerNotFoundException
  mappers
    - BankAccountMapperImpl
  repositories
    - AccountOperationRepository
    - BankAccountRepository
    - CustomerRepository
  services
    - BankAccountService
      - BankAccountServiceImpl
    - BankService
  web
    - BankAccountRestAPI
    - CustomerRestController
  EbankingBackendApplication
  resources
    static
    templates
    application.properties
  test
  target
  .gitignore
  ebanking-backend.iml
  Run: EbankingBackendApplication
```

```
1 AccountOperationRepository.java x 1 BankAccountRepository.java x 1 CustomerRepository.java x 1 BankAccountService.java x 1 BankAccountServiceImpl.java x
  183 }
  184 @Override
  185 public void deleteCustomer(Long customerId){
  186     customerRepository.deleteById(customerId);
  187 }
  188 @Override
  189 public List<AccountOperationDTO> accountHistory(String accountId){
  190     List<AccountOperation> accountOperations = accountOperationRepository.findByIdByBankAccountId(accountId);
  191     return accountOperations.stream().map(op-> dtoMapper.fromAccountOperation(op)).collect(Collectors.toList());
  192 }
  193
  194 @Override
  195 public AccountHistoryDTO getAccountHistory(String accountId, int page, int size) throws BankAccountNotFoundException {
  196     BankAccount bankAccount=bankAccountRepository.findById(accountId).orElse( other: null);
  197     if(bankAccount==null){
  198         throw new BankAccountNotFoundException("Bank Account Not Found");
  199     }
  200     Page<AccountOperation> accountOperations = accountOperationRepository.findByIdByBankAccountId(accountId, PageRequest.of(page, size));
  201     AccountHistoryDTO accountHistoryDTO=new AccountHistoryDTO();
  202     List<AccountOperationDTO> accountOperationDTOs = accountOperations.getContent().stream().map(op -> dtoMapper.fromAccountOperation(op));
  203     accountHistoryDTO.setAccountId(bankAccount.getId());
  204     accountHistoryDTO.setCurrentPage(page);
  205     accountHistoryDTO.setPageSize(size);
  206     accountHistoryDTO.setTotalPages(accountOperations.getTotalPages());
  207     return accountHistoryDTO;
  208 }
```

Recommende... Recommended plugin available for dependency 'Hibernate Core'.  
Configure plugins... Do not suggest this plugin

## BankService.java

```

    package com.ebanking.backend.services;
    import com.ebanking.backend.AccountOperation;
    import com.ebanking.backend.BankAccount;
    import com.ebanking.backend.Customer;
    import com.ebanking.backend.enums.AccountStatus;
    import com.ebanking.backend.enums.OperationType;
    import com.ebanking.backend.exceptions.BalanceNotSufficientException;
    import com.ebanking.backend.exceptions.BankAccountNotFoundException;
    import com.ebanking.backend.exceptions.CustomerNotFoundException;
    import com.ebanking.backend.mappers.BankAccountMapperImpl;
    import com.ebanking.backend.repositories.AccountOperationRepository;
    import com.ebanking.backend.repositories.BankAccountRepository;
    import com.ebanking.backend.repositories.CustomerRepository;
    import com.ebanking.backend.services.BankAccountService;
    import com.ebanking.backend.services.BankAccountServiceImpl;
    import com.ebanking.backend.services.BankService;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Service;
    import org.springframework.transaction.annotation.Transactional;
    import java.util.List;
    import java.util.Optional;

    @Service
    @Transactional
    public class BankService {
        @Autowired
        private BankAccountRepository bankAccountRepository;

        public void consulter() {
            BankAccount bankAccount = bankAccountRepository.findById("9bb7eef0-eb04-4006-9cb8-0b3080ea0559").orElse(null);
            System.out.println("*****");
            if (bankAccount != null) {
                System.out.println(bankAccount.getId());
                System.out.println(bankAccount.getBalance());
                System.out.println(bankAccount.getStatus());
                System.out.println(bankAccount.getCreatedAt());
                System.out.println(bankAccount.getCustomer().getName());
                System.out.println(bankAccount.getAccountOperations().getSimpleName());
                if (bankAccount instanceof CurrentAccount) {
                    System.out.println("Over draft=>" + ((CurrentAccount) bankAccount).getOverDraft());
                }
                if (bankAccount instanceof SaveAccount) {
                    System.out.println("Rate =>" + ((SaveAccount) bankAccount).getInterestRate());
                }
            }
            bankAccount.getAccountOperations().forEach(op -> {
                System.out.println("*****");
                System.out.println(op.getType() + "\t" + op.getOperationDate() + "\t" + op.getAmount());
            });
        }
    }

```

## Package web

BankAccountRestAPI.java:

```

    package com.ebanking.backend.web;
    import com.ebanking.backend.AccountOperation;
    import com.ebanking.backend.BankAccount;
    import com.ebanking.backend.Customer;
    import com.ebanking.backend.enums.AccountStatus;
    import com.ebanking.backend.enums.OperationType;
    import com.ebanking.backend.exceptions.BalanceNotSufficientException;
    import com.ebanking.backend.exceptions.BankAccountNotFoundException;
    import com.ebanking.backend.exceptions.CustomerNotFoundException;
    import com.ebanking.backend.mappers.BankAccountMapperImpl;
    import com.ebanking.backend.repositories.AccountOperationRepository;
    import com.ebanking.backend.repositories.BankAccountRepository;
    import com.ebanking.backend.repositories.CustomerRepository;
    import com.ebanking.backend.services.BankAccountService;
    import com.ebanking.backend.services.BankAccountServiceImpl;
    import com.ebanking.backend.services.BankService;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.http.ResponseEntity;
    import org.springframework.web.bind.annotation.GetMapping;
    import org.springframework.web.bind.annotation.PathVariable;
    import org.springframework.web.bind.annotation.RequestParam;
    import org.springframework.web.bind.annotation.RestController;
    import java.util.List;

    @RestController
    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 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 getHistory(@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.java

The screenshot shows the IntelliJ IDEA interface with the file `CustomerRestController.java` open. The code implements a REST controller for managing customers. It includes methods for listing all customers, getting a specific customer by ID, saving a new customer, updating an existing customer, and deleting a customer. The code uses annotations like `@RestController`, `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping`. It also imports various Spring and Java utility classes.

```
import java.util.List;
import org.springframework.web.util.UriComponentsBuilder;
import org.springframework.web.bind.annotation.*;

@RestController
@AllArgsConstructor
@Stereotype
public class CustomerRestController {
    private BankAccountService bankAccountService;

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

    @GetMapping("customers/{id}")
    public CustomerDTO getCustomer(@PathVariable(name = "id") Long customerId) throws CustomerNotFoundException {
        return bankAccountService.getCustomer(customerId);
    }

    @PostMapping("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); }
}
```

## EBankingBackendApplication.java

The screenshot shows the IntelliJ IDEA interface with the file `EBankingBackendApplication.java` open. This is a Spring Boot application that runs the `EbankingBackendApplication` class. It contains a main method that runs the application. It also includes a command-line runner named `CommandLineRunner` which performs initial data seeding. The code uses `@SpringBootApplication`, `@Bean`, and `CommandLineRunner` annotations. It interacts with the `BankAccountService` to save customers and bank accounts.

```
@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", "Imane", "Mohamed").forEach(name->{
                CustomerDTO customer=new CustomerDTO();
                customer.setName(name);
                customer.setEmail(name+"@gmail.com");
                bankAccountService.saveCustomer(customer);
            });
            bankAccountService.listCustomers().forEach(customer -> {
                try {
                    bankAccountService.saveCurrentBankAccount( initialBalance: Math.random()*9000, overDraft: 9000, customer.getId());
                    bankAccountService.saveSavingBankAccount( initialBalance: Math.random()*12000, overDraft: 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 SaveBankAccountDTO){
                        accountId=((SaveBankAccountDTO) bankAccount).getId();
                    }
                }
            }
        };
    }
}
```

```

ebanking-backend - EbankingBackendApplication.java
...
    ...
    ...
}

//@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(cust->
            CurrentAccount currentAccount=new CurrentAccount();
            currentAccount.setId(UUID.randomUUID().toString());
            currentAccount.setBalance(Math.random()*90000);
            currentAccount.setCreatedAt(new Date());
            currentAccount.setStatus(AccountStatus.CREATED);
            currentAccount.setCustomer(cust);
            currentAccount.setOverDraft(9000);
            bankAccountRepository.save(currentAccount);

            SaveAccount saveAccounts=new Saveaccount();
            ...
        });
    };
}

```

```

ebanking-backend - EbankingBackendApplication.java
...
    ...
    ...
}

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

    return args -> {
        for (int i=0;i<5;i++){
            Customer customer=new Customer();
            ...
            customerRepository.save(customer);

            SaveAccount saveAccount=new Saveaccount();
            ...
            saveAccount.setBalance(Math.random()*90000);
            saveAccount.setCreatedAt(new Date());
            saveAccount.setStatus(AccountStatus.CREATED);
            saveAccount.setCustomer(cust);
            saveAccount.setInterestRate(5.5);
            bankAccountRepository.save(saveAccount);

            bankAccountRepository.findAll().forEach(acc->{
                for (int i=0;i<5;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(acc);
                    accountOperationRepository.save(accountOperation);
                }
            });
        }
    };
}

```

## Application.properties

```

#spring.datasource.url=jdbc:h2:mem:bank
spring.h2.console.enabled=true
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.MariaDBDialect
spring.jpa.show-sql=true

```

The screenshot shows an IDE interface with the project structure on the left and the contents of the application.properties file on the right. The file contains configuration for a Spring application using MySQL as the database, setting up an H2 console, and defining port 8085. It also specifies the database URL, username, password, and dialect.

## Base de données

The screenshot shows the phpMyAdmin interface connected to a MySQL database named 'e-bank'. The database structure is displayed, showing three tables: account\_operation, bank\_account, and customer. The account\_operation table has 126 rows and a size of 32,0 kio. The bank\_account table has 6 rows and a size of 32,0 kio. The customer table has 3 rows and a size of 16,0 kio. A new table creation dialog is open at the bottom, prompting for a name and number of columns (4).

localhost/phpmyadmin/sql.php?db=e-bank&table=account\_operation&pos=0

Affichage des lignes 0 - 24 (total de 120, traitement en 0,0048 seconde(s))

SELECT \* FROM `account\_operation`

	<a href="#">Tout afficher</a>	Nombre de lignes :	25	Filtrer les lignes	Chercher dans cette table	Trier par clé :	Aucun(e)
+ Options	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	<a href="#">id</a>	amount	description	operation_date	type	bank_account_id
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	1	19982.396797742844	credit	2022-05-22 12:22:24	CREDIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	2	5913.63115887403	debit	2022-05-22 12:22:24	DEBIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	3	18185.916323261548	credit	2022-05-22 12:22:24	CREDIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	4	3155.2587472604987	debit	2022-05-22 12:22:24	DEBIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	5	13972.54233396433	credit	2022-05-22 12:22:24	CREDIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	6	1841.3210974530014	debit	2022-05-22 12:22:24	DEBIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	7	10698.059099085278	credit	2022-05-22 12:22:24	CREDIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	8	6091.093006216625	debit	2022-05-22 12:22:24	DEBIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	9	19549.403640304143	credit	2022-05-22 12:22:24	CREDIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	10	5591.854462869458	debit	2022-05-22 12:22:24	DEBIT	15fba02e-f5b-47c2-9f02-8f1865a36967
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	11	14224.247336384004	credit	2022-05-22 12:22:24	CREDIT	15fba02e-f5b-47c2-9f02-8f1865a36967

localhost/phpmyadmin/sql.php?server=1&db=e-bank&table=bank\_account&pos=0

Affichage des lignes 0 - 5 (total de 6, traitement en 0,0018 seconde(s))

SELECT \* FROM `bank\_account`

	<a href="#">Tout afficher</a>	Nombre de lignes :	25	Filtrer les lignes	Chercher dans cette table	Trier par clé :	Aucun(e)		
+ Options	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	<a href="#">type</a>	<a href="#">id</a>	balance	created_at	status	over_draft	interest_rate	customer_id
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	CA	15fba02e-f5b-47c2-9f02-8f1865a36f67	116028.73618159124	2022-05-22 12:22:24	NULL	9000	NULL	3
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	CA	3e01a5ed24e0-4697-8056-35ccb0316e30	121376.6792238842	2022-05-22 12:22:23	NULL	9000	NULL	1
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	CA	4b853b36-d4ee-472d-b47b-deb14e2160b8	124820.14323075124	2022-05-22 12:22:24	NULL	9000	NULL	2
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	SA	858c9251-e60e-4cc4-8dd7-7adaa0543b91	116293.165693335474	2022-05-22 12:22:24	NULL	NULL	5.5	3
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	SA	a6c19130-e4e3-4535-b771-149672754069	90411.41995071522	2022-05-22 12:22:24	NULL	NULL	5.5	2
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	SA	b7dd487-b624-433c-8346-2a6fc92607	109729.72008338496	2022-05-22 12:22:24	NULL	NULL	5.5	1

localhost/phpmyadmin/sql.php?server=1&db=e-bank&table=customer&pos=0

Affichage des lignes 0 - 2 (total de 3, traitement en 0,0016 seconde(s))

SELECT \* FROM `customer`

	<a href="#">Tout afficher</a>	Nombre de lignes :	25	Filtrer les lignes	Chercher dans cette table	Trier par clé :	Aucun(e)
+ Options	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	<a href="#">id</a>	<a href="#">email</a>	<a href="#">name</a>			
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	1	Hassan@gmail.com	Hassan			
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	2	Imane@gmail.com	Imane			
	<a href="#">Éditer</a> <a href="#">Copier</a> <a href="#">Supprimer</a>	3	Mohammed@gmail.com	Mohammed			

Web

```
Swagger UI | Part 3 - Spring Angular Use case | localhost:8085/accounts | Use case JPA, Hibernate Spring | localhost / 127.0.0.1 / e-bank / c | +
```

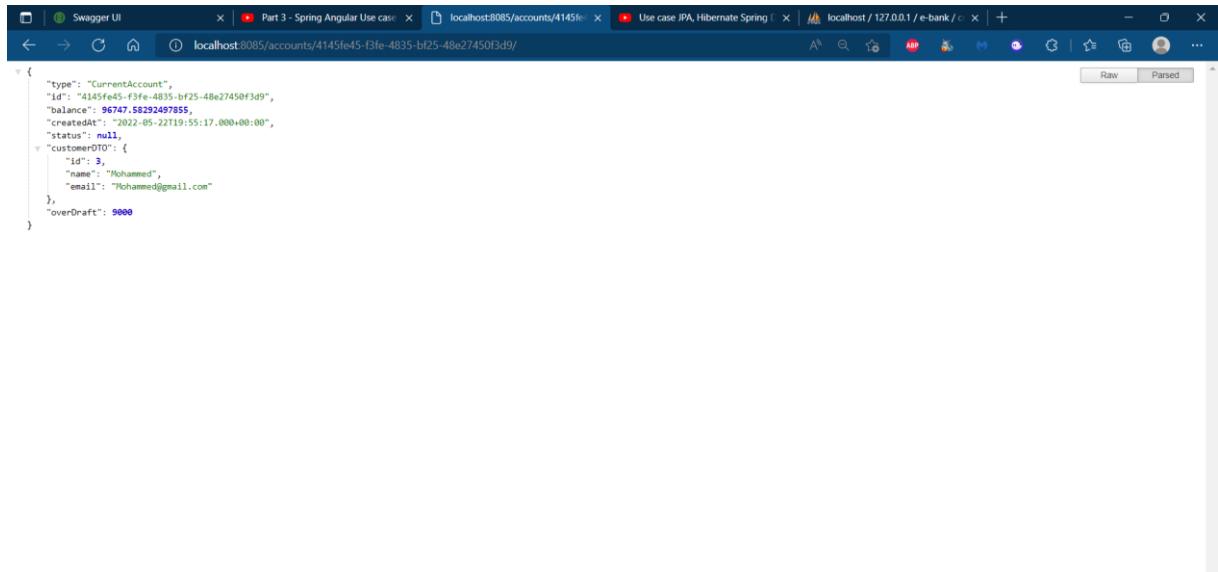
```
Raw Parsed
```

```
[{"id": "4145fe45-f3fe-4835-bf25-48e27450f3d9", "balance": 9647.5829249785, "createdAt": "2022-05-22T19:55:17.000+00:00", "status": null, "customerDTO": {"id": 3, "name": "Mohammed", "email": "Mohammed@gmail.com"}, "overDraft": 9000}, {"id": "51bb1eb0-0226-451d-a5d0-e4efb6ee2ae9", "balance": 138177.7735606708, "createdAt": "2022-05-22T19:55:17.000+00:00", "status": null, "customerDTO": {"id": 2, "name": "Imane", "email": "Imane@gmail.com"}, "interestRate": 5.5}, {"id": "6d5a74d3-90fc-4665-9529-d2051a0fe715", "balance": 129738.1486836921, "createdAt": "2022-05-22T19:55:17.000+00:00", "status": null, "customerDTO": {"id": 3, "name": "Mohammed", "email": "Mohammed@gmail.com"}, "interestRate": 5.5}, {"id": "a2a8c37c-2e7d-40ce-8299-82c862371e51"}]
```

```
Swagger UI | Part 3 - Spring Angular Use case | localhost:8085/accounts/4145fe45-f3fe-4835-bf25-48e27450f3d9/pageOperations?page=1 | Use case JPA, Hibernate Spring | localhost / 127.0.0.1 / e-bank / c | +
```

```
Raw Parsed
```

```
{"accountDTO": {"id": "4145fe45-f3fe-4835-bf25-48e27450f3d9", "balance": 0, "currentPage": 1, "totalPages": 4, "pageSize": 5, "accountOperationDTOs": [{"id": 6, "operationDate": "2022-05-22T19:55:17.000+00:00", "amount": 1671.569978442653, "type": "DEBIT", "description": "debit"}, {"id": 7, "operationDate": "2022-05-22T19:55:17.000+00:00", "amount": 13684.46024961678, "type": "CREDIT", "description": "credit"}, {"id": 8, "operationDate": "2022-05-22T19:55:17.000+00:00", "amount": 2185.81785672395, "type": "DEBIT", "description": "debit"}, {"id": 9, "operationDate": "2022-05-22T19:55:17.000+00:00", "amount": 13366.6741671210785, "type": "CREDIT", "description": "credit"}, {"id": 10, "operationDate": "2022-05-22T19:55:17.000+00:00", "amount": 8472.327344427112, "type": "DEBIT", "description": "debit"}]}
```



The screenshot shows a browser window with multiple tabs open. The active tab displays a JSON object representing a bank account. The JSON structure is as follows:

```
{ "type": "CurrentAccount", "id": "4145fe45-f3fe-4835-bf25-48e27450f3d9", "balance": 96747.58292497855, "createdAt": "2022-05-22T19:55:17.000+00:00", "status": null, "customerDTO": { "id": 3, "name": "Mohammed", "email": "Mohammed@gmail.com" }, "overDraft": 9999 }
```

The JSON object includes fields for account type, ID, balance (96747.58292497855), creation date (2022-05-22T19:55:17.000+00:00), status (null), customer details (id: 3, name: "Mohammed", email: "Mohammed@gmail.com"), and overDraft limit (9999).