



E_Banking Project



Nom HAMZA

Prénom Braimi

Filière Big Data & Cloud Computing

ANNÉE UNIVERSITAIRE : 2022/2023

JEE Project Spring Angular Digital Banking

I. Specifications

We need to create an application that allows us to manage bank accounts. each account belongs to a customer. An account can undergo several **DEBIT** or **CREDIT**-type operations. There are two types of accounts: current accounts and savings accounts.

➤ Part 1: DAO layer

1. Create a Spring Boot project.
2. Create the JPA entities: Customer, Bank Account, Saving Account, Current Account, Account Operation.
3. Create the JPA repository interfaces based on Spring Data.
4. Test the DAO layer.

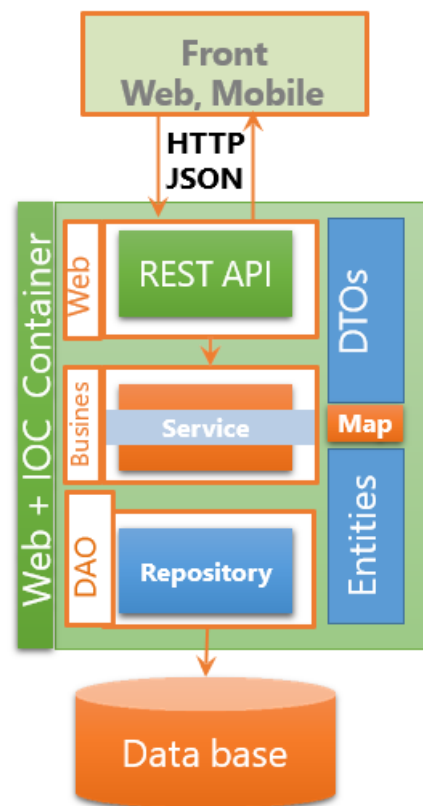
➤ Part 2: Service Layer, DTOs and Mappers

➤ Part 3: Web Layer (RestController)

➤ Part 4: Angular Front End

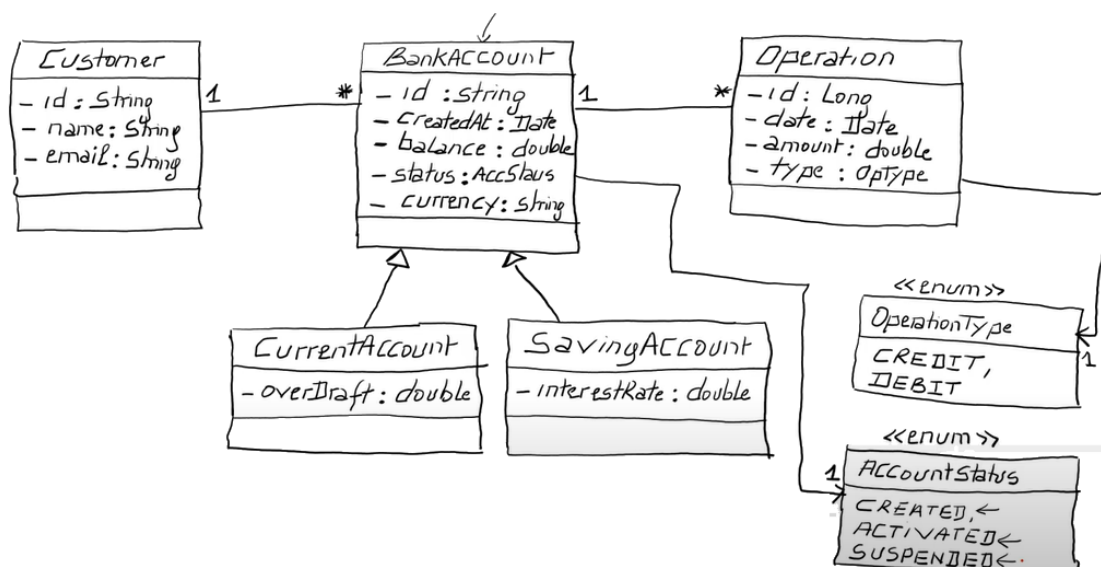
➤ Part 5: Security with Spring Security and JWT

II. Technical Architecture



Web Architecture Client-Side HTML Rendering

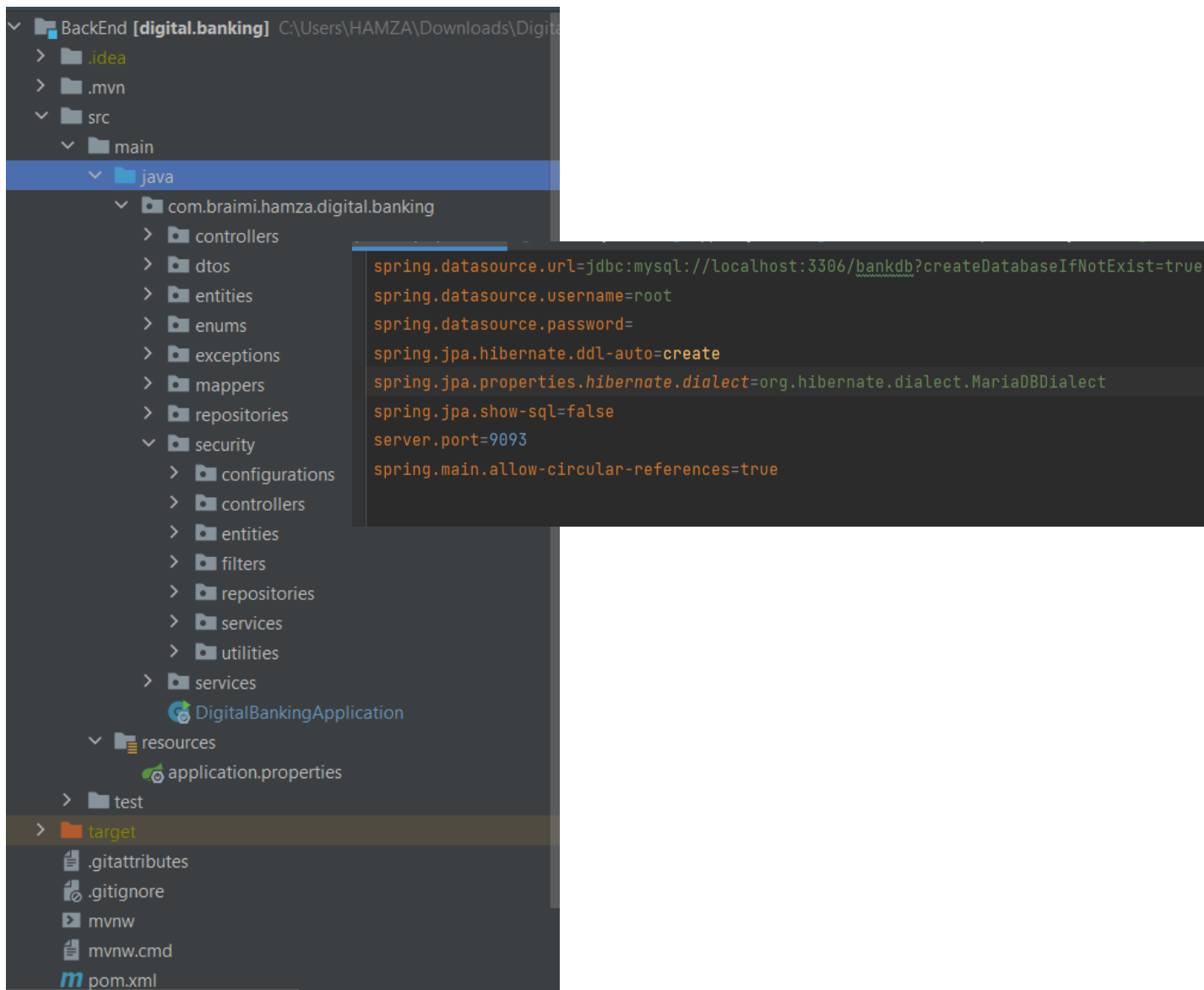
III. Use Case: Digital Banking



creation of a web application client-side HTML rendering that allows to:

- Manage bank customers.
- Manage bank accounts belonging to customers.
- Manage transactions on accounts: payment, transfer, and withdrawal.

IV. Project Structure



V. JPA Entities

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "Type", length = 4)
public class BankAccount {

    no usages
    @Id
    private String id;
    no usages
    private double balance;
    no usages
    private Date createdAt;
    no usages
    @Enumerated(EnumType.STRING)
    private AccountStatus status;

    no usages
    @ManyToOne
    @OnDelete(action = OnDeleteAction.CASCADE)
    private Customer customer;
    no usages
    @OneToMany(mappedBy = "bankAccount", fetch = FetchType.LAZY)
    private List<AccountOperation> accountOperationList;
}
```

```
("CA")
```

```
tAccount extends BankAccount{
```

```
overDraft;
```

```
public enum AccountStatus {
```

```
2 usages
```

```
CREATED, SUSPENDED, ACTIVATED
```

```
}
```

```
@DiscriminatorColumn(name = "SA")
```

```
@Entity
```

```
public class SavingAccount extends BankAccount{
```

```
no usages
```

```
private double interestRate;
```

```
}
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor

@Entity
public class AccountOperation {

    no usages
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id ;
    no usages
    private Date operationDate;
    no usages
    private double amount;
    no usages
    private OperationType type;
    no usages
    private String description;

    no usages
    @ManyToOne
    @OnDelete(action = OnDeleteAction.CASCADE)
    private BankAccount bankAccount;
}
```

```
@Entity
public class Customer {

    no usages
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private String name ;
    no usages
    private String email;

    no usages
    @OneToMany(mappedBy = "customer")
    @OnDelete(action = OnDeleteAction.CASCADE)
    private List<BankAccount> bankAccount = new java.util.ArrayList<>();
}
```

```
10 usages  👤 Your Name
public enum OperationType {
    2 usages
    DEBIT, CREDIT, TRANSFERT
}
```

VI. Repositories

```
public interface BankAccountRepository extends JpaRepository<BankAccount,String> {
    👤 Your Name
    Page<BankAccount> findAll(Pageable pageable);
    1 usage  👤 Your Name
    List<BankAccount> findByCustomer(Customer customer);
}
```

```
public interface AccountOperationRepository extends JpaRepository<AccountOperation,Long> {
    1 usage  👤 Your Name
    List<AccountOperation> findByBankAccountId(String accountId);
    1 usage  👤 Your Name
    Page<AccountOperation> findByBankAccountIdOrderByOperationDateDesc(String accountId, Pageable pageable);
}
```

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    1 usage  👤 Your Name
    @Query("select c from Customer c where c.name like :kw")
    Page<Customer> searchByName(@Param("kw") String keyword, Pageable pageable);

    👤 Your Name
    Page<Customer> findAll(Pageable pageable);
}
```

VII. DTOs

```
@Data
public class
AccountHistoryDTO {

    public String accountId;
    private double balance ;
    private int currentPage;
    private int totalPages;
    private int pageSize;
    private
    List<AccountOperationDTO>
    accountOperationDTOList;
}
```

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

```
@Data
public class BankAccountDTO {

    private String id;
    private double balance;
    private Date createdAt;
    private AccountStatus status;
    private CustomerDTO
    customerDTO;
    private double overDraft;
    private String type ;
    private double interestRate ;
}
```

```
@Data
public class BankAccountsDTO
{

    private List<BankAccountDTO>
    bankAccountDTOS;
    private int totalPages;
}
```

```
@Data
public class CreditDTO {
    private String accountId ;
    private double amount ;
    private String description ;
}
```

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

```
@Data
public class CustomersDTO {

    List<CustomerDTO>
customerDTO;
    int totalpage ;
}
```

```
@Data
public class
TransferRequestDTO {
private String accountSource;
private String
accountDestination;
private double amount ;
private String description;
}
```

```
package
com.braimi.hamza.digital.bank
ing.dtos;

import lombok.Data;

@Data
public class DebitDTO {

    private String accountId ;
    private double amount ;
    private String description ;
}
```

VIII. Mappers

```
package com.braimi.hamza.digital.banking.mappers;

import com.braimi.hamza.digital.banking.dtos.AccountOperationDTO;
import com.braimi.hamza.digital.banking.dtos.CurrentBankAccountDTO;
import com.braimi.hamza.digital.banking.dtos.CustomerDTO;
import com.braimi.hamza.digital.banking.dtos.SavingBankAccountDTO;
import com.braimi.hamza.digital.banking.entities.AccountOperation;
import com.braimi.hamza.digital.banking.entities.CurrentAccount;
import com.braimi.hamza.digital.banking.entities.Customer;
import com.braimi.hamza.digital.banking.entities.SavingAccount;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class BankAccountMapperImplementation {

    @Autowired
    BankAccountMapperImplementation bankAccountMapperImplementation;

    public CustomerDTO fromCustomer(Customer customer){
        CustomerDTO customerDTO = new CustomerDTO();
        BeanUtils.copyProperties(customer,customerDTO);
        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(bankAccountMapperImplementation.fromCustomer(savingAccount.getCustomer()));
        savingBankAccountDTO.setType("saving account");
        return savingBankAccountDTO;
    }

    public SavingAccount fromSavingBankAccountDTO(SavingBankAccountDTO savingBankAccountDTO) {
        SavingAccount savingAccount = new SavingAccount();
        BeanUtils.copyProperties(savingBankAccountDTO, savingAccount);

        savingAccount.setCustomer(bankAccountMapperImplementation.fromCustomerDTO(savingBankAccountDTO.getCustomerDTO()));
        return savingAccount;
    }

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

        currentAccount.setCustomer(bankAccountMapperImplementation.fromCustomerDTO(currentBankAccountDTO.getCustomerDTO()));

        return currentAccount;
    }

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

        currentBankAccountDTO.setCustomerDTO(bankAccountMapperImplementation.fromCustomer(currentAccount.getCustomer()));
        return currentBankAccountDTO;
    }

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

        accountOperationDTO.setAccountId(accountOperation.getBankAccount().getId());
        return accountOperationDTO;
    }
}
```

IX. Exceptions

```
BalanceNotSufficientException.java
1 package com.braimi.hamza.digital.banking.exceptions;
2
3 public class BalanceNotSufficientException extends Exception {
4     public BalanceNotSufficientException(String message) {
5         super(message);
6     }
7 }

BankAccountNotFound.java
1 package com.braimi.hamza.digital.banking.exceptions;
2
3 public class BankAccountNotFound extends Exception {
4     public BankAccountNotFound(String bank_account_not_found) {
5         super(bank_account_not_found);
6     }
7 }

CustomerNotFound.java
1 package com.braimi.hamza.digital.banking.exceptions;
2
3 public class CustomerNotFound extends Exception {
4     public CustomerNotFound(String message) {
5         super(message);
6     }
7 }
```

X. Service Layer

```
package com.braimi.hamza.digital.banking.services;

import com.braimi.hamza.digital.banking.dtos.*;
import com.braimi.hamza.digital.banking.entities.Customer;
import com.braimi.hamza.digital.banking.exceptions.BankAccountNotFound;
import com.braimi.hamza.digital.banking.exceptions.BalanceNotSufficientException;
import com.braimi.hamza.digital.banking.exceptions.CustomerNotFound;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public interface BankAccountService {
    CustomerDTO saveCustomer(CustomerDTO customer);

    SavingBankAccountDTO saveSavingBankAccount(double initialBalance, double interestRate, Long customerId) throws CustomerNotFound;

    CurrentBankAccountDTO saveCurrentBankAccount(double initialBalance, double
```

```
overdraft, Long customerId) throws CustomerNotFoundException;

    List<CustomerDTO> listCustomers(int page);

    List<Customer> listCustomer();

    BankAccountDTO getBankAccount(String accountId) throws BankAccountNotFound;

    void debit(String accountId, double amount, String description) throws
BalanceNotSufficientException, BankAccountNotFound;

    void credit(String accountId, double amount, String description) throws
BankAccountNotFound;

    void transfer(String accountIdSource, String accountIdDestination, double
amount) throws BankAccountNotFound, BalanceNotSufficientException;

    List<BankAccountDTO> bankAccountListOfCustomer(Long customerId);

    BankAccountsDTO getBankAccountList(int page);

    CustomerDTO getCustomer(Long customerId) throws CustomerNotFoundException;

    CustomerDTO updateCustomer(CustomerDTO customerDTO);

    void deleteCustomer(Long customerId);

    List<AccountOperationDTO> accountOperationHistory(String accountId);

    AccountHistoryDTO getAccoutHistory(String accountId, int page, int size) throws
BankAccountNotFound;

    CustomersDTO getCustomerByName(String keyword, int page) throws
CustomerNotFoundException;

    BankAccountDTO updateBankAccount(BankAccountDTO bankAccountDTO);
}
```

BankAccountServiceImplementation

```
package com.braimi.hamza.digital.banking.services;

import com.braimi.hamza.digital.banking.dtos.*;
import com.braimi.hamza.digital.banking.entities.*;
import com.braimi.hamza.digital.banking.enums.OperationType;
import com.braimi.hamza.digital.banking.mappers.BankAccountMapperImplementation;
import com.braimi.hamza.digital.banking.repositories.AccountOperationRepository;
import com.braimi.hamza.digital.banking.repositories.BankAccountRepository;
import com.braimi.hamza.digital.banking.repositories.CustomerRepository;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import lombok.extern.slf4j.Slf4j;

import com.braimi.hamza.digital.banking.exceptions.BalanceNotSufficientException;
import com.braimi.hamza.digital.banking.exceptions.BankAccountNotFound;
import com.braimi.hamza.digital.banking.exceptions.CustomerNotFoundException;
```

```
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.annotation.Order;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.Date;
import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;

@Transactional
@Service
@AllArgsConstructor
@NoArgsConstructor
@Slf4j
public class BankAccountServiceImplementation implements BankAccountService {

    @Autowired
    @Order(1)
    BankAccountRepository bankAccountRepository;
    @Autowired
    @Order(1)
    CustomerRepository customerRepository;
    @Autowired
    @Order(1)
    AccountOperationRepository accountOperationRepository;

    @Autowired
    BankAccountMapperImplementation dtoMapper;

    @Override
    public CustomerDTO saveCustomer(CustomerDTO customerDTO) {
        Customer customer = dtoMapper.fromCustomerDTO(customerDTO);
        customer = customerRepository.save(customer);
        return dtoMapper.fromCustomer(customer);
    }

    @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.setInterestRate(interestRate);
        savingAccount.setCustomer(customer);
        savingAccount = bankAccountRepository.save(savingAccount);

        return dtoMapper.fromSavingBankAccount(savingAccount);
    }
}
```

```
}

@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.setOverDraft(overdraft);
    currentAccount.setCustomer(customer);
    currentAccount = bankAccountRepository.save(currentAccount);
    return dtoMapper.fromCurrentBankAccount(currentAccount);

}

public List<BankAccountDTO> bankAccountListOfCustomer(Long id){
    Customer customer = new Customer();
    customer.setId(id);
    List<BankAccount> bankAccounts =
bankAccountRepository.findByCustomer(customer);
    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 List<CustomerDTO> listCustomers(int page) {
    Page<Customer> customers =
customerRepository.findAll(PageRequest.of(page,6));
    List<CustomerDTO> collect = customers.stream().map(customer ->
dtoMapper.fromCustomer(customer)).collect(Collectors.toList());
    return collect;
}

@Override
public List<Customer> listCustomer() {
    return customerRepository.findAll();
}

@Override
public BankAccountDTO getBankAccount(String accountId) throws
BankAccountNotFound {
```

```
        BankAccount bankAccount =
bankAccountRepository.findById(accountId).orElseThrow(() -> new
BankAccountNotFound("Bank account Not Found"));

        if(bankAccount instanceof SavingAccount){
            SavingAccount savingAccount =(SavingAccount) bankAccount;
            SavingBankAccountDTO savingBankAccountDTO =
dtoMapper.fromSavingBankAccount(savingAccount);
            return savingBankAccountDTO ;
        }else{
            CurrentAccount currentAccount =(CurrentAccount) bankAccount;
            CurrentBankAccountDTO currentBankAccountDTO =
dtoMapper.fromCurrentBankAccount(currentAccount);
            return currentBankAccountDTO ;
        }
    }

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

        accountOperation.setAmount(amount);
        bankAccountRepository.save(bankAccount);
        accountOperationRepository.save(accountOperation);
    }

    @Override
    public void credit(String accountId, double amount , String description) throws
BankAccountNotFound {
        BankAccount bankAccount =
bankAccountRepository.findById(accountId).orElseThrow(() ->new
BankAccountNotFound("Account not found") );

        if (bankAccount == null)
            throw new BankAccountNotFound("bank Account not found");
        AccountOperation accountOperation = new AccountOperation();
        accountOperation.setType(OperationType.CREDIT);
        accountOperation.setOperationDate(new Date());
        bankAccount.setBalance(bankAccount.getBalance() + amount);
        accountOperation.setBankAccount(bankAccount);
        accountOperation.setAmount(amount);
        accountOperation.setOperationDate(new Date());
    }
}
```

```
        accountOperation.setDescription(accountOperation.getDescription());
        bankAccountRepository.save(bankAccount);
        accountOperationRepository.save(accountOperation);
    }

    @Override
    public void transfer(String accountIdSource, String accountIdDestination,
double amount) throws BankAccountNotFound, BalanceNotSufficientException {
        AccountOperationDTO accountSourceOperationDTO = new AccountOperationDTO();
        accountSourceOperationDTO.setAccountId(accountIdSource);
        accountSourceOperationDTO.setAmount(amount);
        debit(accountIdSource, amount, "description test");
        AccountOperationDTO accountDestOperationDTO = new AccountOperationDTO();
        accountDestOperationDTO.setAccountId(accountIdDestination);
        accountDestOperationDTO.setAmount(amount);
        credit(accountIdDestination, amount, "description test");
    }

    @Override
    public BankAccountsDTO getBankAccountList(int page) {

        Page<BankAccount> bankAccounts =
bankAccountRepository.findAll(PageRequest.of(page, 5));
        List<BankAccountDTO> bankAccountDTOList =
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());
        BankAccountsDTO bankAccountsDTO= new BankAccountsDTO();
        bankAccountsDTO.setBankAccountDTOS(bankAccountDTOList);
        bankAccountsDTO.setTotalPage(bankAccounts.getTotalPages());
        return bankAccountsDTO;
    }

    @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) {
        Customer customer = dtoMapper.fromCustomerDTO(customerDTO);
        customer = customerRepository.save(customer);
        return dtoMapper.fromCustomer(customer);
    }
}
```

```
@Override
public BankAccountDTO updateBankAccount(BankAccountDTO bankAccountDTO) {
    BankAccount bankAccount;
    if (bankAccountDTO.getType().equals("saving account")) {
        SavingBankAccountDTO saving = new SavingBankAccountDTO();

        BeanUtils.copyProperties(bankAccountDTO, saving);
        bankAccount = dtoMapper.fromSavingBankAccountDTO(saving);
        bankAccount = bankAccountRepository.save(bankAccount);
        return dtoMapper.fromSavingBankAccount((SavingAccount) bankAccount);
    } else {
        CurrentBankAccountDTO current = new CurrentBankAccountDTO();

        BeanUtils.copyProperties(bankAccountDTO, current);
        bankAccount = dtoMapper.fromCurrentBankAccountDTO(current);
        bankAccount = bankAccountRepository.save(bankAccount);
        return dtoMapper.fromCurrentBankAccount((CurrentAccount) bankAccount);
    }
}

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

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

@Override
public AccountHistoryDTO getAccoutHistory(String accountId, int page, int size)
throws BankAccountNotFound {
    BankAccount bankAccount =
bankAccountRepository.findById(accountId).orElse(null);
    if (bankAccount == null)
        throw new BankAccountNotFound("bank not fount ");
    Page<AccountOperation> accountOperationPage =
accountOperationRepository.findByBankAccountIdOrderByOperationDateDesc(accountId,
PageRequest.of(page, size));
    AccountHistoryDTO accountHistoryDTO = new AccountHistoryDTO();
    List<AccountOperationDTO>
accountOperationDTOList = accountOperationPage.getContent().stream().map(op -
> dtoMapper.fromAccountOperation(op)).collect(Collectors.toList());
    accountHistoryDTO.setAccountOperationDTOList(accountOperationDTOList);
    accountHistoryDTO.setAccountId(bankAccount.getId());
    accountHistoryDTO.setBalance(bankAccount.getBalance());
}
```



```

        accountHistoryDTO.setPageSize(size);
        accountHistoryDTO.setCurrentPage(page);
        accountHistoryDTO.setTotalPages(accountOperationPage.getTotalPages());
        return accountHistoryDTO;
    }

    @Override
    public CustomersDTO getCustomerByName(String keyword,int page) throws
CustomerNotFoundException {
        Page<Customer> customers ;

        customers =
customerRepository.searchByName(keyword,PageRequest.of(page,5));
        List<CustomerDTO> customerDTOS=customers.getContent().stream().map(c->dtoMapper.fromCustomer(c)).collect(Collectors.toList());

        System.out.println(customerDTOS);
        if (customers == null)
            throw new CustomerNotFoundException("customer not fount");
//        List<CustomerDTO> customerDTOS = customers.stream().map(cust->dtoMapper.fromCustomer(cust)).collect(Collectors.toList());
        CustomersDTO customersDTO= new CustomersDTO();
        customersDTO.setCustomerDTO(customerDTOS);
        customersDTO.setTotalpage(customers.getTotalPages());
        return customersDTO;
    }
}

```

XI. Controllers: RESTful Web services

CustomersRestController

```

package com.braimi.hamza.digital.banking.controllers;

import com.braimi.hamza.digital.banking.dtos.BankAccountDTO;
import com.braimi.hamza.digital.banking.dtos.CustomerDTO;
import com.braimi.hamza.digital.banking.dtos.CustomersDTO;
import com.braimi.hamza.digital.banking.services.BankAccountService;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import com.braimi.hamza.digital.banking.exceptions.CustomerNotFoundException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PostAuthorize;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController

```

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

    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping("/customers")
    public List<CustomerDTO> customers(@PathVariable int page) {
        return bankAccountService.listCustomers(page);
    }

    @PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
    @GetMapping("/customers/{id}/accounts")
    public List<BankAccountDTO> accountsListOfCustomer(@PathVariable(name = "id")
Long customerId) {
        return bankAccountService.bankAccountListOfCustomer(customerId);
    }

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

    @PostAuthorize("hasAuthority('ADMIN')")
    @GetMapping("/customers/search")
    public CustomersDTO getCustomerByName(@RequestParam(name = "keyword",
defaultValue = "") String keyword, @RequestParam(name = "page", defaultValue = "0")
int page) throws CustomerNotFoundException {
        CustomersDTO customersDTO = bankAccountService.getCustomerByName("%" +
keyword + "%", page);
        return customersDTO;
    }

    @PostAuthorize("hasAuthority('ADMIN')")
    @PostMapping("/customers")
    public CustomerDTO saveCustomer(@RequestBody CustomerDTO customerDTO) {
        return bankAccountService.saveCustomer(customerDTO);
    }

    @PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
    @PutMapping("/customers/{customerId}")
    public CustomerDTO updateCustomer(@PathVariable Long customerId, @RequestBody
CustomerDTO customerDTO) {
        customerDTO.setId(customerId);
        return bankAccountService.updateCustomer(customerDTO);
    }

    @PostAuthorize("hasAuthority('ADMIN')")
    @DeleteMapping("/customers/{customerId}")
    public void deleteCustomer(@PathVariable Long customerId) {
```

```
bankAccountService.deleteCustomer(customerId);
```

BankAccountRestController

```
package com.braimi.hamza.digital.banking.controllers;

import com.braimi.hamza.digital.banking.dtos.*;
import com.braimi.hamza.digital.banking.exceptions.BankAccountNotFound;
import com.braimi.hamza.digital.banking.services.BankAccountServiceImpl;
import com.braimi.hamza.digital.banking.exceptions.BalanceNotSufficientException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PostAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController

@CrossOrigin("*")
public class BankAccountRestController {

    @Autowired
    BankAccountServiceImpl bankAccountServiceImpl;

    @PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
    @GetMapping("/accounts/{accountId}")
    public BankAccountDTO getBankAccount(@PathVariable String accountId) throws
BankAccountNotFound {
        return bankAccountServiceImpl.getBankAccount(accountId);
    }

    @PostAuthorize("hasAuthority('ADMIN')")
    @GetMapping("/Account/searchAccount")
    public BankAccountsDTO getBankAccount(@RequestParam(name = "page", defaultValue
= "0") int page) throws BankAccountNotFound {
        return bankAccountServiceImpl.getBankAccountList(page);
    }

    @PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
    @GetMapping("/accounts/{accountId}/pageOperations")
    public AccountHistoryDTO getBankAccountOperations(@PathVariable String
accountId, @RequestParam(name = "page", defaultValue = "0") int page,
        @RequestParam(name = "size",
defaultValue = "5") int size) throws BankAccountNotFound {
        return bankAccountServiceImpl.getAccoutHistory(accountId, page,
size);
    }

    @PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
    @PostMapping("/operations/Debit")
    public AccountOperationDTO debit(@RequestBody AccountOperationDTO
accountOperarionDTO) throws BankAccountNotFound, BalanceNotSufficientException {
        this.bankAccountServiceImpl.debit(accountOperarionDTO.getAccountId(),
accountOperarionDTO.getAmount(), accountOperarionDTO.getDescription());
        return accountOperarionDTO;
    }
}
```

```
}

@PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
@PostMapping("/operations/Credit")
public AccountOperationDTO credit(@RequestBody AccountOperationDTO
accountOperarionDTO) throws BankAccountNotFound, BalanceNotSufficientException {
    this.bankAccountServiceImplementation.credit(accountOperarionDTO.getAccountId(),
accountOperarionDTO.getAmount(), accountOperarionDTO.getDescription());
    return accountOperarionDTO;
}

@PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
@PostMapping("/operations/Transfers")
public void transfers(@RequestParam(name = "idSource") String idSource,
@RequestParam(name = "idDestination") String idDestination, @RequestParam(name =
"amount") double amount) throws BankAccountNotFound, BalanceNotSufficientException
{
    this.bankAccountServiceImplementation.transfer(idSource, idDestination,
amount);
}

@PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
@PostMapping("/accounts/debit")
public DebitDTO debit(@RequestBody DebitDTO debitDTO) throws
BankAccountNotFound, BalanceNotSufficientException {
    this.bankAccountServiceImplementation.debit(debitDTO.getAccountId(),
debitDTO.getAmount(), debitDTO.getDescription());
    return debitDTO;
}

@PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
@PostMapping("/accounts/credit")
public CreditDTO credit(@RequestBody CreditDTO creditDTO) throws
BankAccountNotFound {
    this.bankAccountServiceImplementation.credit(creditDTO.getAccountId(),
creditDTO.getAmount(), creditDTO.getDescription());
    return creditDTO;
}

@PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
@PostMapping("/accounts/transfer")
public void transfer(@RequestBody TransferRequestDTO transferRequestDTO) throws
BankAccountNotFound, BalanceNotSufficientException {
    this.bankAccountServiceImplementation.transfer(transferRequestDTO.getAccountSource (
), transferRequestDTO.getAccountDestination(), transferRequestDTO.getAmount());
}

@PostAuthorize("hasAuthority('ADMIN') or hasAuthority('CUSTOMER')")
@PutMapping("/accounts/{accountId}")
public BankAccountDTO updateAccount(@PathVariable String accountId,
@RequestParam BankAccountDTO bankAccountDTO) {
    bankAccountDTO.setId(accountId);
    return bankAccountServiceImplementation.updateBankAccount (bankAccountDTO);
}
```

```
}  
}
```

XII. Test Application

```
package com.braimi.hamza.digital.banking;  
  
import com.braimi.hamza.digital.banking.dtos.BankAccountDTO;  
import com.braimi.hamza.digital.banking.dtos.CustomerDTO;  
import com.braimi.hamza.digital.banking.dtos.SavingBankAccountDTO;  
import com.braimi.hamza.digital.banking.enums.AccountStatus;  
import com.braimi.hamza.digital.banking.enums.OperationType;  
import com.braimi.hamza.digital.banking.entities.AccountOperation;  
import com.braimi.hamza.digital.banking.entities.CurrentAccount;  
import com.braimi.hamza.digital.banking.entities.SavingAccount;  
import com.braimi.hamza.digital.banking.exceptions.CustomerNotFoundException;  
import com.braimi.hamza.digital.banking.repositories.AccountOperationRepository;  
import com.braimi.hamza.digital.banking.repositories.BankAccountRepository;  
import com.braimi.hamza.digital.banking.repositories.CustomerRepository;  
import com.braimi.hamza.digital.banking.security.entities.AppRole;  
import com.braimi.hamza.digital.banking.security.entities.AppUser;  
import com.braimi.hamza.digital.banking.security.services.AccountService;  
import com.braimi.hamza.digital.banking.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 org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import java.util.UUID;  
import java.util.stream.Stream;  
  
@SpringBootApplication  
public class DigitalBankingApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(DigitalBankingApplication.class, args);  
    }  
  
    @Bean  
    CommandLineRunner commandLineRunner(BankAccountService bankAccountService,  
AccountService accountService) {  
        return args -> {  
  
            accountService.addNewRole(new AppRole(null, "ADMIN"));  
            accountService.addNewRole(new AppRole(null, "CUSTOMER"));  
  
            accountService.addNewUser(new AppUser(null, "HAMZA", "1234", new
```

```

ArrayList<>());
    accountService.addNewUser(new AppUser(null, "admin", "1234", new
ArrayList<>());
    accountService.addNewUser(new AppUser(null, "Hafsa", "1234", new
ArrayList<>());
    accountService.addNewUser(new AppUser(null, "Moussa", "1234", new
ArrayList<>());

    accountService.addRoleToUser("Yassine", "CUSTOMER");
    accountService.addRoleToUser("admin", "ADMIN");
    accountService.addRoleToUser("Hafsa", "CUSTOMER");
    accountService.addRoleToUser("Moussa", "CUSTOMER");

    Stream.of("Yassine", "Hafsa", "Moussa").forEach(name -> {
        CustomerDTO customer = new CustomerDTO();
        customer.setName(name);
        customer.setEmail(name + "@hotmail.com");
        bankAccountService.saveCustomer(customer);
    });
    bankAccountService.listCustomers(1).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.getBankAccountList(1).getBankAccountDTOS();
    for (BankAccountDTO bankAccount : bankAccounts) {
        for (int i = 0; i < 10; i++) {
            String accountId;
            if (bankAccount instanceof SavingBankAccountDTO) {
                accountId = bankAccount.getId();
            } else {
                accountId = bankAccount.getId();
            }
            bankAccountService.credit(accountId, 10000 + Math.random() *
120000, "Credit");
            bankAccountService.debit(accountId, 1000 + Math.random() *
9000, "Debit");
        }
    }
};
}

@Bean
CommandLineRunner start(CustomerRepository customerRepository,
                        BankAccountRepository bankAccountRepository,
                        AccountOperationRepository accountOperationRepository)
{
    return args -> {
        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);

SavingAccount savingAccount = new SavingAccount();
savingAccount.setId(UUID.randomUUID().toString());
savingAccount.setBalance(Math.random() * 90000);
savingAccount.setCreatedAt(new Date());
savingAccount.setStatus(AccountStatus.CREATED);
savingAccount.setCustomer(cust);
savingAccount.setInterestRate(5.5);
bankAccountRepository.save(savingAccount);

});
bankAccountRepository.findAll().forEach(acc -> {
    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(acc);
        accountOperationRepository.save(accountOperation);
    }
});
});

}

@Bean
PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

}
```

XIII. Data Base

Customer Table

hamzaHost

bank

bankdb 160,0 KiB

- account_operation 32,0 KiB
- app_role 16,0 KiB
- app_user 16,0 KiB
- app_user_roles 48,0 KiB
- bank_account 32,0 KiB
- customer 16,0 KiB

bankdb.customer: 4 ligne(s) au total (environ)

id	email	name
1	Hamzaz@hotmail.com	Hamza
2	Hafsa@hotmail.com	Hafsa
3	Moussa@hotmail.com	Moussa kouna
4	hamido@hotmail.com	hamido

Bank_Account table

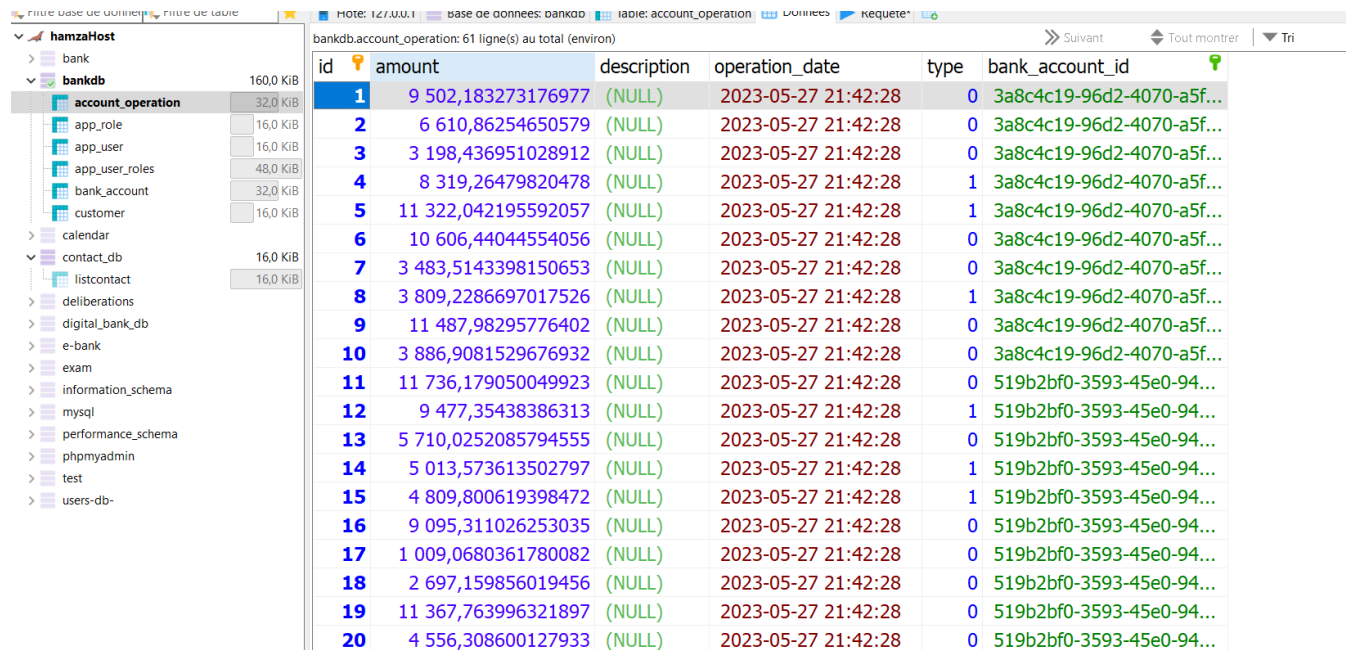
bank

bankdb 160,0 KiB

- account_operation 32,0 KiB
- app_role 16,0 KiB
- app_user 16,0 KiB
- app_user_roles 48,0 KiB
- bank_account 32,0 KiB
- customer 16,0 KiB
- calendar 16,0 KiB

type	id	balance	created_at	status	over_draft	interest_rate	customer_id
SA	3a8c4c19-96d2-4070...	32 392,10198748519	2023-05-27 21:42:28	CREATED	(NULL)	5,5	1
CA	519b2bf0-3593-45e0...	82 153,14421469708	2023-05-27 21:42:27	CREATED	9 000	(NULL)	1
SA	86fc9fc5-699f-44f5-a...	83 315,78922595465	2023-05-27 21:42:28	CREATED	(NULL)	5,5	3
CA	8784483b-5360-4e05...	436,90904822522356	2023-05-27 21:42:28	CREATED	9 000	(NULL)	2
SA	ba2bc58d-7922-4bdf-...	11 205,809994366235	2023-05-27 21:42:28	CREATED	(NULL)	5,5	2
CA	fb5c54ba-191b-40da-...	34 410,512278499074	2023-05-27 21:42:28	CREATED	9 000	(NULL)	3

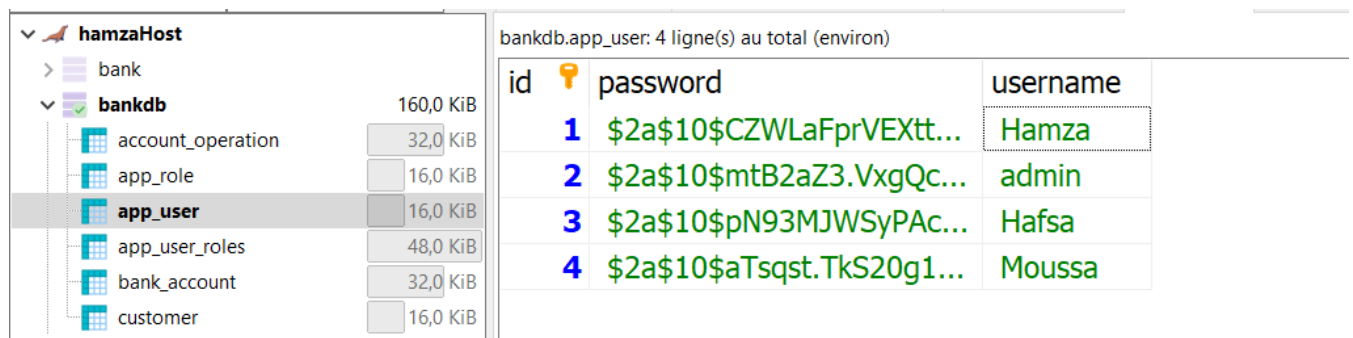
Account Operation table



id	amount	description	operation_date	type	bank_account_id
1	9 502,183273176977	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
2	6 610,86254650579	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
3	3 198,436951028912	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
4	8 319,26479820478	(NULL)	2023-05-27 21:42:28	1	3a8c4c19-96d2-4070-a5f...
5	11 322,042195592057	(NULL)	2023-05-27 21:42:28	1	3a8c4c19-96d2-4070-a5f...
6	10 606,44044554056	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
7	3 483,5143398150653	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
8	3 809,2286697017526	(NULL)	2023-05-27 21:42:28	1	3a8c4c19-96d2-4070-a5f...
9	11 487,98295776402	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
10	3 886,9081529676932	(NULL)	2023-05-27 21:42:28	0	3a8c4c19-96d2-4070-a5f...
11	11 736,179050049923	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...
12	9 477,35438386313	(NULL)	2023-05-27 21:42:28	1	519b2bf0-3593-45e0-94...
13	5 710,0252085794555	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...
14	5 013,573613502797	(NULL)	2023-05-27 21:42:28	1	519b2bf0-3593-45e0-94...
15	4 809,800619398472	(NULL)	2023-05-27 21:42:28	1	519b2bf0-3593-45e0-94...
16	9 095,311026253035	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...
17	1 009,0680361780082	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...
18	2 697,159856019456	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...
19	11 367,763996321897	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...
20	4 556,308600127933	(NULL)	2023-05-27 21:42:28	0	519b2bf0-3593-45e0-94...

➤ For the Security part I add two other tables

App_user table



id	password	username
1	\$2a\$10\$CZWLaFprVEXtt...	Hamza
2	\$2a\$10\$mtB2aZ3.VxgQc...	admin
3	\$2a\$10\$pN93MJWSyPac...	Hafsa
4	\$2a\$10\$aTsqst.TkS20g1...	Moussa

Role_table

> bank		
▼ bankdb	160,0 KiB	
account_operation	32,0 KiB	
app_role	16,0 KiB	
app_user	16,0 KiB	
app_user_roles	48,0 KiB	
bank_account	32,0 KiB	
customer	16,0 KiB	
> calendar		

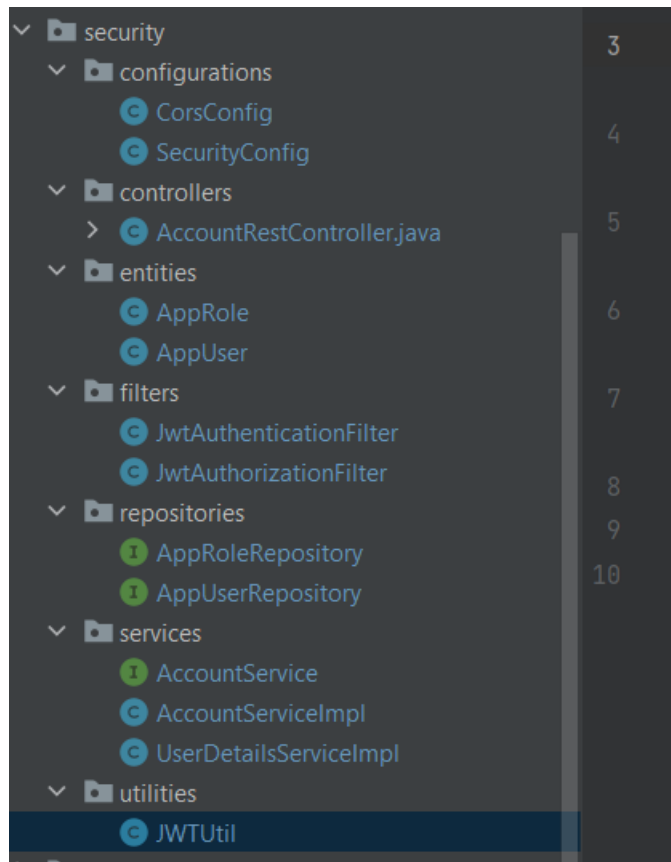
id	role_name
1	ADMIN
2	CUSTOMER

API DOCS: OPEN API (SWAGGER)

customer-rest-controller	
PUT	/customers/{customerId}
GET	/customers
POST	/customers
GET	/customers/{id}
DELETE	/customers/{id}
bank-account-rest-api	
GET	/accounts
GET	/accounts/{accountId}
GET	/accounts/{accountId}/pageOperations
GET	/accounts/{accountId}/operations

SECURITY PART

XIV. Security Structure



XV. Configuration

CorsConfig

```
package com.braimi.hamza.digital.banking.security.configurations;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration
public class CorsConfig {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurerAdapter() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
            }
        };
    }
}
```

```
        .allowedMethods("HEAD", "GET", "PUT", "POST", "DELETE",  
"PATCH");  
    }  
};  
}
```

SecurityConfig

```
package com.braimi.hamza.digital.banking.security.configurations;  
  
import com.braimi.hamza.digital.banking.security.filters.JwtAuthenticationFilter;  
import com.braimi.hamza.digital.banking.security.filters.JwtAuthorizationFilter;  
import com.braimi.hamza.digital.banking.security.repositories.AppUserRepository;  
import com.braimi.hamza.digital.banking.security.services.UserDetailsServiceImpl;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import  
org.springframework.security.config.annotation.authentication.builders.Authenticati  
onManagerBuilder;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import  
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import  
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigu  
rerAdapter;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import  
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilt  
er;  
import org.springframework.web.bind.annotation.CrossOrigin;  
  
@Configuration  
@EnableWebSecurity  
@CrossOrigin("*")  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
    private UserDetailsServiceImpl userDetailsService;  
    private AppUserRepository appUserRepository;  
  
    public SecurityConfig(UserDetailsServiceImpl userDetailsService,  
AppUserRepository appUserRepository) {  
        this.userDetailsService = userDetailsService;  
        this.appUserRepository = appUserRepository;  
    }  
  
    @Override  
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
        auth.userDetailsService(userDetailsService);  
    }  
  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http.csrf().disable();
```

```
http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.cors();
    http.headers().frameOptions().disable();
    http.authorizeRequests().antMatchers("/h2-
console/**", "/refreshToken/**", "/login/**").permitAll();
    http.addFilter(new JwtAuthenticationFilter(authenticationManagerBean(),
appUserRepository));
    http.addFilterBefore(new JwtAuthorizationFilter(),
UsernamePasswordAuthenticationFilter.class);
}

@Bean
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}
}
```

XVI. Entities

➤ App_user

```
package com.braimi.hamza.digital.banking.security.entities;

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

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AppUser {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;
    @ManyToMany(fetch = FetchType.EAGER)
    private Collection<AppRole> roles=new ArrayList<>();
}
```

➤ App_Role

```
package com.braimi.hamza.digital.banking.security.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AppRole {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String roleName;
}
```

XVII. Repositories

```
3 pages  👤 Your Name
public interface AppRoleRepository extends JpaRepository<AppRole, Long> {
    1 usage  👤 Your Name
    AppRole findAppRoleByRoleName(String username);
}
```

```
9 pages  👤 Your Name
public interface AppUserRepository extends JpaRepository<AppUser, Long> {
    2 usages  👤 Your Name
    AppUser findAppUserByUsername(String username);
}
```

XVIII. Service

```
package com.braimi.hamza.digital.banking.security.services;

import com.braimi.hamza.digital.banking.security.entities.AppRole;
import com.braimi.hamza.digital.banking.security.entities.AppUser;

import java.util.List;

public interface AccountService {
```

```
AppUser addNewUser(AppUser appUser);
AppRole addNewRole(AppRole appRole);
void addRoleToUser(String username, String rolename);
AppUser loadUserByUsername(String username);
List<AppUser> listUsers();
}
```

```
package com.braimi.hamza.digital.banking.security.services;

import com.braimi.hamza.digital.banking.security.entities.AppRole;
import com.braimi.hamza.digital.banking.security.entities.AppUser;
import com.braimi.hamza.digital.banking.security.repositories.AppRoleRepository;
import com.braimi.hamza.digital.banking.security.repositories.AppUserRepository;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@Transactional
public class AccountServiceImpl implements AccountService {
    private AppUserRepository appUserRepository;
    private AppRoleRepository appRoleRepository;
    private PasswordEncoder passwordEncoder;

    public AccountServiceImpl(AppUserRepository appUserRepository,
AppRoleRepository appRoleRepository, PasswordEncoder passwordEncoder) {
        this.appUserRepository = appUserRepository;
        this.appRoleRepository = appRoleRepository;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public AppUser addNewUser(AppUser appUser) {
        String pw = appUser.getPassword();
        appUser.setPassword(passwordEncoder.encode(pw));
        return appUserRepository.save(appUser);
    }

    @Override
    public AppRole addNewRole(AppRole appRole) {
        return appRoleRepository.save(appRole);
    }

    @Override
    public void addRoleToUser(String username, String rolename) {
        AppUser appUser = appUserRepository.findAppUserByUsername(username);
        AppRole appRole = appRoleRepository.findAppRoleByRoleName(rolename);
        appUser.getRoles().add(appRole);
    }

    @Override
    public AppUser loadUserByUsername(String username) {
```

```
        return appUserRepository.findAppUserByUsername(username);
    }

    @Override
    public List<AppUser> listUsers() {
        return appUserRepository.findAll();
    }
}
```

```
package com.braimi.hamza.digital.banking.security.services;

import com.braimi.hamza.digital.banking.security.entities.AppUser;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Collection;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    private AccountService accountService;

    public UserDetailsServiceImpl(AccountService accountService) {
        this.accountService = accountService;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        AppUser appUser = accountService.loadUserByUsername(username);
        Collection<GrantedAuthority> authorities = new ArrayList<>();
        appUser.getRoles().forEach(r->{
            authorities.add(new SimpleGrantedAuthority(r.getRoleName()));
        });
        return new User(appUser.getUsername(), appUser.getPassword(), authorities);
    }
}
```


XIX. Filters Package

➤ JwtAuthenticationFilter

```
package com.braimi.hamza.digital.banking.security.filters;

import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.braimi.hamza.digital.banking.security.repositories.AppUserRepository;
import com.braimi.hamza.digital.banking.security.utilities.JWTUtil;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private AuthenticationManager authenticationManager;

    private AppUserRepository appUserRepository;

    public JwtAuthenticationFilter(AuthenticationManager authenticationManager,
AppUserRepository appUserRepository) {
        this.authenticationManager = authenticationManager;
        this.appUserRepository = appUserRepository;
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response) throws AuthenticationException {
        response.setHeader("Access-Control-Allow-Origin",
request.getHeader("Origin"));
        response.setHeader("Access-Control-Allow-Credentials", "true");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS,
DELETE");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "Content-Type, Accept,
```

```

X-Requested-With, remember-me");
    String username =null;
    String password =null;

    try {
        Map<String, String> requestMap = new
ObjectMapper().readValue(request.getInputStream(), Map.class);
        username = requestMap.get("username");
        password = requestMap.get("password");

    }catch (Exception e){

    }

    UsernamePasswordAuthenticationToken authenticationToken =
        new UsernamePasswordAuthenticationToken(username,password);
    return authenticationManager.authenticate(authenticationToken);
}

@Override
protected void successfulAuthentication(HttpServletRequest request,
    HttpServletResponse response, FilterChain chain, Authentication authResult) throws
IOException, ServletException {
    User user =(User) authResult.getPrincipal();
    Algorithm algorithm = Algorithm.HMAC256(JWTUtil.SECRET);
    String jwtAccessToken = JWT.create()
        .withSubject(user.getUsername())
        .withExpiresAt(new
Date(System.currentTimeMillis()+JWTUtil.EXPIRE_ACCESS_TOKEN))
        .withIssuer(request.getRequestURL().toString())
        .withClaim("roles",user.getAuthorities().stream().map(ga-
>ga.getAuthority()).collect(Collectors.toList()))
        .sign(algorithm);

    String jwtRefreshToken = JWT.create()
        .withSubject(user.getUsername())
        .withExpiresAt(new
Date(System.currentTimeMillis()+JWTUtil.EXPIRE_REFRESH_TOKEN))
        .withIssuer(request.getRequestURL().toString())
        .sign(algorithm);

    Map<String,String> idToken = new  HashMap<>();
    idToken.put("jwt",jwtAccessToken);
    idToken.put("refreshToken",jwtRefreshToken);
    idToken.put("roles",user.getAuthorities().stream().map(ga-
>ga.getAuthority()).collect(Collectors.toList()).toString());
    if
    (user.getAuthorities().stream().map(GrantedAuthority::getAuthority).collect(Collectors.toList()).toString().contains("CUSTOMER")) {
        System.out.println(user.getUsername());
        //idToken.put("id",
String.valueOf(appUserRepository.findAppUserByUsername(user.getUsername().trim()).getId()));
        idToken.put("id", String.valueOf(1));
    }
    response.setContentType("application/json");

```

```
        new ObjectMapper().writeValue(response.getOutputStream(), idToken);
    }
}
```

➤ JwtAuthorizationsFilter

```
package com.braimi.hamza.digital.banking.security.filters;

import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.braimi.hamza.digital.banking.security.utilities.JWTUtil;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;

public class JwtAuthorizationFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {

        response.setHeader("Access-Control-Allow-Origin", request.getHeader("Origin"));
        response.setHeader("Access-Control-Allow-Credentials", "true");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, DELETE");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "Content-Type, Accept, X-Requested-With, remember-me");

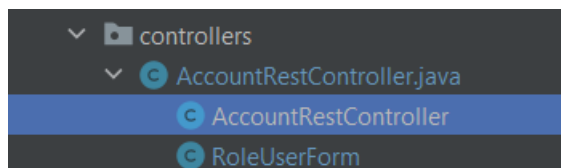
        if (request.getServletPath().equals("/refreshToken")) {
            filterChain.doFilter(request, response);
        } else {
            String authorizationToken = request.getHeader(JWTUtil.AUTH_HEADER);
            if (authorizationToken != null && authorizationToken.startsWith(JWTUtil.PREFIX)) {
                try {
                    String jwt = authorizationToken.substring(JWTUtil.PREFIX.length());
                    Algorithm algorithm = Algorithm.HMAC256(JWTUtil.SECRET);
                    JWTVerifier jwtVerifier = JWT.require(algorithm).build();
```

```
DecodedJWT decodedJWT = jwtVerifier.verify(jwt);
String username = decodedJWT.getSubject();
String[] roles =
decodedJWT.getClaim("roles").asArray(String.class);
Collection<GrantedAuthority> authorities = new ArrayList<>();
for (String r:roles){
    authorities.add(new SimpleGrantedAuthority(r));
}
UsernamePasswordAuthenticationToken authenticationToken =
    new
UsernamePasswordAuthenticationToken(username,null,authorities);

SecurityContextHolder.getContext().setAuthentication(authenticationToken);
filterChain.doFilter(request,response);

} catch (Exception e){
    response.setHeader("error-message",e.getMessage());
    response.sendError(HttpServletResponse.SC_FORBIDDEN);
}
} else {
    filterChain.doFilter(request,response);
}
}
}
```

XX. Controllers



```
package com.braimi.hamza.digital.banking.security.controllers;

import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.braimi.hamza.digital.banking.security.services.AccountService;
import com.braimi.hamza.digital.banking.security.utilities.JWTUtil;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.Data;
import com.braimi.hamza.digital.banking.security.entities.AppRole;
import com.braimi.hamza.digital.banking.security.entities.AppUser;
import org.springframework.security.access.prepost.PostAuthorize;
import org.springframework.web.bind.annotation.*;
```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.security.Principal;
import java.util.*;
import java.util.stream.Collectors;

@RestController
@CrossOrigin("*")
public class AccountRestController {
    AccountService accountService;
    public AccountRestController(AccountService accountService){
        this.accountService = accountService;
    }

    @GetMapping(path = "/users")
    @PostAuthorize("hasAuthority('USER')")
    public List<AppUser>appUsers() {
        return accountService.listUsers();
    }

    @PostMapping(path = "/users")
    @PostAuthorize("hasAuthority('ADMIN')")
    public AppUser saveUser(@RequestBody AppUser appUser) {
        return accountService.addNewUser(appUser);
    }

    @PostMapping(path = "/roles")
    @PostAuthorize("hasAuthority('ADMIN')")
    public AppRole saveRole(@RequestBody AppRole appRole) {
        return accountService.addNewRole(appRole);
    }

    @PostMapping(path = "/addRoleToUser")
    public void addRoleToUser(@RequestBody RoleUserForm roleUserForm) {
accountService.addRoleToUser(roleUserForm.getUsername(),roleUserForm.getRoleName())
;
    }

    @GetMapping(path = "/refreshToken")
    public void refreshToken(HttpServletRequest request, HttpServletResponse
response) throws Exception {
        String authToken = request.getHeader(JWTUtil.AUTH_HEADER);
        if (authToken!=null && authToken.startsWith(JWTUtil.PREFIX)){
            try {
                String jwt = authToken.substring(JWTUtil.PREFIX.length());
                Algorithm algorithm = Algorithm.HMAC256(JWTUtil.SECRET);
                JWTVerifier jwtVerifier = JWT.require(algorithm).build();
                DecodedJWT decodedJWT = jwtVerifier.verify(jwt);
                String username = decodedJWT.getSubject();
                AppUser appUser = accountService.loadUserByUsername(username);
                String jwtAccessToken = JWT.create()
                    .withSubject(appUser.getUsername())
                    .withExpiresAt(new
Date(System.currentTimeMillis()+JWTUtil.EXPIRE_ACCESS_TOKEN))
                    .withIssuer(request.getRequestURL().toString())

```

```

        .withClaim("roles",appUser.getRoles().stream().map(r->r.getRoleName()).collect(Collectors.toList()))
        .sign(algorithm);

        Map<String,String> idToken = new HashMap<>();
        idToken.put("jwt",jwtAccessToken);
        idToken.put("refreshToken",jwt);
        idToken.put("roles",appUser.getRoles().stream().map(r->r.getRoleName()).collect(Collectors.toList()).toString());
        if(appUser.getRoles().stream().map(AppRole::getRoleName)
        .collect(Collectors.toList()).toString().contains("CUSTOMER")){
            idToken.put("id", appUser.getId().toString());
        }
        response.setContentType("application/json");
        new ObjectMapper().writeValue(response.getOutputStream(),idToken);

    }catch (Exception e){
        throw e;
    }
    }else {
        throw new RuntimeException("Refresh token Required!!!");
    }
}

@GetMapping(path = "/profile")
public AppUser profile(Principal principal){
    return accountService.loadUserByUsername(principal.getName());
}
}

@Data
class RoleUserForm{
    private String username;
    private String roleName;
}

```

XXI. JWTUtil

```

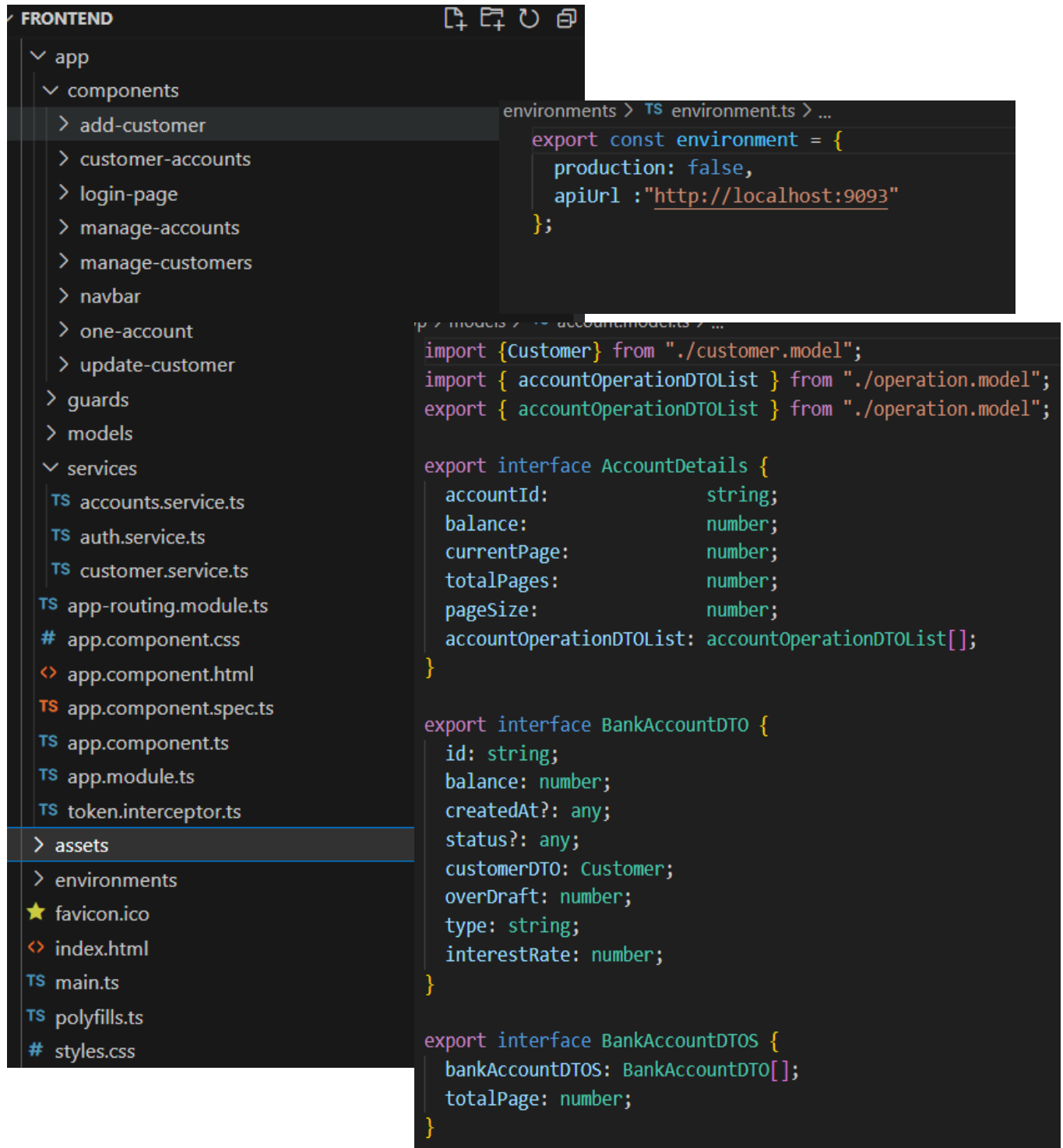
package com.braimi.hamza.digital.banking.security.utilities;

public class JWTUtil {
    public static final String SECRET = "K$74Xv2zSb3%3zsa3jazC@X&ZQPu$wJp";
    public static final String AUTH_HEADER = "Authorization";
    public static final long EXPIRE_ACCESS_TOKEN = 60 * 60 * 1000;
    public static final long EXPIRE_REFRESH_TOKEN = 480 * 60 * 1000;
    public static final String PREFIX = "Bearer ";
}

```

FRONTEND PART

XXII. Project Structure



The screenshot displays the project structure of the frontend part of a digital banking application. The left sidebar shows a tree view of the 'FRONTEND' directory. The main editor area shows the content of 'environments/environment.ts' and 'models/account.model.ts'.

Project Structure (Left Sidebar):

- FRONTEND
 - app
 - components
 - add-customer
 - customer-accounts
 - login-page
 - manage-accounts
 - manage-customers
 - navbar
 - one-account
 - update-customer
 - guards
 - models
 - services
 - accounts.service.ts
 - auth.service.ts
 - customer.service.ts
 - app-routing.module.ts
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
 - token.interceptor.ts
 - assets
 - environments
 - favicon.ico
 - index.html
 - main.ts
 - polyfills.ts
 - styles.css

Environment Configuration (environments/environment.ts):

```
export const environment = {  
  production: false,  
  apiUrl: "http://localhost:9093"  
};
```

Model Definitions (models/account.model.ts):

```
import { Customer } from "../customer.model";  
import { accountOperationDTOList } from "../operation.model";  
export { accountOperationDTOList } from "../operation.model";  
  
export interface AccountDetails {  
  accountId: string;  
  balance: number;  
  currentPage: number;  
  totalPages: number;  
  pageSize: number;  
  accountOperationDTOList: accountOperationDTOList[];  
}  
  
export interface BankAccountDTO {  
  id: string;  
  balance: number;  
  createdAt?: any;  
  status?: any;  
  customerDTO: Customer;  
  overDraft: number;  
  type: string;  
  interestRate: number;  
}  
  
export interface BankAccountDTOS {  
  bankAccountDTOS: BankAccountDTO[];  
  totalPage: number;  
}
```

```
export interface Customer {
  id: number;
  name: string;
  email: string;
}

export interface CustomerDTOS {
  customerDTO: Customer[];
  totalpage: number;
}
```

```
export interface accountOperationDTOList {
  id: number;
  operationDate: string;
  amount: number;
  type: string;
  description: string;
  accountId: string;
}
```

```
export class Tokens {
  jwt!: string;
  refreshToken!: string;
  roles!: string;
}
```

XXIII. Services

➤ Account Service

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

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

  constructor(private http: HttpClient) { }

  getAccount(accountId: string, page: number, size: number): Observable<AccountDetails> {
    return
    this.http.get<AccountDetails>(environment.apiUrl + "/accounts/" + accountId + "/pageOperations?page=" + page + "&size=" + size);
  }

  makeDebit(accountId: string, amount: number, description: string) {
    let data1 = { accountId: accountId, amount: amount, description: description };
    return this.http.post(environment.apiUrl + "/accounts/debit", data1);
  }
}
```



```

makeCredit(accountId:string, amount:number, description:string){
  let data1 = {accountId:accountId, amount:amount,description:description}
  return this.http.post(environment.apiUrl+"/accounts/credit", data1);
}

makeTransfer(accountDestination:string, accountSource:string, amount:number,
description:string){
  let data1 = {accountSource, accountDestination,amount,description:description}
  return this.http.post(environment.apiUrl+"/accounts/transfer", data1);
}

updateAccount(bankAccount: BankAccountDTO):Observable<any> {
  return
this.http.put<any>(environment.apiUrl+"/accounts/"+bankAccount.id,bankAccount);
}
}

```

➤ Auth.service

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { of, Observable } from 'rxjs';
import { catchError, mapTo, tap } from 'rxjs/operators';
import { Tokens } from "../models/tokens";
import { environment } from "../../environments/environment";
import { Router } from "@angular/router";
import Swal from "sweetalert2";

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

  private readonly JWT_TOKEN = 'JWT_TOKEN';
  private readonly REFRESH_TOKEN = 'REFRESH_TOKEN';
  private readonly ROLES = 'ROLES';
  private loggedInUser!: string;

  constructor(private http: HttpClient, private router: Router) {}

  login(user: { username: string, password: string }): Observable<boolean> {
    return this.http.post<any>(environment.apiUrl+'/login', user)
      .pipe(

```

```
tap(tokens => this.doLoginUser(user.username, tokens)),
mapTo(true),
catchError(error => {
  Swal.fire({
    position: 'center',
    icon: 'error',
    title: "The given information are incorrect, please try again !",
    showConfirmButton: false,
    timer: 2500
  });
  return of(false);
}));
}

logout() {
  this.doLogoutUser()
}

isLoggedIn() {
  return !!this.getJwtToken();
}

refreshToken() {
  return this.http.post<any>(environment.apiUrl+'/refresh', {
    'refreshToken': this.getRefreshToken()
  }).pipe(tap((tokens: Tokens) => {
    this.storeJwtToken(tokens.jwt,tokens.roles);
  }));
}

getJwtToken() {
  return localStorage.getItem(this.JWT_TOKEN);
}

private doLoginUser(username: string, tokens: Tokens) {
  this.loggedUser = username;
  this.storeTokens(tokens);
}

private doLogoutUser() {
  this.loggedUser = 'null';
  this.removeTokens();
}

private getRefreshToken() {
```

```

    return localStorage.getItem(this.REFRESH_TOKEN);
  }

  private storeJwtToken(jwt: string, roles: string) {
    localStorage.setItem(this.JWT_TOKEN, jwt);
    localStorage.setItem(this.ROLES, roles);
    localStorage.setItem("id", "1");
  }

  private storeTokens(tokens: Tokens,) {
    localStorage.setItem(this.JWT_TOKEN, tokens.jwt);
    localStorage.setItem(this.REFRESH_TOKEN, tokens.refreshToken);
    localStorage.setItem(this.ROLES, tokens.roles);
    localStorage.setItem("id", "1");
  }

  private removeTokens() {
    localStorage.removeItem(this.JWT_TOKEN);
    localStorage.removeItem(this.REFRESH_TOKEN);
    localStorage.removeItem(this.ROLES);
    localStorage.removeItem("id");
  }
}

```

➤ Customer.service

```

import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {Observable} from "rxjs";
import {Customer, CustomerDTOS} from "../models/customer.model";
import {environment} from "../../environments/environment";
import {BankAccountDTOS} from "../models/account.model";

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

  constructor(private http: HttpClient) {
  }
}

```

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

public getOneCustomer(id: number): any {
    return this.http.get<any>(environment.apiUrl + "/customers/" + id);
}

public searchCustomers(keyword: string, page: number): Observable<CustomerDTOS> {
    return this.http.get<CustomerDTOS>(environment.apiUrl +
"/customers/search?keyword=" + keyword + "&page=" + page)
}

public searchAccountByCustomer(page: number): Observable<BankAccountDTOS> {
    return this.http.get<BankAccountDTOS>(environment.apiUrl +
"/Account/searchAccount?page=" + page)
}

public saveCustomer(customer: Customer): Observable<Array<Customer>> {
    return this.http.post<Array<Customer>>(environment.apiUrl + "/customers",
customer);
}

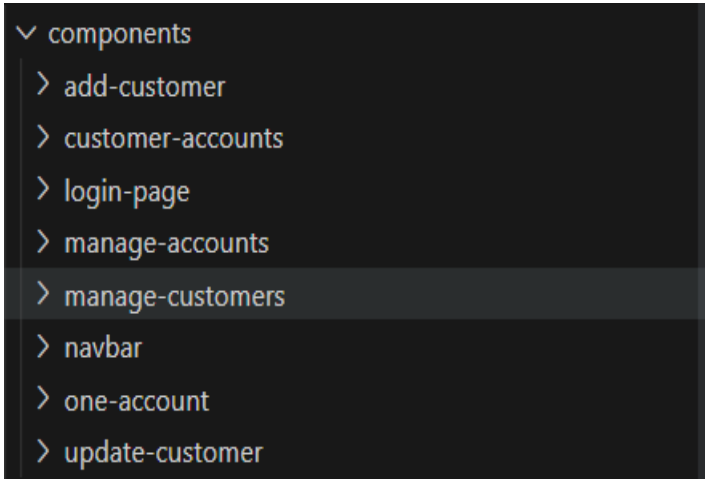
public deleteCustomer(id: number) {
    return this.http.delete(environment.apiUrl + "/customers/" + id);
}

updateCustomer(customer: Customer): Observable<Array<Customer>> {
    return this.http.put<Array<Customer>>(environment.apiUrl + "/customers/" +
customer.id, customer);
}

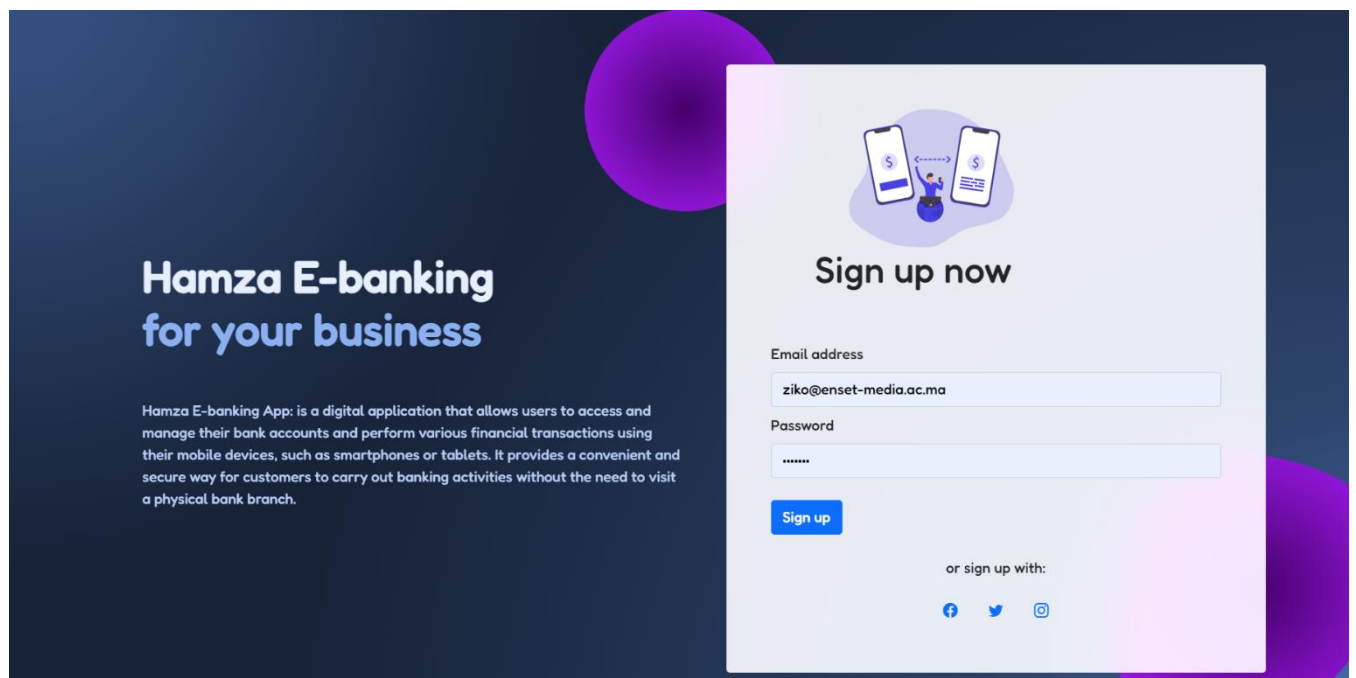
getCustomerById(id: number): Observable<Customer> {
    return this.http.get<Customer>(environment.apiUrl + "/customers/" + id);
}

public getAccountsOfCustomer(id: number): any {
    return this.http.get<any>(environment.apiUrl + "/customers/" + id + "/accounts");
}
}
```

XXIV. Components



XXV. Login Page



HTML Part

```
<section class="background-radial-gradient overflow-hidden">
  <style>
    .background-radial-gradient {
      background-color: hsl(218, 41%, 15%);
      background-image: radial-gradient(650px circle at 0% 0%,
        hsl(218, 41%, 35%) 15%,
        hsl(218, 41%, 30%) 35%,
        hsl(218, 41%, 20%) 75%,
        hsl(218, 41%, 19%) 80%,
        transparent 100%),
        radial-gradient(1250px circle at 100% 100%,
        hsl(218, 41%, 45%) 15%,
        hsl(218, 41%, 30%) 35%,
        hsl(218, 41%, 20%) 75%,
        hsl(218, 41%, 19%) 80%,
        transparent 100%);
    }

    #radius-shape-1 {
      height: 220px;
      width: 220px;
      top: -60px;
      left: -130px;
      background: radial-gradient(#44006b, #ad1fff);
      overflow: hidden;
    }

    #radius-shape-2 {
      border-radius: 38% 62% 63% 37% / 70% 33% 67% 30%;
      bottom: -60px;
      right: -110px;
      width: 300px;
      height: 300px;
      background: radial-gradient(#44006b, #ad1fff);
      overflow: hidden;
    }

    .bg-glass {
      background-color: hsla(0, 0%, 100%, 0.9) !important;
      backdrop-filter: saturate(200%) blur(25px);
    }
  </style>

```

```

</style>

<div class="container px-4 py-5 px-md-5 text-center text-lg-start my-5">
  <div class="row gx-lg-5 align-items-center mb-5">
    <div class="col-lg-6 mb-5 mb-lg-0" style="z-index: 10">
      <h1 class="my-5 display-5 fw-bold ls-tight" style="color: hsl(218, 81%, 95%)">
        Hamza E-banking <br />
        <span style="color: hsl(218, 81%, 75%)">for your business</span>
      </h1>
      <p class="mb-4 opacity-70" style="color: hsl(218, 81%, 85%)">
        Hamza E-banking App:
        is a digital application that allows users to access and manage their bank
        accounts and perform various financial transactions using their mobile devices, such
        as smartphones or tablets. It provides a convenient and secure way for customers to
        carry out banking activities without the need to visit a physical bank branch.
      </p>
    </div>

    <div class="col-lg-6 mb-5 mb-lg-0 position-relative">
      <div id="radius-shape-1" class="position-absolute rounded-circle shadow-5-
strong"></div>
      <div id="radius-shape-2" class="position-absolute shadow-5-strong"></div>

      <div class="card bg-glass">
        <div class="card-body px-4 py-5 px-md-5">
          <svg id="Layer_1" data-name="Layer 1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
          viewBox="0 0 363.74 303.29" width="350" height="150" class="illustration
styles_illustrationTablet__1DW0a">
            <defs>
              <linearGradient id="linear-gradient" x1="206.7" y1="169.12" x2="203.96"
y2="145.92"
                gradientTransform="matrix(1, 0, 0, -1, 0, 352)"
gradientUnits="userSpaceOnUse">
                <stop offset="0.01"></stop>
                <stop offset="0.08" stop-opacity="0.69"></stop>
                <stop offset="0.21" stop-opacity="0.32"></stop>
                <stop offset="1" stop-opacity="0"></stop>
              </linearGradient>
            </defs>
            <title>13</title>
            <path
              d="M195.09,31.58a102.82,102.82,0,0,0-19.56,2.2c-24.14,5.37-44.09,21.89-
62.17,38.75-15.12,14.1-30,29.35-37.87,48.48-7.52,18.35-8,39.13-16.71,57-5.51,11.33-
14,20.87-20.75,31.52-11.39,18-17.23,41.84-
7.12,60.64,5.76,10.69,15.85,18.3,25.92,25.08,25.79,17.36,54.25,31.87,85,36.4,11.74,1.7

```

```

3,23.64,2,35.5,2.57,10.23.51,20.58,1.27,30.64-.64,25.17-4.78,44.23-25.14,67.56-
35.72,16.59-7.53,35-9.93,52.62-14.45s35.64-11.86,46.72-26.34c9.89-12.93,12.66-
29.36,14.38-45.13,1.66-15.19-7.64-29.41-13.33-43.38-12.3-30.19-25.15-60.74-47.6-
85A161.55,161.55,0,0,0,242.23,35.3a212.17,212.17,0,0,0-37.46-
3.73C201.55,31.51,198.32,31.49,195.09,31.58Z"
    transform="translate(-25.71 -31.52)" fill="#4942E4" opacity="0.18"
style="isolation: isolate;"></path>
    <path
      d="M177.2,158.7a2.48,2.48,0,0,0-2-2.06,3.45,3.45,0,0,0-2.84.8,3,3,0,0,0-
1,1.57,4.35,4.35,0,0,0,.11,1.93,7.11,7.11,0,0,0,1,2.51c1,1.42,3.36,1.92,4.08,0a9.76,9.
76,0,0,0,.07-1.87C176.73,160.6,177.3,159.71,177.2,158.7Z"
      transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path
      d="M143,75.22l27.39,140.39a12.5,12.5,0,0,1-
9.87,14.66l99.75,242.12a12.5,12.5,0,0,1-14.68-9.83h0L57.68,91.87a12.5,12.5,0,0,1,9.86-
14.66h0L128.3,65.35A12.5,12.5,0,0,1,143,75.17v0Z"
      transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <rect x="72.28" y="71.83" width="83.47" height="163.52" rx="12.08"
      transform="translate(-53.01 -6.84) rotate(-11.04)"
      fill="#473f47"></rect>
    <path
      d="M139.6,79.16l25.84,132.45a11.54,11.54,0,0,1-9.25,13.44l-
54.54,10.64a11.65,11.65,0,0,1-6-
.38A11.37,11.37,0,0,1,88,226.72L62.17,94.26a11.55,11.55,0,0,1,9.27-
13.44L80.65,79,81,80.68a5.52,5.52,0,0,0,4.4,4.26,5.66,5.66,0,0,0,2.17,0L112.89,80a5.61
,5.61,0,0,0,4.44-6.49L117,71.92l9-
1.74a11.73,11.73,0,0,1,8.51,1.57A11.29,11.29,0,0,1,139.6,79.16Z"
      transform="translate(-25.71 -31.52)" fill="#fff"></path>
    <circle cx="88.3" cy="116.15" r="18.81" fill="#4942E4" opacity="0.18"
style="isolation: isolate;"></circle>
    <path
      d="M115.59,161.53l-.65-3.36a9.73,9.73,0,0,1-5.37-.44l.34-
2.38a9,9,0,0,0,5,.44c2.38-.46,3.69-2.15,3.31-4s-1.88-2.7-4.54-3.21c-3.66-.65-6.06-
1.76-6.62-4.64s1-5.24,3.94-6.36l-.65-3.35,2.06-.4.63,3.22a9.13,9.13,0,0,1,4.55.31-
.37,2.39a8.28,8.28,0,0,0-4.41-.3c-2.55.5-3.21,2.24-
2.94,3.56.34,1.73,1.75,2.35,4.86,3,3.71.7,5.77,2.08,6.37,5,.51,2.62-.82,5.44-
4.12,6.69l.67,3.44Z"
      transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path
      d="M359.24,92.07,336.5,236.62a12.79,12.79,0,0,1-14.62,10.64l-62.56-
9.85a12.78,12.78,0,0,1-10.67-
14.59h0L271.4,78.25A12.78,12.78,0,0,1,286,67.6h0L62.56,9.84A12.78,12.78,0,0,1,359.24,9
2Z"
      transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <rect x="220.33" y="114.59" width="167.28" height="85.39" rx="12.36"

```



```

        transform="translate(75.65 401.61) rotate(-81.06)"
fill="#473f47"></rect>
    <path
        d="M354.6,94.66,333.14,231a11.81,11.81,0,0,1-13.59,9.69l-56.15-
8.84a12,12,0,0,1-5.62-2.45,11.62,11.62,0,0,1-4.36-
10.94L274.88,82.12a11.81,11.81,0,0,1,13.61-9.68L298,73.93l-
.27,1.71a5.68,5.68,0,0,0,2.75,5.64,6.24,6.24,0,0,0,2.09.75l26.09,4.1a5.74,5.74,0,0,0,6
.54-4.69l.25-1.62,9.21,1.45a12,12,0,0,1,7.63,4.48A11.55,11.55,0,0,1,354.6,94.66Z"
        transform="translate(-25.71 -31.52)" fill="#fff"></path>
    <path
        d="M237,143.3h-5a1.5,1.5,0,0,1,0-3h5a1.5,1.5,0,0,1,0,3Zm-10,0h-
5a1.5,1.5,0,0,1,0-3h5a1.5,1.5,0,0,1,0,3Zm-10,0h-5a1.5,1.5,0,0,1,0-
3h5a1.5,1.5,0,0,1,0,3Zm-10,0h-5a1.5,1.5,0,0,1,0-3h5a1.5,1.5,0,0,1,0-
3h5a1.5,1.5,0,0,1,0,3Zm-10,0h-5a1.5,1.5,0,0,1,0-3h5a1.5,1.5,0,0,1,0-
3h5a1.5,1.5,0,0,1,0,3Z"
        transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path
        d="M177.62,149.21a1.41,1.41,0,0,1-1-.37l-6.76-5.91a1.5,1.5,0,0,1-.11-
2.13,1.14,1.14,0,0,1,.11-.11l6.76-6.19a1.5,1.5,0,0,1,2,2.21l-
5.52,5.06,5.49,4.81a1.5,1.5,0,0,1,.16,2.11l0A1.47,1.47,0,0,1,177.62,149.21Z"
        transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path
        d="M241.74,149.21a1.48,1.48,0,0,1-1.13-.51,1.51,1.51,0,0,1,.14-2.12l5.5-
4.81-5.52-5.06a1.5,1.5,0,0,1,2-
2.21l6.76,6.19a1.53,1.53,0,0,1,.49,1.12,1.55,1.55,0,0,1-.51,1.12l-
6.76,5.91A1.5,1.5,0,0,1,241.74,149.21Z"
        transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path d="M233.93,250.16a29.5,29.5,0,1,1-59,0,21.24,21.24,0,0,1,.08-
2.26,29.49,29.49,0,0,1,58.89,2.26Z"
        transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path
        d="M192.05,263.55c-1,4.26.22,8.81-.66,13.05a29.6,29.6,0,0,1-14.67-
16.37,18.51,18.51,0,0,1,.11-4.4,11.67,11.67,0,0,0,0-
4.71A10.83,10.83,0,0,0,175,247.9a29.45,29.45,0,0,1,12.32-21.77,1.52,1.52,0,0,1,0,.61-
1.26,5.61a16.76,16.76,0,0,1,0,5,5.76,5.76,0,0,1-2.74,4,15.44,15.44,0,0,0-
1.58.78,5,5,0,0,0-1.2,1.44,12.19,12.19,0,0,0-1.8,3.76,4.19,4.19,0,0,0,.94,3.89l8.22-
1.6a2.62,2.62,0,0,1,2.49.37c.71.77.27,2,.42,3.05.2,1.32,1.38,2.28,1.93,3.51C193.75,258
.77,192.59,261.23,192.05,263.55Z"
        transform="translate(-25.71 -31.52)" fill="#020202" opacity="0.14"
style="isolation: isolate;"></path>
    <path
        d="M181.13,235a4.84,4.84,0,0,0,1.75,2,20.93,20.93,0,0,0,11.93,4.18A7.57,7.5
7,0,0,0,191,243c-1,1-1.21,3-.06,3.8a3.33,3.33,0,0,0,1.35.52,12.25,12.25,0,0,0,6.57-
.53,11.19,11.19,0,0,1,3.87-1.16,8.29,8.29,0,0,1,3.12.89,7.79,7.79,0,0,0,4.5.83c1.52-
.31,2.9-1.77,2.58-3.28-.08-.36-.24-.75-.06-1.06a1.21,1.21,0,0,1,.56-.41l14-6.2c.89-

```

```

.4,1.9-1,2-1.93s-.88-1.78-1.79-2.25c-4.81-2.51-10.42-2.83-15.84-3.11-9.62-
.47a83.91,83.91,0,0,0-14,.1c-1.61.19-4.46.21-5.69,1.41A5.66,5.66,0,0,0,181.13,235Z"
    transform="translate(-25.71 -31.52)" fill="#020202" opacity="0.4"
style="isolation: isolate;"></path>
    <path
      d="M233.93,250.16a29.44,29.44,0,0,1-11.23,23.16,3.66,3.66,0,0,1-1-
1,3,3,0,0,1,.8-4.17l0,0c-.5.26-2.45-.38-3-.47l-3-.47c-1.46-.23-4-.13-4.85-
1.6a3.38,3.38,0,0,0-.34-.58,2,2,0,0,0-.63-.43c-.91-.44-2-.95-2.2-1.94-.15-.73.21-1.56-
.16-2.2-.12-.21-.3-.36-.42-.56a1.89,1.89,0,0,1,.21-1.85c1.28-2.24,3.93-3.32,6.47-
3.75s5.18-.37,7.65-1.09a34.35,34.35,0,0,1,4.66-1.43c.89-.12,2-.21,2.32-1,.19-.47,0-
1,.19-1.5.08-.2.21-.38.29-.58a1.22,1.22,0,0,0-.78-1.54l-.22-.05c.21-.55.38-1.26-.06-
1.65a1.16,1.16,0,0,0-1.08-.17,9.26,9.26,0,0,0-2.71,1.79,7.43,7.43,0,0,1-9.43-
1.11,2.61,2.61,0,0,1-.82-1.33,2.14,2.14,0,0,1,1.25-2.12,6,6,0,0,1,2.55-.5q-.49-3.21-1-
6.42c-4.91,1.31-6.9-3.36-7.6-7.24s2.06-5.26,5.61-
5.52A29.48,29.48,0,0,1,233.93,250.16Z"
      transform="translate(-25.71 -31.52)" fill="#020202" opacity="0.14"
style="isolation: isolate;"></path>
    <path
      d="M195.69,192.51H214.6a4.93,4.93,0,0,1,5,4.86v.14s-3.6,14.71-
3.6,19.65c0,2.71.85,10.79.85,10.79H193.46s1.08-8.45,1.06-11.28c0-4.82-3.86-19.17-3.86-
19.17a4.93,4.93,0,0,1,4.87-5Z"
      transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
    <path
      d="M194.78,223.28c.94,0,1.87-.16,2.8-.24,1.21-.11,2.42-.09,3.64-
.08l19.32.24a5.75,5.75,0,0,1,2.14.29,2,2,0,0,1,1.32,1.57c.09.93-.69,1.69-
1.43,2.26a26.39,26.39,0,0,1-11.65,5.1c-6.52,1.11-13.21-.27-19.59-2a7.72,7.72,0,0,1-
2.87-1.23,19.22,19.22,0,0,1-2.1-2.49,8.22,8.22,0,0,1-1.55-
1.92C183.06,220.65,193.28,223.29,194.78,223.28Z"
      transform="translate(-25.71 -31.52)" fill="#473f47"></path>
    <path d="M208.06,195.27a2.16,2.16,0,0,1-2.35,1.88c-2.32-.11-2.48-1.88-2.48-
1.88v-9.21h4.83Z"
      transform="translate(-25.71 -31.52)" fill="#fca89e"></path>
    <path d="M203.23,190.61s3,4.72,4.83,2.45v-3.68Z" transform="translate(-25.71
-31.52)"
      fill="url(#linear-gradient)"></path>
    <path
      d="M201.19,188.27c1.21,3.75,7.23,3.86,9.25.88a8.23,8.23,0,0,0,.95-
6,20.76,20.76,0,0,0-1.84-5.85c-.62-1.16-1.25-2.19-2.65-2.36a8.35,8.35,0,0,0-
4.62,1,6.74,6.74,0,0,0-3,3.28,6.61,6.61,0,0,0-
.21,4.41,13.46,13.46,0,0,0,.91,2.1A18.46,18.46,0,0,1,201.19,188.27Z"
      transform="translate(-25.71 -31.52)" fill="#fca89e"></path>
    <path
      d="M203,240a8.12,8.12,0,0,1-2.21,1.23c-1.69.71-5,2-
4.55,3.5.68,2,3,1.82,4.61,1.35a30.47,30.47,0,0,0,6.94-3.42Z"
      transform="translate(-25.71 -31.52)" fill="#473f47"></path>
    <path

```

```

        d="M207.1,234.91c5.75-4.2,8.75-7.3,15.21-11.23,1.53-.93,2.59-1.21,5.25-.16,2.27.89,2,4.79.79,6.42-3.72,4.88-18,11.62-20.92,12.64-1.25.44-5.66-3.27-3.87-5.05A31.12,31.12,0,0,1,207.1,234.91Z"
        transform="translate(-25.71 -31.52)" fill="#473f47"></path>
    <path
        d="M204.44,243a19.68,19.68,0,0,0,3,3.12c2,1.5,3.29,2,4.87,1.27,2.3-1.07.19-3.32-1.59-5.11a24.92,24.92,0,0,0-4.53-3.73Z"
        transform="translate(-25.71 -31.52)" fill="#473f47"></path>
    <path d="M204.74,237.48l2.45,2a5.88,5.88,0,0,1,1.28,1.49c.8,1.49-2,4.53-3.75,2.74s-3.4-3.7-3.4-3.7Z"
        transform="translate(-25.71 -31.52)" fill="#fca89e"></path>
    <path
        d="M197.42,240.68c-6.68-3.36-10.78-4.58-15.07-10.63a7.09,7.09,0,0,1-1.11-5.35c.52-1.55,2.78-2.48,5.45-1.84,2.24.53,17.25,12.29,19.59,14.51,1,1-2.06,5.73-4.59,4.91A24.93,24.93,0,0,1,197.42,240.68Z"
        transform="translate(-25.71 -31.52)" fill="#473f47"></path>
    <path
        d="M211.17,176.91a5.18,5.18,0,0,1-1.12,1.66,6.45,6.45,0,0,1-3.59,1.75,25.64,25.64,0,0,1-6-.2,1.74,1.74,0,0,1-1.17,1.4,29.18,29.18,0,0,0,1.2,4.07c.48,1.31,1.18,2.51,1.71,3.79-1.64-1.13-2.89-3.06-4.08-4.62a8.69,8.69,0,0,1-1.51-2.49,3.06,3.06,0,0,1,.27-2.8,3.27,3.27,0,0,0,.65-.95,2.37,2.37,0,0,0,0-.92,4.9,4.9,0,0,1,1.7-3.85,9.69,9.69,0,0,1,3.84-2,8.47,8.47,0,0,1,3.77-.41,6.24,6.24,0,0,1,4.43,4.18A2.44,2.44,0,0,1,211.17,176.91Z"
        transform="translate(-25.71 -31.52)" fill="#473f47"></path>
    <path
        d="M199.89,185.8a1.61,1.61,0,0,0-1.06.68,1.17,1.17,0,0,0-.05.9,2.78,2.78,0,0,0,.46.8,1.62,1.62,0,0,0,.37.37,1.55,1.55,0,0,0,.81.2h.65a.62.62,0,0,0,.32-.09.68.68,0,0,0,.22-.31,2.27,2.27,0,0,0,0-1.41,2.52,2.52,0,0,0-.61-1.15C200.72,185.53,200.26,185.7,199.89,185.8Z"
        transform="translate(-25.71 -31.52)" fill="#fca89e"></path>
    <rect x="197.29" y="149.54" width="6.55" height="12.62" rx="1.24"
    fill="#473f47"></rect>
    <polygon points="172.05 161.79 157.38 147.72 151.31 131.76 146.42 132.7 152.21 151.29 168.61 173.74 172.05 161.79"
        fill="#4942E4"></polygon>
    <circle cx="279.69" cy="117.69" r="18.81" fill="#4942E4" opacity="0.18"
    style="isolation: isolate;"></circle>
    <path
        d="M302.24,162.81.51-3.39a9.78,9.78,0,0,1-4.91-2.21l1.12-2.13a9,9,0,0,0,4.6,2.1c2.39.36,4.19-.79,4.47-2.7s-.87-3.17-3.21-4.54c-3.23-1.84-5.12-3.69-4.68-6.6s2.68-4.6,5.84-4.66l.51-3.38,2.08.32-.49,3.24a9.2,9.2,0,0,1,4.19,1.81-1.15,2.13a8.2,8.2,0,0,0-4.05-1.75c-2.58-.39-3.78,1-4,2.36-.26,1.74.86,2.8,3.59,4.43,3.25,1.9,4.73,3.89,4.31,6.88-.4,2.64-2.59,4.86-6.12,4.93l-.52,3.47Z"
        transform="translate(-25.71 -31.52)" fill="#4942E4"></path>

```

```

    <rect x="93.53" y="188.37" width="61.5" height="17.55" transform="translate(-
62.07 -3.21) rotate(-11.35)"
      fill="#4942E4"></rect>
    <rect x="284.13" y="169.52" width="3.65" height="26.84"
transform="translate(38.03 408.05) rotate(-81.72)"
      fill="#4942E4"></rect>
    <rect x="311.62" y="175.61" width="3.65" height="22.67"
transform="translate(57.6 438.69) rotate(-81.72)"
      fill="#4942E4"></rect>
    <rect x="279.81" y="179.77" width="3.65" height="20.29"
transform="translate(27.43 409.74) rotate(-81.72)"
      fill="#4942E4"></rect>
    <rect x="299.25" y="185.65" width="3.65" height="14.19"
transform="translate(41.27 431.4) rotate(-81.72)"
      fill="#4942E4"></rect>
    <rect x="315.93" y="189.28" width="3.65" height="11.79"
transform="translate(53.15 449.99) rotate(-81.72)"
      fill="#4942E4"></rect>
    <rect x="282.83" y="183.68" width="3.65" height="28.58"
transform="translate(22.05 419.63) rotate(-81.72)"
      fill="#4942E4"></rect>
    <rect x="311.22" y="190.29" width="3.65" height="21.01"
transform="translate(44.69 451.11) rotate(-81.95)"
      fill="#4942E4"></rect>
    <rect x="293.6" y="180.86" width="3.65" height="52.54"
transform="translate(22.2 438.13) rotate(-81.72)"
      fill="#4942E4"></rect>
    <path
      d="M227.49,186.81a2.51,2.51,0,0,1,2-
2.06,3.42,3.42,0,0,1,2.83.8,3.07,3.07,0,0,1,1,1.57,4.21,4.21,0,0,1-
.12,1.93,7.19,7.19,0,0,1-1,2.51c-1,1.42-3.36,1.92-4.09,0a9.76,9.76,0,0,1-.07-
1.87C228,188.71,227.39,187.82,227.49,186.81Z"
      transform="translate(-25.71 -31.52)" fill="#fca89e"></path>
    <path
      d="M214.92,192.57c.14-.83.89-.63,2.4.15a51,51,0,0,1,10.15,7.38c.21-3.33.27-
6.68.18-10,2.29-.15,3.61.38,5.77,1.17a52.18,52.18,0,0,1-1.37,14.25c-.42,1.62-1,3.47-
2.62,4-2,.67-3.79-1.08-5.54-2.18-1.59-1-4.19-1.06-5.46-2.42-1-1-1.11-3-1.48-4.28-.62-
2.16-1.45-4.35-1.89-6.55A4.88,4.88,0,0,1,214.92,192.57Z"
      transform="translate(-25.71 -31.52)" fill="#4942E4"></path>
  </svg>
  <form>
    <!-- 2 column grid layout with text inputs for the first and last names
-->
    <div class="row mb-5 px-5">
      <h1>Sign up now</h1>
    </div>

```

```

        <!-- Email input -->
        <form [formGroup]="loginForm" (ngSubmit)="login()">
        <div class="form-outline mb-2">
            <label class="form-label" for="form3Example3">Email address</label>
            <input type="email" formControlName="username" id="form3Example3"
class="form-control" />

        </div>

        <!-- Password input -->
        <div class="form-outline mb-4">
            <label class="form-label" for="form3Example4">Password</label>
            <input type="password" formControlName="password" id="form3Example4"
class="form-control" />

        </div>

        <!-- Submit button -->
        <button type="submit" [disabled]="!loginForm.valid" class="btn btn-
primary btn-block mb-4">
            Sign up
        </button>
    </form>
    <!-- Register buttons -->
    <div class="text-center">
        <p>or sign up with:</p>
        <button type="button" class="btn btn-link btn-floating mx-1">
            <i class="bi bi-facebook text-sucess"></i>
        </button>

        <button type="button" class="btn btn-link btn-floating mx-1">
            <i class="bi bi-twitter"></i>
        </button>

        <button type="button" class="btn btn-link btn-floating mx-1">
            <i class="fa-brands bi-instagram"></i>
        </button>

    </div>
</form>
</div>
</div>

```

```
        </div>
    </div>
</div>
</section>
<!-- Section: Design Block -->
```

Typescript Part

```
import {Component, OnInit} from '@angular/core';
import {FormGroup, FormBuilder} from '@angular/forms';
import {Router} from '@angular/router';
import {AuthService} from "../../services/auth.service";

@Component({
  selector: 'app-login',
  templateUrl: './login-page.component.html',
  styleUrls: ['./login-page.component.css']
})
export class LoginPageComponent implements OnInit {

  loginForm!: FormGroup;

  constructor(private authService: AuthService, private formBuilder: FormBuilder,
private router: Router) {
  }

  ngOnInit() {
    this.loginForm = this.formBuilder.group({
      username: [''],
      password: ['']
    });
  }

  get f() {
    return this.loginForm.controls;
  }

  login() {

    this.authService.login(
      {
        username: this.f['username'].value,
        password: this.f['password'].value
      }
    )
  }
}
```

```

    )
    .subscribe(success => {
      if (success) {
        if (localStorage.getItem("ROLES")!.includes("ADMIN")) {
          this.router.navigate(['/customers']);
          document.location.reload();
        } else {
          this.router.navigate(['/customer-accounts/' +
localStorage.getItem("id")!]);
          setTimeout(function () {
            document.location.reload();
          }, 1000);
        }
      }
    });
  }
}
}

```

XXVI. Admin Space

Digital Banking
All Customers
Manage Accounts
Add Customer
Logout

Manage Customers

Keyword : Search

ID	Name	Email	Actions
1	Hamza	Hamzaz@hotmail.com	Delete Update View Accounts
2	Hafsa	Hafsa@hotmail.com	Delete Update View Accounts
3	Moussa kouna	Moussa@hotmail.com	Delete Update View Accounts
4	hamido	hamido@hotmail.com	Delete Update View Accounts

0

JEE Project Spring Angular - Digital Banking

Digital Banking All Customers Manage Accounts Add Customer

Logout



Add a new customer

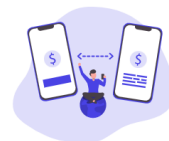
Name:

email:

Save

Digital Banking All Customers Manage Accounts Add Customer

Logout



Account Operations

Account ID :3a8c4c19-96d2-4070-a5f2-9d5850b83b24

Balance :32,392.10

ID	Date	Type	Amount
61	27-49-2023	DEBIT	12.00
1	27-42-2023	DEBIT	9,502.18
2	27-42-2023	DEBIT	6,610.86
3	27-42-2023	DEBIT	3,198.44
4	27-42-2023	CREDIT	8,319.26

0 1 2

Operations

☐ DEBIT: ☐ CREDIT: ☐ TRANSFER:


Amount :

Description :

Save Operation

Customer Space

[Digital Banking](#) [My Accounts](#) [Edit profile](#) [Logout](#)




Welcome :Hamza

Hamza(Hamzaz@hotmail.com)

Account Id	Balance	Type	Actions
3a8c4c19-96d2-4070-a5f2-9d5850b83b24	32392.10198748519 DHs	saving account	View All Operations
519b2bf0-3593-45e0-9415-ed81f0ce77b0	82153.14421469708 DHs	Current account	View All Operations

[Digital Banking](#) [My Accounts](#) [Edit profile](#) [Logout](#)



Update Customer Profile

Name:

email:

[Save](#)