ENSET

PROJET JEE

spring

ANGULAR

DIGITAL BANKING

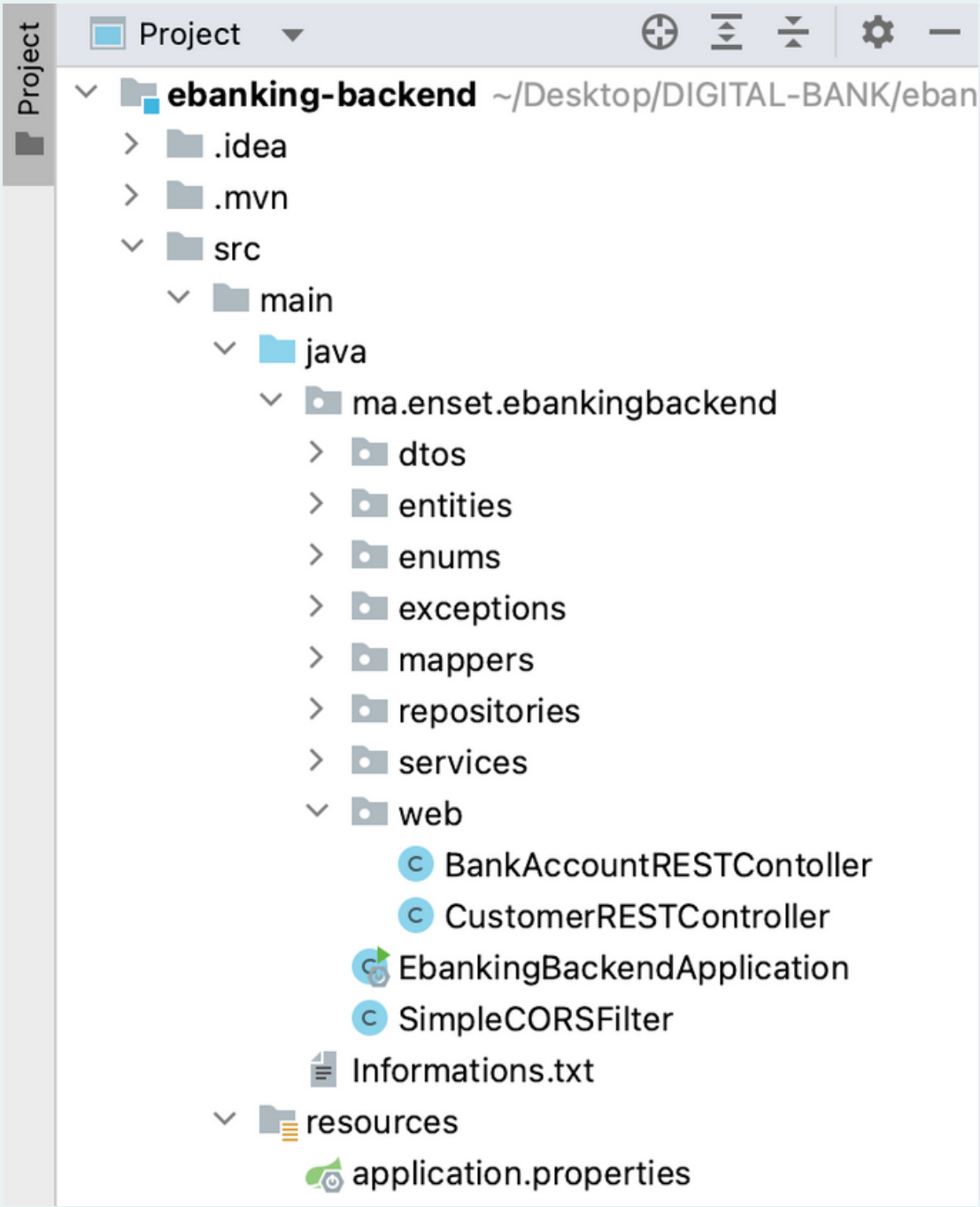
RÉALISÉ PAR :

ANAS NEDDAY

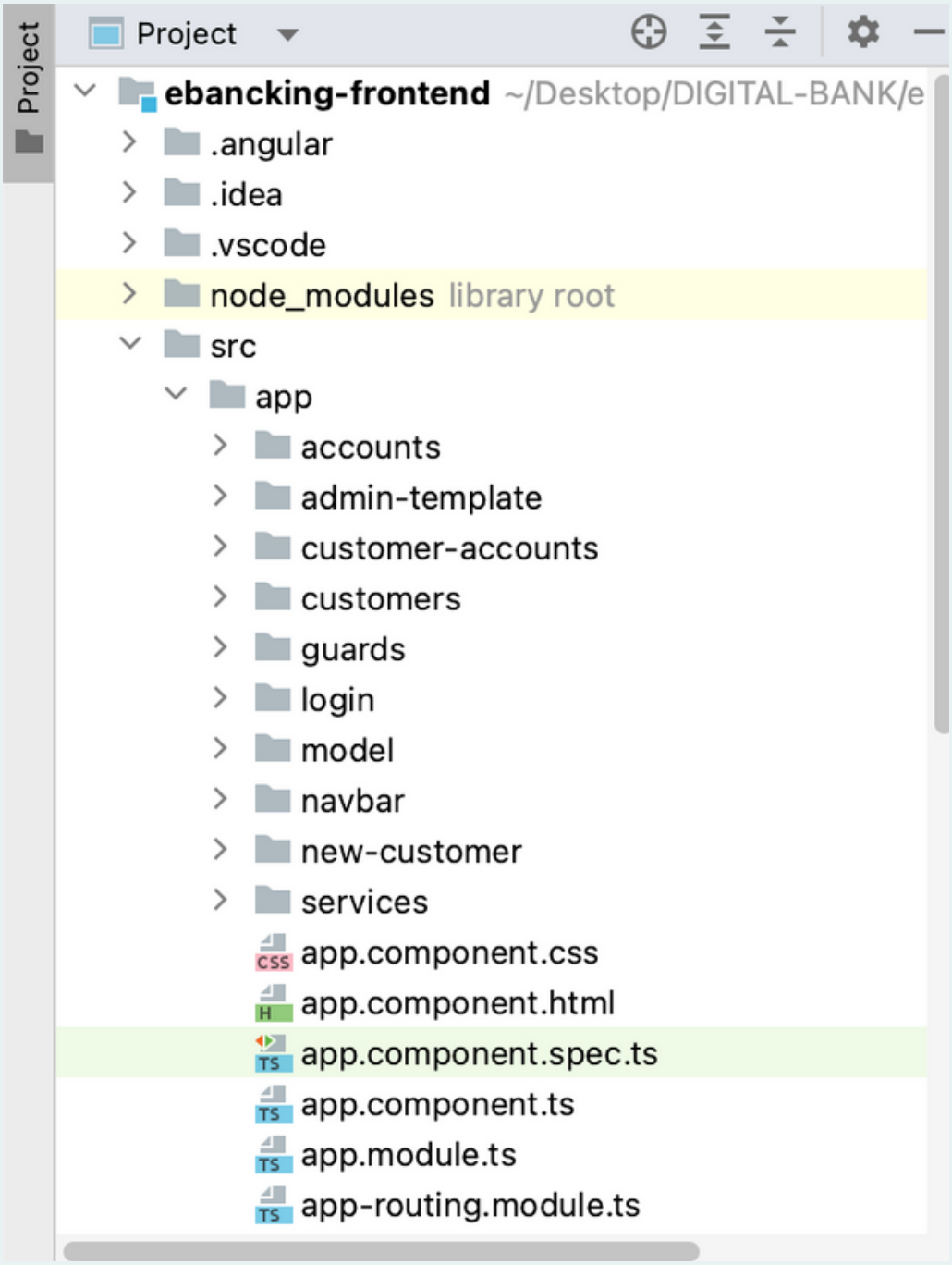
ENCADRÉ PAR:

MOHAMED YOUSSFI

BACKEND STRUCTURE :



FRONTEND STRUCTURE :



Class AccountOperation

```java
package ma.enset.ebankingbackend.entities;

import ...


17 usages
@Entity
@Data
@NoArgsConstructor @AllArgsConstructor
public class AccountOperation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long ID;
    private Date operationDate;
    private double amount;
    private String description;

    @Enumerated(EnumType.STRING)
    private OperationType type;

    @ManyToOne
    private BankAccount bankAccount;

}
```

Class BankAccount

```java
2 inheritors
@Entity
@Data
@Inheritance(strategy = InheritanceType.JOINED)

@NoArgsConstructor @AllArgsConstructor
public class BankAccount {
    @Id
    private String ID;
    private String RIB;
    private double balance;
    private Date createdAt;
    @Enumerated(EnumType.STRING)
    private AccountStatus status;

    @ManyToOne
    private Customer customer;
    @OneToMany(mappedBy = "bankAccount", fetch = FetchType.LAZY)

    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<AccountOperation> accountOperations;

}
```

Class CurrentAccount

```
17 usages
@Data
@Entity
//@DiscriminatorValue("CA")
@AllArgsConstructor @NoArgsConstructor
public class CurrentAccount extends BankAccount{
    private double overDraft;

}
```

class Customer

```
16 usages
@Entity
@Data
@NoArgsConstructor @AllArgsConstructor
public class Customer {
    //ID, firstName, lastName, email, phoneNumber, bankAccounts
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long ID;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    @OneToMany(mappedBy = "customer")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<BankAccount> bankAccounts;
}
```

```
com.fasterxml.jackson.annot
ss
```

Class SavingAccount

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

REPOSITORIES :

```
public interface AccountOperationRepository extends
JpaRepository<AccountOperation, Long> {
    List<AccountOperation> findByBankAccountID(String id);
    List<AccountOperation> findByBankAccount(BankAccount bankAccount);

    Page<AccountOperation> findByBankAccountID(String id,Pageable pageable);


    Page<AccountOperation> findByBankAccountIDOrderByOperationDateDesc(String
id, Pageable pageable);
}
```

```
package ma.enset.ebankingbackend.repositories;

import ma.enset.ebankingbackend.entities.BankAccount;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface BankAccountRepository extends JpaRepository<BankAccount, String>
{
    List<BankAccount> findByCustomerID(Long ID);
}
```

```
package ma.enset.ebankingbackend.repositories;

import ma.enset.ebankingbackend.entities.Customer;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface CustomerRepository extends JpaRepository<Customer, Long> {
    List<Customer> findByFirstNameContainsOrLastNameContains(String firstName, String
lastName);
}
```

INTERFACE BANKACCOUNTSERVICE

```
import java.util.List;

public interface BankAccountService {
    CustomerDTO saveCustomer(CustomerDTO customer);
    CurrentAccountDTO saveCurrentBankAccount(double balance, double overDraft,
Long customerID) throws CustomerNotFoundException;
    SavingAccountDTO saveSavingBankAccount(double balance, double interestRate,
Long customerID) throws CustomerNotFoundException;
    List<CustomerDTO> listCustomers();
    BankAccountDTO getBankAccount(String id) throws
BankAccountNotFoundException;
    void deposit(String id, double amount, String description) throws
BankAccountNotFoundException;
    void withdraw(String id, double amount, String description) throws
BankAccountNotFoundException, InsufficientBalanceException;
    void transfer(String idSource, String idDestination, double amount) throws
BankAccountNotFoundException, InsufficientBalanceException;

    List<BankAccountDTO> listBankAccounts();

    CustomerDTO getCustomer(Long id) throws CustomerNotFoundException;

    CustomerDTO updateCustomer(CustomerDTO customerDTO);

    void deleteCustomer(Long id) throws CustomerNotFoundException;
    List<AccountOperationDTO> accountOperationsHistory(String ID) throws
BankAccountNotFoundException;

    AccountHistoryDTO accountOperationsHistory(String id, int page, int size) throws
BankAccountNotFoundException;

    List<CustomerDTO> searchCustomers(String keyword);

    List<BankAccountDTO> getCustomerBankAccounts(Long id) throws
CustomerNotFoundException;
}
```

```
@Service
@Transactional
@AllArgsConstructor // Inject all the fields
// Inject a logger
@Slf4j //Equivalent to Logger log = LoggerFactory.getLogger(BankAccountServiceImpl.class);
public class BankAccountServiceImpl implements BankAccountService{
    private CustomerRepository customerRepository;
    private BankAccountRepository bankAccountRepository;
    private AccountOperationRepository accountOperationRepository;

    private BanKAccountMapperImpl dtoMapper;
    @Override
    public CustomerDTO saveCustomer(CustomerDTO customerDTO) {
        log.info("Saving customer {}", customerDTO);
        Customer customer = dtoMapper.fromCustomerDTOToCustomer(customerDTO);
        customerRepository.save(customer);
        return dtoMapper.fromCustomerToCustomerDTO(customer);
    }

    @Override
    public List<CustomerDTO> searchCustomers(String keyword) {
        log.info("Searching customers with keyword {}", keyword);
                                                                                                return
customerRepository.findByFirstNameContainsOrLastNameContains(keyword,keyword).stream()
.map(dtoMapper::fromCustomerToCustomerDTO).collect(Collectors.toList());
    }

    @Override
        public CurrentAccountDTO saveCurrentBankAccount(double balance, double overDraft,
Long customerID) throws CustomerNotFoundException {
        log.info("Saving Current bank account with balance {}", balance);
        CurrentAccount bankAccount=new CurrentAccount();
        Customer customer = customerRepository.findById(customerID).orElse(null);
        if (customer == null) throw new CustomerNotFoundException("Customer not found");

        bankAccount.setID(UUID.randomUUID().toString());
        bankAccount.setRIB("00012"+customerID+"0001");
        bankAccount.setBalance(balance);
        bankAccount.setCreatedAt(new java.util.Date());
        bankAccount.setStatus(AccountStatus.CREATED);
        bankAccount.setOverDraft(overDraft);
        bankAccount.setCustomer(customerRepository.findById(customerID).orElse(null));
                                                                                                return
dtoMapper.fromCurrentAccountToCurrentAccountDTO(bankAccountRepository.save(bankAcco
unt));
    }
```

```
@Override
    public SavingAccountDTO saveSavingBankAccount(double balance, double interest,
Long customerID) throws CustomerNotFoundException {
        log.info("Saving Saving bank account with balance {}", balance);
        SavingAccount bankAccount=new SavingAccount();

        Customer customer = customerRepository.findById(customerID).orElse(null);
        if (customer == null) throw new CustomerNotFoundException("Customer not
found");

        bankAccount.setID(UUID.randomUUID().toString());
        bankAccount.setRIB("00012"+customerID+"0001");
        bankAccount.setBalance(balance);
        bankAccount.setCreatedAt(new java.util.Date());
        bankAccount.setStatus(AccountStatus.CREATED);
        bankAccount.setInterestRate(interest);

 bankAccount.setCustomer(customerRepository.findById(customerID).orElse(null));

        return
dtoMapper.fromSavingAccountToSavingAccountDTO(bankAccountRepository.save(ba
nkAccount));
    }

    @Override
    public List<CustomerDTO> listCustomers() {
        List<Customer> customers=customerRepository.findAll();
        /*Programmation impérative
        List<CustomerDTO> customerDTOS=new ArrayList<>();
        for (Customer customer:customers){
            customerDTOS.add(dtoMapper.fromCustomerToCustomerDTO(customer));
        }
        */
        return
customers.stream().map(dtoMapper::fromCustomerToCustomerDTO).collect(Collecto
rs.toList());
    }


    }
```

```
@Override
    public BankAccountDTO getBankAccount(String ID) throws
BankAccountNotFoundException {
        BankAccount bankAccount=getBankAccountEntity(ID);

 bankAccount.setAccountOperations(accountOperationRepository.findByBankAccoun
t(bankAccount));
        if (bankAccount instanceof CurrentAccount)
            return
dtoMapper.fromCurrentAccountToCurrentAccountDTO((CurrentAccount)
bankAccount);
        else
            return dtoMapper.fromSavingAccountToSavingAccountDTO((SavingAccount)
bankAccount);
    }

    //Get the bank account entity from the database
    private BankAccount getBankAccountEntity(String ID) throws
BankAccountNotFoundException {
        return bankAccountRepository.findById(ID)
                .orElseThrow(()->new BankAccountNotFoundException("Bank account not
found"));
    }

    @Override
    public void deposit(String ID, double amount, String description) throws
BankAccountNotFoundException {
        BankAccount bankAccount=getBankAccountEntity(ID);

        //Create an operation<credit>
        AccountOperation accountOperation=new AccountOperation();
        accountOperation.setAmount(amount);
        accountOperation.setType(OperationType.CREDIT);
        accountOperation.setBankAccount(bankAccount);
        accountOperation.setDescription(description);
        accountOperation.setOperationDate(new java.util.Date());
        //Update the balance
        bankAccount.setBalance(bankAccount.getBalance()+amount);
        //Save the operation and the bank account
        accountOperationRepository.save(accountOperation);
        bankAccountRepository.save(bankAccount);
```

```
  @Override
    public void withdraw(String ID, double amount, String description) throws
BankAccountNotFoundException, InsufficientBalanceException {
        BankAccount bankAccount=getBankAccountEntity(ID);

        if (bankAccount.getBalance()<amount)
           throw new InsufficientBalanceException("Insufficient balance");

        //Create an operation<debit>
        AccountOperation accountOperation=new AccountOperation();
        accountOperation.setAmount(amount);
        accountOperation.setType(OperationType.DEBIT);
        accountOperation.setBankAccount(bankAccount);
        accountOperation.setDescription(description);
        accountOperation.setOperationDate(new java.util.Date());
        //Update the balance
        bankAccount.setBalance(bankAccount.getBalance()-amount);
        //Save the operation and the bank account
        accountOperationRepository.save(accountOperation);
        bankAccountRepository.save(bankAccount);
    }

    @Override
    public void transfer(String IDSource, String IDDestination, double amount) throws
BankAccountNotFoundException, InsufficientBalanceException {
        //Debit the source account
        withdraw(IDSource,amount,"Transfer to "+IDDestination);
        //Credit the destination account
        deposit(IDDestination,amount,"Transfer from "+IDSource);

    }
    @Override
    public List<BankAccountDTO> listBankAccounts() {
        List<BankAccount> bankAccounts=bankAccountRepository.findAll();
        return bankAccounts.stream().map(bankAccount -> {
            if (bankAccount instanceof CurrentAccount)
                return
dtoMapper.fromCurrentAccountToCurrentAccountDTO((CurrentAccount)
bankAccount);
            else
                return dtoMapper.fromSavingAccountToSavingAccountDTO((SavingAccount)
bankAccount);
        }).collect(Collectors.toList());
    }
```

```
  @Override
    public CustomerDTO getCustomer(Long id) throws
CustomerNotFoundException {
        Customer customer=customerRepository.findById(id).orElseThrow(()->new
CustomerNotFoundException("Customer not found"));
        return dtoMapper.fromCustomerToCustomerDTO(customer);
    }

    @Override
    public CustomerDTO updateCustomer(CustomerDTO customerDTO) {
        log.info("Updating customer {}", customerDTO);
        Customer customer =
dtoMapper.fromCustomerDTOToCustomer(customerDTO);
        customerRepository.save(customer);
        return dtoMapper.fromCustomerToCustomerDTO(customer);
    }

    @Override
    public void deleteCustomer(Long id) throws CustomerNotFoundException {
        Customer customer=customerRepository.findById(id).orElseThrow(()->new
CustomerNotFoundException("Customer not found"));
        customerRepository.delete(customer);
    }

    @Override
    public List<AccountOperationDTO> accountOperationsHistory(String ID) throws
BankAccountNotFoundException {
        BankAccount bankAccount=getBankAccountEntity(ID);
        List<AccountOperation>
accountOperations=accountOperationRepository.findByBankAccountID(ID);
        return
accountOperations.stream().map(dtoMapper::fromAccountOperationToAccountO
perationDTO).collect(Collectors.toList());
    }
```

```
 @Override
    public AccountHistoryDTO accountOperationsHistory(String ID, int page, int size) throws
BankAccountNotFoundException {
        BankAccount bankAccount = getBankAccountEntity(ID);
        //Page<AccountOperation> accountOperations =
accountOperationRepository.findByBankAccountID(ID, PageRequest.of(page, size));
        //Order by date
        Page<AccountOperation> accountOperations =
accountOperationRepository.findByBankAccountIDOrderByOperationDateDesc(ID,
PageRequest.of(page, size));

        AccountHistoryDTO accountHistoryDTO = new AccountHistoryDTO();
        List<AccountOperationDTO>accountOperationsDTOs =
accountOperations.getContent().stream().map(dtoMapper::fromAccountOperationToAccountOper
ationDTO).collect(Collectors.toList());
        accountHistoryDTO.setOperations(accountOperationsDTOs);
        accountHistoryDTO.setID(bankAccount.getID());
        accountHistoryDTO.setRIB(bankAccount.getRIB());
        accountHistoryDTO.setBalance(bankAccount.getBalance());
        accountHistoryDTO.setType(bankAccount instanceof CurrentAccount ? "Current Account" :
"Saving Account");

        //pages
        accountHistoryDTO.setTotalPages(accountOperations.getTotalPages());
        accountHistoryDTO.setCurrentPage(page);
        accountHistoryDTO.setSize(size);
return accountHistoryDTO;

        }

    @Override
    public List<BankAccountDTO> getCustomerBankAccounts(Long id) throws
CustomerNotFoundException {
        Customer customer=customerRepository.findById(id).orElseThrow(()->new
CustomerNotFoundException("Customer not found"));
        List<BankAccount> bankAccounts=bankAccountRepository.findByCustomerID(id);
        return bankAccounts.stream().map(bankAccount -> {
            if (bankAccount instanceof CurrentAccount)
                return dtoMapper.fromCurrentAccountToCurrentAccountDTO((CurrentAccount)
bankAccount);
            else
                return dtoMapper.fromSavingAccountToSavingAccountDTO((SavingAccount)
bankAccount);
        }).collect(Collectors.toList());
    }

}
```

BANKACCOUNTRESTCONTOLLER

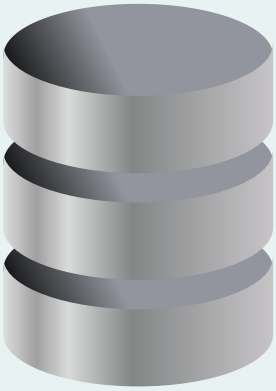
```
package ma.enset.ebankingbackend.web;

import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import ma.enset.ebankingbackend.dtos.*;
import ma.enset.ebankingbackend.exceptions.BankAccountNotFoundException;
import ma.enset.ebankingbackend.exceptions.InsufficientBalanceException;
import ma.enset.ebankingbackend.services.BankAccountService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

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

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

    @GetMapping("/bankAccounts")
    public List<BankAccountDTO> listBankAccounts() {
        return bankAccountService.listBankAccounts();
    }

    @GetMapping("/bankAccounts/{ID}/history")
    public List<AccountOperationDTO> getHistory(@PathVariable String ID) throws
BankAccountNotFoundException {
        return bankAccountService.accountOperationsHistory(ID);
    }

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



}
```

```
 @PostMapping("/bankAccounts/deposit")
    public DepositDTO deposit(@RequestBody DepositDTO deposit) throws
BankAccountNotFoundException {
        bankAccountService.deposit(deposit.getAccountID(), deposit.getAmount(),
deposit.getDescription());
        return deposit;
    }

    @PostMapping("/bankAccounts/withdraw")
    public WithdrawDTO withdraw(@RequestBody WithdrawDTO withdrawDTO) throws
BankAccountNotFoundException, InsufficientBalanceException {
        bankAccountService.withdraw(withdrawDTO.getAccountID(),
withdrawDTO.getAmount(), withdrawDTO.getDescription());
        return withdrawDTO;
    }

    @PostMapping("/bankAccounts/transfer")
    public void transfer(@RequestBody TransferDTO transferDTO) throws
BankAccountNotFoundException, InsufficientBalanceException {
        bankAccountService.transfer(transferDTO.getAccountIDSource(),
transferDTO.getAccountIDDestination(), transferDTO.getAmount());
    }
```

CUSTOMERRESTCONTROLLER

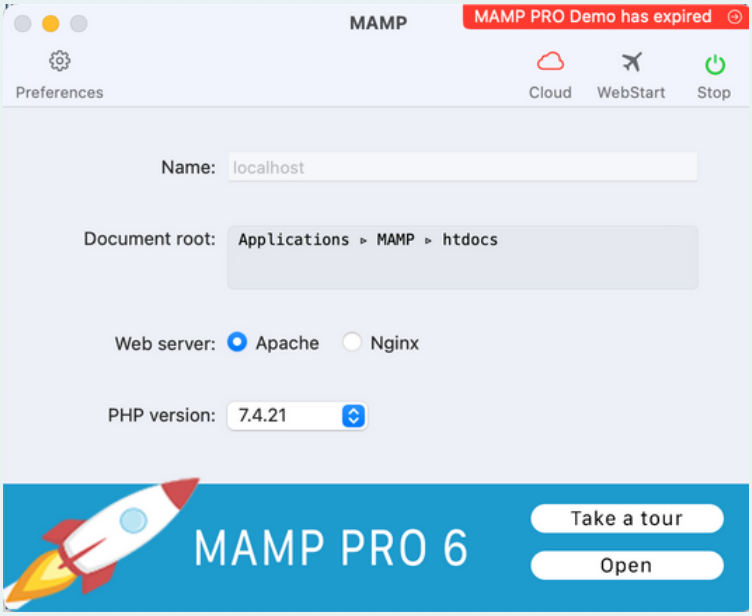
```
@RestController
@AllArgsConstructor
@Slf4j
@CrossOrigin("*")
public class CustomerRESTController {
    private BankAccountService bankAccountService;

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

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

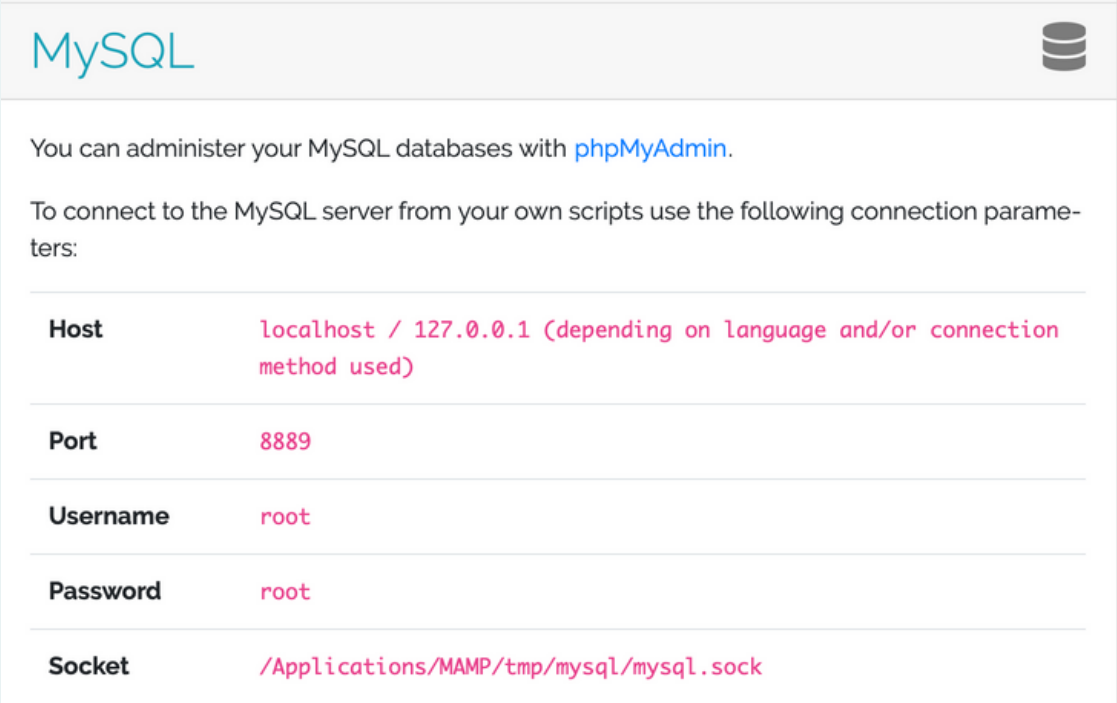
```
  @PostMapping("/customers")
    public CustomerDTO saveCustomer(@RequestBody CustomerDTO customerDTO) {
        return bankAccountService.saveCustomer(customerDTO);
    }


    @PutMapping("/customers/{id}")
    public CustomerDTO updateCustomer(@PathVariable Long id,@RequestBody CustomerDTO
customerDTO) {
        customerDTO.setID(id);
        return bankAccountService.updateCustomer(customerDTO);
    }

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

    @GetMapping("/customers/{id}/bankAccounts")
    public List<BankAccountDTO> getCustomerBankAccounts(@PathVariable Long id) throws
CustomerNotFoundException {
        return bankAccountService.getCustomerBankAccounts(id);
    }
}
```



BASE DE DONNÉES :

MAMP SERVER





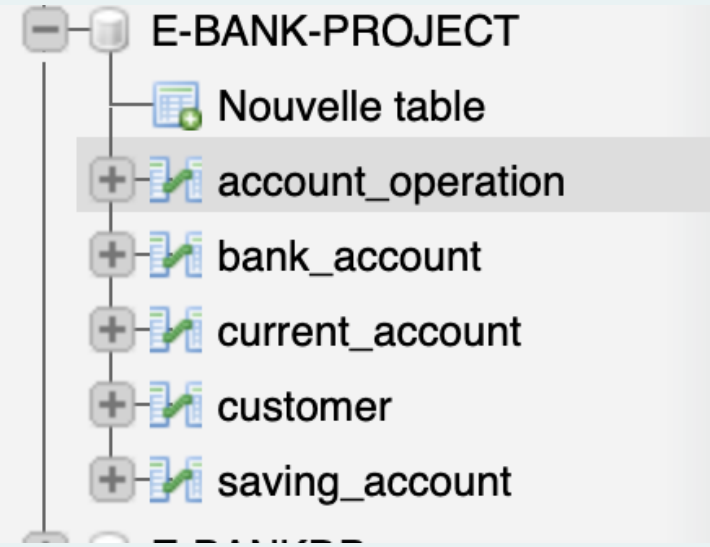
MySQL

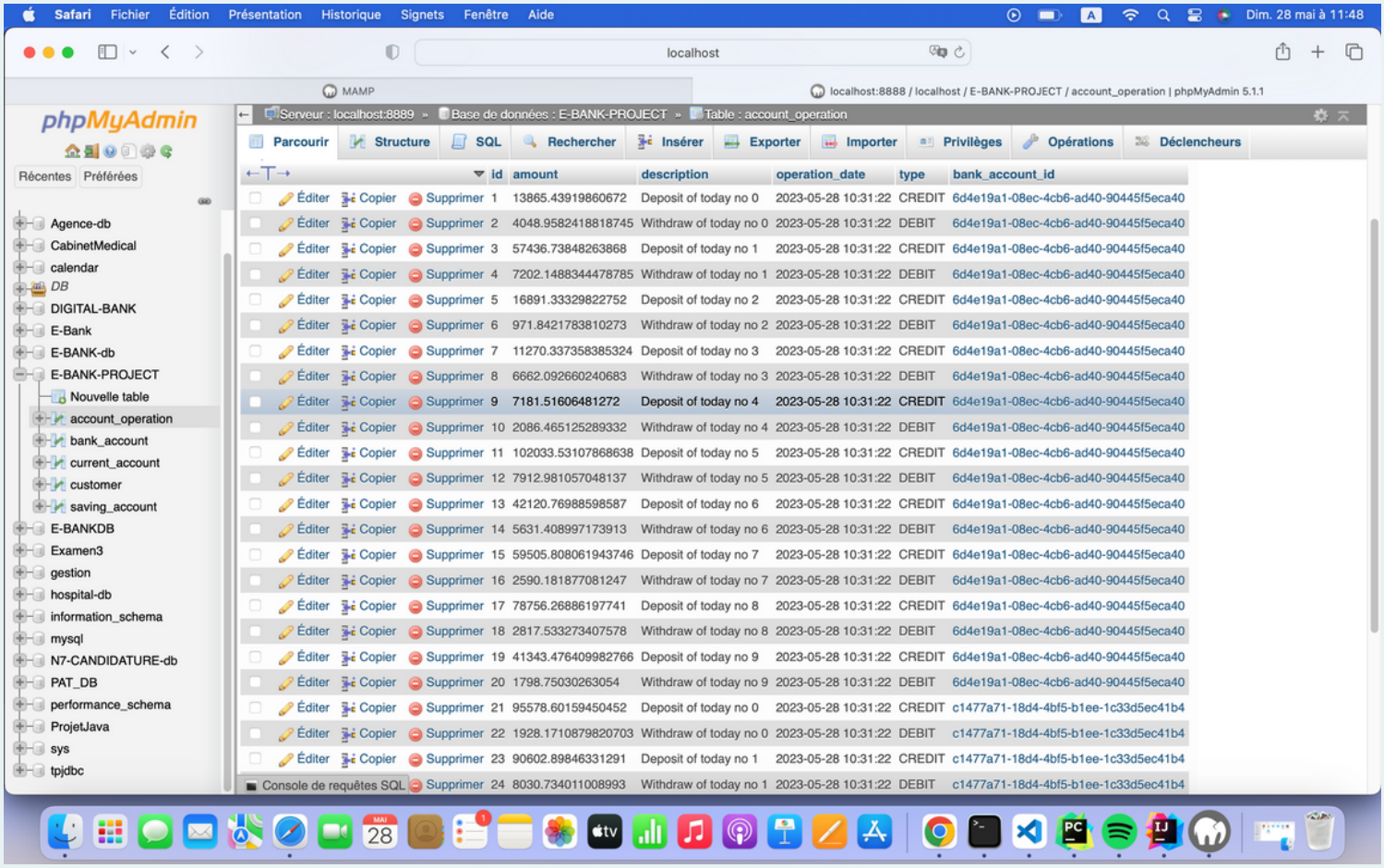
You can administer your MySQL databases with phpMyAdmin.

To connect to the MySQL server from your own scripts use the following connection parameters:

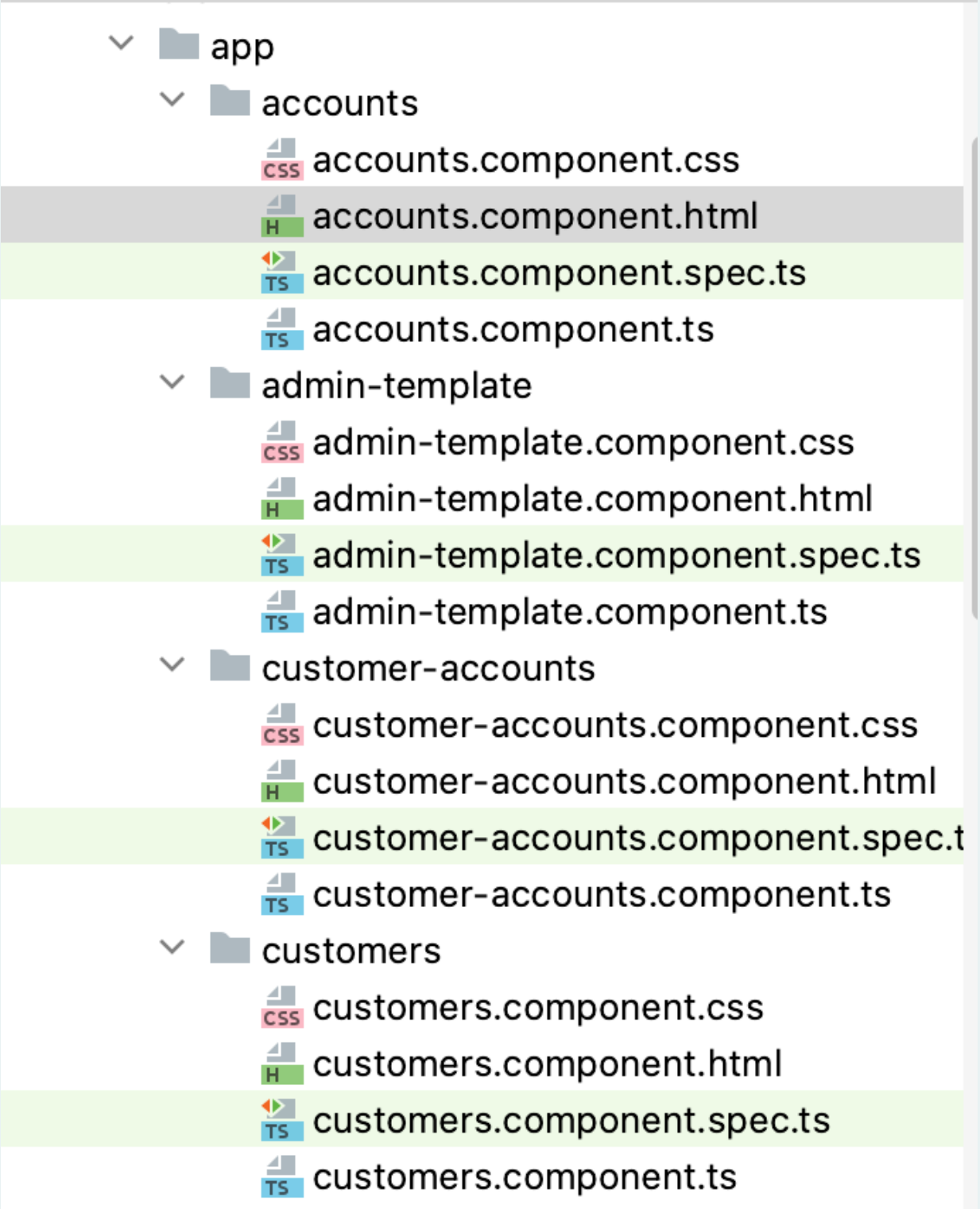
Host	localhost / 127.0.0.1 (depending on language and/or connection method used)
Port	8889
Username	root
Password	root
Socket	/Applications/MAMP/tmp/mysql/mysql.sock

```
spring.datasource.url=jdbc:mysql://localhost:8889/E-BANKDB?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root

server.port=8082

spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
spring.jpa.show-sql=true
```

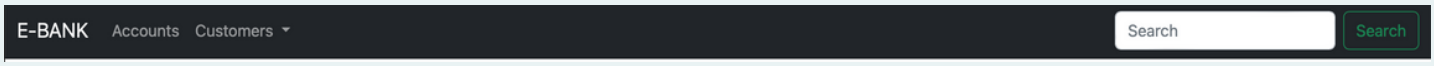




	id	amount	description	operation_date	type	bank_account_id
Éditer Copier Supprimer	1	13865.43919860672	Deposit of today no 0	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	2	4048.9582418818745	Withdraw of today no 0	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	3	57436.73848263868	Deposit of today no 1	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	4	7202.1488344478785	Withdraw of today no 1	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	5	16891.33329822752	Deposit of today no 2	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	6	971.8421783810273	Withdraw of today no 2	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	7	11270.337358385324	Deposit of today no 3	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	8	6662.092660240683	Withdraw of today no 3	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	9	7181.51606481272	Deposit of today no 4	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	10	2086.465125289332	Withdraw of today no 4	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	11	102033.53107868638	Deposit of today no 5	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	12	7912.981057048137	Withdraw of today no 5	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	13	42120.76988598587	Deposit of today no 6	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	14	5631.408997173913	Withdraw of today no 6	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	15	59505.808061943746	Deposit of today no 7	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	16	2590.181877081247	Withdraw of today no 7	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	17	78756.26886197741	Deposit of today no 8	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	18	2817.533273407578	Withdraw of today no 8	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	19	41343.476409982766	Deposit of today no 9	2023-05-28 10:31:22	CREDIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	20	1798.75030263054	Withdraw of today no 9	2023-05-28 10:31:22	DEBIT	6d4e19a1-08ec-4cb6-ad40-90445f5eca40
Éditer Copier Supprimer	21	95578.60159450452	Deposit of today no 0	2023-05-28 10:31:22	CREDIT	c1477a71-18d4-4bf5-b1ee-1c33d5ec41b4
Éditer Copier Supprimer	22	1928.1710879820703	Withdraw of today no 0	2023-05-28 10:31:22	DEBIT	c1477a71-18d4-4bf5-b1ee-1c33d5ec41b4
Éditer Copier Supprimer	23	90602.89846331291	Deposit of today no 1	2023-05-28 10:31:22	CREDIT	c1477a71-18d4-4bf5-b1ee-1c33d5ec41b4
Éditer Copier Supprimer	24	8030.734011008993	Withdraw of today no 1	2023-05-28 10:31:22	DEBIT	c1477a71-18d4-4bf5-b1ee-1c33d5ec41b4

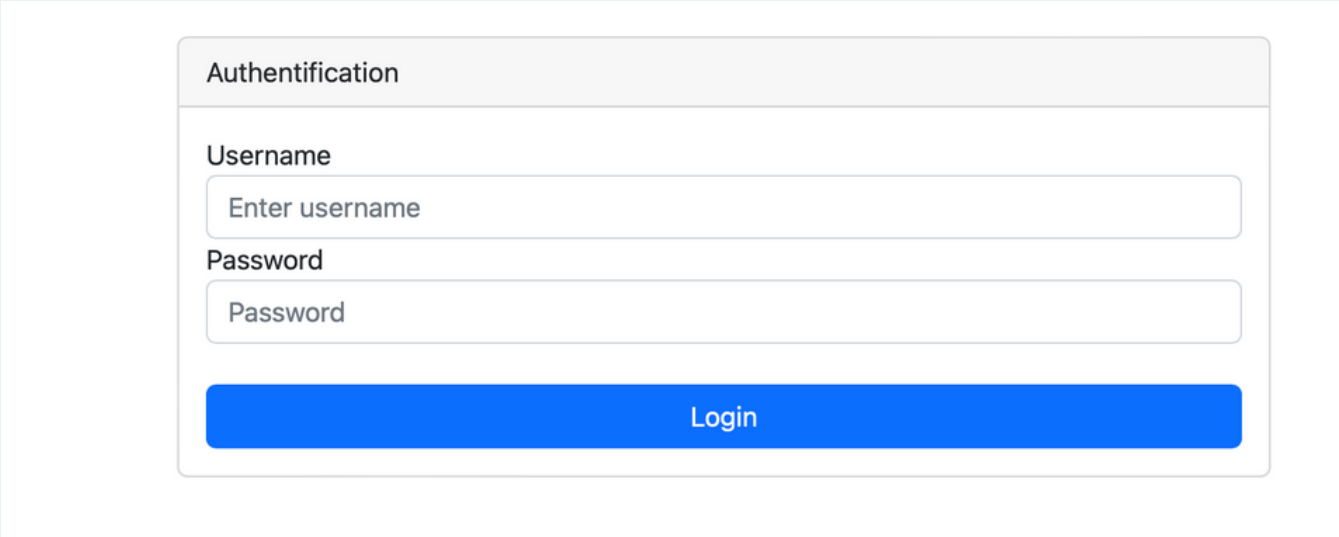

app
accounts
accounts.component.css
accounts.component.html
accounts.component.spec.ts
accounts.component.ts
admin-template
admin-template.component.css
admin-template.component.html
admin-template.component.spec.ts
admin-template.component.ts
customer-accounts
customer-accounts.component.css
customer-accounts.component.html
customer-accounts.component.spec.t
customer-accounts.component.ts
customers
customers.component.css
customers.component.html
customers.component.spec.ts
customers.component.ts

NAVBAR :



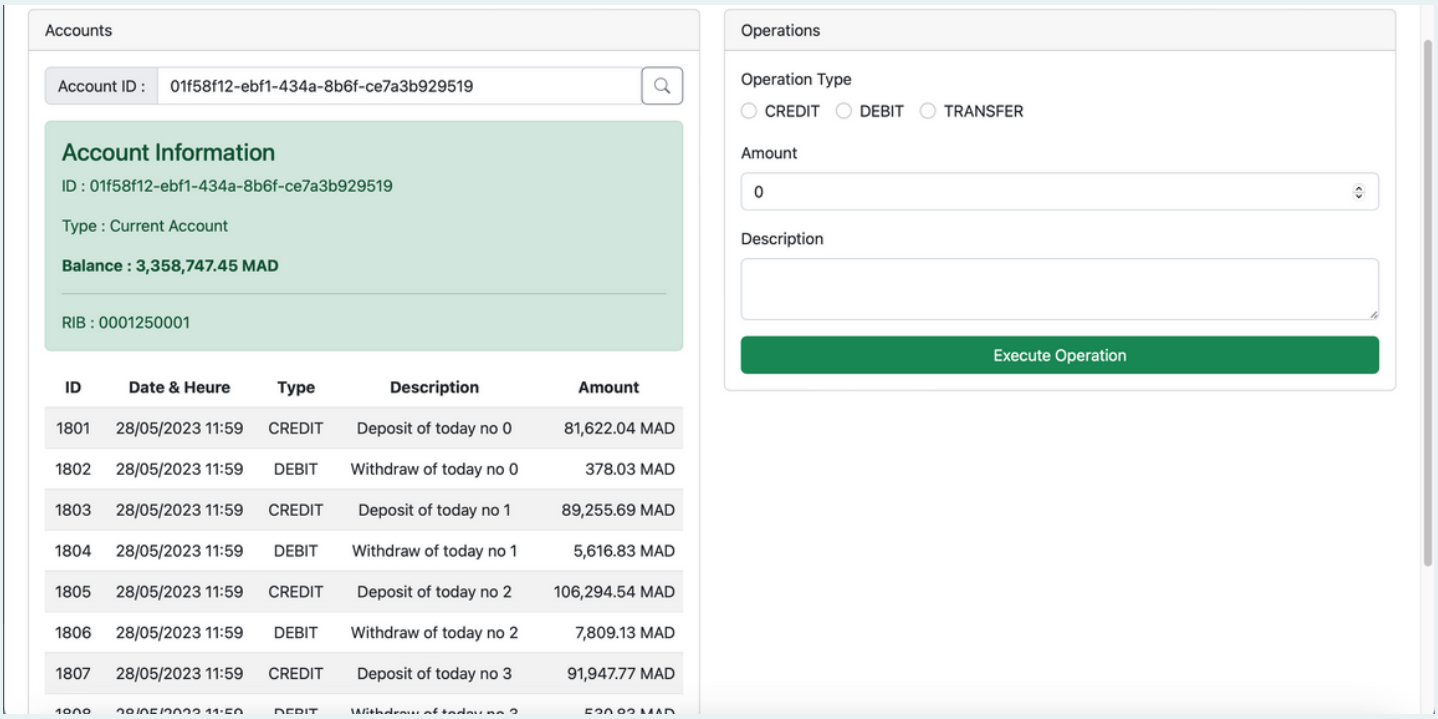

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">E-BANK</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link" routerLink="/admin/accounts">Accounts</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown"
role="button" data-bs-toggle="dropdown" aria-expanded="false">
            Customers
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="/admin/customers">Search
customers</a></li>
            <li><a class="dropdown-item" href="/admin/new-customer">New
customer</a></li>
          </ul>
        </li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search"
aria-label="Search">
        <button class="btn btn-outline-success"
type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

LOGIN PAGE

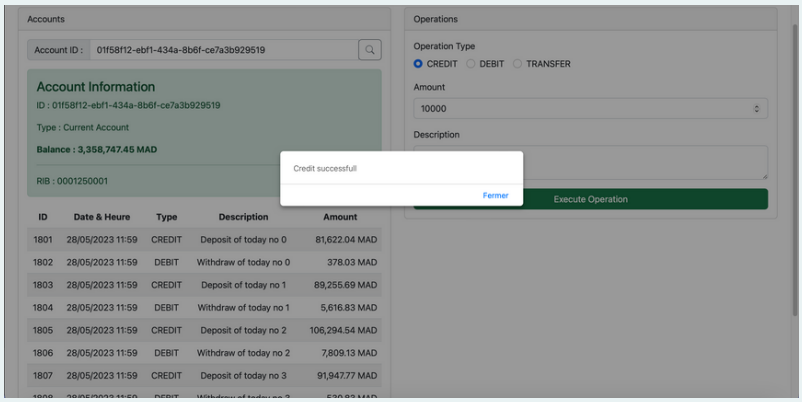


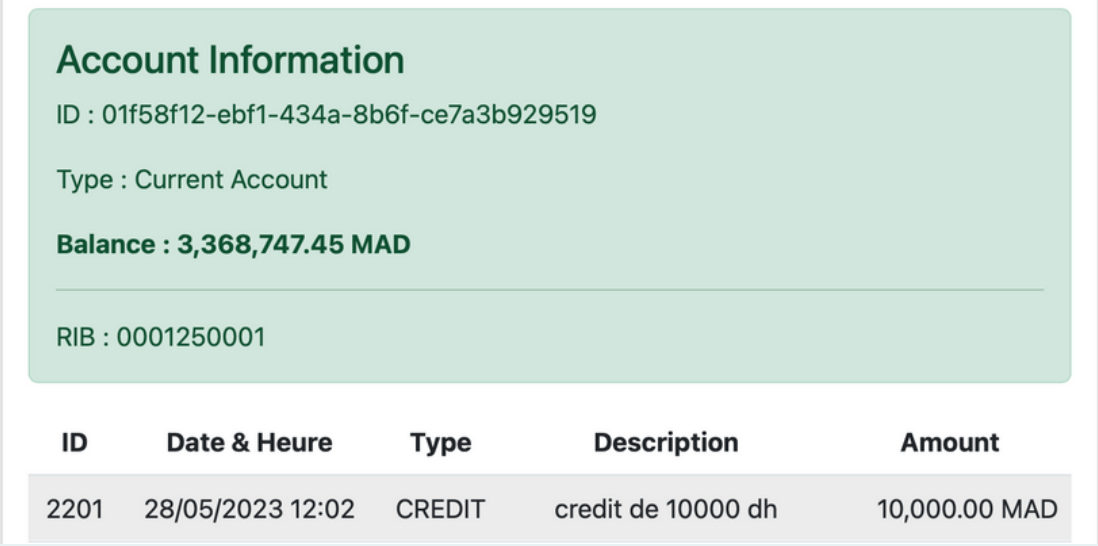

```
<div class="container">
  <div class="col-md-6 offset-3">
      <div class="card mt-4">
        <div class="card-header">
          Authentification
        </div>

     <div class="card-body">
          <!--error message-->
          <div class="alert alert-danger" *nglf="errorMessage">
            {{errorMessage}}
          </div>
          <form class="" [formGroup]="userFormGroup" (ngSubmit)="handleLogin()">
            <div class="form-group">
              <label for="username">Username</label>
              <input type="text" class="form-control" id="username" placeholder="Enter username"
formControlName="username">
            </div>
            <div class="form-group">
              <label for="password">Password</label>
              <input type="password" class="form-control" id="password" placeholder="Password"
formControlName="password">
            </div>
            <div class="d-grid gap-2 mt-4">
             <button type="submit" class="btn btn-primary">Login</button>
             </div>
          </form>
        </div>
      </div>
    </div>
</div>
```

INTERFACE GRAPHIQUE :



CREDIT :





DEBIT :

Account Information

ID : 01f58f12-ebf1-434a-8b6f-ce7a3b929519

Type : Current Account

Balance : 3,368,170.45 MAD

RIB : 0001250001

ID	Date & Heure	Type	Description	Amount
2202	28/05/2023 12:03	DEBIT	Debit de 577 Dh	577.00 MAD

TRANSFERT :

Operations

Operation Type

CREDIT DEBIT TRANSFER

Destination

3c617dbb-18f3-4a0b-b4f8-1434197e3ec0

Amount

800 dh

Description

transfer a Marouane

Execute Operation

DÉPÔT GITHUB :