BankApplication

Banque Misr Training Program 2025
Back-end Development with Java Spring Boot

Team Information

Team Number	Team Leader
Team 3	Ziad Sheref

Team Members

Name	Email
Ziad Sheref	ziyadsherif@gmail.com
إسراء محمد السيد محمد	israamohamed2315@gmail.com
عمرو خالد سلطان	amrsultan2822@gmail.com
ندى محيى الدين حافظ عبدالعال	nadamohey24@gmail.com
Abdullah Moussa	22-101114@students.eui.edu.eg

Submission Date 11 August 2025

1.Models

1.Customer.java

```java
package com.sprints.BankApplication.model;
import ...
@Data
@Entity
@Table(name = "customer")
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false)
    @NotBlank(message = "Name is required")
    @Size(max = 100, message = "Name must not exceed 100 characters")
    private String name;

    @Column(unique = true)
    @Email(message = "Email should be valid")
    @NotBlank(message = "Email is required")
    private String email;

    @Size(max = 20, message = "Phone must not exceed 20 characters")
    private String phone;

    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<BankAccount> accounts;

    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<Transaction> transactions;

    public Customer() {}

    public Customer(Integer id, String name, String email, String phone, List<BankAccount> accounts, List<Transaction> transactions) {...}
```

2.BankAccount.java

```java
import ...
@Data
@Entity
@Table(name = "bank_account")
public class BankAccount {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false, unique = true)
    @NotBlank(message = "Account number is required")
    @Size(max = 20, message = "Account number must not exceed 20 characters")
    private String accountNumber;

    @Column(nullable = false)
    @NotBlank(message = "Account type is required")
    @Size(max = 50, message = "Account type must not exceed 50 characters")
    private String accountType;

    @NotNull(message = "Balance is required")
    @DecimalMin(value = "0.0", inclusive = true, message = "Balance must be non-negative")
    private BigDecimal balance;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @OneToMany(mappedBy = "account", cascade = CascadeType.ALL)
    private List<Transaction> transactions;

    public BankAccount() {}

    public BankAccount(Integer id, String accountNumber, String accountType, BigDecimal balance, Customer customer, List<Transaction> transactions) {...}
```

3.Transaction.java

```java
package com.sprints.BankApplication.model;

import ...

@Data
@Entity
@Table(name = "transaction")
public class Transaction {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false)
    @NotBlank(message = "Transaction type is required")
    @Size(max = 50, message = "Transaction type must not exceed 50 characters")
    private String type;

    @Column(nullable = false)
    @NotNull(message = "Amount is required")
    @DecimalMin(value = "0.01", message = "Amount must be greater than 0")
    private BigDecimal amount;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @ManyToOne
    @JoinColumn(name = "account_id")
    private BankAccount account;

    public Transaction() {}

    public Transaction(Integer id, String type, BigDecimal amount, Customer customer, BankAccount account) {...}
```

2.Repositorys

1.CustomerRepository

```java
package com.sprints.BankApplication.repository;

import com.sprints.BankApplication.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository  9 usages
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    Optional<Customer> findByEmail(String email);  no usages
}
```

2. BankAccountRepository

```java
package com.sprints.BankApplication.repository;

import ...

@Repository  6 usages
public interface BankAccountRepository extends JpaRepository<BankAccount, Integer> {
    List<BankAccount> findByAccountType(String type);  1 usage

    List<BankAccount> findByBalanceGreaterThan(BigDecimal amount);  1 usage

    @Query("SELECT a FROM BankAccount a WHERE a.balance BETWEEN :min AND :max")  1 usage
    List<BankAccount> findAccountsInRange(Double min, Double max);

    @Modifying  1 usage
    @Transactional
    @Query("UPDATE BankAccount b SET b.balance = :balance WHERE b.id = :accountId")
    int updateBalance(@Param("accountId") Integer accountId, @Param("balance") Double balance);
}
```

3. TransactionRepository

```java
package com.sprints.BankApplication.repository;

import ...

@Repository  3 usages
public interface TransactionRepository extends JpaRepository<Transaction, Integer> {

    List<Transaction> findByAccountId(Integer accountId);  1 usage
    @Modifying  1 usage
    @Transactional
    @Query("DELETE FROM Transaction t WHERE t.account.id = :accountId")
    int deleteByAccountId(@Param("accountId") Integer accountId);
}
```

3. Dtos

1.CustomerDTO

```java
package com.sprints.BankApplication.dto;

import ...

@Data  29 usages
public class CustomerDto {


    private Integer id;

    @NotBlank(message = "Name is required")
    @Size(max = 100, message = "Name must not exceed 100 characters")
    private String name;
    @Email(message = "Email should be valid")
    @NotBlank(message = "Email is required")
    private String email;
    @Size(max = 20, message = "Phone must not exceed 20 characters")
    private String phone;

    public CustomerDto() {}

    public CustomerDto(Integer id, String name, String email, String phone) {...}
```

2.BankAccountDTO

```java
package com.sprints.BankApplication.dto;

import ...
@Data  31 usages
public class BankAccountDto {

    private Integer id;

    @NotBlank(message = "Account number is required")
    @Size(max = 20, message = "Account number must not exceed 20 characters")
    private String accountNumber;
    @NotBlank(message = "Account type is required")
    @Size(max = 50, message = "Account type must not exceed 50 characters")
    private String accountType;
    @NotNull(message = "Balance is required")
    @DecimalMin(value = "0.0", inclusive = true, message = "Balance must be non-negative")
    private BigDecimal balance;
    @NotNull(message = "Customer ID is required")
    private Integer customer_id;

    public BankAccountDto() {}

    public BankAccountDto(Integer id, String accountNumber, String accountType, BigDecimal balance, Integer customer_id) {...}
```

3.TransactionDTO

```java
package com.sprints.BankApplication.dto;

import ...
@Data  27 usages
public class TransactionDto {

    @NotBlank(message = "Transaction type is required")
    @Size(max = 50, message = "Transaction type must not exceed 50 characters")
    private String type;
    @NotNull(message = "Amount is required")
    @DecimalMin(value = "0.01", message = "Amount must be greater than 0")
    private BigDecimal amount;
    @NotNull(message = "Customer ID is required")
    private Integer customer_id;
    @NotNull(message = "Account ID is required")
    private Integer account_id;

    public TransactionDto() {}

    public TransactionDto(String type, BigDecimal amount, Integer customer_id, Integer account_id) {...}
```

4.Exception

```java
@ControllerAdvice
public class GlobalExceptionHandler {

    // Handle validation
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Object> handleValidationExceptions(
            MethodArgumentNotValidException ex, WebRequest request) {

        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach(( ObjectError error) -> {
            String fieldName = ((FieldError) error).getField();
            String message = error.getDefaultMessage();
            errors.put(fieldName, message);
        });

        Map<String, Object> body = new HashMap<>();
        body.put("timestamp", LocalDateTime.now());
        body.put("status", HttpStatus.BAD_REQUEST.value());
        body.put("errors", errors);

        return new ResponseEntity<>(body, new HttpHeaders(), HttpStatus.BAD_REQUEST);
    }

    // Handle runtime exceptions
    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<Object> handleRuntimeException(RuntimeException ex, WebRequest request) {
        Map<String, Object> body = new HashMap<>();
        body.put("timestamp", LocalDateTime.now());
        body.put("status", HttpStatus.BAD_REQUEST.value());
        body.put("message", ex.getMessage());

        return new ResponseEntity<>(body, new HttpHeaders(), HttpStatus.BAD_REQUEST);
    }
}
```

5.Controllers

1.CustomerController

```java
import ...

@RestController
@RequestMapping("/customers")
public class CustomerController {

    private final CustomerService customerService;  7 usages

    public CustomerController(CustomerService customerService) { this.customerService = customerService; }

    @PostMapping
    public ResponseEntity<CustomerDto> createCustomer(@Valid @RequestBody CustomerDto customerDto) {...}

    @GetMapping
    public ResponseEntity<List<CustomerDto>> getAllCustomers() {
        List<CustomerDto> customers = customerService.getAllCustomers();
        return ResponseEntity.ok(customers);
    }

    @GetMapping("/{id}")
    public ResponseEntity<CustomerDto> getCustomerById(@PathVariable Integer id) {
        CustomerDto customer = customerService.getCustomerById(id);
        if (customer == null) {...}
        return ResponseEntity.ok(customer);
    }

    @GetMapping("/search")
    public ResponseEntity<CustomerDto> getCustomerByEmail(@RequestParam String email) {
        CustomerDto customerDto = customerService.findByEmail(email);
        if (customerDto == null) {...}
        return ResponseEntity.ok(customerDto);
    }
    @PutMapping("/{id}")
    public ResponseEntity<CustomerDto> updateCustomer(@PathVariable Integer id, @Valid @RequestBody CustomerDto customerDto) {...}
```

2.BankAccountController

```java
@RestController
@RequestMapping("/account")
public class BankAccountController {
    private final BankAccountService bankAccountService;  11 usages

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

    @PostMapping
    public ResponseEntity<BankAccountDto> createAccount(@Valid @RequestBody BankAccountDto dto) {BankAccountDto created = bankAccountService.createBankAccount(dto);
        return ResponseEntity.status(HttpStatus.CREATED).body(created);
    }

    @GetMapping
    public ResponseEntity<List<BankAccountDto>> getAllAccounts() {return ResponseEntity.ok(bankAccountService.getAllBankAccounts());}

    @GetMapping("/{id}")
    public ResponseEntity<BankAccountDto> getAccountById(@PathVariable Integer id) {BankAccountDto account = bankAccountService.getBankAccountById(id);
        return account != null ? ResponseEntity.ok(account) : ResponseEntity.notFound().build();
    }

    @PutMapping("/{id}")
    public ResponseEntity<BankAccountDto> updateAccount(@PathVariable Integer id, @Valid @RequestBody BankAccountDto dto) {
        BankAccountDto updated = bankAccountService.updateBankAccount(id, dto);
        return updated != null ? ResponseEntity.ok(updated) : ResponseEntity.notFound().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteAccount(@PathVariable Integer id) {boolean deleted = bankAccountService.deleteBankAccount(id);
        return deleted ? ResponseEntity.noContent().build() : ResponseEntity.notFound().build();
    }

    @GetMapping("/type/{type}")
    public ResponseEntity<List<BankAccountDto>> getByAccountType(@PathVariable String type) {...}
```

3.TransactionController

```java
@RequestMapping("/transactions")
public class TransactionController {

    private final TransactionService transactionService;  7 usages
    public TransactionController(TransactionService transactionService){...}

    @PostMapping
    public ResponseEntity<TransactionDto> createTransaction(@Valid @RequestBody TransactionDto transactionDto) {
        TransactionDto created = transactionService.createTransaction(transactionDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(created);
    }
    @GetMapping
    public ResponseEntity<List<TransactionDto>> getAllTransactions() {
        List<TransactionDto> transactions = transactionService.getAllTransactions();
        return ResponseEntity.ok(transactions);
    }
    @GetMapping("/account/{accountId}")
    public ResponseEntity<List<TransactionDto>> getTransactionsByAccount(@PathVariable Integer accountId) {
        List<TransactionDto> transactions = transactionService.getTransactionsByAccountId(accountId);
        return ResponseEntity.ok(transactions);
    }

    @GetMapping("/{id}")
    public ResponseEntity<TransactionDto> getTransactionById(@PathVariable Integer id) {
        TransactionDto transaction = transactionService.getTransactionById(id);
        if (transaction == null) {...}
        return ResponseEntity.ok(transaction);
    }

    @PutMapping("/{id}")
    public ResponseEntity<TransactionDto> updateTransaction(@PathVariable Integer id, @Valid @RequestBody TransactionDto transactionDto) {
        try {
            TransactionDto updated = transactionService.updateTransaction(id, transactionDto);
            return ResponseEntity.ok(updated);
```

6.Services

1.CustomerService

```java
package com.sprints.BankApplication.service;
import ...
@Service  5 usages
public class CustomerService {
    private final CustomerRepository customerRepository;  9 usages

    @Autowired
    public CustomerService(CustomerRepository customerRepository) { this.customerRepository = customerRepository; }

    public CustomerDto createCustomer(CustomerDto customerDto){  3 usages
        Customer customer = new Customer();
        customer.setName(customerDto.getName());
        customer.setEmail(customerDto.getEmail());
        customer.setPhone(customerDto.getPhone());
        Customer savedCustomer = customerRepository.save(customer);
        return mapToDto(savedCustomer);
    }

    private CustomerDto mapToDto(Customer customer) {  5 usages
        if (customer == null) return null;
        CustomerDto dto = new CustomerDto();
        dto.setId(customer.getId());
        dto.setName(customer.getName());
        dto.setEmail(customer.getEmail());
        dto.setPhone(customer.getPhone());
        return dto;
    }
    public List<CustomerDto> getAllCustomers(){return customerRepository.findAll().stream().map(this::mapToDto).collect(Collectors.toList());}  1 usage

    public CustomerDto getCustomerById(Integer id){return customerRepository.findById(id).map(this::mapToDto).orElse( other: null);}  1 usage

    public CustomerDto findByEmail(String email) {return customerRepository.findByEmail(email).map(this::mapToDto).orElse( other: null);}  1 usage
```

2.BankAccountController

```java
@Service  5 usages
public class BankAccountService {
    private final BankAccountRepository bankAccountRepository;  16 usages
    private final CustomerRepository customerRepository;  3 usages

    @Autowired
    public BankAccountService(BankAccountRepository bankAccountRepository, CustomerRepository customerRepository) {...}

    private BankAccountDto mapToDto(BankAccount bankAccount) {  7 usages
        if (bankAccount == null) return null;
        BankAccountDto dto = new BankAccountDto();
        dto.setId(bankAccount.getId());
        dto.setAccountNumber(bankAccount.getAccountNumber());
        dto.setAccountType(bankAccount.getAccountType());
        dto.setBalance(bankAccount.getBalance());
        dto.setCustomer_id(bankAccount.getCustomer() != null ? bankAccount.getCustomer().getId() : null);
        return dto;
    }

    public BankAccountDto createBankAccount(BankAccountDto bankAccountDto) {  3 usages
        BankAccount bankAccount = new BankAccount();
        bankAccount.setAccountNumber(bankAccountDto.getAccountNumber());
        bankAccount.setAccountType(bankAccountDto.getAccountType());
        bankAccount.setBalance(bankAccountDto.getBalance());

        Optional<Customer> customerOptional = customerRepository.findById(bankAccountDto.getCustomer_id());
        customerOptional.ifPresent(bankAccount::setCustomer);

        BankAccount savedAccount = bankAccountRepository.save(bankAccount);
        return mapToDto(savedAccount);
    }

    public List<BankAccountDto> getAllBankAccounts() {return bankAccountRepository.findAll().stream().map(this::mapToDto).collect(Collectors.toList());}
```

```java
    public BankAccountDto getBankAccountById(Integer id) {  1 usage
        Optional<BankAccount> bankAccount = bankAccountRepository.findById(id);
        return bankAccount.map(this::mapToDto).orElse( other: null);
    }

    public BankAccountDto updateBankAccount(Integer id, BankAccountDto bankAccountDto) {  1 usage

        Optional<BankAccount> existingAccountOpt = bankAccountRepository.findById(id);

        if (existingAccountOpt.isPresent()) {...}
        return null;
    }

    public boolean deleteBankAccount(Integer id) {  1 usage
        if (bankAccountRepository.existsById(id)) {
            bankAccountRepository.deleteById(id);
            return true;
        }
        return false;
    }

    public List<BankAccountDto> findByAccountType(String type) {return bankAccountRepository.findByAccountType(type).stream()  1 usage
            .map(this::mapToDto).collect(Collectors.toList());}

    public List<BankAccountDto> findByBalanceGreaterThan(BigDecimal amount) {return bankAccountRepository.findByBalanceGreaterThan(amount).stream()
            .map(this::mapToDto).collect(Collectors.toList());}

    public List<BankAccountDto> findAccountsInRange(Double min, Double max) {return bankAccountRepository.findAccountsInRange(min, max).stream()  1 us
            .map(this::mapToDto).collect(Collectors.toList());}

    public String transferMoney(Integer senderAccountId, Integer receiverAccountId, BigDecimal amount) {...}

    public void changeAccountBalance(Integer accountId, Double newBalance) {...}
```

6.TransactionService

```java
@Service  5 usages
public class TransactionService {

    private final TransactionRepository transactionRepository;  9 usages
    private final CustomerRepository customerRepository;  3 usages
    private final BankAccountRepository bankAccountRepository;  3 usages

    public TransactionService(TransactionRepository transactionRepository, CustomerRepository customerRepository, BankAccountRepository bankAccountRepos

    private TransactionDto mapToDto(Transaction transaction) {  5 usages
        TransactionDto dto = new TransactionDto();
        dto.setType(transaction.getType());
        dto.setAmount(transaction.getAmount());
        dto.setCustomer_id(transaction.getCustomer().getId());
        dto.setAccount_id(transaction.getAccount().getId());
        return dto;
    }

    public TransactionDto createTransaction(TransactionDto transactionDto) {  3 usages
        Transaction transaction = new Transaction();
        transaction.setType(transactionDto.getType());
        transaction.setAmount(transactionDto.getAmount());

        Optional<Customer> customerOpt = customerRepository.findById(transactionDto.getCustomer_id());
        Optional<BankAccount> accountOpt = bankAccountRepository.findById(transactionDto.getAccount_id());

        if (customerOpt.isPresent() && accountOpt.isPresent()) {...} else {
            throw new RuntimeException("Customer or Bank Account not found");
        }
    }

    public List<TransactionDto> getAllTransactions() {return transactionRepository.findAll().stream().map(this::mapToDto).collect(Collectors.toList());}

    public List<TransactionDto> getTransactionsByAccountId(Integer accountId) {...}
```

7.Run

1.Create

```java
package com.sprints.BankApplication;
import ...

@SpringBootApplication
public class BankApplication {

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

    @Bean
    public CommandLineRunner demo(
            CustomerService customerService,
            BankAccountService bankAccountService,
            TransactionService transactionService) {
        return String[] args -> {
            CustomerDto cust1 = customerService.createCustomer(new CustomerDto( id: null,  name: "Omar",  email: "omar@example.com",  phone: "0123456789"));
            CustomerDto cust2 = customerService.createCustomer(new CustomerDto( id: null,  name: "Ahmed",  email: "ahmed@example.com",  phone: "0987654321"));

            BankAccountDto acc1 = bankAccountService.createBankAccount(
                    new BankAccountDto( id: 1,  accountNumber: "A101",  accountType: "Savings", new BigDecimal( val: "1500.00"), cust1.getId()));

            BankAccountDto acc2 = bankAccountService.createBankAccount(
                    new BankAccountDto( id: 2,  accountNumber: "A102",  accountType: "Checking", new BigDecimal( val: "2500.00"), cust2.getId()));

            transactionService.createTransaction(
                    new TransactionDto( type: "Deposit", new BigDecimal( val: "1500.00"), cust1.getId(), acc1.getId()));

            transactionService.createTransaction(
                    new TransactionDto( type: "Withdrawal", new BigDecimal( val: "200.00"), cust2.getId(), acc2.getId()));
        };
    }
}
```

2.Output

```
1 •  SELECT * FROM bank_app_db.customer;
```

Result Grid | Filter Rows: | Edit:

id	email	name	phone
1	omar@example.com	Omar	0123456789
2	ahmed@example.com	Ahmed	0987654321
NULL	NULL	NULL	NULL

```
1 •  SELECT * FROM bank_app_db.bank_account;
```

Result Grid | Filter Rows: | Edit:

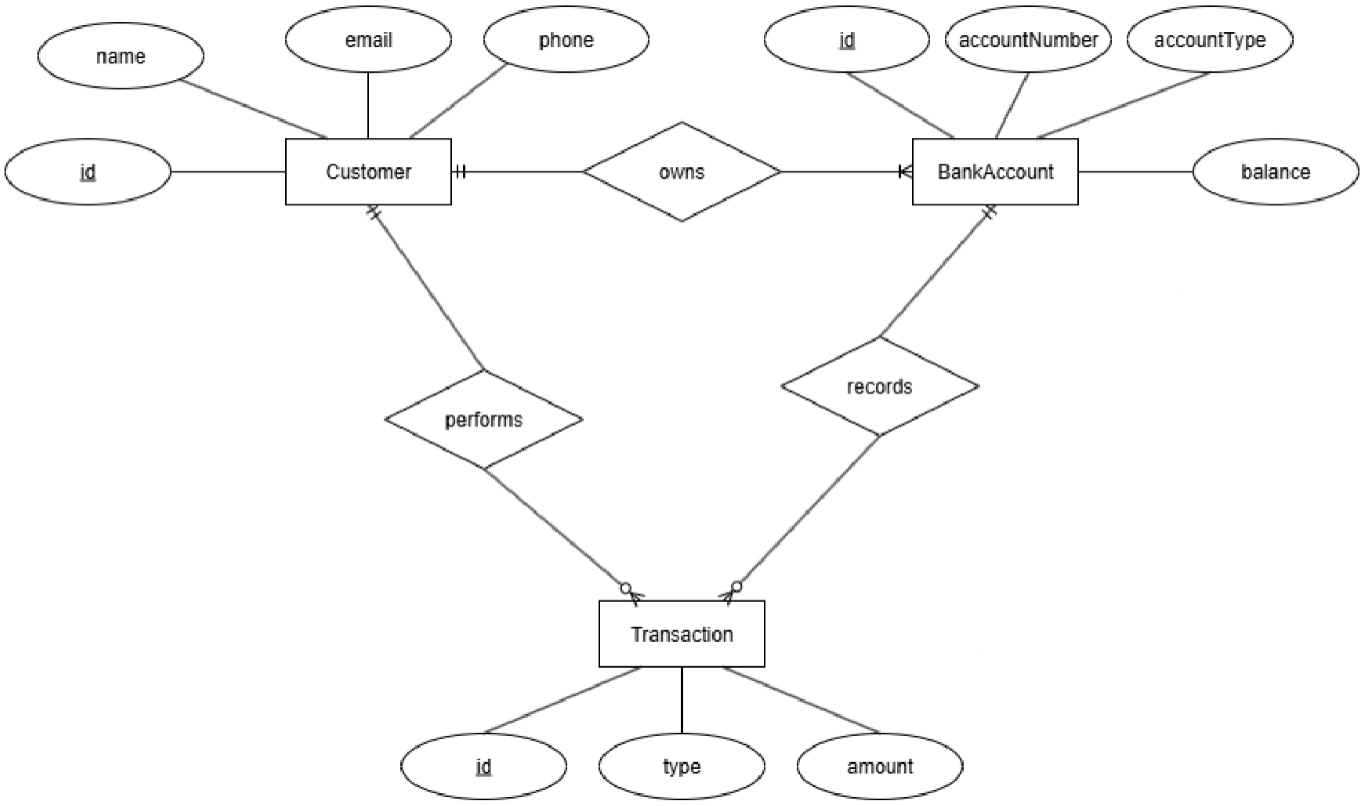
id	account_number	account_type	balance	customer_id
1	A101	Savings	1500.00	1
2	A102	Checking	2500.00	2
NULL	NULL	NULL	NULL	NULL

```
1 •  SELECT * FROM bank_app_db.transaction;
```

Result Grid | Filter Rows: | Edit:

id	amount	type	account_id	customer_id
1	1500.00	Deposit	1	1
2	200.00	Withdrawal	2	2
NULL	NULL	NULL	NULL	NULL

Erd


name
email
phone
id
accountNumber
accountType
id
Customer
owns
BankAccount
balance
performs
records
Transaction
id
type
amount