package com.fis.bankapplication.services;

import java.util.List;

import java.util.Optional;

import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.fis.bankapplication.dao.AccountDao;

import com.fis.bankapplication.dao.TransactionDao;

import com.fis.bankapplication.exception.CustomerNotFoundException;

import com.fis.bankapplication.exception.InsufficientBalance;

import com.fis.bankapplication.models.Account;

import com.fis.bankapplication.models.Transaction;

@Service

@Transactional

public class TransactionServiceImpl implements TransactionService {

@Autowired

private TransactionDao transactionDAO;

@Autowired

private AccountDao accountDAO;

// to deposite in one's own account

@Override

public boolean deposit(int accountId, double amount) throws InsufficientBalance, CustomerNotFoundException {

Optional<Account> account = accountDAO.findById(accountId);

if (account.isPresent()) {

if (amount > 0) {

transactionDAO.deposit(accountId, amount);

Transaction transaction = new Transaction();

transaction.setTransactionId(transaction.getTransactionId());

transaction.setTranscationDate(transaction.getTranscationDate());

transaction.setModeOfTransaction("Deposit");

transaction.setFromAccount(accountId);

transaction.setToAccount(accountId);

transaction.setAmount(amount);

transaction.setAccount(account.get());

transactionDAO.save(transaction);

return true;

}

else {

throw new InsufficientBalance("Amount should be positive number");

}

}else

{

throw new CustomerNotFoundException("Accout is not present");

}

}

// to withdraw amount from one own's account

@Override

public boolean withdraw(int accountId, double amount) throws InsufficientBalance, CustomerNotFoundException {

Optional<Account> account = accountDAO.findById(accountId);

if (account.isPresent()) {

Account userBalance = account.get();

if (userBalance.getBalance() >= amount) {

transactionDAO.withdraw(accountId, amount);

Transaction transaction = new Transaction();

transaction.setTransactionId(transaction.getTransactionId());

transaction.setTranscationDate(transaction.getTranscationDate());

transaction.setModeOfTransaction("Withdraw");

transaction.setFromAccount(accountId);

transaction.setToAccount(accountId);

transaction.setAmount(amount);

transaction.setAccount(account.get());

transactionDAO.save(transaction);

return true;

} else {

throw new InsufficientBalance("Insufficient Balance to withdraw");

}

} else {

throw new CustomerNotFoundException("Account with id not present" + accountId);

}

}

// to transfer the amount from account to another

@Override

public boolean fundTransfer(int fromAccountId, int toAccountId, double amount) throws InsufficientBalance {

transactionDAO.deposit(toAccountId, amount);

transactionDAO.withdraw(fromAccountId, amount);

Transaction transaction = new Transaction();

transaction.setTransactionId(transaction.getTransactionId());

transaction.setTranscationDate(transaction.getTranscationDate());

transaction.setModeOfTransaction("Fund Transfer");

transaction.setFromAccount(fromAccountId);

transaction.setToAccount(toAccountId);

transaction.setAmount(amount);

transactionDAO.save(transaction);

return true;

}

// get all the transaction from database

@Override

public List<Transaction> getAllTranscation() {

List<Transaction> list = transactionDAO.findAll();

return list;

}

// @Override

// public List<Transaction> getAllTranscationByAccountId(int accountId){

// return transactionDAO.getAllTranscationByAccountId(accountId);

// }

}