package com.db.awmd.challenge.web;  
  
import com.db.awmd.challenge.domain.Account;  
import com.db.awmd.challenge.exception.DuplicateAccountIdException;  
import com.db.awmd.challenge.service.AccountsService;  
import javax.validation.Valid;  
  
import com.db.awmd.challenge.service.EmailNotificationService;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.math.BigDecimal;  
  
@RestController  
@RequestMapping("/v1/accounts")  
@Slf4j  
public class AccountsController {  
  
 private final AccountsService accountsService;  
  
 @Autowired  
 public AccountsController(AccountsService accountsService) {  
 this.accountsService = accountsService;  
 }  
  
 @PostMapping(consumes = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseEntity<Object> createAccount(@RequestBody @Valid Account account) {  
 *log*.info("Creating account {}", account);  
  
 try {  
 this.accountsService.createAccount(account);  
 } catch (DuplicateAccountIdException daie) {  
 return new ResponseEntity<>(daie.getMessage(), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 return new ResponseEntity<>(HttpStatus.*CREATED*);  
 }  
  
 @GetMapping(path = "/{accountId}")  
 public Account getAccount(@PathVariable String accountId) {  
 *log*.info("Retrieving account for id {}", accountId);  
 return this.accountsService.getAccount(accountId);  
 }  
  
 @PutMapping("/{accountId}/{accountId}/{amount}")  
 public String transferAmounts(@PathVariable String accountIdFrom,@PathVariable String accountIdTo, @PathVariable BigDecimal amount){  
 BigDecimal bdZero=new BigDecimal("0");  
 BigDecimal balanceOfFromAccount=accountsService.getAccount(accountIdFrom).getBalance();  
 BigDecimal balanceOfToAccount=accountsService.getAccount(accountIdTo).getBalance();  
  
 if(balanceOfFromAccount.compareTo(balanceOfToAccount)>0)  
 {  
 EmailNotificationService emailNotificationService=new EmailNotificationService();  
 Account accountFrom=new Account(accountIdFrom);  
 Account accountTo=new Account(accountIdFrom);  
// Logic/functionality here for UPDATE/deducting balance "From To account holder"  
accountFrom.setBalance(accountsService.getAccount(accountIdFrom).getBalance().subtract(amount));  
 //accountsService.getAccountsRepository().save(accountFrom);  
  
// Logic/functionality here for UPDATE/Adding amount of "To account holder" accountTo.setBalance(accountsService.getAccount(accountIdFrom).getBalance().add(amount));  
 //accountsService.getAccountsRepository().save(accountTo);  
 emailNotificationService.notifyAboutTransfer(accountFrom,"Successfully transferred amounts "+amount+" from "+accountIdFrom+" to "+accountIdTo);  
 emailNotificationService.notifyAboutTransfer(accountTo,"Successfully transferred amounts "+amount+" from "+accountIdFrom+" to "+accountIdTo);  
 return "Successfully transferred amounts "+amount+" from "+accountIdFrom+" to "+accountIdTo;  
 }  
 return "Transfer unsuccessful. Balance amount is Zero ";  
 }  
  
}