**UPGRADE NEW GRAD CODING CHALLENGE**

*Assumptions and Used Practices*

1. I have used Spring Boot framework to build this library application.
2. I have used h2 database as the in memory store.
3. I have used Spring JPA, Hibernate to perform database operations.
4. For simplicity, I have created a data.sql file which automatically populate the data in tables when the application loads.
5. I have used a one-one relationship of user and wallet and each user is distinguished uniquely by his/her email ID so that means if a user want to create a wallet with same data then it will generate an error message which will be returned in JSON format.
6. Each wallet is associated with multiple accounts and have a many to one relationship which is shown in Account.java class.
7. For creating a Wallet, walletId is automatically generated and is the primary key of User entity so we need to post only the rest of the body with
8. To check the current account balance, a user must have account in his wallet, so I have created an end point of creating new Account, and furthermore he can also see all his accounts based on the walletId and the API for these are mentioned below.
9. For every transaction I have marked the type as Withdrawal and Deposit.
10. For transferring amount between two accounts, I have taken two variables, fromAccountId and toAccountId, and these attributes are being set by the getters and setters. However, if a user is performing just a withdrawal/deposit transaction then only one account is associated so I have kept the values of fromAccountId and toAccountId values 0 in that case.
11. For getting Last N transactions of a particular user, I am searching that with accountId as foreign key in the Transaction table.
12. For asynchronous requests, I have used CompletableFuture of Java 8 and implemented that in spring boot with @async and @enableAsync annotations. I have set the maximum pool size of 4 which can be seen in VirtualWalletApplication class.
13. For exceptions, I have used @ExceptionHandler annotation which automatically handles some basic exceptions such as no value found or wrong input format.
14. For Error Handling, I have made a customErrorType class which is used in ResponseEntity to generate response.

The URI Resource for 6 basic end points are as follows

1. Create a new Wallet for a user.

<http://localhost:8080/user/create>

1. Return current account balance

[http://localhost:8080/users/{WalletId}/accounts/{accountId}](http://localhost:8080/users/%7bWalletId%7d/accounts/%7baccountId%7d)

1. Perform a withdrawal transaction on an account.

[http://localhost:8080/withdrawal/{accountId}/{amount}](http://localhost:8080/withdrawal/%7baccountId%7d/%7bamount%7d)

1. Perform a deposit transaction on an account

[http://localhost:8080/deposit/{accountId}/{amount}](http://localhost:8080/deposit/%7baccountId%7d/%7bamount%7d)

1. Perform a transfer from one account to another.

[http://localhost:8080/transfer/{fromAccountId}/{toAccountId}/{amount}](http://localhost:8080/transfer/%7bfromAccountId%7d/%7btoAccountId%7d/%7bamount%7d)

1. Return last N transactions for an account.

[http://localhost:8080/transaction/{accountId}/](http://localhost:8080/transaction/%7baccountId%7d/)

Other end points

1. Create Accounts for a specific wallet.

[http://localhost:8080/users/{WalletId}/accounts](http://localhost:8080/users/%7bWalletId%7d/accounts)

1. Get all accounts for a specific wallet

[http://localhost:8080/users/{WalletId}/accounts](http://localhost:8080/users/%7bWalletId%7d/accounts)

For simplicity I have populated the tables on startup and the dataset is given below and can also be found at data.sql in resources folder.

**INSERT** **INTO** **USER** **VALUES**('101','Prabhsimran.06@gmail.com','201','Prabhsimran Singh','MoneyBank');

**INSERT** **INTO** **USER** **VALUES**('102','Ranjoodhparmar@gmail.com','202','Ranjoodh Singh','MyBank');

**INSERT** **INTO** Account **VALUES**(1001,500,'101');

**INSERT** **INTO** ACCOUNT **VALUES**(1002,1000,'101');

**INSERT** **INTO** ACCOUNT **VALUES**(1003,2000,'101');

**INSERT** **INTO** TRANSACTION **VALUES**(301,200,1001,'2017-11-01 16:25:25',1002,'Withdrawal',1001);

**INSERT** **INTO** TRANSACTION **VALUES**(302,200,1001,'2017-09-05 11:21:00',1002,'Deposit',1002);

**INSERT** **INTO** TRANSACTION **VALUES**(303,500,1002,'2016-01-11 07:45:25',1003,'Deposit',1002);

**INSERT** **INTO** TRANSACTION **VALUES**(304,500,1002,'2017-12-06 12:01:25',1003,'Withdrawal',1003