Assignment

SPIKE

1. **Library Level Solution**
    a. need to consider the precision and how big a money amount can be
        i. **ignoring it for simplification**
    b. trust the caller to pass the user identifier, Authentication and Authorisation should be an other module work.
    c. focus on behavioural test first, as the library is still in an early stage.
        i. target to have high code coverage
        ii. avoid focus too much on unit test, as early stage library may have structural change quite often. (and also due to time limit)
    d. There should be a **Transaction Request** Record
        i. to store the action / event to be handled by the system
        ii. should store a **reference** to indicate the intension / action trigger by the user
            1. the transaction need to be trace-able
        iii. possible record structure

```
class TransactionRequest {
  sourceType: TRANSACTION_REQUEST_SOURCE_TYPE, // ['USER', 'WALLET']
  sourceId: string,
  type: TRANSACTION_REQUEST_TYPE, // ['DEPOSIT', 'WITHDRAW', 'TRANSFER']
  destinationType: TRANSACTION_REQUEST_SOURCE_TYPE, // ['USER', 'WALLET']
  destinationId: string,
  typeMeta: Object, // if using postgressql, should use JSONB, keep it as small as possible
  ref: string, //defined by use case trigger, generate by client side
}
```

    e. Should care about **Transaction** Record and **State** Record
        i. Transaction Record
            1. only focus on one wallet transaction
            2. to store all history
            3. to recalculate the state if there is request for double checking the State Record
            4. should store the change instead of a new sum up number
            v. possible record structure

```
class WalletTransaction {
  walletId: ID,
  transactionRequestId: ID,
```

```
  amountChange: number,
}
```

b. State Record

   a. to provide a summary of a user

   b. to avoid calculation during read

      a. since it will not be scalable when the number of Transaction Record of one user increase

   c. to provide an easy way if there is a need to do comparison among users

   d. possible record structure

```
class WalletState {
  userId: ID
  amount: number,
  lastTransactionRequestId: ID,
}
```

f. 4 main use cases

- Assuming some basic error handling will be considered, like
  - record not found
- **User can deposit money into his wallet**
  - Assume user deposit $50
  - Failure Use Cases: NONE ?
  - related state change
    - one new TransactionRequest record

```
transactionRequest = {
  sourceType: 'USER',
  sourceId: _userId_,
  type: 'DEPOSIT',
  detinationType: 'WALLET',
  destincateId: _userWalletId_
  typeMeta: {
    amount: 50.0
  },
  ref: `app:deposite:${userId}:${walletId}:${uuid}`
}
```

    - one WalletTransaction record

```
walletTransaction = {
  walletId: _userWalletId_,
  transactionRequestId: transactionRequest.id,
  amountChange: 50.0,
}
```

- update WalletState record

```
walletState = {
  userId: _userId_,
  amount: _originalAmount_ + 50,
  lastTransactionRequestId: _transactionRequest_,
  lastWalletTransactionId: _walletTransactionId_,
}
```

- **User can withdraw money from her wallet**
  - Assume user withdraw $50
  - Failure use case:
    - current amount not enough
  - related State Change
    - one new TransactionRequest record

```
transactionRequest = {
  sourceType: 'USER',
  sourceId: userId,
  type: 'WITHDRAW',
  detinationType: 'WALLET',
  destincateId: _userWalletId_
  typeMeta: {
    amount: 50.0
  },
  ref: `app:deposite:${userId}:${walletId}:${uuid}`
}
```

    - one WalletTransaction record

```
walletTransaction = {
  walletId: _userWalletId_,
  transactionRequestId: transactionRequest.id,
  amountChange: -50.0,
}
```

    - update WalletState record

```
walletState = {
  userId: _userId_,
  amount: _originalAmount_ - 50,
  lastTransactionRequestId: _transactionRequest_,
  lastWalletTransactionId: _walletTransactionId_,
}
```

- **User can send money to another user**
  - Assume User A transfer $50 to User B

- Failure cases
  - User A doesn’t have enough money
- related State Change
  - one new TransactionRequest record

```
transactionRequest = {
  sourceType: 'WALLET',
  sourceId: _userAWalletId_,
  type: 'WITHDRAW',
  detinationType: 'WALLET',
  destincateId: _userBWalletId_
  typeMeta: {
    amount: 50.0
  },
  ref: `app:deposite:${_userAWalletId_}:${_userBWalletId_}:${uuid}`
}
```

  - one WalletTransaction record to UserA

```
walletTransaction = {
  walletId: _userAWalletId_,
  transactionRequestId: transactionRequest.id,
  amountChange: -50.0,
}
```

  - one WalletTransaction record to UserB

```
walletTransaction = {
  walletId: _userBWalletId_,
  transactionRequestId: transactionRequest.id,
  amountChange: 50.0,
}
```

  - update WalletState record to UserA

```
walletState = {
  userId: _userId_,
  amount: _originalAmount_ - 50,
  lastTransactionRequestId: _transactionRequest_,
  lastWalletTransactionId: _walletTransactionId_,
}
```

  - update WalletState record to UserB

```
walletState = {
  userId: _userId_,
  amount: _originalAmount_ + 50,
  lastTransactionRequestId: _transactionRequest_,
  lastWalletTransactionId: _walletTransactionId_,
}
```

- **User can check her wallet balance**
  - just read the WalletState with the wallet Id

2. **System Level (just brain storming, not included in the assignment)**
   a. What DB should be used
   b. for the field storing money related amount, a proper number-related type should be used
   c. consider using RPC API standard
      i. assuming in the application will run in a micro-service architecture
   d. need to consider IdempotentId in API call
   e. assuming when record are stored in DB, there will be createdAt and modifiedAt and also related mechanism
   f. the data structure now will need to be created in sequence to DB. is it good?
   g. surely need to do Authentication and Authorisation