Solution Design overview

1. Solution Design consists of 3 layered design.
   a) Controller
   b) Service
   c) Database Access Layer

   TransferServiceEndpoint acts as Controller where the 3 endpoints are defined i.e. /createAccount, /transferAmount and /viewAccount

   Controller performs certain validations and calls Service Layer (TransferServiceRepository) for further processing. Service Layer executes the request and use Database Access Layer(AccountDao, TransferDao) for add/access/update database.

2. PESSIMISTIC_WRITE Database Locks are used to prevent incorrect state of Account record in case, multiple simultaneous transactions are being executed on the same account.
   Benefit of using PESSIMISTIC_WRITE lock instead of using OPTIMISTIC_WRITE /Default lock type in this scenario is enabling LOCK RETRY. If current transaction is not able to acquire the LOCK in configured hint time of 1000ms then retry will be made 3 times.

3. Scalability has been ensured by using Sequences. It enable unique key generation for JPA entities even when the multiple instances of application are deployed across multiple JVMs.

4. **An alternative approach for implementation** could be using JMS.
   Upon receiving a transfer request, we can store the request in DB, send the transfer request message in JMS queue and return the transfer_Id to as response of rest call with HTTP status code 202.

   On service layer, MDB configured for this queue can execute the request. Multiple instances of MDBs would be processing the request so simultaneous requests will be processed. Maximum number of MDB instances could be configured.

   In order to ensure that only one request for the two accounts be executed, groupid functionality of MDB should be used. Implementation should also be done to read DLQ and take appropriate action. User/GUI is required to check track the transfer request later using transfer id.

   In order to get transfer status/confirmation in response of same rest call, busy waiting using FUTURE (polling transfer request status change in database) or web-socket could be used.

Limitations

1. In case, transfer is successfully executed but user miss the response due to network issues. User could issue same request again with an impression that last request failed.
   Implementation could be made to handle duplicate transfer requests issued by the user.
2. Constant values and messages hardcoded in the code could be defined in property file and as system property using deployment configuration. i.e. defining in wildfly configuration.
3. Security should be implemented.

4. Deployment could be automated using Chef or puppet script.
5. H2 database has been used for this assignment. This should be replaced by some commercial database.
6. Unit test cases to be created corresponding to test cases mentioned in this document .

Deployment Instructions

1. Setup Wildfly
2. Add following h2 datasource configuration in wildfly standalone configuration:

```
        <datasource jndi-name="java:jboss/datasources/TRANSFER_SERVICE_DS" pool-
name="TRANSFER_SERVICE_DS" enabled="true" use-java-context="true">
          <connection-url>jdbc:h2:file:C:\Localdata\domainservice;MV_STORE=FALSE;INIT=CREATE SCHEMA IF NOT EXISTS
TRANSFERSERVICE;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
          <driver>h2</driver>
          <security>
            <user-name>sa</user-name>
            <password>sa</password>
              </security>
            </datasource>
```

   Replace “C:\Localdata” in above configuration by your user folder path
3. Start Wildfly
4. Clone repository available at github link below or use source code in attachment https://github.com/project000001/assignment
5. Compile the project and extract the ear file generated as ../transferservice/app/target/transferservice.app.ear
6. Copy EAR file to wildfly’s deployment folder
7. Once ear is deployed, Rest service is available for use
8. Below are the templates for accessing the rest services available in this application:
   a) Create Account

      http://localhost:8080/transferservice/rest/createAccount?name=<accountName>&balance=<balance>
      Replace <accountName> by an account name string and <balance> by value of initial amount(minimum 100)

      Example:
      http://localhost:8080/transferservice/rest/createAccount?name=user1&balance=1000
      returns below JSON with created account details.
      {"accountId":3,"name":"user1","balance":1000.0}

   b) View Account

      http://localhost:8080/transferservice/rest/viewAccount?accountId=<accountId>
      Replace <accountId> by numeric accountId

Example:
http://localhost:8080/transferservice/rest/viewAccount?accountId=3
returns below JSON with related account details.
{"accountId":3,"name":"user1","balance":1000.0}

Below mentioned error message is displayed in case of account Id is not available
Referred account no found

c) Transfer Amount
http://localhost:8080/transferservice/rest/transferAmount?drAccountId=<drAccountId>
&crAccountId=<crAccountId>&amount=<amount>

Replace <drAccountId> by numeric accountId of account to be debited
Replace <crAccountId> by numeric accountId of account to be credited
Replace <amount> by value of account to be transferred

Example:
http://localhost:8080/transferservice/rest/transferAmount?drAccountId=4&crAccountId=5
&amount=10

Below JSON with details of the transfer request and related accounts is returned
{"pk":2,"drAccount":{"accountId":4,"name":"abcd","balance":990.0},"crAccount":{"acco
untId":5,"name":"abc","balance":1010.0},"amount":10.0,"transferDate":151656744716 8}

Error message should be displayed in case of account Ids are not available or amount is not available in account to be debited.

TEST Cases:

1. Account created with 1000 Euro balance.
   Rest command:
   http://localhost:8080/transferservice/rest/createAccount?name=user1&balance=1000
   Expected output:
   JSON with details of the created account should be displayed
   {"accountId":3,"name":"user1","balance":1000.0}

2. Error displayed while creating account with initial amount less than 100.
   Rest command:
   http://localhost:8080/transferservice/rest/createAccount?name=user1&balance=10
   Expected output error message:
   Below mentioned error message should be returned.
   amount less than minimal allowed balance

3. Account details should be displayed for viewing request for valid AccountId.
   Rest command:
   http://localhost:8080/transferservice/rest/viewAccount?accountId=3
   Expected output:
   JSON with related account details should be displayed.
   {"accountId":3,"name":"user1","balance":1000.0}

4. Account details should be displayed for viewing details of non-existing accountId:
   Rest command:
   http://localhost:8080/transferservice/rest/viewAccount?accountId=31
   Expected output:
   Below mentioned error message should be returned.
   Referred account no found

5. Amount should be transferred between accounts with available balance in debit account.
   REST Command:
   http://localhost:8080/transferservice/rest/transferAmount?drAccountId=4&crAccountId=5&amount=10
   Expected Output:
   Below json with related transfer details and related accounts should be displayed
   {"pk":2,"drAccount":{"accountId":4,"name":"abcd","balance":990.0},"crAccount":{"accountId":5,"name":"abc","balance":1010.0},"amount":10.0,"transferDate":151656744716

6. Error should be displayed when transferring amount when required amount is not available in debit account.
   REST Command:
   http://localhost:8080/transferservice/rest/transferAmount?drAccountId=4&crAccountId=5&amount=10000
   Expected Output:
   Below mentioned error message should be displayed
   Account balance low