INTRODUCTION

This is simple rest service using spring boot 2.0.4 and maven 3.x. The application provides transfer service that transfers money from one account to another.

ARCHITECTURE

The application has three tier architecture that consists presentation layer, business layer and data access layer. The presentation layer is provided by controller classes. Users can interact with presentation layer. There are two controller classes as called AccountController and TransferController under the controller folder. It contains application logic and it passes input data to services. Other layer is business logic layer that is provided by service classes which serve as an intermediary for data exchange between the presentation layer and data access layer. Last layer is the data access layer for interaction with models and performing database operations. This layer is provided by repository interfaces . There are two repository interfaces named TransferRepository and AccountRepository.

Hsqldb is used to store persistent data. There are two tables that called account and transfer. Tables are created by using data.sql script files under the 'transfer/src/main/resources/account-table/' and 'transfer/src/main/resources/transfer-table/' folders. Foreign keys are not used when creating relations between transfer and account tables, because it is small application. It can be difficult to manage the consistency of the data in large applications.

Database operations are implemented by using Spring Data JPA. It focuses on using JPA in order to store data in a relational databases. It has the ability to create repository implementations automatically, at runtime, from a repository interface. There are two repository interfaces under the repository folder. AccountRepository interface works with Account class and TransferRepository interface works with Transfer class under the Entity folder. The Account and the Transfer classes are annotated with @Entity, indicating that is a JPA entity. These entities will be mapped to database tables named Account and Transfer.

The application handles concurrent transfer requests with Optimistic Locking approach. Optimistic Locking with Spring Data JPA is implemented by the JPA implementation used. In order to use optimistic locking, Account entity has to include a version property with @Version annotation. While using it, each transaction which reads data holds the value of the version property. Before the transaction wants to make an update. It checks the version property again. If the value is changed in the meantime an ObjectOptimisticLockingFailureException will be thrown. Otherwise, the transaction commits the update and increments a value version property.

The concurrent writing request to the account table has been tested by the ConcurrentTransferRequestTest test class. The accountOfTransferRequest1 can be thought as a one request and the accountOfTransferRequest2 as an another request. In case of both of them wanted to update to balance variable, the ObjectOptimisticLockingFailureException exception was thrown.

Unit tests of almost all requests have been developed using Junit Test framework . Tests are located under the 'transfer/src/test/java/com/ingenico/epayment/transfer/' folder.

Logback was used to logged the transactions. By default ,Spring boot uses Logback for its logging.

Custom exceptions were constructed for making easier to understand application under the Exception folder. When throwing exceptions, Http status codes are considered.

LIMITATIONS

In this application, users can have only one account. In real solutions, if users would be stored in customer entity and also the relations would be constructed between customer and account entities, users could have more than one account in payment solution.

BUILDING AND RUNNING

This application is packaged as a jar which has Tomcat 8 embedded. No Tomcat or JBoss installation is necessary. You run it using the java -jar command.

- Clone this repository
- Make sure you are using JDK 1.8 and Maven 3.x
- You can build the project and run the tests by running mvn clean package
- Once successfully built, you can run the service by this command:

```
java -jar target/transfer-0.0.1-SNAPSHOT.jar
```

The application will be started on 8080 port.