Revolute code challenge

Author: Artem Bogdan
Date: 12.06.2019

The document describes high level design, implementation details, limitations and possible improvements

Prerequisites

Maven - mandatory
Git client - mandatory
SoapUI or any other REST testing tool - desirable

Assumptions

- Design follows ‘make it simple’ requirement and does not provide prodiction-like solution
- Authentication is done via http header *Authorization* as per OAuth 2.0 protocol. However there is no real check against identity provider. Any REST call having this header is considered as authenticated.
- Authorization is not supported as per requirement.

Repository

https://github.com/ArtemBogdan/revolut

How to run

1. Navigate to the root directory (where pom.xml file is located)
2. Run `mvnw clean compile exec:java` command
3. To stop press Ctrl-C from the same console where step #2 was executed

Design

Data-model is consisted of two database tables

account

The primary table for storing bank accounts.

Column	Type	Constraint	Description

account_id	serial	PRIMARY KEY	Primary key which is generated automatically by database after every insert
customer_name	VARCHAR(50)		The name of the account's owner.
amount	NUMERIC(10, 2)	NOT NULL DEFAULT 0	Account balance. Can be changed only as part of financial transaction.
status_code	integer	NOT NULL DEFAULT 1	1 - Active 2 - Suspended 3 - Terminated
created_when	TIMESTAMP	not null default CURRENT_DATE	Automatically populated with the current timestamp
user_name	user_name	VARCHAR(20)	A user who created the account.

Transaction

Table for tracking account's transactions. Three scenarios are possible:

- Deposit: *account_to* is populated, *account_from* is null
- Withdraw: *account_to* is null, *account_from* is populated
- Transfer: *account_to* is populated, *account_from* is populated

Amount is always populated with positive value. At least one of (*account_to*, *account_from*) must be populated. A transaction decreases balance for *account_from* (if populated) and at the same time increases balance for *account_to* (if provided) by the value of *amount.*

Column	Type	Constraint	Description
transaction_id	serial	PRIMARY KEY	Auto generated transaction id
account_from	integer	REFERENCES account	Reference to account table
account_to	integer	REFERENCES account	Reference to account table
amount	NUMERIC(10, 2)	NOT NULL	Transaction amount. Always populated with positive value
timestamp	TIMESTAMP	not null default CURRENT_DATE	Autogenerated transaction timestamp

user_name	VARCHAR(20)		A user who made the transaction

Implementation details

Application layer language: Java 8
Build tool: Maven
Database: Java based In-memory PostgreSQL (org.postgresql)
Web layer: Sparkjava (com.sparkjava)
Service and database classes are declared as interfaces allowing to replace implementation transparently when needed.

Transaction management (concurrency)

All concurrent access is made using database locking (not as java synchronization) which allows horizontal scaling application layer (multiple application nodes). Database transactions are controlled by application (service) layer.

Database layer classes

Interface com.revolut.codechallenge.artembogdan.dao.Database - declares methods for getting database connection and two wrappers for transactional operation:
transactionalOperation - it takes function which must be executed in a whole transaction. The wrapper takes care of committing transaction and rolling back in case of database exception.
readOperation - operation for reading data which does not require commit or rollback.

Interface com.revolut.codechallenge.artembogdan.dao.DatabaseOperations - make operations with database tables. It returns entities with the structure similar to corresponding database table.

Service layer classes

Interface com.revolut.codechallenge.artembogdan.service.AccountService - the main service accessible to the API consumers. Following operations are supported

- createAccount - creates new account
- getAccount - finds the account by the number
- modifyAccount - modifies account; Since balance amount can be modified only via transaction then the only field which can be changed is customer_name.
- getAccountTransactions - returns account with its' transactions
- makeTransfer - transfer money from one account to another. It locks both accounts to avoid concurrent access, makes validation that balance is enough, than it creates transaction record and modifies balance of both accounts
- deposite - locks account, creates transaction record and increases account's balance

- withdraw - locks account, check if balance is enough, than creates transaction record and decreases account's balance

Web layer

Class com.revolut.codechallenge.artembogdan.web.RestController - the class which routes endpoints to AccountService API. Main features:

- It automatically start http server on default http://localhost:4567 address
- Requests and responses are accepted by the http protocol and are supposed to be in JSON format
- It handles business exceptions from service layer and wraps them into JSON with meaningful information.
- It authenticates the user by Authorization http header. The header value is supposed to start with 'Bearer '.

Endpoints

Endpoint	HTTP Method	Description	Payload sample
/account/:id	GET	Gets account	
/account/:id/transactions/	GET	Gets account with all transaction	

/account/	POST	Creates new account with zero balance	{
			"customerName": "Some name"
			}

/account/transfer/	POST	Transfer money between accounts	{
			"accountFrom": 1,
			"accountTo": 2,
			"amount": 12.22
			}

/account/deposite/	POST	Add money to account	{
			"accountTo": 1,
			"amount": 20.20
			}

/account/withdraw/	POST	Withdraw money	{
			"accountFrom": 1,
			"amount": 9.48
			}

Error handling

Error handling at the service layer is supported via ServiceException class. The class contains error code defined in ErrorCatalogue enumeration. Following codes are supported:

ACCESS_DENIED - Rest controller did not receive Authorization header
UNKNOWN_ERROR - Unexpected exception
DATABASE_ERROR - Database rejected the operation
INVALID_INPUT_PARAMETER - Validation of input parameter is failed
NOT_FOUND - Requested entity is missing in database
LOCK_TIMEOUT - Database lock was not acquired within configured time
ACOUNT_MUST_BE_ACTIVE - Attempt to make transaction against terminated or suspended account
LOW_ACCOUNT_BALANCE - The balance is low to make a withdrawal or transfer