Wallet Operation Microservice

Developer : Oguzhan Demirer
Author : Oguzhan Demirer

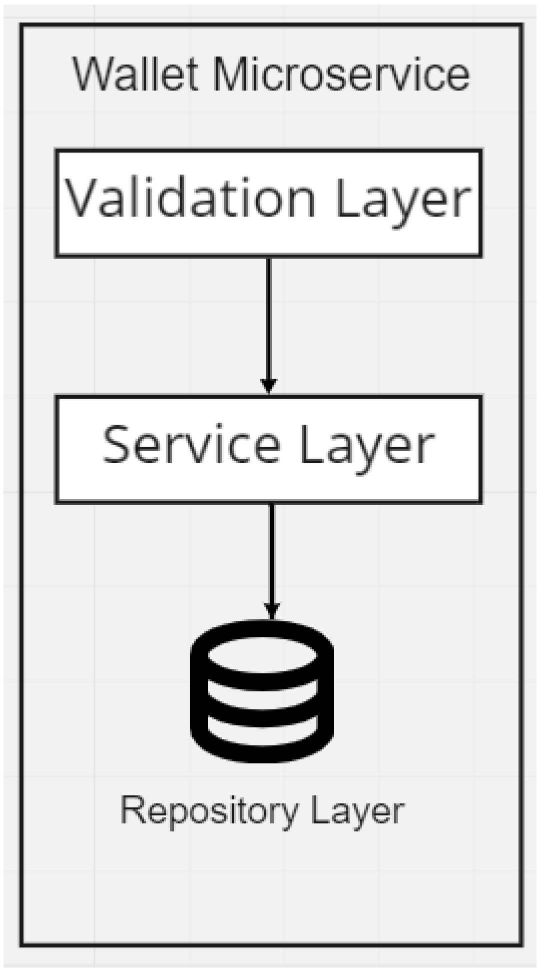
Purpose of Service

This microservice is only responsible for Wallet operations. Simply Credit/Withdrawal and see transaction history for each customer with current balance.

Technology

- Java.
- Spring Boot.
- H2 in memory database & JPA Persistance, SQL.
- Maven.
- JUnit for unit tests.

Service Architecture & Design



Service simply contains three layers. These layers are;

Validation Layer

Our application has custom implemented annotations for validation. Annotations implemented in **ValidationLayer.java.** Simply we are checking

- **Transaction id uniqueness.**
- **Balance sufficient for withdrawal.**
- **Customer id and wallet.**

Our custom Validation Annotations are;

- **@CreditOperationValidation**
- **@WithdrawalOperationValidation**
- **@WalletDetailsValidation**

Service Layer

WalletOperationService.java, As best practice in microservices each transaction designed as atomic and independent. Simply we have three services;

walletDetails() – Gets customer balance & transactions
withdrawalTransaction() – Withdrawal transaction.
creditTransaction() – Deposit money transaction.

Repository Layer

We have Wallet.java and Transaction.java tables. Each wallet ONLY has one owner.(customer id). We are using ORM for fetch Transacions with customer id column. It has One-To-Many relationship which means that each customer may have more than one transaction. **Transaction.java** has transaction id as a key. Each transaction id must be unique.

Application Models

WalletOperationServiceResponse.java – This is base response of our microservice. We are using Java generics for type safety and flexibility.
WalletDetailResponse.java – Response object for walletDetails(). Object contains Wallet and all transactions which wallet has.
TransactionResponse.java – Response object for Credit/Withdrawal transactions.
TransactionRequest.java - Request object for Credit/Withdrawal transactions.

Exception Handler

Our application has its own lightweight exception handler **WalletOperationServiceExceptionHandler.java .** We are properly catch specific business logic violations and exceptional cases and logging them properly with exception time. These implemented exception cases are ;

walletServiceValidationException( ): This is our validation / business logic exception. For example, Validation exception thrown if transaction id in request already exists, there are not enough balance for requested amount withdrawal and customer id does not exist. We are returning meaningful message with HTTP 400, BAD REQUEST.

unexpectedInternalTechnicalError ( ): This is out internal server error handler. Implemented for catch unexpected technical runtime errors. We are returning meaningful message with HTTP 500, INTERNAL SERVER ERROR.

Unit Tests

Unit tests implemented for our microservice. Aim of tests are measure robustness of service and transaction correctness. Unit test cases implemented for microservice ;

- Data & Business Validations.
- Application Transaction Correctness.

Application Usage & Test Tips

Application is simple Spring Boot application so you can start it easily. You can test service with Postman.

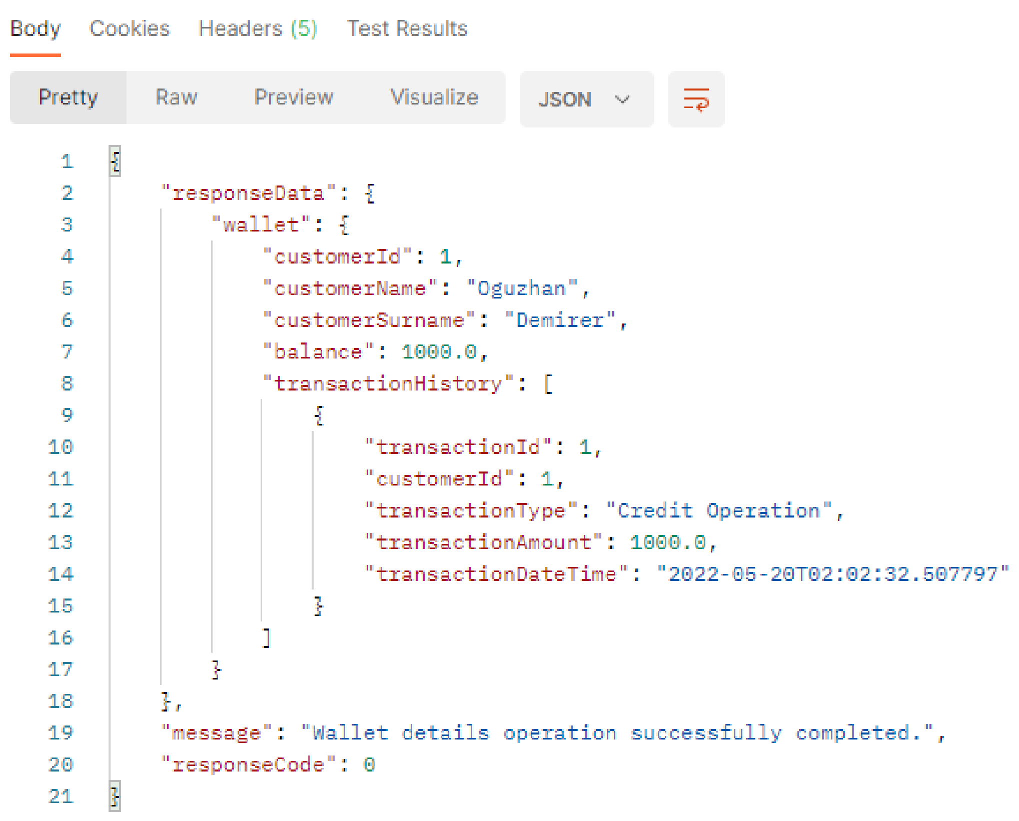
IMPORTANT : In resources directory there are data.sql script for initially create 4 customers with 4 deposit operations each of them. In your requests you can use these customer id's. Initial databse data as below;

"CustomerId: 1", "Balance : 1000" -> HAS 1 transaction with "transactionId : 1"
"CustomerId: 2", "Balance : 1000" -> HAS 1 transaction with "transactionId : 2"
"CustomerId: 3", "Balance : 1000" -> HAS 1 transaction with "transactionId : 3"
"CustomerId: 4", "Balance : 1000" -> HAS 1 transaction with "transactionId : 4"

Important: You can use these customerId's while you are doing your tests. This microservice only responsible for wallet operations so new customer creation or update operations CANNOT be done with this microservice. It is out of scope for this microservice.

To see customer balance and transaction history (CustomerId : 1)

http://localhost:8080/wallet/walletDetails/1 (GET METHOD)



```
{
    "responseData": {
        "wallet": {
            "customerId": 1,
            "customerName": "Oguzhan",
            "customerSurname": "Demirer",
            "balance": 1000.0,
            "transactionHistory": [
                {
                    "transactionId": 1,
                    "customerId": 1,
                    "transactionType": "Credit Operation",
                    "transactionAmount": 1000.0,
                    "transactionDateTime": "2022-05-20T02:02:32.507797"
                }
            ]
        }
    },
    "message": "Wallet details operation successfully completed.",
    "responseCode": 0
}
```

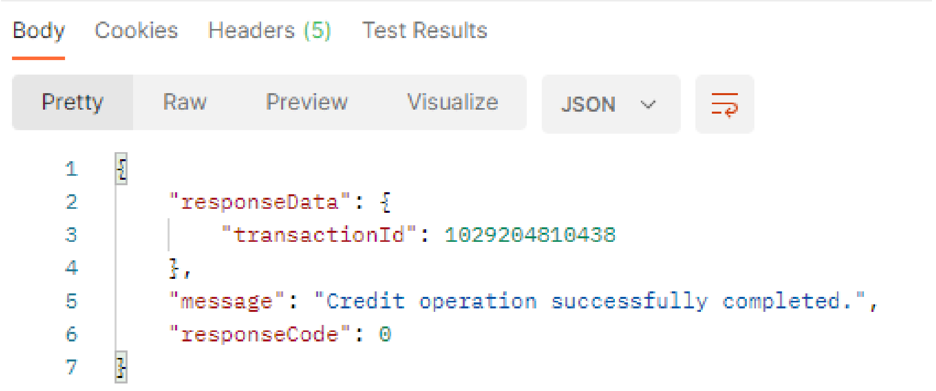
Credit / Withdrawal operation

http://localhost:8080/wallet/credit (POST METHOD)

http://localhost:8080/wallet/withdrawal (POST METHOD)

Request Body Example;

```
{
  "transactionId": 1029204810438, //Must be Unique
  "customerId": 1, // CustomerId must be exist. Use 1,2,3 OR 4
  "walletOperationAmount": 115.0 //Amount for wallet operation
}
```



```
{
    "responseData": {
        "transactionId": 1029204810438
    },
    "message": "Credit operation successfully completed.",
    "responseCode": 0
}
```