DDD with SpringBoot Exercise
RESTFUL API FOR PAYMENTS

Tomislav Landeka (tomo.landeka02@gmail.com)
https://www.linkedin.com/in/tlandeka/
https://github.com/tlandeka/domain-driven-design-in-spring-boot

Foreword

This document describes the application design that I have made. Before You start the read I would like to mention that I do not have for example a 100% test coverage, but I presented the way I write tests. These are the features that have not finished completely(because I wanted to spend more time on the architecture), but I have done it partially which means that I have shown the way I do it:

- I do not have 100% test coverage
- I did not validate **all** client inputs
- The API returns HTTP Bad Request for every problem that happens
- I have not logged the key information

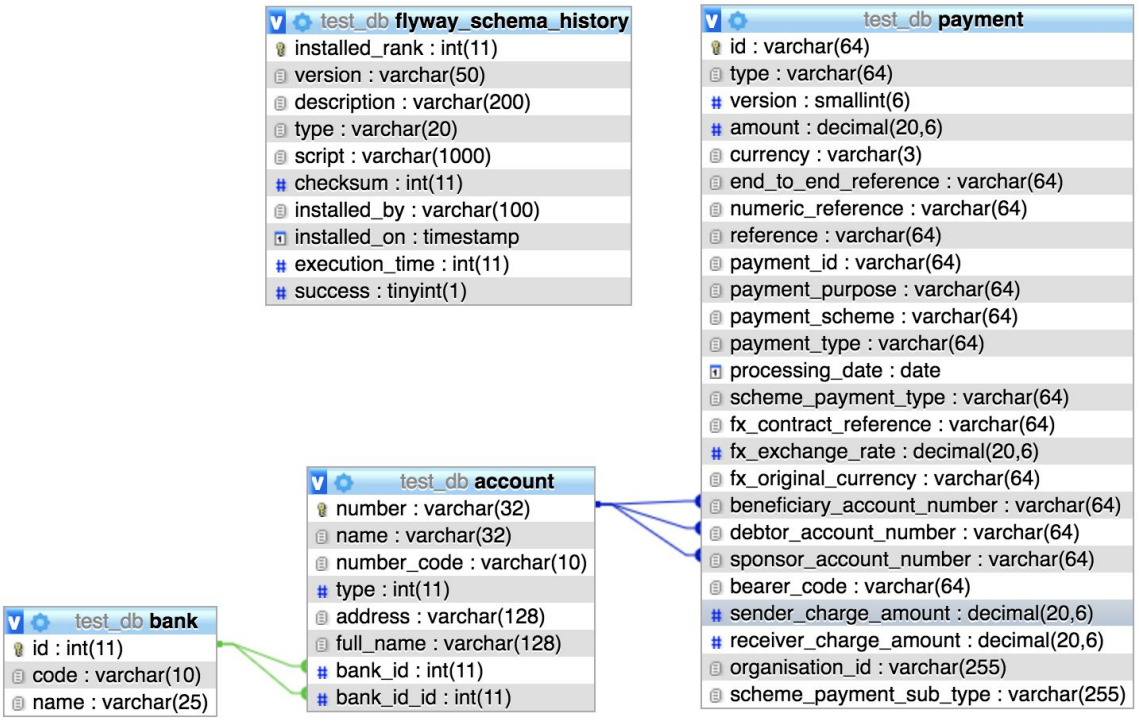
The problem and the solution

I created a Restful API for transfer money from one bank account to another. The API must be able to receive HTTP request with JSON body content.
The JSON content contains information like:

- Basic information: payment id, amount, currency, paymentType, description, etc.
- Money sender account details + bank details
- Money receiver account details + bank details
- Sponsor account details + bank details

Database design

I have made an application that can create one payment and store it to the database but with a condition that account and bank details already exist in the database(Let's assume that someone already added the account and bank in our payment system) therefore I have made database migration that inserts the test “account and bank” data into database. According to that I have designed the database this way:



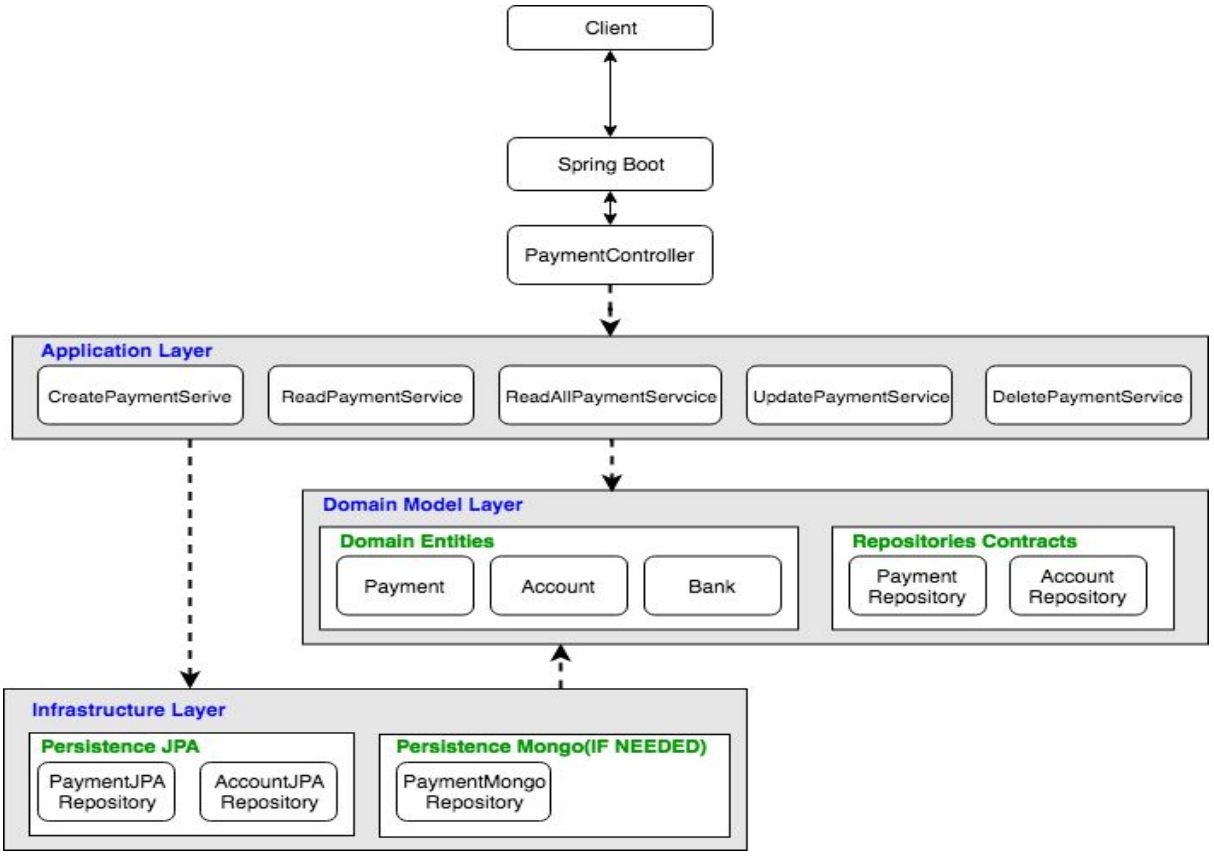
This design tells that one bank can have multiple accounts, and one account can have multiple payments. The table **flyway_schema_history** is auto-generated table by the migration tool.

Implementation

While I was working on implementation I followed some approaches from Domain Driven Design(DDD). The DDD follows the best practices, SoC, SOLID principles, decoupling the dependencies, etc. When we talk about tests, this is a test coverage statistic:

Element	Class %	Method %	Line %
application	93% (15/16)	63% (99/155)	75% (218/290)
controller	100% (2/2)	0% (0/6)	12% (2/16)
domain	100% (4/4)	47% (20/42)	71% (74/103)
infrastructure	100% (3/3)	77% (7/9)	80% (12/15)
ExcerciseApplication	100% (1/1)	0% (0/1)	33% (1/3)

Architecture design

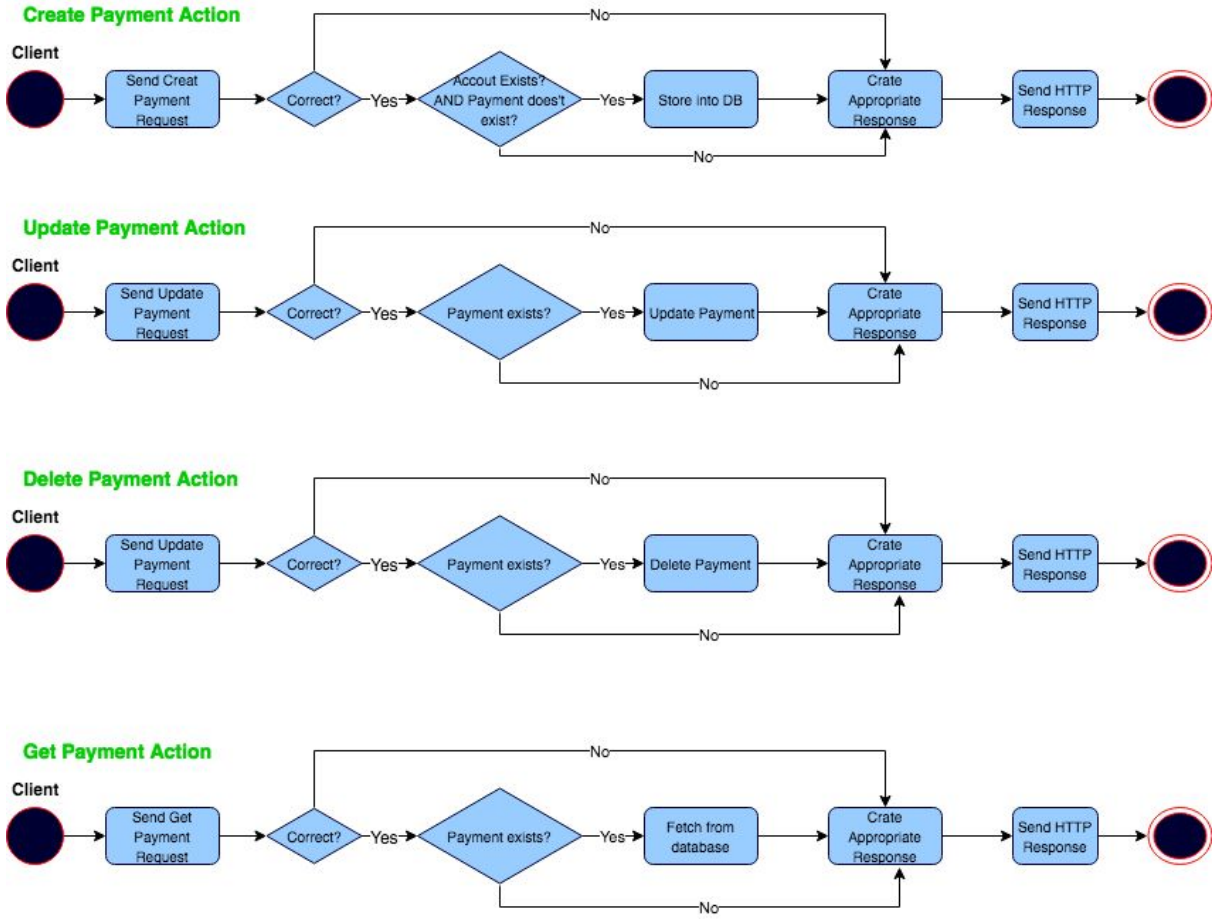


Source code

The source code of the application is available on this link: https://github.com/tlandeka/domain-driven-design-in-spring-boot.
Take a look on the README.md file in order to run the application.

Activity flow diagrams

Next picture shows activity flow diagrams of the application for four actions:



Basic API usage

The API provides five actions in order to manage the payments.

Create payment action

Method: **POST**
Route: /payments/{paymentId}
Body example:


```
{
  "type": "Payment",
  "id": "4ee3a8d8-ca7b-4290-a52c-dd5b6165ec22",
  "version": 0,
  "organisation_id": "743d5b63-8e6f-432e-a8fa-c5d8d2ee5fcb",
  "attributes": {
    "amount": "100.21",
    "beneficiary_party": {
      "account_name": "W Owens",
      "account_number": "31926819",
      "account_number_code": "BBAN",
      "account_type": 0,
      "address": "1 The Beneficiary Localtown SE2",
      "bank_id": "403000",
      "bank_id_code": "GBDSC",
      "name": "Wilfred Jeremiah Owens"
    },
    "charges_information": {
      "bearer_code": "SHAR",
      "sender_charges": [
        {
          "amount": "5.00",
          "currency": "GBP"
        },
        {
          "amount": "10.00",
          "currency": "USD"
        }
      ],
      "receiver_charges_amount": "1.00",
      "receiver_charges_currency": "USD"
    },
    "currency": "GBP",
    "debtor_party": {
      "account_name": "EJ Brown Black",
      "account_number": "GB29XABC10161234567801",
```

```
      "account_number_code": "IBAN",
      "address": "10 Debtor Crescent Sourcetown NE1",
      "bank_id": "203301",
      "bank_id_code": "GBDSC",
      "name": "Emelia Jane Brown"
    },
    "end_to_end_reference": "Wil piano Jan",
    "fx": {
      "contract_reference": "FX123",
      "exchange_rate": "2.00000",
      "original_amount": "200.42",
      "original_currency": "USD"
    },
    "numeric_reference": "1002001",
    "payment_id": "123456789012345678",
    "payment_purpose": "Paying for goods/services",
    "payment_scheme": "FPS",
    "payment_type": "Credit",
    "processing_date": "2017-01-18",
    "reference": "Payment for Em's piano lessons",
    "scheme_payment_sub_type": "InternetBanking",
    "scheme_payment_type": "ImmediatePayment",
    "sponsor_party": {
      "account_number": "56781234",
      "bank_id": "123123",
      "bank_id_code": "GBDSC"
    }
  }
}
```


Get Payment

Method: **GET**
Route: /payments/{paymentId}

Get All Payments

Method: **GET**
Route: /payments

Delete Payment

Method: **GET**
Route: /payments/{paymentId}

Update Payment

I made it possible to update only three properties just to show an example of updating. These are the properties that can be updated: *payment_type, currency* and *payment_scheme*.

Method: **PATCH**
Route: /payments/{paymentId}
Body example:

```
{
  "payment_type": "TestPayment",
  "currency": "BAM",
  "payment_scheme": "SWIFT"
}
```