Payment gateway

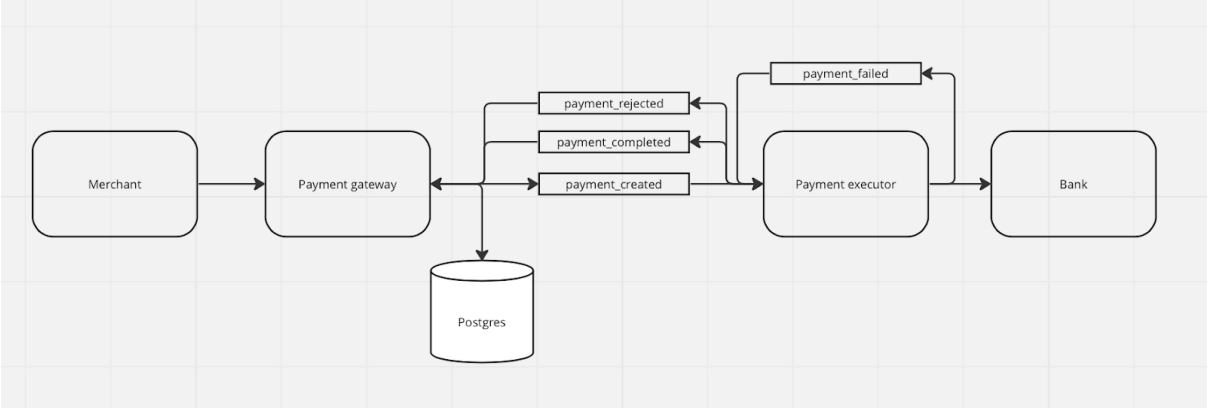
Functional requirements

- A merchant should be able to process a payment through the payment gateway and receive either a successful or unsuccessful response.
- A merchant should be able to retrieve the details of a previously made payment.

Nonfunctional requirements

- The solution should be easy to scale up.
- The solution should avoid double charging
- The solution should be highly available and fault-tolerant

High-level design



Workflow of payments

- Merchant invoke method pay of payment gateway service
- Payment gateway service stores payment to database and publishes event to topic “patment_created”
- Payment executor reads topic “patment_created” and invoke acquiring bank API
- If the payment accepts by acquiring bank then payment executor publishes message to topic “payment_completed”
- If the acquiring bank API throws an exception then payment will be placed to the “payment_failed” topic. And the payment will be retried.
- If the acquiring bank does not accept payment at all payment will be placed to “patment_rejected” topic.
- During all this processes payment gateway awaits payment terminated status in topics “payment_completed” or “patment_rejected”

To avoid double charging we have to use a unique payment id. Merchant should ask for this id from the payment gateway service.
All the services use this payment id as an idempotency token.
Using this payment id the merchant can gather information about payments.

If we have a lot of payments we can scale up payment executor service. It will evenly spread RPS to bank services.
We can split our database using, for example, merchant Id as a shard key.

The solution doesn't cover

- Security
- 3d payment verification
- Other payment methods (it covers only bank cards)
- Database sharding
- Merchant to the acquiring bank matching (in production different merchants can have different banks)

To run this solution

- Download it from github https://github.com/Ref83/checkout
- If you use MacOS run script run.sh
    - It runs docker-compose.yml to initialise infrastructure (Postgres and Kafka)
    - build, test and run all solutions (Gateway, Executor and Bank)
- If you use the other OS.
    - It runs docker-compose.yml to initialise infrastructure (Postgres and Kafka)
    - Build and run all three solutions in your IDE

Try solution

You can use swagger to invoke methods of payments gateway service.
http://localhost:7000/swagger/index.html

Or you can use curl like this

```
curl -X 'POST' \
  'http://localhost:7000/payment_gateway/pay' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "paymentId": "89e87e44-2e96-4fd4-8a0f-2f7cea96cfc1",
  "cardNumber": "0000000000000000",
  "expiry": "03/26",
  "cvv": "234",
  "cardHolder": "card holder",
  "amount": 100,
  "currency": "EUR",
  "merchantId": "123"
}'
```

Prepared cases

To get errors from the acquiring bank API use the following card holders:

- "scammer" - throws exception "Fraud transaction detected"
- "empty account" - throws exception "Not enough money"
- "timeout" - emulates timeout

Payment.Domain.Core

Core domain of payments. It is better to place it in nuget.