Hand In 1

Project Description

EventBank is a overly simplified banking system to study event driven architectures. The first use case that we are interested is to execute **card payments** that come in from an external payment provider. The service that are involved are the payment service that validates the card authorization and the accounts service that managed the balances of the customer accounts of the bank. The second feature of the EventBank is that accounts can be opened. Because the **opening of a bank account** can contain a lot of logic we use a process engine to manage the opening of the accounts.

Portfolio Description

- E1: todo
- E2: We implemented a event notification between the cards service and the notifications service
- E3: We implemented a process to open an account (todo jonathan)
- E4: We thought in-depth about the tradeoff between commands and events. The result is the risk based approach and asynchronous execution of payments.
- E5: We implemented a saga pattern for the payment execution with stateful retry.
- E6: We created the hand-in documentation and polished the project.

The messaging parts were mainly implemented by Gian-Luca. The orchistration parts (process engine) were implemented by Jonathan. However, we did the concept of the processes together and also reviewed each others code.

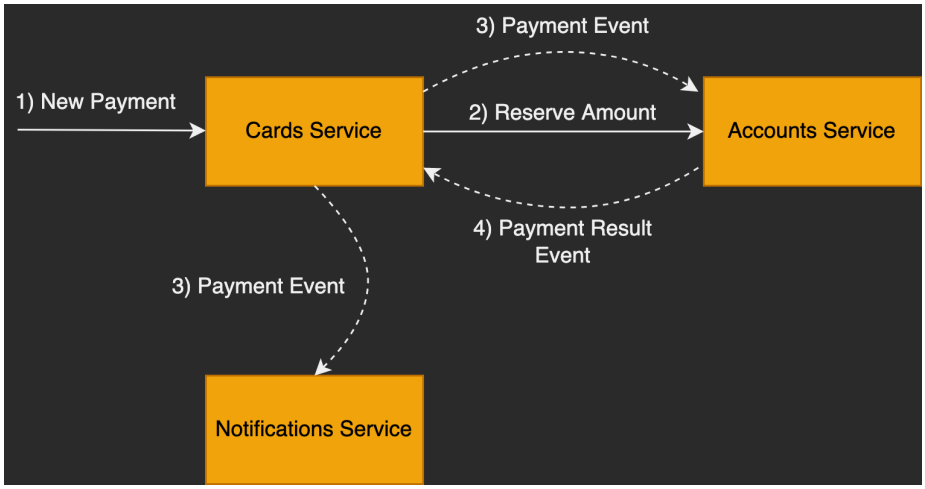
Implementation of the Card Payments

The card payments have some important requirements that we need to meet.

- Cards payments need to be processed fast. In practice we would have a hard time limit in which we either have to process the card payment or we have to reject it
- Card payments need to be atomic. If we remove the money from a customer bank account we need to make sure the card payment is accepted. Otherwise the customer might be confused and pays again which leads to a double payment. If we accept the card payment we need to make sure the money is removed from the customer bank account.

Those requirements are very simplified compared to a real banking system. And one might argue that this is not a typical use case for event-driven systems. However, it would be kind of boring to study event driven architectures if we only have to deal with a very simple use case that has no requirements where event driven architectures struggle.

The following figure depicts the architecture of the services that handles the card payments and the different interactions.



Description:

1. The payment service receives the card payment from the external payment provider. It checks the authorization and business rules.
2. The cards service sends a synchronous message to the accounts service to reserve the balance of the payment. If the balance is not available the accounts service will reject the reservation. This step is optional and only executed if the cards service categorizes the payment as risky. In our mock scenario a payment is risky is the amount is higher than 1000. In a real world scenario this would be determined by the a much more complicated process.
3. The payments service now starts a new card payment sage. After this point the payment service already response to the external payment provider with with a success result. The first step of the sage is to send a payment event. The accounts service subscribes to the payment event and will execute the payment. Other services could also listen to the payment event and perform other actions such as fraud detection. In fact the notification service also listens to this type of events and sends a mock notification to the customer.
4. After the payment was successfully executed the accounts service will send a payment success event. When the payment was successfully executed the cards service will close the saga. If no such event is received the payment will be retied again and after a certain number of reties a error will be logged.

Implementation of the Account Opening

todo Jonathan