КР №3 Косталевский Даниил Кириллович БПИ-236

По условию задания нужно было создать серверную часть онлайн-магазина, которая отвечает за работу с оплатами и заказами. Если говорить подробнее - нужно было сделать два микросервиса:

1. Payments-service, который позволяет создавать счет, пополнять баланс, просматривать баланс счета
2. Order-service, который позволяет создавать заказы, просматривать список заказов, просматривать статус отдельного взятого заказа

рассмотрим реализацию каждого из этих сервисов подробнее.

Payments-service
Напомню, что этот сервис отвечает за создание и управление пользовательским счетом, принимает запросы на пополнение баланса.

Рассмотрим сущности этого сервиса:

1. Inbox и outbox - отвечают за обработку входящих и исходящих сообщений через базу данных. Inbox хранит входящие сообщения от kafka в таблице inbox_messages. Outbox отвечает за сбор исходящий сообщений в таблице outbox_events перед публикацией в kafka. такой подход позволяет достигать at-most-once, обрабатывать одно сообщение несколько раз, отлавливать некорректные обработки, предотвращать потерю сообщений и гарантирует атомарную запись бизнес-данных и событий в одной транзакции.
2. Account - модель счета. Является основой этого сервиса. Поля - user_id и balance.
3. Transaction log - хранит историю операций.
4. Rest-контроллер paymentsController.
5. Сервисный слой accountService.

логика взаимодействия следующая: точкой входа является REST-контроллер, который слушает три эндпоинта: /accounts - для создания нового счета. получается DTO CreateAccountRequest, делегирует дальше в сервис accountService. Второй эндпоинт - /topup - получается user_id и сумму и делегирует в сервис. Третий эндпоинт - get /accounts - возвращает текущее состояние счета.
Далее контроллер делигирует сервису, где в зависимости от эндпоинта реализуется

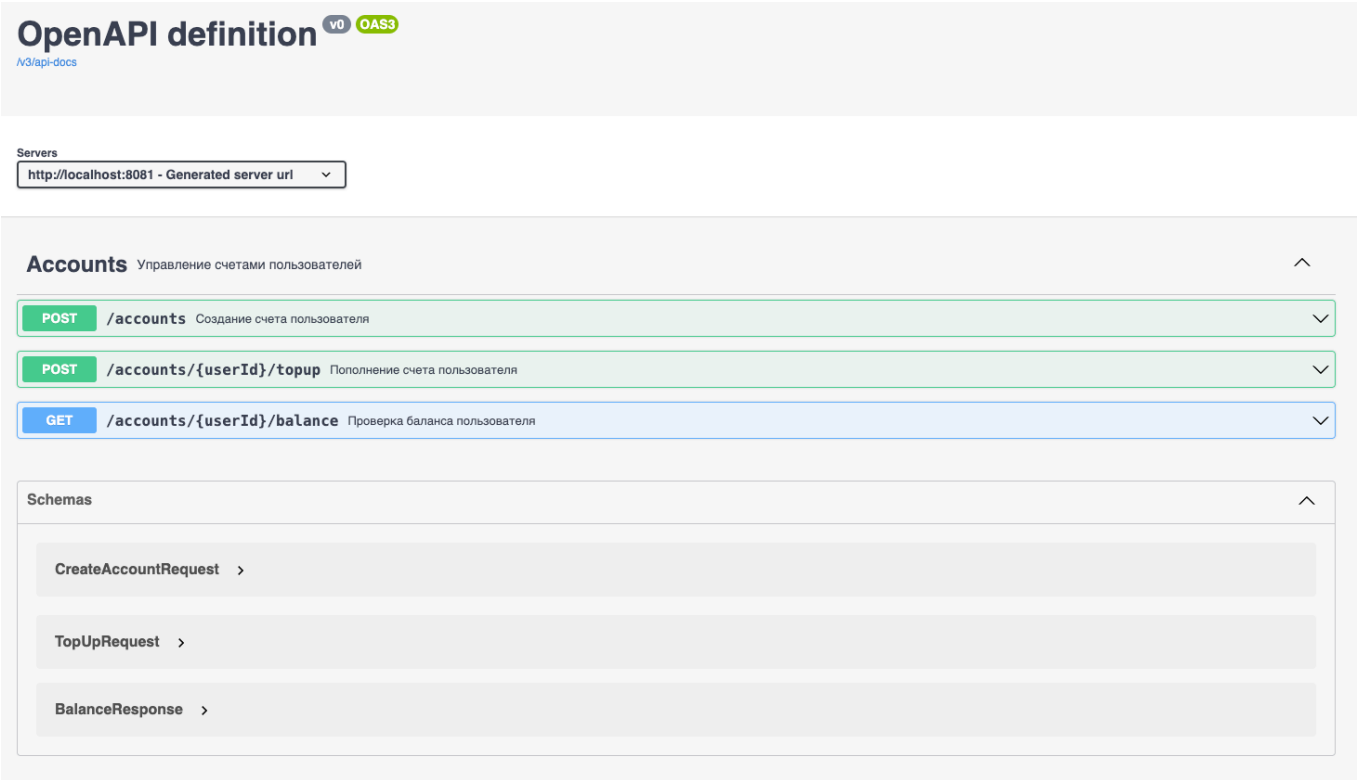
логика. createAccount - проверка на существование аккаунта по такому id и сохранение в БД с помощью accountRepository. TopUp - загружает запись счета, увеличивает balance на введенную сумму, сохраняет получившийся объект и передает данные transaction_log. TryDebit - загружает счет и следит за тем, чтобы balance >= amount, если средства есть, то сумма вычитается, если не хватает, то возвращает сообщение об этом. Каждый представленный метод обернут в Transactional и гарантирует атомарность операций.
Обмен сообщениями производится через inbox. Метод inboxListener принимает событие о создании заказа, сериализует его в объект и сохраняет в таблицу запись об этом событии. Далее производится обработка входящего сообщения и по итогам работы сервиса формируется paymentResultEvent, которое сообщает об успешности операции. Далее это событие записывается в таблицу и outboxPublisher отправляет его обратно.

Покрытие тестами

payments-service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.example.shop.payments.entity		46 %		n/a	29	47	42	88	29	47	0	4
com.example.shop.payments.service		71 %		90 %	9	22	22	81	8	17	1	5
com.example.shop.payments.config		0 %		n/a	5	5	17	17	5	5	1	1
com.example.shop.payments.controller		0 %		n/a	4	4	8	8	4	4	1	1
com.example.shop.payments		0 %		n/a	2	2	3	3	2	2	1	1
Total	344 of 702	50 %	1 of 10	90 %	49	80	92	197	48	75	4	12

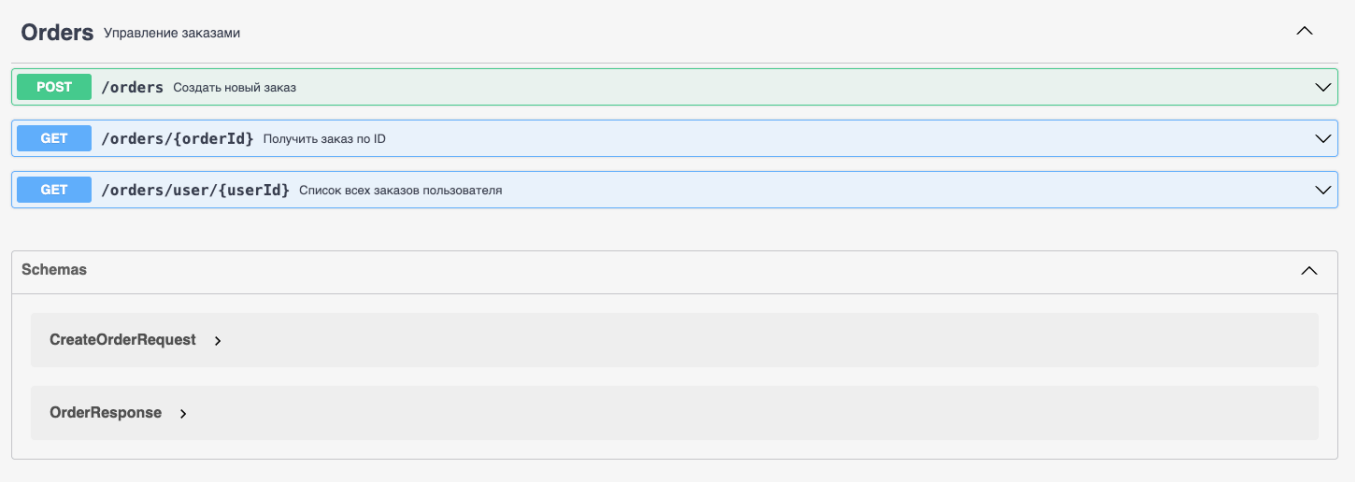
Документация api



Orders-service

Обеспечивает работу с заказами (создание и управление). Позволяет создать заказ, просмотреть статус конкретного заказа, получить список заказов конкретного пользователя.

Компоненты в данном сервисе взаимодействуют следующим образом: все начинается с REST-контроллера, который принимает CreateOrderRequest (запрос на создание заказа) и дальше вызывает метод createOrder в сервисной части. Внутри сервисной части генерируется orderId, создается сущность заказа, которая сохраняется в таблицу, формируется событие для payments-service, которое тоже сохраняется в таблицу со статусом нового. Эта транзакция коммитится (гарантируется атомарность записи заказа и события). После этого outbox publisher выбирает все события со статусом нового из бд, выполняет блокирующую отправку в kafka, обновляет статус события с нового на отправленное. Потом после логики в другом сервисе приходит событие с новым статусом: оплачено или провалено. Обновленная сущность сохраняется в бд.



Помимо всех остальных сервисов был корректно реализован docker-compose, все микросервисы были упакованы и корректно разворачиваются с помощью docker

compose up.


shop
Running (6/6)
13.07%
12 seconds ago
postgres-pa
9141c73dff85
postgres:15
Running
0.21%
5432:5432
13 seconds ago
postgres-or
b94fa89c6992
postgres:15
Running
0.33%
5433:5432
13 seconds ago
orders-servi
ed703731ed0
shop-orders-service
Running
4%
8082:8082
12 seconds ago
payments-s
f246e8f13726
shop-payments-servi
Running
3.47%
8081:8081
12 seconds ago
kafka-1
fc3608d9deac
confluentinc/cp-kafk
Running
4.68%
9092:9092
12 seconds ago
zookeeper-1
cca4e534cff3
confluentinc/cp-zool
Running
0.38%
2181:2181
13 seconds ago