

### 3. Какой сервис должен реализовывать какую логику и почему?

Распределение логики между различными сервисами в архитектуре – сложная и очень важная задача. От правильности её выполнения во многом зависит эффективность и масштабируемость всей системы. Вот некоторые принципы, которые помогут в этом:

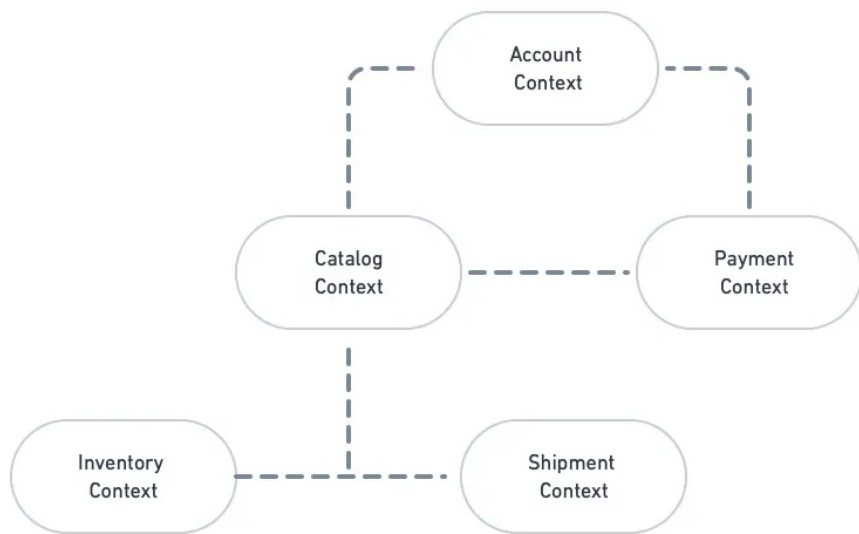
- Принцип единственной ответственности: Каждый микросервис должен быть ответственен за одну конкретную функцию или бизнес-процесс. Это помогает сделать сервисы более модульными, масштабируемыми и удобными для поддержки.
- Разделение команды и сервиса: Этот принцип предполагает, что каждая команда должна быть ответственна за один или несколько микросервисов, что позволяет повысить автономность и упростить координацию.
- Определение границ сервисов по контексту: Границы сервиса должны определяться на основе предметной области и контекста, в котором они функционируют. Например, если у вас есть функциональность, которая используется в нескольких контекстах, то, возможно, имеет смысл выделить её в отдельный микросервис.

Приведём пример: предположим, у нас есть система онлайн-магазина. Мы можем иметь следующие сервисы:

- Сервис управления инвентарем: этот сервис отвечает за отслеживание товаров на складе, их цен и состояния.
- Сервис обработки заказов: этот сервис управляет всеми аспектами обработки заказов (процессом), включая проверку доступности товаров на складе(через Сервис управления инвентарем), обработку платежей(через Сервис управления платежами) и управление доставкой(через Сервис управления клиентами).
- Сервис управления платежами: этот сервис отвечает за обработку платежей, включая взаимодействие с платежными шлюзами и управление счетами клиентов(с помощью Сервис управления клиентами).
- Сервис управления клиентами: этот сервис управляет всеми данными, связанными с клиентами, включая адреса для доставки.

Каждый из этих сервисов работает автономно и обладает своей собственной бизнес-логикой, которую он должен реализовывать, и общается с другими сервисами через определённые интерфейсы для обмена данными.

Это позволяет делить сложную систему на небольшие, управляемые части, которые могут разрабатываться и поддерживаться независимо друг от друга, что упрощает процесс разработки и поддержки.



**4. Какой сервис должен просто быть простым исполнителем, а кто умным оркестратором процесса... с обработкой ошибок, таймерами бизнес-процесса и т.д.?**

В архитектуре роль "умного оркестратора" обычно достается тому сервису, который охватывает наиболее сложные и многоэтапные бизнес-процессы, требующие координации работы нескольких других сервисов. В нашем примере выше это сервис заказов.

Почему так происходит? Давайте разберемся:

- **Центральность бизнес-процесса:** Процесс создания и обработки заказа является центральным для большинства коммерческих предприятий. В рамках этого процесса задействованы многие другие сервисы (инвентаризация, платежи, доставка и т.д.). Таким образом, управление этим процессом обычно требует наиболее широкой координации.
- **Сложность бизнес-логики:** В процессе создания и обработки заказов используется сложная бизнес-логика. Такой процесс может включать различные этапы и условия (проверка наличия товара, обработка платежей, обработка возможных исключений и т.д.), которые нужно тщательно координировать.

- Обработка ошибок и исключений: В процессе обработки заказов могут возникать различные исключительные ситуации (например, нехватка товара, проблемы с платежами и т.д.). "Умный оркестратор" должен быть способен обрабатывать такие ситуации и координировать необходимые действия между другими сервисами.

Подобные причины делают сервис обработки заказов наиболее подходящим кандидатом на роль "умного оркестратора" в контексте нашего примера. Однако это не значит, что другие сервисы не могут играть подобную роль в других контекстах или в других аспектах бизнес-процессов. Ключевым моментом является выбор того сервиса, который наиболее эффективно и надежно сможет координировать сложные межсервисные взаимодействия. Просто помните - что размазывать весь бизнес-процесс на мелкие куски иногда неэффективно, лучше выделять более крупные микросервисы, которые будут управлять большими кусками процессов.

