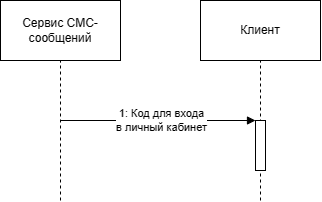
# 

## **Асинхронное взаимодействие**

Асинхронное взаимодействие предполагает необязательность какого-либо ответа от получателя или отправитель не ждёт ответа, на мой взгляд отличным примером демонстрации асинхронного взаимодействия могут являться уведомления.

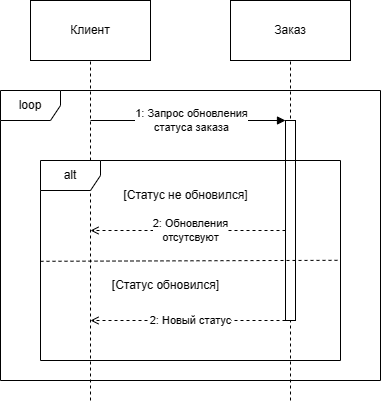
При отправке СМС-уведомлений мы не можем получить напрямую от сервиса информацию о том, доставлено сообщение или нет. Простым примером асинхронного взаимодействия при отправке СМС-сообщения может служить отправка кода для входа в личный кабинет.

Отправка СМС-сообщений с кодом доступа в личный кабинет:



Допустим, что мы ограничены в отправке запросов на клиент со стороны сервера и можем только отвечать на запросы клиента из-за протокола https, тогда асинхронное взаимодействие будет реализовано посредством long polling, когда клиент периодически запрашивает, например, обновление статуса доставки:

Обновление статуса доставки:

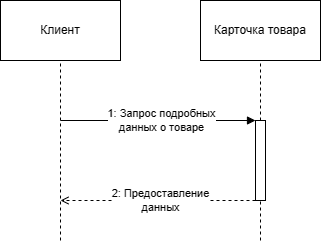


## **Синхронное взаимодействие**

При условии использования протокола https примерами синхронного взаимодействия могут послужить любые действия, требующие от сервера какие-либо запросы по предоставлению данных.

При открытии карточки товара из каталога товаров идёт запрос по получению данных как о самом товаре (состав, вес и т.п.) так и о возможных топпингах(добавках). Цена и название попадают в карточку товара через кэш из каталога товаров.

Открытие карточки товара:



В качестве второго примера может послужить запрос на создание заказа из списка товаров в корзине. В этом случае пользователь ждёт обработку запроса для продолжения работы на сайте, так как серверу нужно сменить страницу оформления заказа на страницу информации по готовности заказа.

Оформление заказа:



# **Механика масштабирования**

## **Загруженность сервисов**

Самыми загруженными сервисами могут считаться сервис заказов, товаров и корзины.

Сервис заказов будет являться самым загруженным из всех, так как этому сервису необходимо будет принимать данные о результатах оплат, о состояниях заказов, о составе заказов и тому подобное. Стадия обработки заказа является самой трудоёмкой и ресурсоемкой, так как ведётся в режиме реального времени (обновление статусов) и сопряжена со сложной бизнес-логикой (расчётное время доставки, учёт способа оплаты и прочее).

Сервис товаров будет вторым по загруженности, потому что данные по товарам – наиболее важная информация для клиентов до начала обработки заказа. Первым делом, когда пользователь попадает на сайт онлайн-пиццерии, от клиента на сервер поступает запрос о предоставлении актуальный данных по товарам, для составления каталога. Именно объём запрашиваемой информации будет играть основную роль в загруженности к данному сервису.

Предположительно, третьим по загруженности будет сервис корзины по той причине, что расчёт итоговой стоимости всех товаров в корзине связан со сложной бизнес-логикой (учёт скидок, акций…) и частотой запросов на перерасчёт стоимости, так как каждый добавленный товар необходимо добавлять к уже имеющимся товарам.

## **Масштабирование**

При масштабировании нужно учитывать не только способность масштабировать производительность вычислительной мощности оборудования, но и способность масштабирования БД.

### **Масштабирование вычислительных мощностей**

Из описания, данного задания не ясно каковы у данной сети пиццерий планы на счёт скорости масштабирования бизнеса. Если бизнес не планирует расти дальше, то вариант вертикального масштабирования вычислительных мощностей будет приемлемым, так как на старте это не потребует больших усилий, по сравнению с созданием инфраструктуры для горизонтального масштабирования, а когда через какой-то долгий промежуток времени придёт мысль об увеличении бизнеса, в принципе уже не будет играть особой роли какая стратегия масштабирования вычислительных мощностей была выбрана из-за появления новых технологий. Соответственно, если бизнес планирует расти дальше и не останавливаться, то стратегия горизонтального масштабирования покажет себя с лучшей стороны, так как на этапе роста, пока размеры бизнеса сравнительно (с будущими размерами компании) небольшие, проще будет “обкатать” технологию горизонтального масштабирования. Однако при малых размерах всё ещё не факт, что на старте стратегия горизонтального масштабирования будет выгоднее вертикального в среднесрочной перспективе.

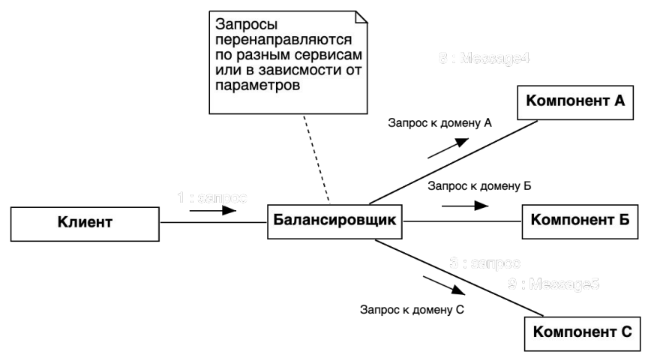
### **Масштабирование БД**

При выборе стратегии масштабирования базы данных в данном задании определяющим фактором будет являться выбранная ранее микросервисная архитектура. Микросервисная архитектура, подталкивает к выбору федеративного способа масштабирования, при котором каждый микросервис имеет свою базу данных, отвечающую за хранение информации по его предметной области. Так в базе данных микросервиса “Заказ” будут храниться только те записи, что имеют непосредственное отношение к заказам (статусы заказа, стоимость заказа, время приёма в обработку заказа и т.д.).

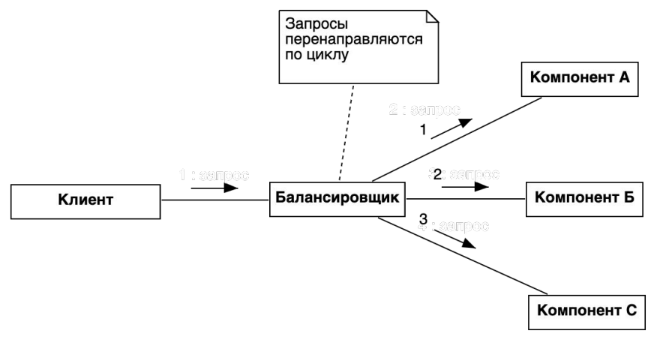
Естественно, как и в случае с масштабирование вычислительных мощностей, у каждого микросервиса можно по особому организовать свою личную БД. Лучшим образом для каждого микросервиса подойдёт шардирование, так как данные в этих базах и так разделены по общим признакам. В применении иных способов устройства баз данных микросервисов не будет особого смысла, так как разработанное федеративное решение для всех микросервисов можно просто незначительно изменить до шардирования (схожий принцип устройства позволяет) для применения в пределах одного микросервиса. Это повлечет удешевление разработки всей системы.

## **Механизм балансировки**

Так как в данном случае имеется микросервисная архитектура, соответственно, балансировка между сервисами будет осуществляться по значимым признакам. Сервис товара не сможет обработать запрос, предназначенный для сервиса оплаты.



Соответственно, если в сервисы запросы приходят уже “отсортированные” по признаку, значит, что сложность обработки запросов, поступающих в один сервис будет примерно одинакова. Получается, из-за балансировки запросов на этапе распределения по микросервисам внутри самих микросервисов можно применить циклическую балансировку. Балансировка по весам будет избыточной в данном случае.



# **Механизм кеширования данных**

В данном проекте для удобства и практичности стоит сделать три вида кэша (многоуровневое кэширование):

1. Пользовательский кэш :

* Будет располагаться на устройстве клиента в виде кэша мобильного приложения/cookie-файлов браузера.
* В него будут входить личные данные клиента (история заказов, адреса доставки, избранные товары и тд…)
* Весь кэш, кроме истории заказов, не будет иметь TTL, но при каждой новой сессии будет делаться запрос на сервер для проверки актуальности кэша. Информация по заказу будет храниться только месяц.

2. Кэш сервиса :

* Будет располагаться в ОЗУ сервиса.
* Хранит частые однотипные обработанные запросы пользователей.
* Будет обновляться принудительным сбросом кэша при появлении новых данных.

3. Кэш запросов к БД :

* Располагается в ОЗУ сервиса.
* Хранит частые однотипные запросы к базе данных.
* Будет обновляться принудительным сбросом кэша при появлении новых данных.

Схема многоуровневого кэширования проекта:

