**Rest API**

Архитектурный стиль, набор правил, который описывает, как наиболее эффективно использовать http и строить свою API так, чтобы ей было удобно пользоваться, чтобы она выдерживала нагрузки, легко масштабировалась и т.д

***Концепции***:   
1) **Модель взаимодействия** – это клиент – сервер

2) **Многоуровневость** - Может быть сколько угодно серверных слоёв. Причём с точки зрения клиента, это никакой роли не играет. А внутри там может быть сколько угодно серверных слоёв с каким-нибудь балансированием, распределением, проксированием, микросервисной архитектурой и т.д.

3) **Сервер не должен обладать состоянием**То есть клиент посылает запрос, сервер отдаёт ответ, и никакого промежуточного состояния нет. То есть при каждом новом общении сервер и клиент общаются как будто бы в первый раз

4) **Единообразный унифицированный интерфейс**

Допустим, есть какой-то интернет магазин. Нам всеми сущностями мы можем выполнять CRUD операции

POST  
DELETE

PUT/PATCH

GET

GET

Для каждой операции мы используем правильный семантический метод. (В конце с s наиболее семантически правильно)

+ форматы взаимодействия, заголовки, всё должно быть единообразно

Запрос должен содержать всю необходимую информацию для его выполнения

Так же важна семантика. С помощью get запроса мы должны получать данные, а изменять или обновлять мы не должны !

Идемпотентность – это свойство, которое означает, что повторной идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект, не изменяющий состояние сервера. Корректно реализованные методы GET, PUT и DELETE идемпотентны, но не метод POST

**5) Кеширование**

Может быть осуществлено методом http, проставлением определённых заголовков, так и средствами сторонними на сервере (Redis, mcache и т.д)

Get и post могут быть кэшируемыми, PUT и DELETE не кэшируются

**6) Версионирование**

/api/users

Новая версия /api/v2/users

Те пользователи, которые работали со старой api, так же и дальше продолжают использовать старую версию, а новая v2

**7) Документирование**

Open API и Swagger

**SOAP**

SOAP (Simple Object Access Protocol) и REST (Representational State Transfer) — это два разных подхода к созданию веб-сервисов, и выбор между ними зависит от конкретных требований вашего проекта. Вот несколько сценариев, когда использование SOAP может быть более подходящим, чем REST:

1. **Требования к безопасности**: SOAP поддерживает расширенные стандарты безопасности, такие как WS-Security, которые обеспечивают шифрование, подпись и аутентификацию сообщений. Это делает SOAP предпочтительным выбором для приложений, где безопасность критически важна, например, в банковских или финансовых системах.
2. **Транзакции**: SOAP поддерживает стандарты, такие как WS-AtomicTransaction, которые позволяют управлять распределенными транзакциями. Это может быть полезно в сложных бизнес-процессах, где требуется согласованность данных.
3. **Формальный контракт**: SOAP использует WSDL (Web Services Description Language) для описания интерфейсов веб-сервисов. Это обеспечивает строгую типизацию и формальный контракт между клиентом и сервером, что может быть полезно в корпоративных средах, где необходимо четко определить интерфейсы.
4. **Поддержка сложных операций**: SOAP хорошо подходит для сложных операций, которые требуют более сложной логики обработки сообщений, таких как маршрутизация и надежная доставка сообщений (например, с использованием WS-ReliableMessaging).
5. **Интеграция с существующими системами**: Если ваша система уже использует SOAP или интегрируется с другими системами, которые используют SOAP, может быть проще продолжать использовать этот протокол.
6. **Стандартизация и совместимость**: SOAP является более стандартизированным протоколом и может быть предпочтительным выбором в средах, где требуется высокая степень совместимости и соблюдение стандартов

Протокол обмена структурированными сообщениями. (XML)

Никто не запрещает в рамках одного сервиса использовать SOAP и REST

В SOAP контроллерах описан WSDL и он возвращает XML

Если Rest использует только http, то Soap может использовать с любым протоколом прикладного уровня (SMTP, FTP, HTTP и т.д.)

REST – множество окон (endpoint’ов). SOAP – одно окно, в которое необходимо передать название операции, которую мы хотим выполнить.  
SOAP – это протокол, который задаёт уже определённые рамки  
  
Сообщения, переданные в SOAP, обладают определённой структурой, и состоят они из 4х частей.

Envelope – определяет начало и конец сообщения  
Header – это что-то вроде заголовков в обычный http запросах

Body – полезная нагрузка

Fault – инф-я об ошибках

**GraphQL**

Язык запросов. Проблематика – есть подробная карточка товара, но так же есть место, где нужно отобразить мало данных

Клиент сам может определять, какие данные ему нужны.

В GraphQL есть два типа основных запросов

Query (get) и mutation (post) subscription (realtime)

Описывается схема GraphQL похоже на типы type Post {  
id  
}

1. Query   
   Часто запрашиваемые поля можно вынести в фрагменты
2. Mutation

**Websocket**

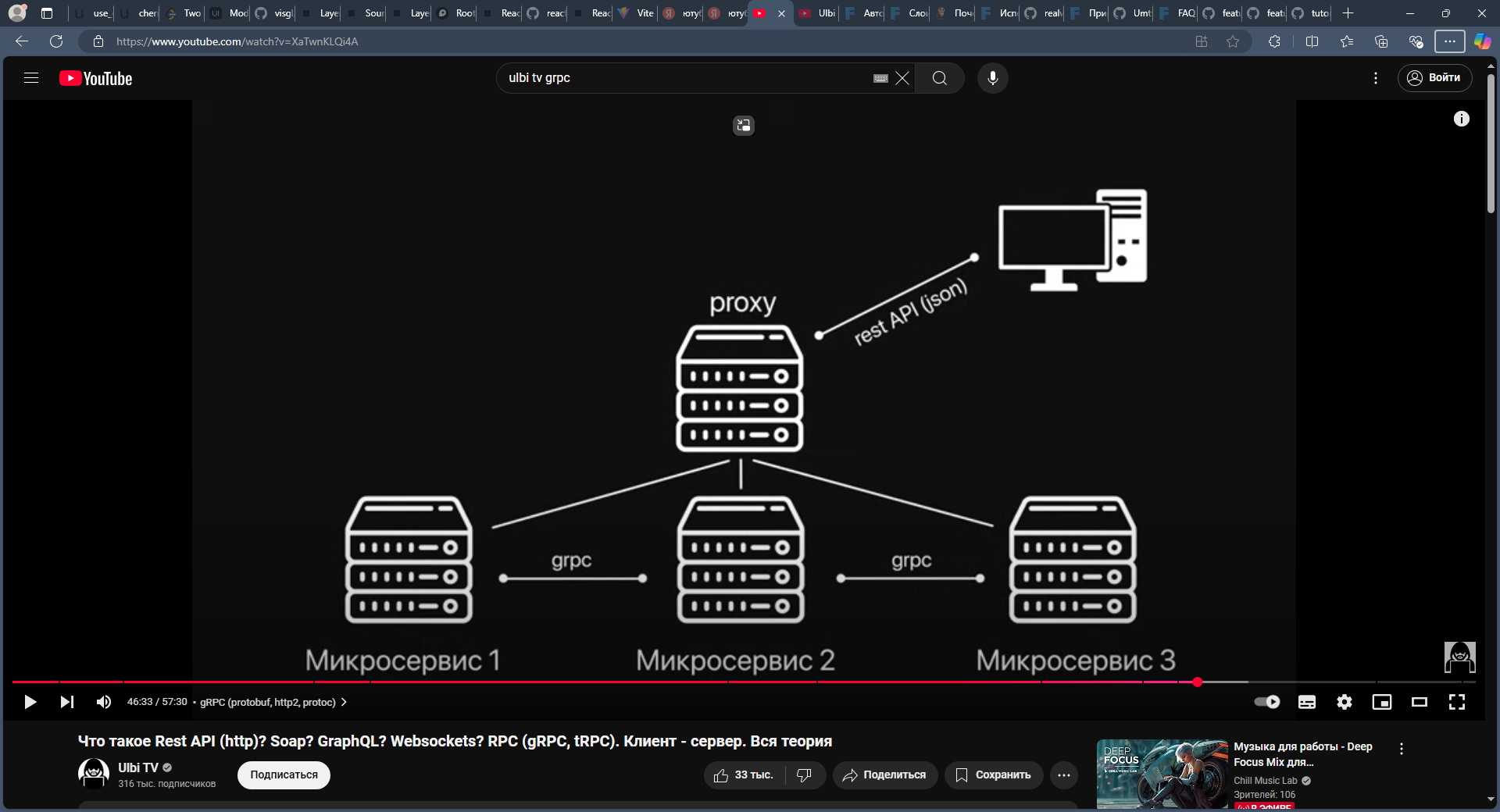
Все пользователи устанавливают непрерывное соединение с сервером.

Вместо http указывается ws

**RPC**

Это удалённый вызов процедур.   
  
У нас есть клиент, и есть сервер. Сервер реализует какой-то метод, при этом на клиенте этот метод не реализован. С помощью RPC клиент способен вызвать этот метод, как будто этот метод реализован у него.

**gRPC**

Фреймворк от google. Он очень популярен в микросервисной архитектуре.   


Преимущества:

1. **HTTP 2.** Http 2 по тестам быстрее на 10-15% http1. В http 2 используется бинарный формат, за счёт чего его можно лучше сжать, лучше обработать. Реализованы потоки данных.Вместо json испльзуется бинарный формат, он называется protobuf
2. **Работает в разы быстрее**
3. **Простой и быстрый стриминг данных**
4. **Вместо json используется бинарный формат (protobuf), который имеет строгую типизацию.**
5. **Инструментарий из под коробки:**

**-** генерация кода для многих ЯП (protoc)

**-** Аутентификация

**-** потоковая передача данных

**…**……………………………..

1. **Удобный вызов процедур**

**TRPC  
(Typesafe RPC)**