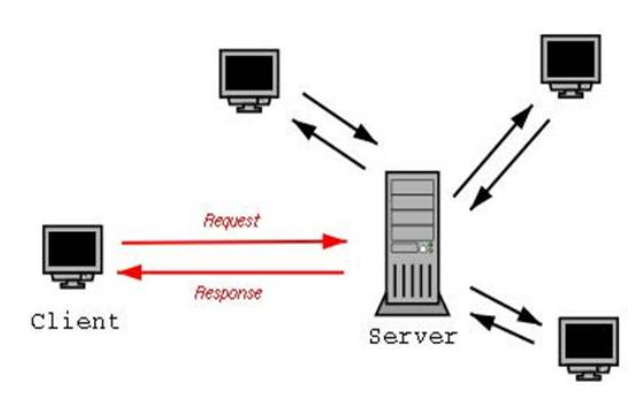
Лекция. Архитектура ПО

# Клиент-серверная архитектура

«Клиент — сервер» (англ. client–server) — архитектура, в которой сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами.

Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов. Типичный пример – это браузер.



Также стоит заметить, что**в основе взаимодействия клиент-сервер лежит принцип того, что такое взаимодействие начинает клиент**, сервер лишь отвечает клиенту и сообщает о том, может ли он предоставить услугу клиенту и если может, то на каких условиях. Т.е. программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине).

Программное обеспечение клиента и сервера обычно установлено на разных машинах, но также они могут работать и на одном компьютере. Данная концепция взаимодействия была разработана в первую очередь для того, чтобы разделить нагрузку между участниками процесса обмена информацией, а также для того, чтобы разделить программный код поставщика и заказчика.

Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

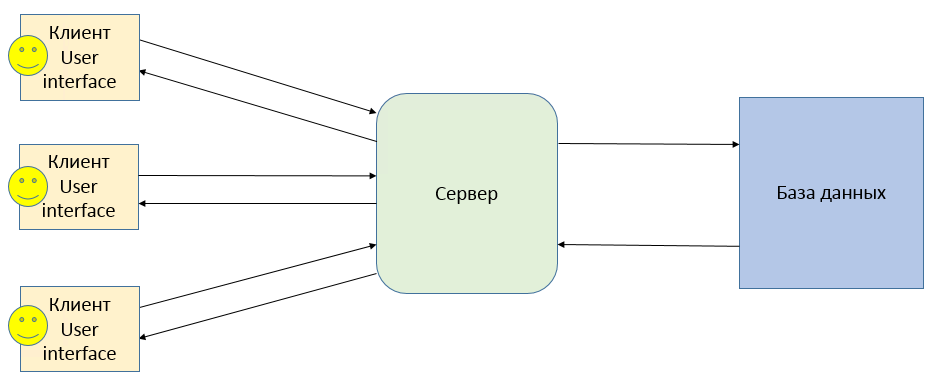
### Многоуровневая архитектура «клиент — сервер»

— разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Частные случаи многоуровневой архитектуры:

Трехуровневая архитектура - архитектурная модель программного комплекса, предполагающая наличие трех компонентов:

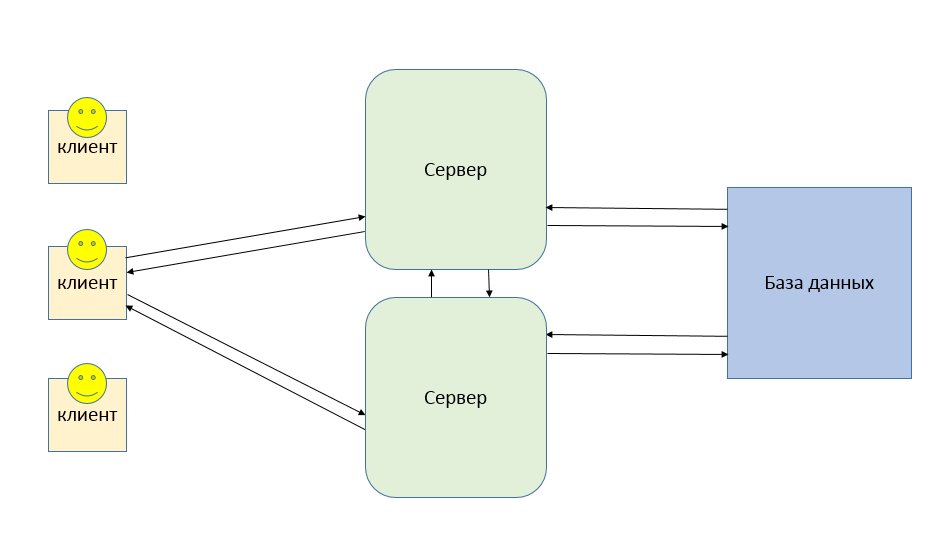
* клиент,
* сервер приложений (для которых подключено клиентское приложение)
* сервер баз данных (с этими работает сервер приложений).



Пример: Вы оформляете кредит онлайн или же пришли в банк, где сотрудник банка оформляет для Вас кредит. Кредит оформляется через веб-приложение (это клиент), после заполнения формы заполнения заявки формируется запрос, который отправляется на сервер, сервер в свою очередь запрашивает или сохраняет данные в базу данных.

Если нагрузка на сервер планируется большая, то схема может быть такой – рисунок ниже.

При оформлении заявки от клиента запрос поступает на балансировщик, который в свою очередь определяет какой из серверов менее нагружен и передается запрос ему.



**Преимущества клиент-сервисной архитектуры:**

* Отсутствие дублирования кода программы-сервера программами-клиентами.
* Так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются.
* Все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

**Недостатки клиент-сервисной архитектуры:**

* Неработоспособность сервера может сделать неработоспособной всю вычислительную сеть. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п.
* Поддержка работы данной системы требует отдельного специалиста — системного администратора.
* Высокая стоимость оборудования.

## UML диаграммы

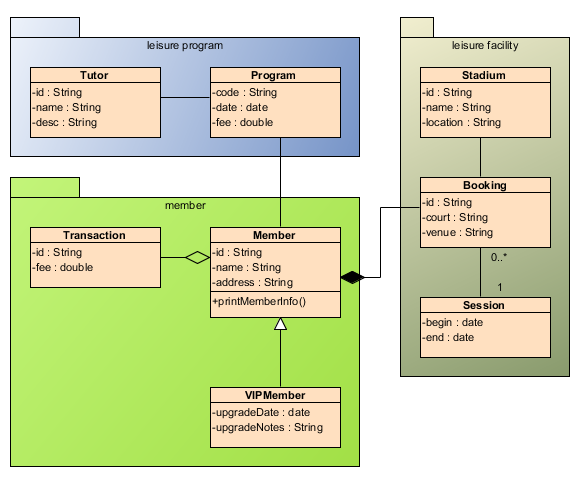
В ходе проектирования ПО архитектором или опытным программистом создается проектная документация, включающая текстовые описания, диаграммы, модели будущей программы. В этом помогает язык UML.  
  
UML — является графическим языком для визуализации, описания параметров, конструирования и документирования различных систем (программ в частности).

Диаграммы бывают разные:

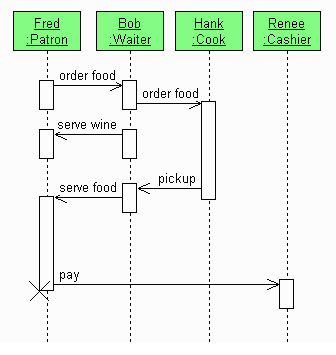
* Диаграмма вариантов использования (use case diagram)
* Диаграмма классов (class diagram)
* Диаграмма состояний (statechart diagram)
* Диаграмма последовательности (sequence diagram)
* Диаграмма кооперации (collaboration diagram)
* Диаграмма компонентов (component diagram)
* Диаграмма развертывания (deployment diagram)

**Примеры диаграмм:**

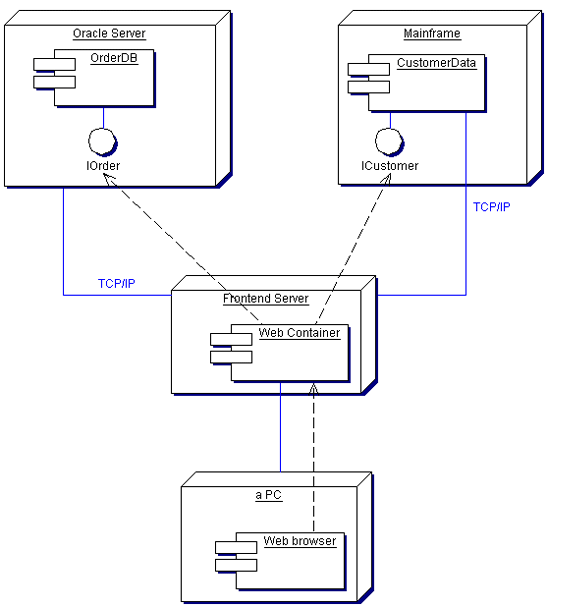
**Диаграмма классов**

****

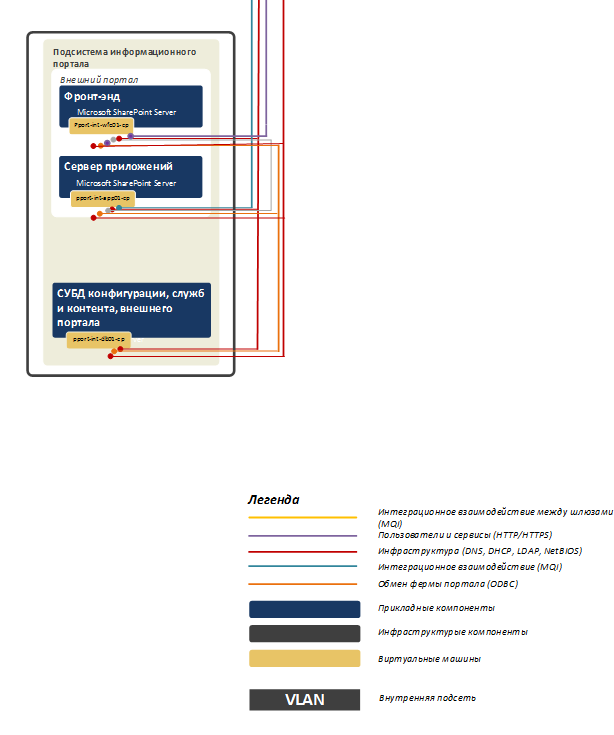
**Диаграмма последовательности**

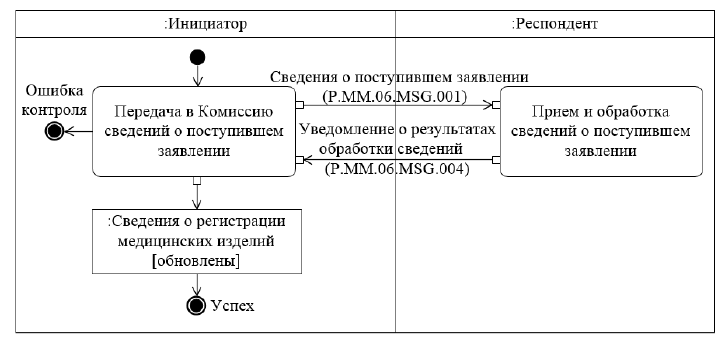
****

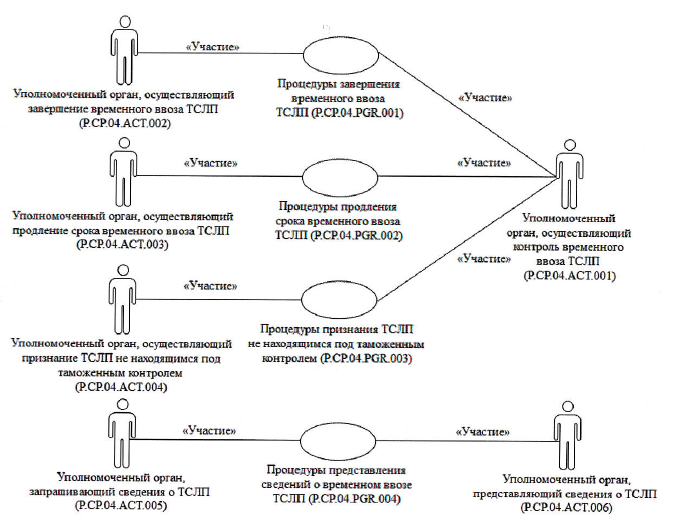
**Диаграммы развертывания**

****

Схемы могут выглядеть и так:



Так же диаграммы uml часто используются и для описания бизнес процессов ПО. Например:



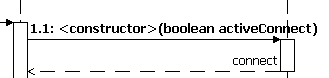
**Статья с примерами диаграмм** <https://habr.com/ru/post/74330/>

<https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema13/tema13_3>

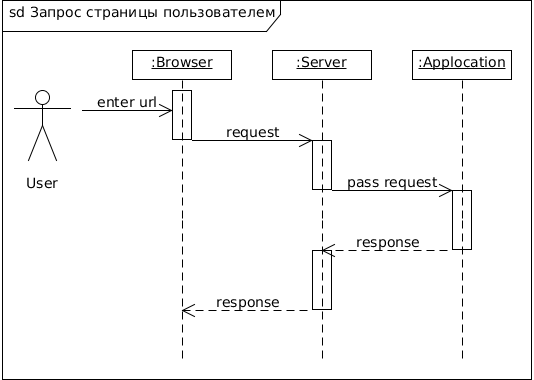
### Виды взаимодействия между клиентом и сервером

Виды взаимодействия между клиентом и сервером:

**Синхронное** - Клиент посылает сообщение серверу и ждет, пока тот примет и обработает сообщение. Как правило, один объект передает синхронное сообщение второму, второй – третьему и т.д., образуя вложенный поток сообщений. В любом случае клиент, инициирующий поток сообщений, должен дождаться его завершения, т.е. возврата управления. Это самый распространенный тип сообщений.



**Асинхронное** - Клиент посылает сообщение серверу и, не дожидаясь ответа, продолжает выполнять следующие операции



- – ***синхронное*** сообщение (англ. synchronous message).

-  – ***асинхронное*** сообщение (англ. asynchronous message).

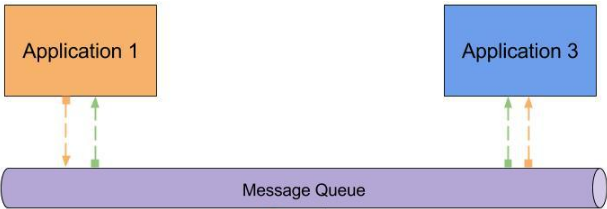
- – ***возвращающее*** сообщение (англ. reply message), обозначающее возврат значения или управления от сервера обратно клиенту. Стрелки этого вида зачастую отсутствуют на диаграммах, поскольку неявно предполагается их существование после окончания процесса выполнения операции.

Пример: Обращение в паспортный стол оформить прописку – синхронное взаимодействие. Обращение в МФЦ для обращения прописки – асинхронное взаимодействие.

# Сервис-ориентированная архитектура - SOA

Сервис-ориентированная архитектура (Service-oriented Architecture (SOA)) представляет собой подход в проектировании, при котором предоставляется возможность обмена данными между двумя и более приложениями, разработанными на разных технологиях. Чаще всего для обмена данными между такими приложениями используются веб-сервисы с помощью протокола SOAP (<https://ru.wikipedia.org/wiki/SOAP>), а для обмена сообщениями используют платформо-независимый язык (вроде XML или JSON). В рамках SOA веб-сервисы предоставляют доступ к базе данных через http.

Так же может использоваться менеджер очередей. Есть несколько приложений, которые асинхронно общаются друг с другом с помощью платформо-независимых сообщений. Очередь сообщений улучшает масштабируемость и усиливает изолированность приложений. Им не нужно знать, где находятся другие приложения, сколько их и даже что они собой представляют. Однако все эти приложения должны использовать один язык обмена сообщениями, т. е. заранее определённый текстовый формат представления данных.



Очередь сообщений может упростить взаимодействие приложений, но она не способна решить проблему разных форматов языков. Впрочем, была сделана попытка превратить очередь сообщений из простого канала связи в посредника, доставляющего сообщения и преобразующего их в нужные форматы/языки. ESB стал следующей ступенью в естественной эволюции простой очереди сообщений.

## SOAP

 — протокол обмена структурированными сообщениями в распределённой вычислительной среде. SOAP протокол используется для обмена произвольными сообщениями в формате XML. (Спецификация <https://www.w3.org/TR/soap/>)

SOAP может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTP, HTTPS и др. Чаще всего SOAP используется поверх HTTP.

Сообщение SOAP выглядит так:

Envelope – Корневой элемент, который определяет сообщение и пространство имен, использованное в документе.

Header – Содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации.

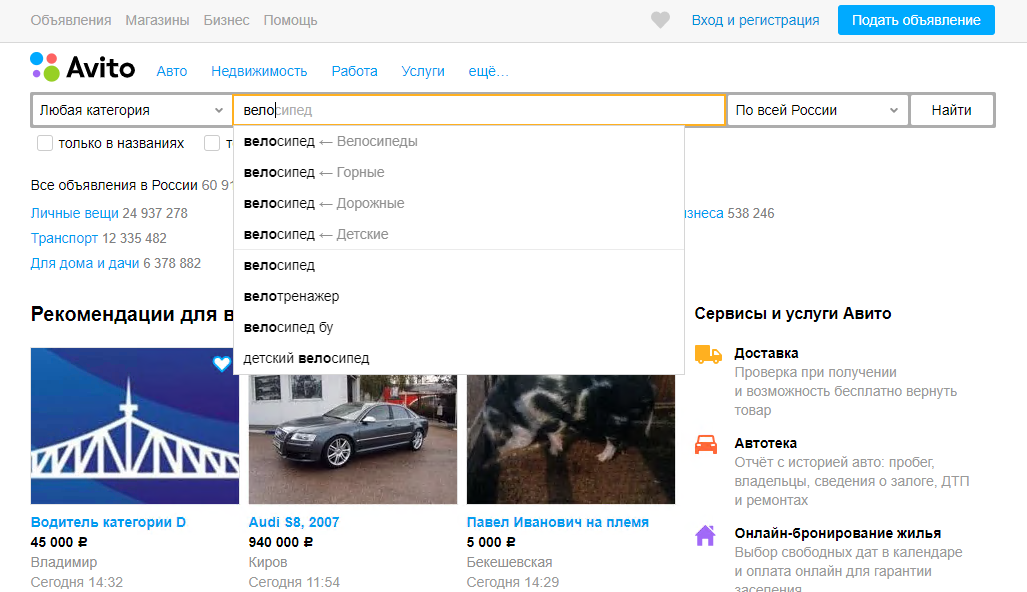
Body – Содержит сообщение, которым обмениваются приложения.

Fault – Необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений

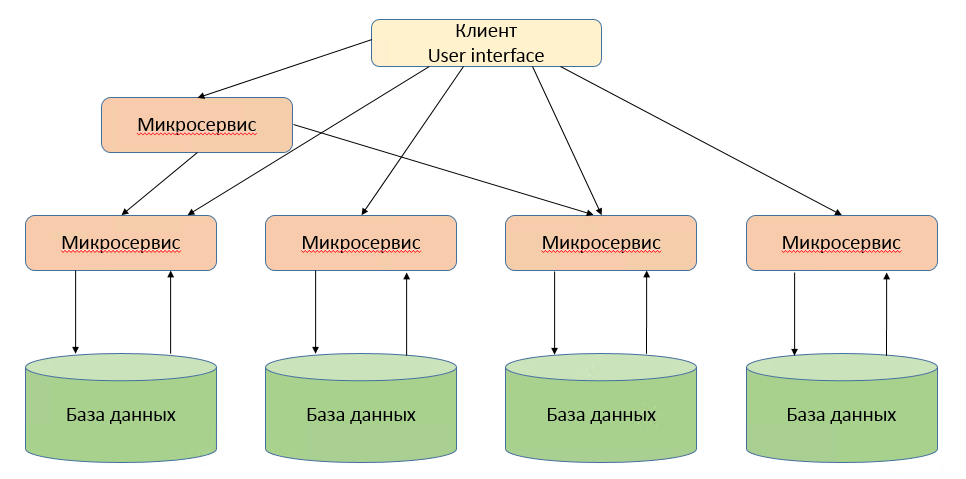
# Микросервисная архитектура

Микросервисы — это небольшие, автономные, совместно работающие сервисы.

Например, сайт авито avito.ru



Здесь могут быть реализованы много различных микросервисов, которые работают все вместе и имеют один интерфейс пользователя (user interface - UI). Это может быть микросервис авторизации, микросервис для поиска объявлений, микросервис для модерации объявлений, микросервис для приема платежей, микросервис для нахождения дубликатов.



Каждая часть такой программы имеет свою базу данных, кодовую базу. Каждый такой микросервис работает автономно.

При этом обмен данными между самими сервисами ведется через сетевые протоколы. Чаще всего используют http протоколы с rest api (<https://ru.wikipedia.org/wiki/REST>) надстройкой.

**REST** (от англ. ***Re****presentational****S****tate****T****ransfer* — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой  системы.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют *«REST-запрос»*), а необходимые данные передаются в качестве параметров запроса. В отличие от веб-сервисов (веб-служб) на основе SOAP, не существует «официального» стандарта для REST веб-API. Дело в том, что REST является **архитектурным стилем**, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют такие стандарты, как HTTP, URL, JSON и XML.

**Преимущества микросервисной архитектуры:**

* Декомпозиция монолита (большую программу разделяем на несколько маленьких)
* Проще тестировать отдельные микросервисы, покрытие выше.
* Независимость релизов
* Если упадет один микросервис, то ПО в целом останется работоспособным, не будет работать какая-то отдельно взятая функциональность
* Независимое масштабирование (пример: для одного микросервиса необходим 10 мощных серверов, для другого – одна виртуальная машина)
* Возможность пробовать новые технологии (разные микросервисы могут работать на разных тенологиях)

**Минусы микросервисной архитектуры:**

* Расходы на настройку новой функциональности (настройкая окружения, настройка среды разворачивания микросервиса, настройка сборщика релизов, иногда на настройку уходит столько же времени, сколько и на разработку).
* Тестировать проще один сервис, но для сквозного тестирования необходимо описывать тесты проверки всех функциональностей (это может быть сделать неудобно и трудоемко, т.к. приходится подготавливать окружение для каждого сервиса).
* При использовании в разработке разных технологий компании необходимо иметь ресурсы с разными компетенциями

Монолитная vs Микросервисная архитектура

<https://proglib.io/p/monolitnaya-vs-mikroservisnaya-arhitektura-2019-09-16>