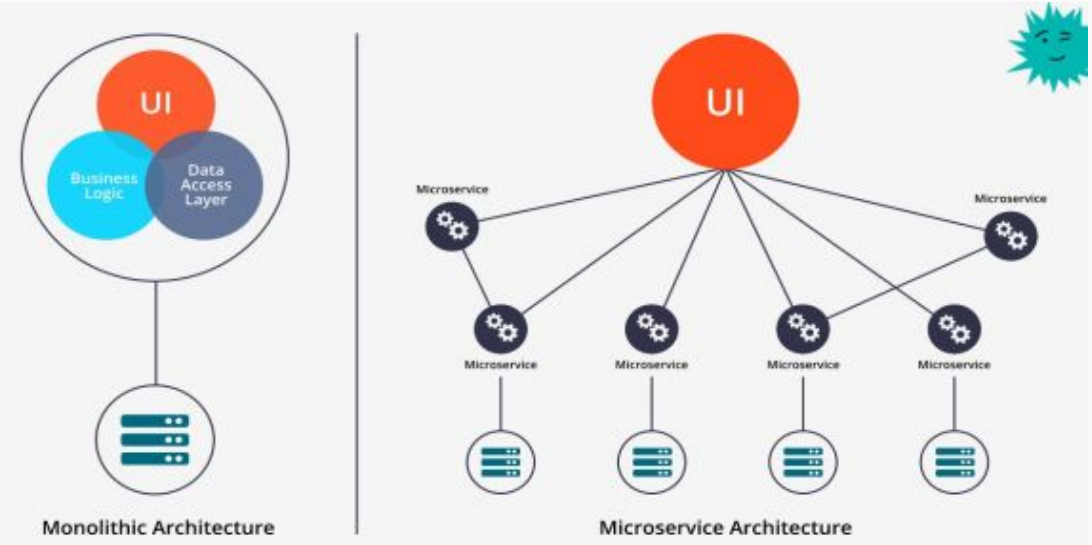
Архитектура ПО

Архитектура ПО

Архитектура ПО — совокупность важнейших решений об организации программной системы.

Хорошая архитектура — это архитектура делающая процесс разработки и сопровождения программы более простым и эффективным. Программное обеспечение с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать.



Высокоуровневое и детальное проектирование

Проектирование на высоком уровне или **HLD** относится к общей системе, проектированию, которое включает описание архитектуры и дизайна системы и является общим системным проектированием, включающим:

- Системная архитектура
- Дизайн базы данных
- Краткое описание систем, сервисов, платформ и взаимосвязей между модулями.

В HLD (который основан на бизнес-требованиях и ожидаемых результатах) включена схема, представляющая каждый аспект проектирования.

- Он содержит описание аппаратных, программных интерфейсов, а также пользовательских интерфейсов.
- Это также известно как макроуровень / системный дизайн
- Он создан solution architect.
- Рабочий процесс типичной пользовательской операции подробно описан в HLD вместе со спецификациями производительности.

Высокоуровневое и детальное проектирование

LLD, или **проектирование на низком уровне,** — это этап в процессе разработки программного обеспечения, на котором определяются подробные компоненты системы и их взаимодействие.

- Это подробное описание каждого модуля, включающее фактическую логику для каждого компонента системы и подробное описание каждого модуля.
- Он также известен как микроуровень / детальный дизайн.
- Он создан дизайнерами и разработчиками.
- Это включает в себя преобразование проекта высокого уровня в более подробный план с описанием конкретных алгоритмов, структур данных и интерфейсов.
- LLD служит руководством для разработчиков во время написания кода, обеспечивая точную и эффективную реализацию функций системы.

Высокоуровневое и детальное проектирование

Высокоуровневое проектирование включает определение общей структуры системы, основных модулей и их взаимодействий. На этом этапе создаются архитектурные диаграммы, такие как диаграммы компонентов и диаграммы развёртывания.

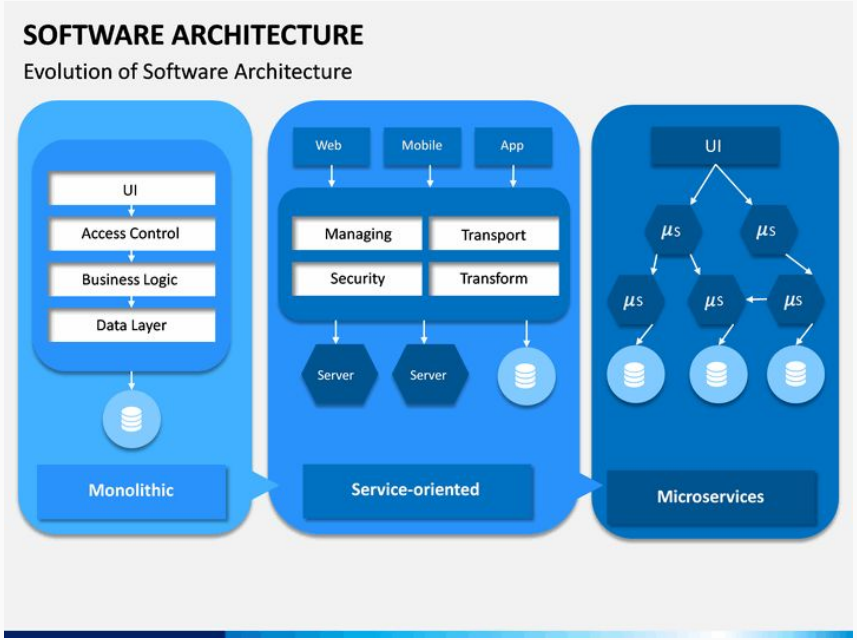
Детальное проектирование фокусируется на разработке конкретных компонентов и их взаимодействий, включая описание алгоритмов и структур данных. Для каждого компонента определяются набор структур данных, перечень свойств системы и реализующих эти свойства алгоритмов, а также связи с другими компонентами.

Высокоуровневое проектирование закладывает «скелет» продукта, а детальное проектирование «обрастает» этот «скелет» деталями.

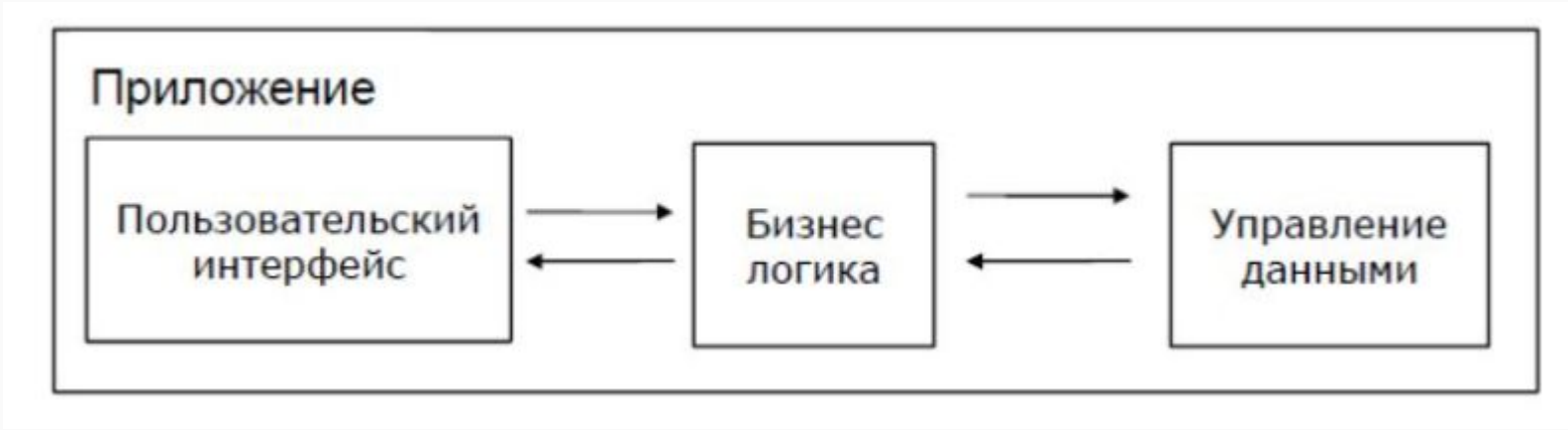
Проектные документы высокого уровня помогают руководителям проектов и архитекторам понять, как будет работать система, а проектные документы низкого уровня более подробны и предназначены для программистов.

Зачем нужна архитектура?

Архитектура программного обеспечения — это как скелет здания. Она определяет, какие компоненты будут в системе, как они взаимодействуют и каковы их основные функции. В сложных системах, где есть много модулей, функций и зависимостей, архитектура помогает разработчикам держать все под контролем. Она позволяет избежать хаоса, делает систему более понятной и управляемой.



Базовые функции информационных систем



- Слой представления - все, что связано с взаимодействием с пользователем: нажатие кнопок, движение мыши, отрисовка изображения, вывод результатов поиска и т.д.
- Бизнес логика - правила, алгоритмы реакции приложения на действия пользователя или на внутренние события, правила обработки данных.
- Слой доступа к данным - хранение, выборка, модификация и удаление данных, связанных с решаемой приложением прикладной задачей

Одноуровневая архитектура

Одноуровневая клиент-серверная архитектура, также известная как одноуровневая архитектура, представляет собой простейшую форму клиент-серверной модели.

В этой архитектуре клиент и сервер взаимодействуют напрямую, без промежуточных уровней.

Клиент отправляет запросы на сервер, а сервер обрабатывает эти запросы и возвращает результаты клиенту.

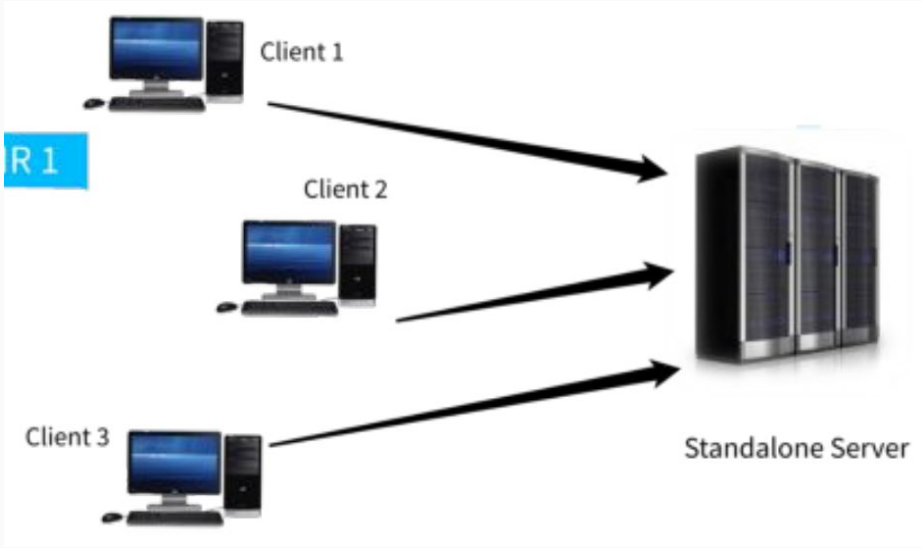
Одноуровневая архитектура

- Отсутствие GUI (графического интерфейса)
- Простые приложения или программы
- Приложения не масштабируются

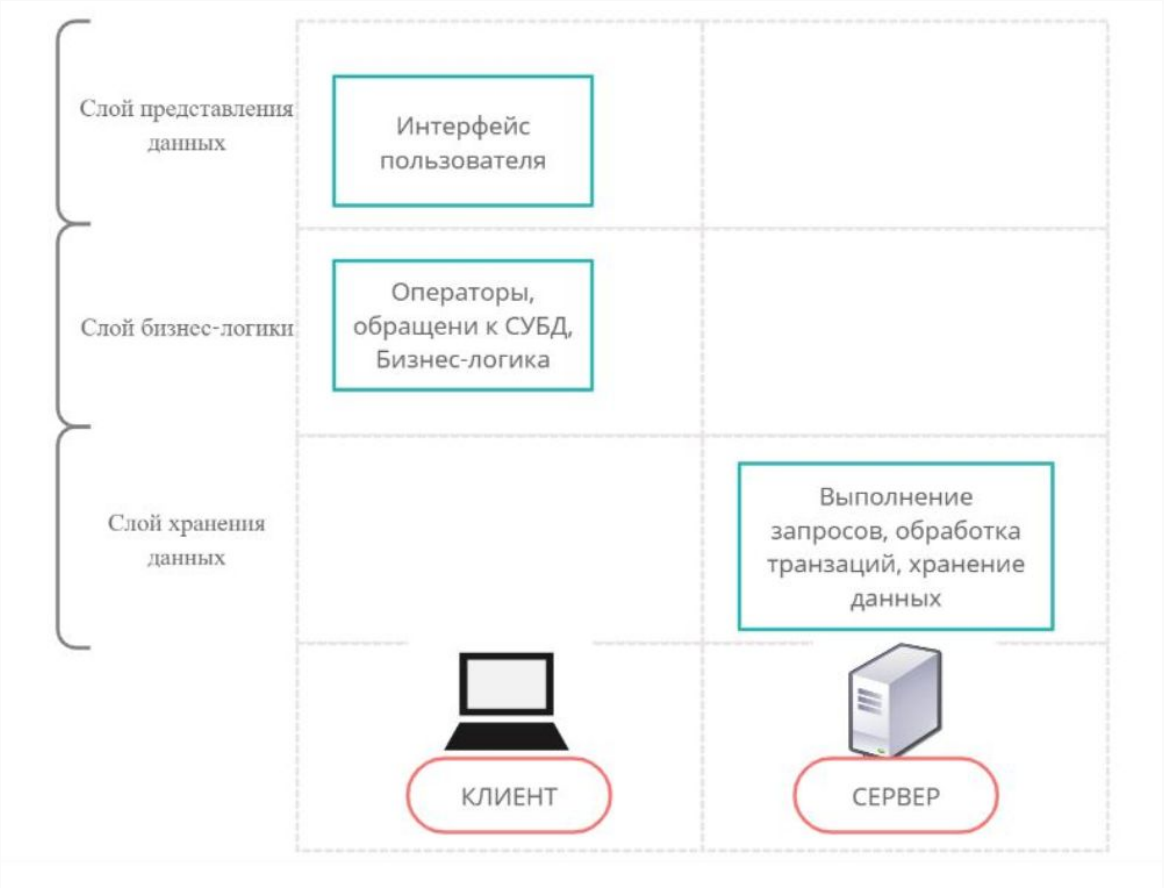
Двухуровневая архитектура

- **Клиент** содержит слои презентации, бизнес-логики и передачи данных.
- **Сервер** включает хранилища и базы данных.ё

➔ Разделение ответственности
➔ Более простое обновление

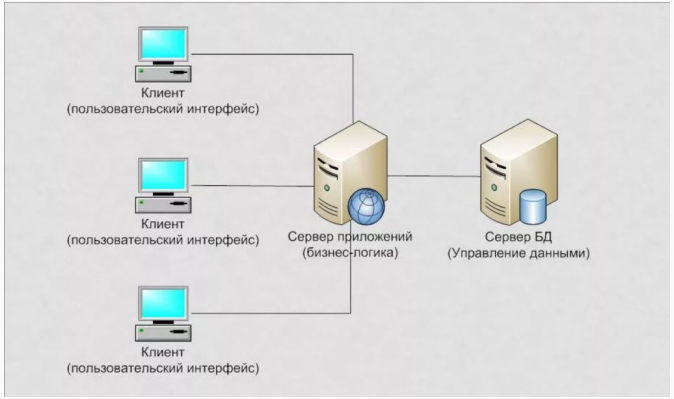


Двухуровневая архитектура

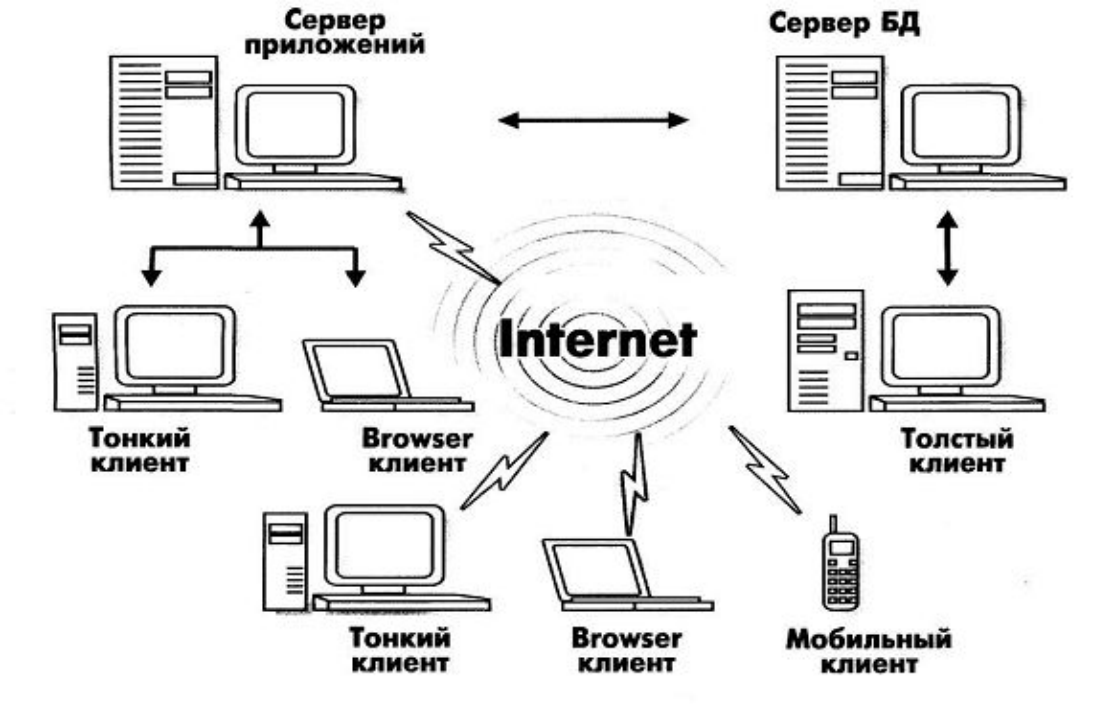


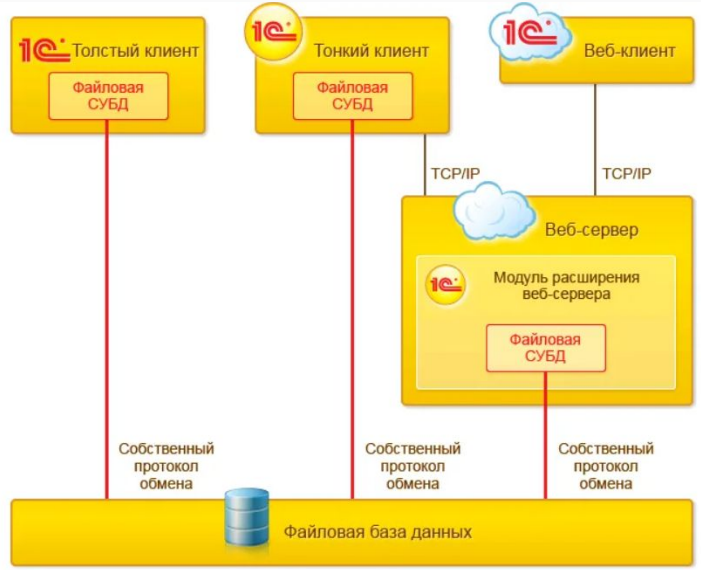
Трехуровневая и n-уровневая архитектура

- ➔ Тонкий клиент
- ➔ Высокая степень гибкости и масштабируемости

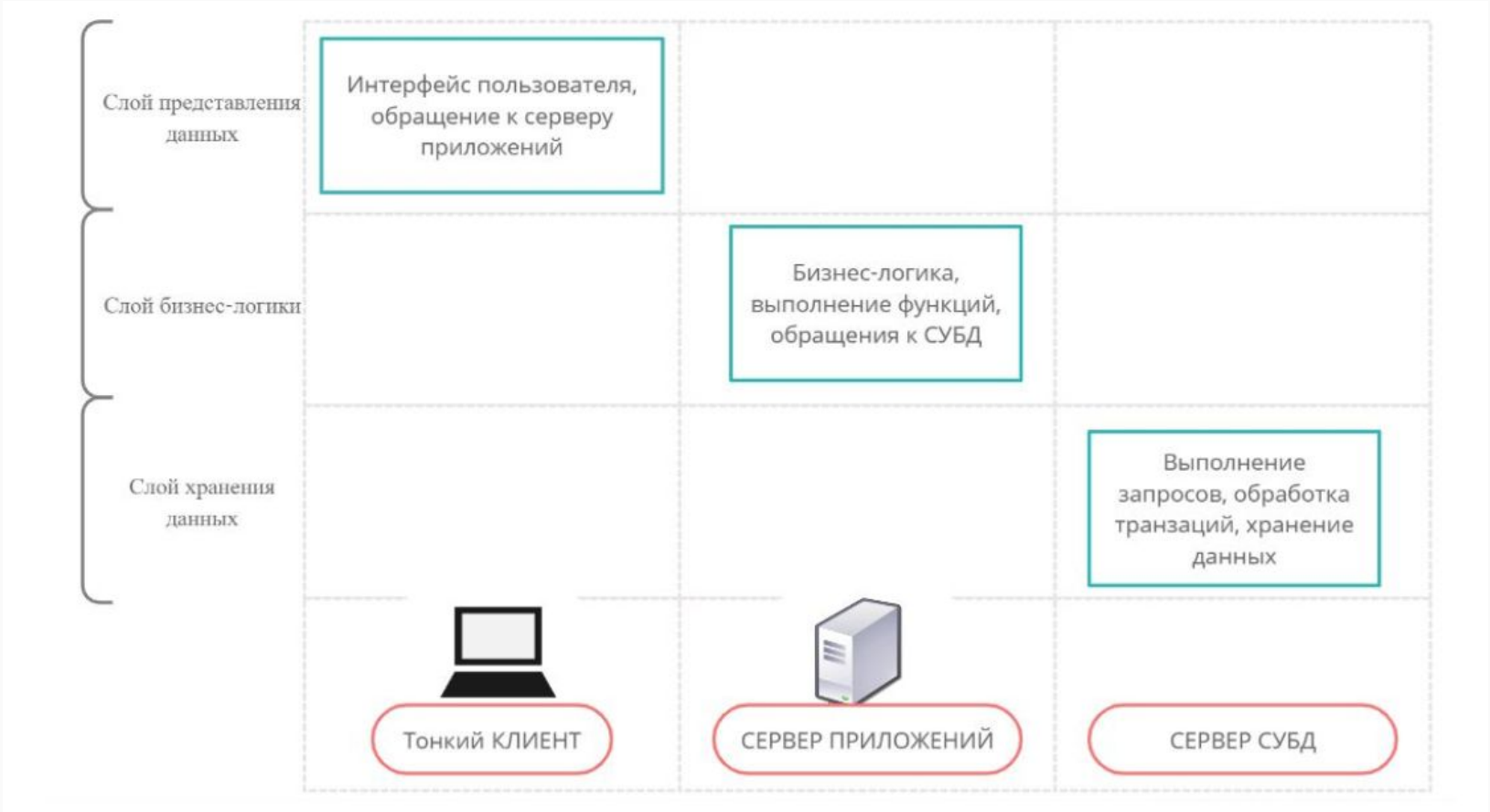


Трехуровневая и n-уровневая архитектура



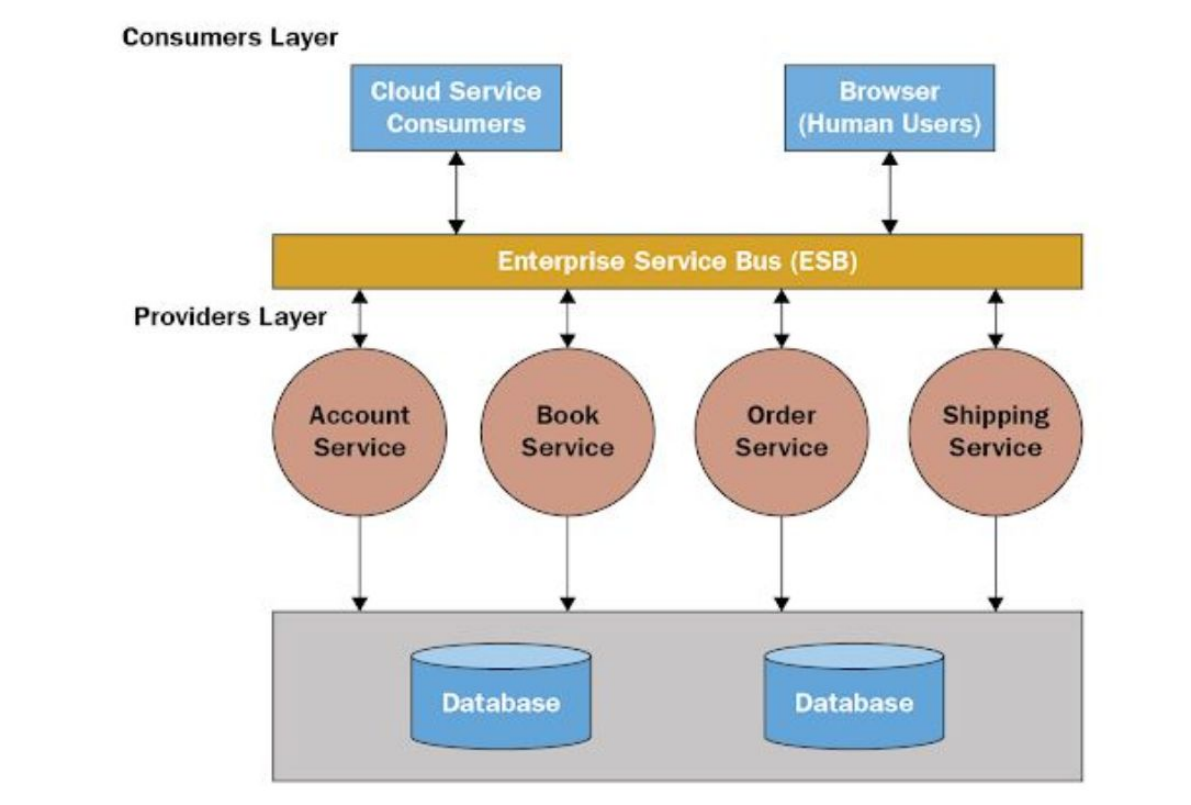


Трехуровневая и n-уровневая архитектура



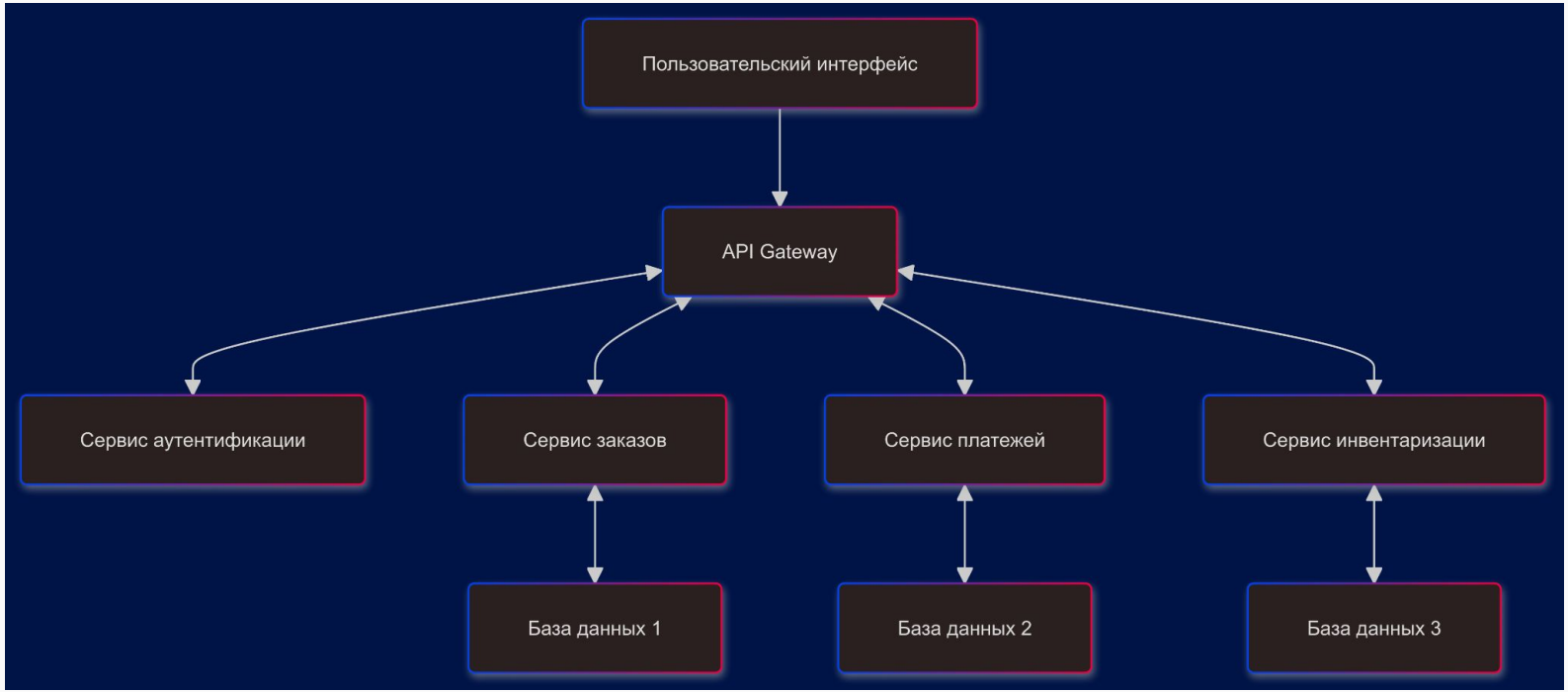
SOA (Сервис-ориентированная архитектура)

- SOA - это парадигма
- Во главе стола - сервисы
- Что используют в SOA:
  - ESB (шина)
  - Web - services
  - Microservices



Микросервисы

➔ Один сервис - одна конкретная бизнес задача
➔ Акцент не только на техническую часть, но и на организационную структуру



Микросервисы

Преимущества
- ➔ Гибкость
- ➔ Небольшие специализированные команды
- ➔ Небольшая база кода
- ➔ Сочетание технологий
- ➔ Изоляция ошибок
- ➔ Масштабируемость
- ➔ Изоляция данных

Недостатки
- ➔ Сложность решения
- ➔ Перегрузки и задержки сети
- ➔ Более частые сбои
- ➔ Целостность данных
- ➔ Управление
- ➔ Высокая стоимость владения

Бессерверная архитектура

Приложения размещаются сторонней службой, что устраняет необходимость в управлении серверным программным обеспечением и оборудованием со стороны разработчика. Приложения разбиты на отдельные функции, которые можно вызывать и масштабировать по отдельности.

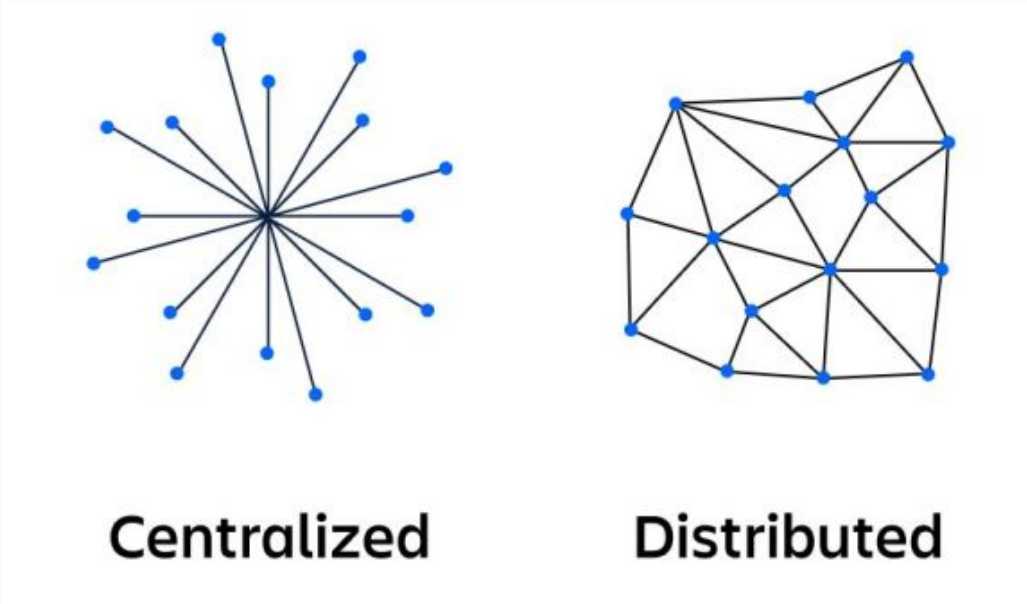
Бессерверная архитектура

Преимущества
➔ Простота развертывания
➔ Снижение затрат
➔ Улучшенная масштабируемость
➔ Возможность сосредоточиться на пользовательском опыте

Недостатки
➔ Привязка к поставщику
➔ Перегрузки и задержки сети
➔ Не подходит для больших задач

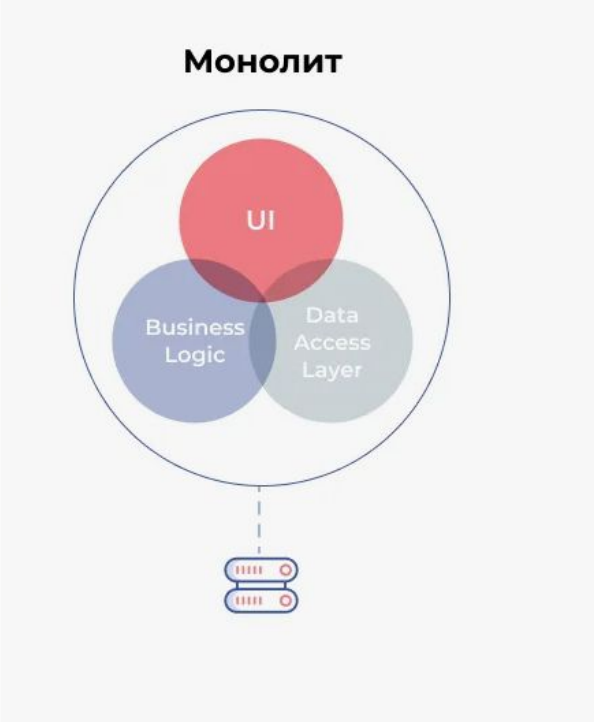
Распределенная архитектура

Распределенная система — это набор компьютерных программ, использующих вычислительные ресурсы нескольких отдельных вычислительных узлов для достижения одной общей цели. Распределенные системы направлены на устранение узких мест или единых точек отказа в системе



Монолит

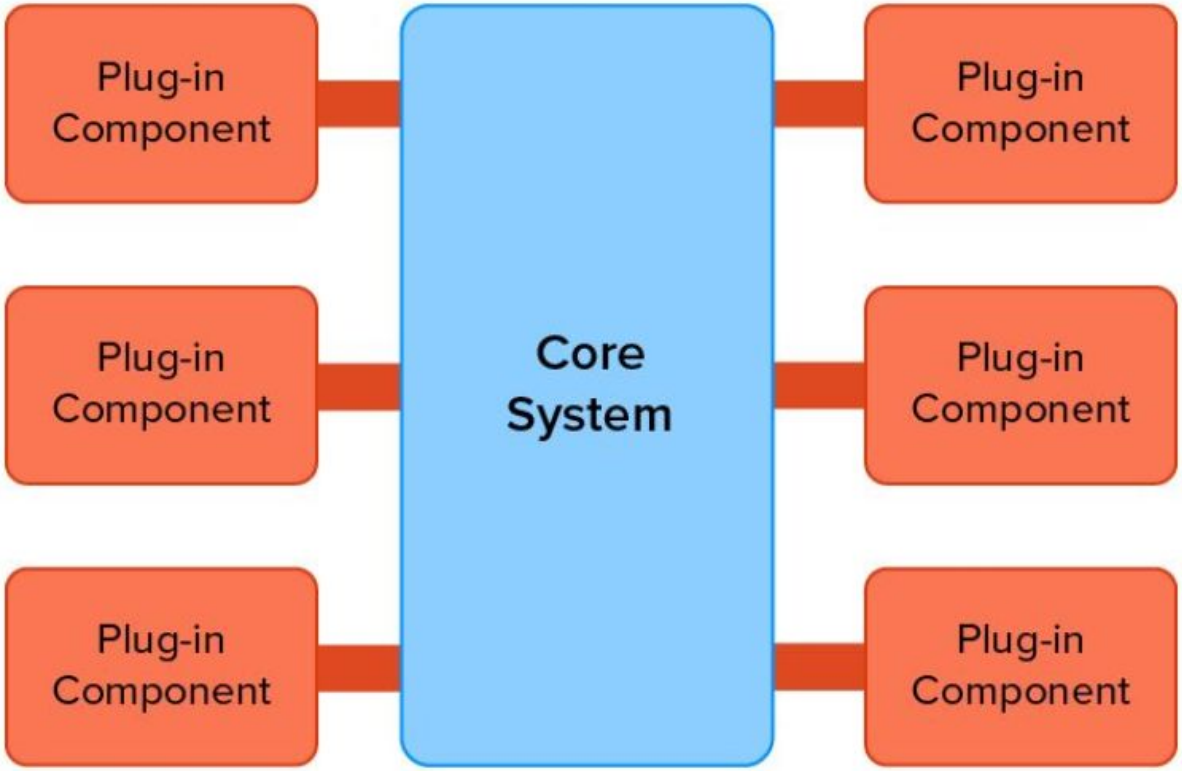
Монолитное приложение - это приложение, построенное как единое целое, где вся логика по обработке запросов помещается внутрь одного процесса



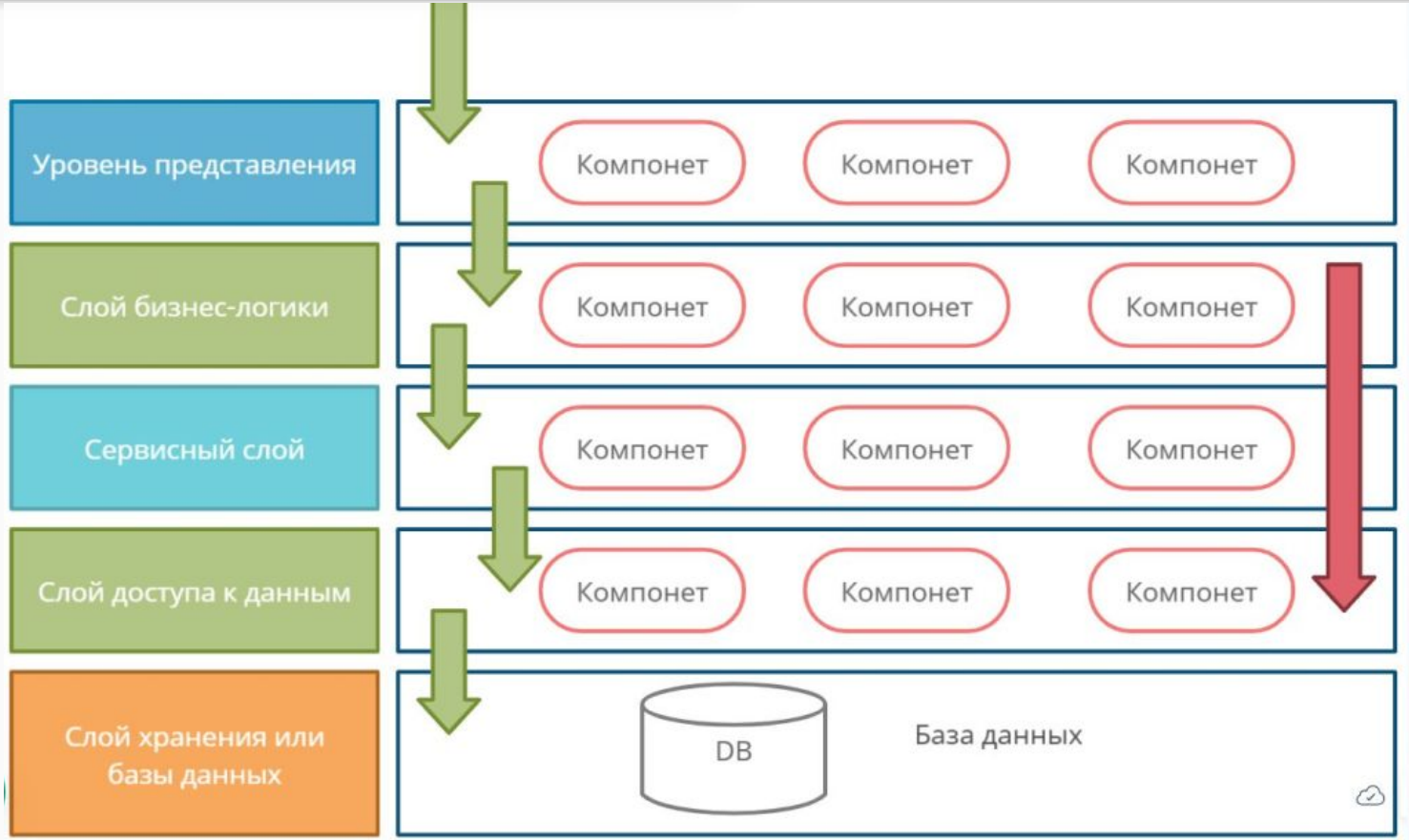
Стили монолита

- Микроядро (Microkernel)
- Многоуровневая архитектура (n-tier)
- Модульная архитектура

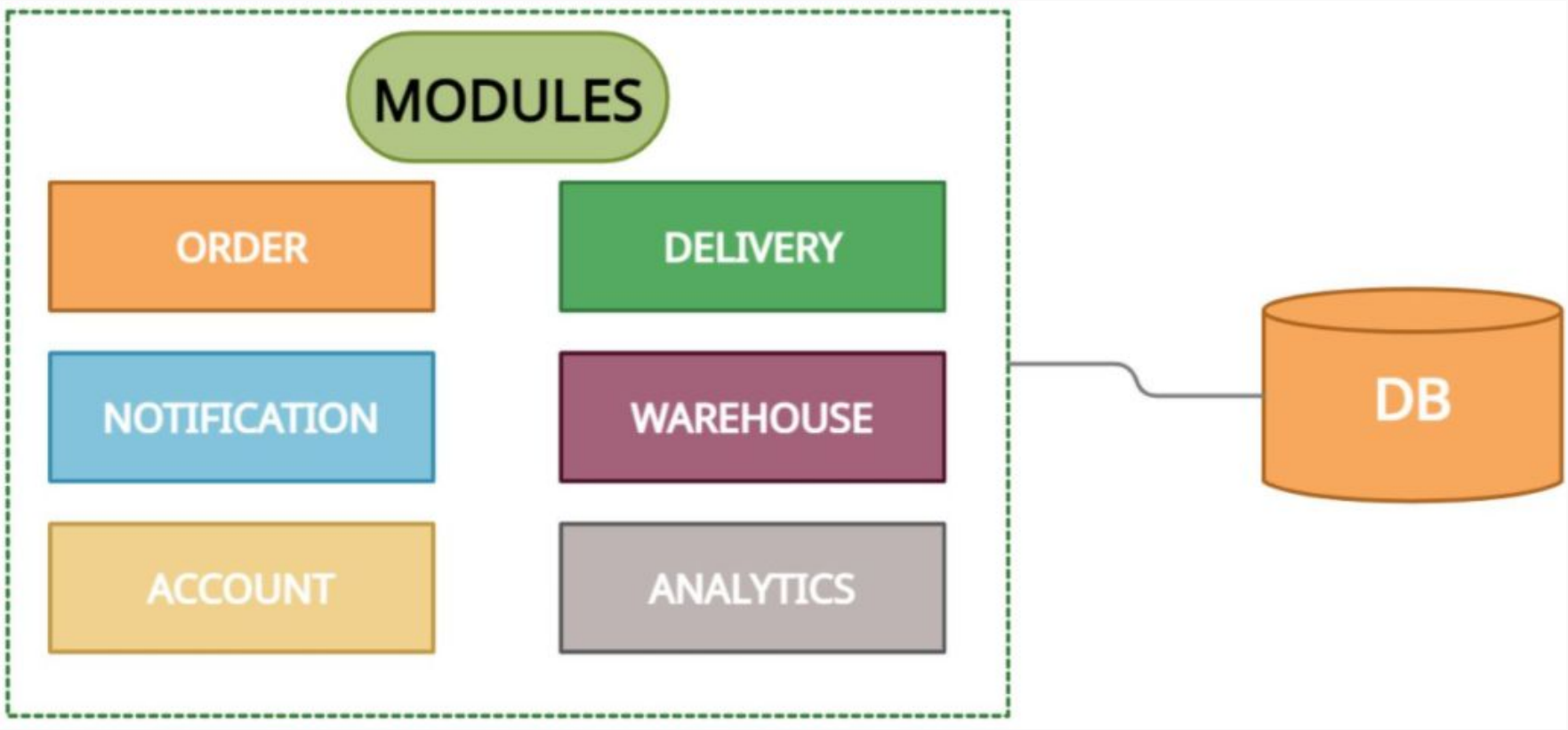
Микроядро


Plug-in Component
Plug-in Component
Plug-in Component
Core System
Plug-in Component
Plug-in Component
Plug-in Component

Многоуровневая архитектура



Модульная архитектура



Монолит

Преимущества:
➔ Его легко разрабатывать
➔ Его легко дорабатывать
➔ Его легко разворачивать
➔ Легко отслеживать ошибки
➔ Просто масштабировать

Недостатки:
➔ Кодовая база со временем становится громоздкой
➔ Ограниченная гибкость
➔ Сложно внедрять новые технологии
➔ Сложно масштабировать

В каких случаях это хорошо
➔ Для запуска MVP
➔ Для небольшой команды
➔ Нет архитектора для реализации сложной архитектуры
➔ Маленькое приложение, где нет богатой бизнес-логики

Архитектурный шаблон

Архитектурный шаблон - это обобщенное часто используемое решение распространенной задачи в архитектуре ПО в заданном контексте.

Шаблон - это решение задачи в определенном контексте.
➔ Многоуровневая архитектура
➔ Клиент - сервер
➔ Управляемая событиями архитектура
➔ Сервис-ориентированная архитектура
➔ Архитектура на основе микросеврвисов

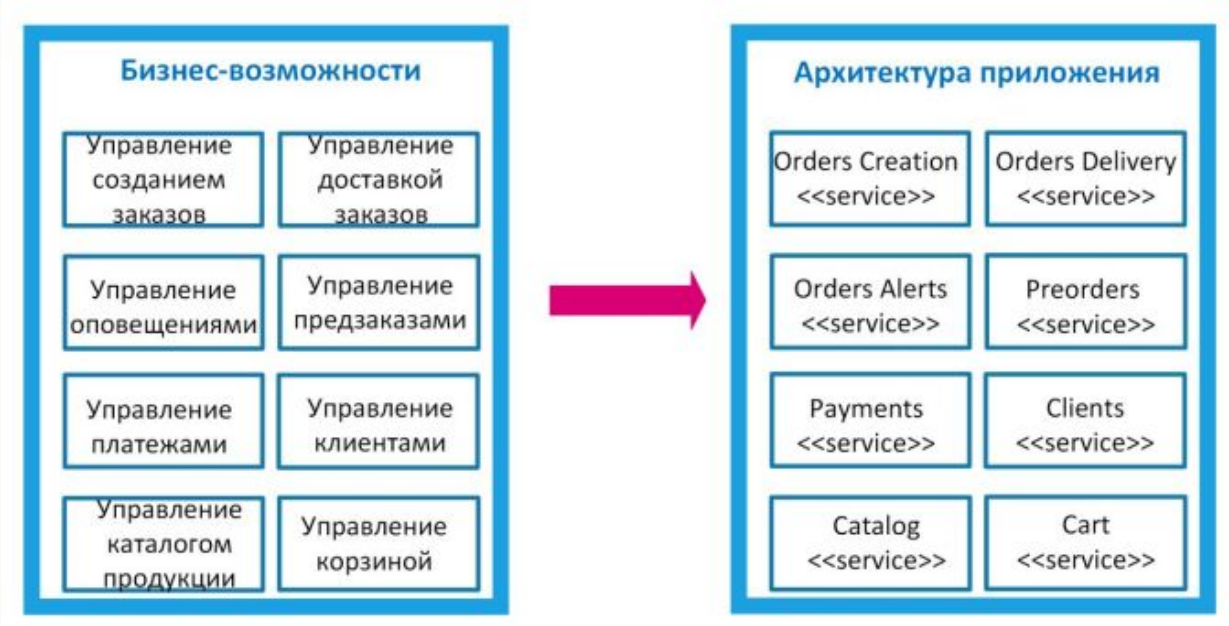
Как перейти от монолита к микросервисам

➔ Архитектура должна быть стабильной
➔ Сервис должен реализовывать небольшой набор сильно связанных функций.
➔ Сервисы должны быть слабо связаны
➔ Сервис должен быть тестируемым
➔ Каждый сервис должен быть достаточно маленький, чтобы ее разрабатывала команда «две пиццы», т.е. команда из 6-10 человек.
➔ Каждая команда, владеющая одной или несколькими службами, должна быть автономной.

Команда должна иметь возможность разрабатывать и развертывать свои сервисы с минимальным сотрудничеством с другими командами

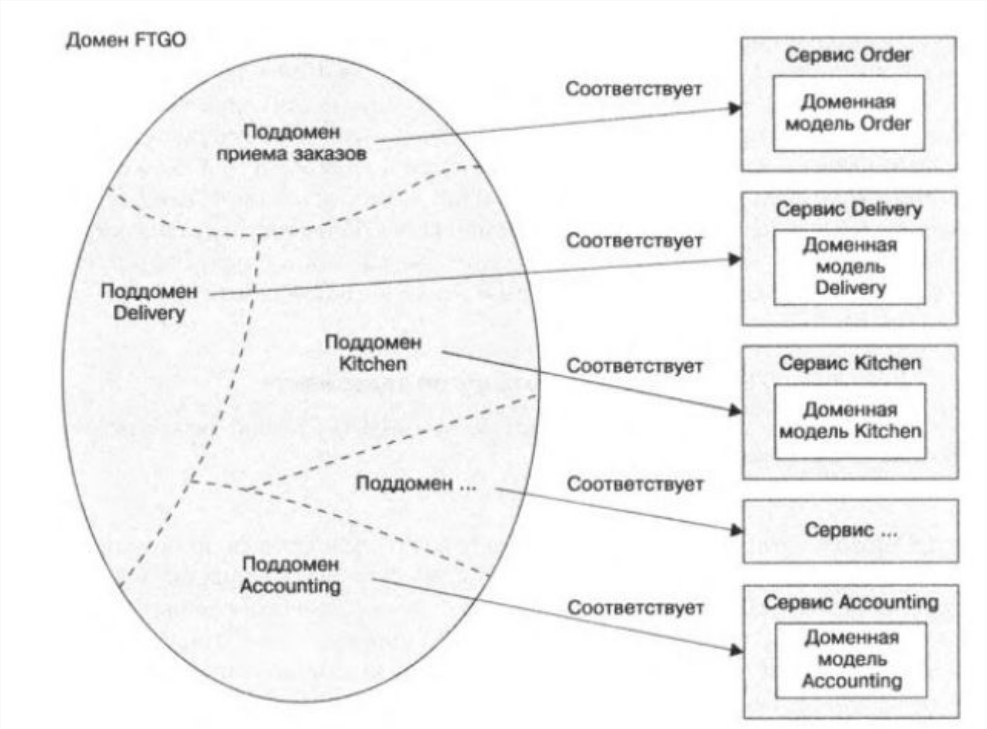
Разбиение по бизнес-возможностям

Один из наиболее известных способов разбиения на микросервисы — это определение бизнес-возможностей приложения и создание по одному микросервису на каждую из них. Бизнес-возможности представляют собой функции, которые будут доступны пользователям при работе с приложением.



Разбиение по субдоменам

Примерами поддоменов являются принятие заказов, управление заказами, управление кухней, доставка и финансовая отчетность.



Как архитектура влияет на проект

Правильная архитектура делает сложную систему более устойчивой, легко модифицируемой и масштабируемой. Например, если ваша система будет поддерживать миллионы пользователей, архитектура должна предусматривать балансировку нагрузки и работу с большими объемами данных. Без этого система может «упасть» при увеличении количества пользователей.

Архитектура также влияет на то, как быстро можно внести изменения в систему. Хорошо продуманная архитектура позволяет добавлять новые функции или изменять старые, не нарушая работу всей системы. Это особенно важно в условиях, когда требования постоянно меняются, и необходимо быстро адаптироваться к новым задачам.

Основные принципы объектно-ориентированного проектирования (ООП)

Инкапсуляция — это принцип, согласно которому данные и методы, которые работают с этими данными, объединяются в одном объекте, скрывая внутреннюю реализацию от внешнего мира.

Наследование — это механизм, позволяющий одному классу (дочернему) унаследовать свойства и методы другого класса (родительского).

Полиморфизм позволяет объектам разных классов обрабатывать вызовы одних и тех же методов по-разному.

Абстракция — это процесс выделения важных характеристик объекта и игнорирования несущественных деталей.

Принципы SOLID

Принцип единственной ответственности (Single Responsibility Principle, SRP)
Принцип единственной ответственности гласит, что каждый класс должен иметь только одну причину для изменения, то есть он должен отвечать за одну конкретную задачу.

Принцип открытости/закрытости (Open/Closed Principle, OCP)
Классы должны быть открыты для расширения, но закрыты для изменения. Это значит, что мы должны иметь возможность добавлять новый функционал через расширение классов, а не через изменение существующего кода.

Принцип подстановки Барбары Лисков (Liskov Substitution Principle, LSP)
Объекты подклассов должны быть заменяемы объектами родительских классов без изменения правильности программы. Это означает, что подклассы должны полностью соответствовать поведению базового класса.

Принцип разделения интерфейса (Interface Segregation Principle, ISP)
Клиенты не должны зависеть от интерфейсов, которые они не используют. Другими словами, лучше создавать более узкие и специфические интерфейсы, чем один большой и общий.

Принцип инверсии зависимостей (Dependency Inversion Principle, DIP)
Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Принципы SOLID

S	Принцип единственной ответственности	Для каждого объекта должно быть определено единственное назначение и это должно быть инкапсулировано классом
O	Принцип открытости/ закрытости	Программное обеспечение должно быть открыто для расширения, но закрыто для модификации
L	Принцип подстановки Лисков	Любой подкласс всегда должен использоваться вместо своего родительского класса
I	Принцип разделения интерфейса	Много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения
D	Принцип инверсии зависимостей	Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций

Принципы DRY, KISS и YAGNI

DRY: Don’t Repeat Yourself (Не повторяйся)
Принцип DRY говорит о том, что каждый фрагмент информации или логики в коде должен иметь единственное, однозначное представление. Другими словами, не стоит дублировать код — если вы видите повторяющийся код, его нужно вынести в отдельный метод, функцию или класс.

KISS: Keep It Simple, Stupid (Делай проще, глупец)
Принцип KISS напоминает разработчикам, что код должен быть как можно проще. Сложность увеличивает вероятность ошибок и затрудняет понимание и поддержку кода. Лучший код — это тот, который легко читать и понимать даже не самому опытному разработчику.

YAGNI: You Aren’t Gonna Need It (Вам это не понадобится)
Принцип YAGNI призывает не добавлять функциональность в код до тех пор, пока она действительно не понадобится. Часто разработчики пытаются предугадать будущие потребности и добавляют в код избыточные функции или возможности, которые могут никогда не понадобиться.

Принципы работы с паттернами

Паттерны проектирования играют важную роль в создании масштабируемых и поддерживаемых систем. Для того чтобы эффективно работать с паттернами, необходимо понимать ключевые принципы их использования, такие как повторное использование кода, поддержка расширяемости и гибкости системы.

Повторное использование кода — это один из основных принципов, поддерживаемых паттернами проектирования. Паттерны позволяют разработчикам создавать общие решения, которые можно использовать в разных частях проекта или даже в других проектах.

Поддержка расширяемости и гибкости системы
Расширяемость и гибкость — это важные характеристики качественной программной системы. Паттерны проектирования помогают создавать системы, которые легко адаптируются к изменениям и могут быть расширены без существенных изменений в исходном коде.