**ВНУТРЕННЯЯ СТРУКТУРА ПРИЛОЖЕНИЯ**

1. **КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА**

Клиент-серверная архитектура - это модель организации приложений, в которой задачи распределены между клиентами и серверами. В такой архитектуре клиент, обычно являющийся пользователем или программой, запрашивает услуги или ресурсы у сервера, который отвечает на запросы, предоставляя необходимые данные или функциональность.

В клиент-серверной архитектуре используется три компонента:

* **Клиент** — программа, которую мы используем в интернете. Чаще всего это браузер, но может быть и другая отдельная программа
* **Сервер** — компьютер, на котором хранится сайт или приложение. Когда мы заходим на сайт магазина, мы обращаемся к серверу, на котором находится сайт
* **База данных** — программа, в которой хранятся все данные приложения. Для магазина это будет база клиентов, товаров и заказов

Рассмотрим достоинства и недостатки данной архитектуры.

**Достоинства**:

1. Отсутствие дублирования. Весь сайт или приложение хранится на одном компьютере-сервере. Это позволяет использовать его с разных устройств, будь то компьютер или мобильный телефон
2. Надежность: при использовании клиент-серверной архитектуры, отказ одного компонента системы не влияет на работу других компонентов
3. Безопасность: централизованное управление сервером обеспечивает возможность контроля доступа и защиты данных, что делает клиент-серверную архитектуру более безопасной по сравнению с другими IT-архитектурами
4. Производительность. Серверы обычно производительнее, чем компьютеры пользователей. Это позволяет обрабатывать тысячи запросов от сотни разных пользователей одновременно.
5. Масштабируемость. Клиент-серверная сеть обладает высокой степенью масштабируемости. При необходимости можно добавить больше клиентов и серверов в сеть, что позволит увеличить мощность сервера и обработать больше запросов без остановки работы сети.
6. Поддержка. Клиент-серверная архитектура позволяет взаимодействовать с различными платформами и технологиями, используя открытые стандарты для обмена данными и коммуникаций.

**Недостатки**:

1. Перегрузка сервера. Популярные порталы могут получать большое количество запросов одновременно. Например, при десятке миллионов запросов в секунду сервер может не выдержать и отключиться
2. Зависимость от сервера: клиент не может работать без сервера. Если сервер(а) недоступен или имеет проблемы, все клиенты будут неработоспособны или испытывать проблемы с функциональностью
3. Ограниченность: при использовании клиент-серверной архитектуры возникают ограничения на количество одновременно подключенных клиентов и на пропускную способность сети
4. Высокая стоимость оборудования

<https://itelon.ru/blog/arkhitektura-klient-server/>

<https://habr.com/ru/articles/495698/#plus>

<https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit>

<https://testengineer.ru/client-server-chto-eto>

1. **БЕССЕРВЕРНАЯ АРХИТЕКТУРА**

Бессерверная архитектура – это способ создания и запуска приложений и сервисов без необходимости управления инфраструктурой. Приложение по‑прежнему будет работать на серверах, но управление этими серверами облачный провайдер полностью берет на себя. Команде больше не придется заниматься выделением ресурсов, масштабированием и обслуживанием серверов для запуска приложений, баз данных и систем хранения данных.

**Достоинства**:

1. Не нужно управлять серверами и средами исполнения, что сберегает время и силы на разработку.
2. Экономия на ресурсах. Бессерверные функции запускаются только тогда, когда они необходимы, что потенциально позволяет сэкономить значительные ресурсы.
3. Масштабируемость. Модульная организация бессерверных приложений упрощает разработчикам масштабирование приложений. В облаке над новыми функциями можно работать независимо, а затем постепенно добавлять их в работающее приложение.
4. Надежность и доступность. Приложение, разбитое на слабо связанные функции, создает гораздо меньше зависимостей между ними. Поэтому, если одна функция дает сбой, приложение продолжит работу.

**Недостатки**:

1. Ограничения технологического стека. Бессерверная архитектура менее гибкая, чем микросервисы. Эти платформы часто поддерживают только отдельные языки программирования, фреймворки и инструменты
2. Привязка к провайдеру. Полностью бессерверные приложения тесно связаны с платформой поставщика услуг, для которой они были созданы. Но каждая облачная платформа поддерживает разные технологии, а это означает, что перенастройка бессерверных функций для платформы другого провайдера может оказаться трудной задачей.
3. Низкая гибкость: у многих людей возникают проблемы с подключением к одному поставщику бессерверной среды в течение длительного периода.
4. Безопасность. Поставщик облачных служб может одновременно запускать код нескольких своих клиентов на одном сервере. Данные вашего приложения могут быть раскрыты, если общий сервер настроен неправильно.
5. Производительность. Когда функции вызываются после продолжительного периода бездействия, возникают так называемые «холодные» старты (часто встречаются в бессерверных установках), которые приводят к тому, что выполнение кода приостанавливается на большее число секунд.

<https://hashdork.com/ru/serverless-vs-microservices/>

<https://timeweb.cloud/blog/monolity-mikroservisy-besservernaya-arhitektura>

**ПОСТРОЕНИЕ ПРИЛОЖЕНИЯ**

1. **МОНОЛИТНАЯ АРХИТЕКТУРА**

Концепция монолитного программного обеспечения заключается в том, что различные компоненты приложения объединяются в одну программу на одной платформе. Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения. Все части программного обеспечения унифицированы, и все его функции управляются в одном месте.

Монолитная архитектура удобна для работы небольших групп, поэтому многие стартапы выбирают этот подход при создании приложения. Компоненты монолитного программного обеспечения взаимосвязаны и взаимозависимы, что помогает программному обеспечению быть самодостаточным.

**Достоинства:**

1. Простой деплоймент. Благодаря монолитному ядру разработчикам не нужно развертывать изменения или обновления по отдельности, поскольку они могут сделать это сразу и сэкономить много времени
2. Простота и скорость разработки.  В монолитной архитектуре разработчик может быстро начать реализовывать бизнес-логику, вместо того чтобы тратить время на размышления о межпроцессном взаимодействие
3. Меньше сквозных проблем. Большинство приложений зависят от множества межкомпонентных задач, таких как контрольные журналы, ведение логов, ограничение скорости и т. д. Монолитные приложения гораздо легче учитывают эти вопросы благодаря своей единой кодовой базе. К этим задачам проще подключать компоненты, когда все работает в одном приложении
4. Производительность. При правильной сборке монолитные приложения обычно более производительны, чем приложения на основе микросервисов благодаря общему коду и памяти.

**Недостатки:**

1. Привязанность к одному стеку технологий. Добавление новых технологий или фреймворков чревато перепроектированием всего приложения. Такое скудное технологическое разнообразие ограничивает ваши возможности и замедляет дальнее развитие приложения
2. Громоздкость. С ростом проекта монолит так же растет, становится все более запутанным и непонятным для других разработчиков.
3. Ограниченная гибкость, так как каждое обновление приложения требует полного повторного развертывания.
4. Масштабируемость. Масштабирование монолитного приложения может стать сложной задачей, поскольку все приложение должно масштабироваться вместе, а не только необходимые части. Это отсутствие гибкости часто увеличивает затраты и снижает эффективность при работе с большими нагрузками.
5. Поддержка. поддержка монолитной кодовой базы становится все более сложной задачей по мере роста сложности и размера приложения. Эта трудность связана с тесной связью компонентов, из-за чего разработчикам сложнее модифицировать или отлаживать приложение, не затрагивая другие части.
6. Производительность. Является как плюсом, так и минусом. Достоинством она может служить при разработке небольших приложений. Однако если проект растет, число пользователей и функций увеличивается, то и производительность сильно падает.
7. Неработоспособность при частичном отказе. Если что-то в монолите работает некорректно или в целом не работает, то и весь монолит не будет работать.

<https://habr.com/ru/companies/otus/articles/476024/>

<https://aws.amazon.com/ru/compare/the-difference-between-monolithic-and-microservices-architecture>

<https://habr.com/ru/companies/haulmont/articles/758780/>

<https://appmaster.io/ru/blog/monolitnaia-i-mikroservisnaia-arkhitektura#preimushchestva-monolitnoi-arkhitektury>

1. **МИКРОСЕРВИСНАЯ АРХИТЕКТУРА**

**Микросервисная архитектура** — один из подходов проектирования, при котором единое приложение строится как набор небольших сервисов. Каждый модуль работает в собственном процессе и взаимодействует с другими модулями. Модули работают независимо и построены вокруг бизнес-потребностей и выполняют определённую функцию. Например, хранят данные пользователей или отвечают за аутентификацию.

**Достоинства**:

1. Масштабируемость. Каждый микросервис можно масштабировать независимо от других, исходя от его конкретных потребностей. Такая детализированная масштабируемость позволяет эффективно распределять ресурсы и обеспечивает оптимальную производительность, особенно при работе с меняющимися требованиями к рабочей нагрузке
2. Технологическое разнообразие. Каждый сервис работает обособленно, так что можно подобрать для него разные технологии и фреймворки – если только их интерфейсы не зависят от технологий. Это открывает целый мир возможностей и позволяет командам выбирать лучшие инструменты для реализации конкретных задач.
3. Простота разработки. Над каждым микросервисом может работать отдельная команда разработчиков, что одновременно и ускоряет работу, и упрощает ее. Команды не обязаны координировать развертывание.
4. Эффективный деплоймент. Более короткое время запуска и возможность развертывания микросервисов независимо друг от друга действительно выгодны для CI / CD. По сравнению с обычным монолитом он намного плавнее.
5. Работоспособность при частичном отказе системы. В микросервисах изолируемые разломы лучше по сравнению с монолитным подходом. Хорошо спроектированная распределенная система переживет сбой одного сервиса.

**Недостатки**:

1. Сложность. Несмотря на простоту разработки (ведь разрабатывать отдельные сервисы намного легче, чем целый монолит), микросервисы сложно координировать. Для координации взаимодействия между сервисами, обеспечения согласованности данных и обработки сбоев необходимо скрупулезное проектирование и внедрение.
2. Тестирование и отладка. Отладка микросервисных приложений может оказаться более сложной задачей, поскольку затронутые микросервисы могут разрабатываться разными людьми и командами. Это означает, что для отладки потребуется координировать тесты, обсуждения и сбор отзывов между многими членами команды, а это дополнительные затраты времени и ресурсов.
3. Транзакции. Транзакции легче проводить в монолите. Решением этой проблемы на микросервисах является Saga Pattern. Хорошее решение, но все же слишком громоздкое для реализации на практике.
4. Оборудование. Помимо вышеперечисленных сложностей, для микросервисов также может потребоваться больше оборудования, чем для традиционных монолитов. Иногда микросервисы могут превзойти один монолит, если есть его части, которые требуют масштабирования до предела.

<https://habr.com/ru/companies/haulmont/articles/758780/>

<https://habr.com/ru/companies/slurm/articles/674600/>

**ТИПЫ ВЕБ-ПРИЛОЖЕНИЯ**

1. **SPA**

SPA или Single Page Application — это одностраничное веб-приложение, которое загружается на одну HTML-страницу. Благодаря динамическому обновлению с помощью JavaScript, во время использования не нужно перезагружать или подгружать дополнительные страницы. На практике это означает, что пользователь видит в браузере весь основной контент, а при прокрутке или переходах на другие страницы, вместо полной перезагрузки нужные элементы просто подгружаются.

**Достоинства**:

1. Высокая скорость. Все ресурсы загружаются за одну сессию, а во время действий на странице данные просто меняются, что очень экономит время.
2. Гибкость и отзывчивость пользовательского интерфейса. За счет того, что веб-страница всего одна, проще построить насыщенный интерфейс, хранить сведения о сеансе, управлять состояниями представлений и анимацией;
3. Кэширование данных. Приложение отправляет всего один запрос, собирает данные, а после этого может функционировать в offline-режиме.
4. Кроссплатформенность. Работает на любых устройствах и операционных системах.
5. Поддержка. Легче мониторить и исправлять ошибки.

**Недостатки**:

1. Безопасность. Высокая уязвимость к XSS-атакам.
2. Не сохраняется история перехода по страницам, поэтому не поддерживаются навигационные кнопки браузера.
3. Дольше загружается при первом посещении.
4. Может возникнуть проблема совместимости с другими браузерами
5. Требуется JavaScript
6. **MPA**

MPA или Multi Page Application — это многостраничные приложения, которые работают по традиционной схеме. Это означает, что при каждом незначительном изменении данных или загрузке новой информации страница обновляется. Такие приложения тяжелее, чем одностраничные, поэтому их использование целесообразно только в тех случаях, когда нужно отобразить большое количество контента.

**Достоинства**:

1. Безопасность. Меньшая уязвимость к атакам.
2. Простая SEO-оптимизация, так как можно оптимизировать каждую из страниц под нужные запросы.
3. Меньший стек технологий. Как правило для разработки многостраничного приложения требуется меньший стек технологий.
4. Пользовательский опыт. За счет простого интерфейса и навигации многостраничные приложения намного удобнее для пользователей.

**Недостатки**:

1. Тесная связь между бэкендом и фронтендом, поэтому их не получается развивать параллельно
2. Длительные сроки разработки
3. Низкая скорость взаимодействия, так как MPA перезагружают контент, когда пользователь взаимодействует с приложением.
4. Поддерживать и обновлять многостраничные сайты намного дольше и тяжелее, чем одностраничные.

 Стоит также отметить, что SPA часто является частью архитектуры многостраничного вебсайта. К примеру, основная часть ресурса может быть многостраничной, а личный кабинет реализован в виде SPA.

<https://webcase.com.ua/blog/razrabotka-odnostranichnyh-prilozhenij-spa-webcase/#f1>

<https://vc.ru/seo/108149-odnostranichnye-spa-i-mnogostranichnye-pwa-veb-prilozheniya>

<https://www.purrweb.com/ru/blog/odnostranichnye-prilozheniya-polnyj-gid-po-razrabotke/>

1. **ПРОГРЕССИВНЫЕ ВЕБ-ПРИЛОЖЕНИЯ**

Progressive Web App (PWA) — это адаптация сайта компании под мобильное устройство в виде приложения. PWA приложения повторяют функционал мобильного приложения. Веб-страницы размещаются на домене, а на устройствах открываются через браузеры. Можно назвать прогрессивные веб-приложения продвинутыми версиями сайта, которые адаптируются под устройство пользователя. При этом специальная разработка под Android и IOS не нужна.

**Достоинства**:

1. Пользовательский опыт. Так как это приложение практически не отличается от мобильного, пользователю удобно и привычно его использовать. Также там приходят знакомые всем пользователям Push-уведомления, что может вовлечь пользователя в приложение и бренд еще больше.
2. Скорость разработки. Для создания PWA приложений существуют специальные конструкторы, что ускоряет и упрощает разработку.
3. Меньший вес, чем у мобильного приложения
4. Быстрая скорость работы (выше, чем у сайта) и возможность перейти в офлайн-режим.
5. Разработка дешевле на 70%, чем у мобильных приложений

**Недостатки**:

1. Ограниченный функционал, так как PWA приложения не могут в полной мере использовать весь функционал устройства.
2. Для добавления PWA нужно посетить первый раз веб-приложение в браузере
3. Большой расход батареи

<https://vc.ru/marketing/141463-pwa-prilozheniya-chto-eto-takoe-i-dlya-chego-biznesu-sozdavat-prilozhenie-iz-sayta>  
<https://habr.com/ru/articles/418923/>

Какие архитектуры могут быть выбраны для разработки нашего приложения?

В первую очередь, в начале разработки, пока приложение не стало популярным, можно использовать клиент-серверную архитектуру. Однако с распространением приложения и ростом пользователей (а пользователей в каждом ресторане/кафе может быть очень много, около 50 человек) нужно будет приобретать и обеспечивать большее количество серверов, в результате чего можно будет рассмотреть переход на бессерверную архитектуру, так как Она отлично работает для приложений, насыщенных клиентами, и приложений, которые быстро растут и нуждаются в неограниченном масштабе.

Касательно внутреннего устройства приложения, будет разумно использовать микросервисную архитектуру в целях распределения нагрузки между серверами и сервисами.

Так как мы разрабатываем веб-приложение, будет разумно разрабатывать не просто веб-сайт, а прогрессивное веб-приложение, чтобы пользователям было удобно и быстро его использовать.

В совокупности микросервисная архитектура и тип PWA могут обеспечить быстрые сроки разработки, а также упростить ее.