Web-architecture

▼ Что такое Web-archetecture

это процесс проектирования, создания и внедрения компьютерной программы в Интернете. Часто эти программы представляют собой веб-сайты, содержащие полезную информацию для пользователя, и веб-разработчики могут разрабатывать эти программы для определенной цели, компании или бренда. Веб-архитектура включает в себя каждый компонент приложения и помогает веб-разработчикам создавать проекты, улучшающие взаимодействие с пользователем.

▼ Важность Web-archetecture

Она служит основой для каждого элемента цифрового приложения или программы. Поскольку это основа, качество архитектуры часто напрямую влияет на качество конечного продукта. Кроме того, эффективная веб-архитектура может обеспечить адаптируемость, если работодатель потребует значительных изменений в уже начатом проекте. Важно разработать надежную основу веб-архитектуры, чтобы проект мог развиваться эффективно и в рамках установленного бюджета.

▼ Уровни архитектуры веб-приложений

1. Уровень представления (PL) Уровень представления PL отображает пользовательский интерфейс и упрощает взаимодействие с пользователем. Уровень представления имеет компоненты пользовательского интерфейса, которые визуализируют и показывают данные для пользователей. Также существуют компоненты пользовательского процесса, которые задают взаимодействие с пользователем. PL предоставляет всю необходимую информацию клиентской стороне. Основная цель уровня представления - получить входные данные, обработать запросы пользователей, отправить их в службу данных и показать результаты.
2. Уровень обслуживания данных (DSL) Уровень службы данных DSL передает данные, обработанные уровнем бизнес-логики, на уровень

представления. Этот уровень гарантирует безопасность данных, изолируя бизнес-логику со стороны клиента.

3. Уровень бизнес-логики (BLL) Слой бизнес-логики BLL несет ответственность за надлежащий обмен данными. Этот уровень определяет логику бизнес-операций и правил. Вход на сайт - это пример уровня бизнес-логики.
4. Уровень доступа к данным (DAL) Уровень доступа к данным DAL предлагает упрощенный доступ к данным, хранящимся в постоянных хранилищах, таких как двоичные файлы и файлы XML. Уровень доступа к данным также управляет операциями CRUD - создание, чтение, обновление, удаление.

▼ Типы архитектуры веб-приложений

▼ Одностраничные веб-приложения 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. Такого эффекта удается добиться с помощью продвинутых фреймворков JavaScript: Angular, React, Ember, Meteor, Knockout.

Приемущества:

- Высокая скорость — все ресурсы загружаются за одну сессию, а во время действий на странице данные просто меняются, что очень экономит время;
- 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;
- упрощенная разработка — код можно начинать писать с файла file://URL, не используя сервер, не нужен отдельный код для рендера страницы на стороне сервера;

- кэширование данных — приложение отправляет всего один запрос, собирает данные, а после этого может функционировать в offline-режиме.

Недостатки:

- Seo оптимизация требует решений в виде серверного рендеринга — из-за того, что контент загружается при помощи технологии AJAX, которая подразумевает динамическое изменение содержания станицы, а для оптимизации важна устойчивость;
- нагрузка на браузер — из-за того, что клиентские фреймворки тяжелые, они довольно долго загружаются;
- необходима поддержка JavaScript — без JS нельзя полноценно пользоваться полным функционалом приложения;
- утечка памяти в Java Script — из-за плохой защиты, SPA больше подвержена действиям злоумышленников и утечке памяти.

▼ **Многостраничные веб-приложения 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.

Преимущества

- Простая SEO оптимизация — можно оптимизировать каждую из страниц приложения под нужные ключевые запросы;
- привычность для пользователей — за счет простого интерфейса и классической навигации.

Недостатки

- Тесная связь между бекендом и фронтендом, поэтому их не получается развивать параллельно;сложная разработка — требуют использования фреймворков как на стороне клиента, так и на стороне сервера, что увеличивает сроки и бюджет разработки.
- сложная разработка — требуют использования фреймворков как на стороне клиента, так и на стороне сервера, что увеличивает сроки и

бюджет разработки.

▼ ***SPA или MPA***

При выборе типа веб-приложения нужно ориентироваться на то, зачем именно вы его создаете. Многостраничный сайт подойдет интернет-магазину с большим количеством товаров и услуг, а если у вас, к примеру, инфобизнес, где можно изложить всю информацию в рамках сжатого веб-пространства — подойдет одностраничный сайт.

Выбор SPA целесообразен, если:

- есть необходимость в многофункциональном, насыщенном пользовательском интерфейсе;
- есть необходимость в интерфейсе API — использовании готовых блоков для построения приложения.

Выбор MPA целесообразен, если:

- приложения используются только для чтения информации;
- есть необходимость в использовании приложения в браузерах без поддержки JavaScript.

▼ **Прогрессивные приложения PWA**

Progressive Web Application взаимодействуют с пользователем, как приложение. Они могут устанавливаться на главный экран смартфона, отправлять push-уведомления и работать в офлайн-режиме.

Преимущества

- Кроссплатформенность — могут работать сразу с несколькими операционными системами;
- высокая скорость работы и возможность запуска и отображения данных в офлайн-режиме с моментальной загрузкой;
- высокая скорость установки;
- быстрая разработка — для создания PWA, не нужен отдельный сайт, достаточно изменить уже существующий.

Недостатки

Не все браузеры поддерживают основные функции таких приложений (например, Firefox и Edge).

▼ Архитектура микросервисов

Микросервисная архитектура — это подход, который помогает не только ускорить разработку продукта, но и сделать ее гибкой и управляемой: проект из неделимого целого превращается в систему связанных между собой блоков — сервисов.

Приложение с микросервисной архитектурой разделено на небольшие не зависящие друг от друга компоненты — микросервисы. У каждого из них своя бизнес-задача: например, управлять каталогом, хранить и обновлять содержимое корзины или проводить оплату заказа.

Благодаря тому, что части приложения автономны, его, как и любую распределённую систему, легко развивать и обновлять: добавление или улучшение отдельных функций никак не повлияет на остальные компоненты.

Преимущества.

Чтобы запустить новые функции или обновить существующие, достаточно изменить один модуль приложения. Это позволяет ускорить разработку и чаще выпускать обновления.

Для каждого сервиса можно использовать свой язык программирования, способ хранения данных, необходимые библиотеки.

Микросервисная архитектура позволяет вести гибкую разработку и при необходимости быстро изменить состав команды или требования к продукту.

Управление ресурсами, инфраструктурой и функциональностью приложения можно доверить разным сервисам и оптимизировать каждый из них по отдельности.

Если нагрузка на ресурсы микросервисов увеличится, они масштабируются автоматически. А гибкий процесс разработки позволит усилить команду дополнительными сотрудниками.

Проблемы внутри одного сервиса не нарушат работу системы в целом и не приведут к появлению новых ошибок.

Недостатки

Сбой одного сервиса не приведёт к полному отказу приложения, но любая распределённая система имеет и другие слабые места:

потенциальные проблемы связи её элементов друг с другом, сетевые задержки, возможная неконсистентность данных.

Вы сэкономите, если будете платить только за те ресурсы, которые потребляют микросервисы, но должны будете предусмотреть расходы на внедрение облачных технологий, отдельное развёртывание каждого нового сервиса и его покрытие отдельными тестами и мониторингами.

Контролировать качество решения отдельных бизнес-задач проще и эффективнее, чем оценивать систему в целом, но настроить рабочие процессы большой команды разработчиков не так уж легко.

▼ **Бессерверная архитектура**

это технологическое решение, основанное на событиях и запросах, которое позволяет разработчикам приложений создавать в облаке эффективные рабочие среды, обладающие всеми вычислительными ресурсами, необходимыми для организации бесперебойного процесса разработки. Подобные фреймворки очень удобны, особенно в условиях сжатых сроков и ресурсоемкости поставленных задач.

Преимущества:

- **Развернуть и запустить.** Ресурсы инфраструктуры управляются поставщиком облака. Таким образом, внутренняя ИТ-служба может сосредоточиться на бизнес-сценарии использования программных приложений, а не на управлении базовым оборудованием. Функции позволяют пользователям развертывать сборки приложений и файлы конфигурации, необходимые для предоставления требуемых аппаратных ресурсов.
- **Отказоустойчивой.** Поскольку кодирование бессерверных приложений логически отделено от базовой инфраструктуры, сбои оборудования имеют минимальное влияние на процесс разработки программного обеспечения. Пользователи не обязаны самостоятельно управлять приложениями.
- **Низкие эксплуатационные расходы.** В задачах инфраструктуры и управления операциямиуправляются поставщиками облачных, позволяя организации сосредоточить свои усилия на создание программных функций. Приложения выпускаются быстрее, что приводит к более быстрой обратной связи с конечными

пользователями и, следовательно, к дальнейшим улучшениям в течение следующих циклов выпуска программного обеспечения.

- **Оптимизированный биллинг на основе использования.** Модель выставления счетов с оплатой по мере использования особенно хорошо подходит для малых и средних (SMB) организаций, которым не хватает капитала для создания локальных центров обработки данных и управления ими. Текущая модель OpEx дополнительно оптимизирована функции настроены против потенциального чрезмерного выделения ресурсов и недостаточного использования ресурсов.
- **Встроенные интеграции.** Большинство поставщиков облачных услуг предлагают интеграцию с различными службами, которые позволяют пользователям сосредоточиться на создании высококачественных приложений, а не на их настройке.

Недостатки:

- поддерживает только определенные языки программирования. Большинство позволяют развертывать функции, написанные только на определенных языках. Правда они поддерживают большинство основных языков.
- Бессерверные фреймворки каждого производителя обладают собственной спецификой. В подавляющем большинстве бессерверные платформы обладают собственной спецификой, которая зависит от производителя.
- «Холодные» бессерверные функции снижают производительность. Фундаментальным ограничением бессерверных архитектур является то, что запуск «холодных» функций, то есть тех, которые не запускались на протяжении длительного времени, занимает дополнительное время. Время холодного старта может составлять всего несколько секунд, но в высокопроизводительных средах, когда счет идет на миллисекунды — это вечность.
- Вы не можете сделать бессерверное приложение целиком бессерверным. Запускать все в бессерверной среде — нецелесообразно или экономически невыгодно, поэтому эта модель предполагает способ запускать в ней определенные части приложения, которым постоянно требуется высокая

производительность и значительные вычислительные ресурсы. Можно сказать, что бессерверное решение — это способ дополнить другие типы архитектур приложений. Таким образом, Serverless — не универсальная архитектура, что ограничивает варианты ее применения.

▼ **Монолитная архитектура**

это традиционная модель программного обеспечения, которая представляет собой единый модуль, работающий автономно и независимо от других приложений. Монолитом часто называют нечто большое и неповоротливое, и эти два слова хорошо описывают монолитную архитектуру для проектирования ПО. Монолитная архитектура — это отдельная большая вычислительная сеть с единой базой кода, в которой объединены все бизнес-задачи. Чтобы внести изменения в такое приложение, необходимо обновить весь стек через базу кода, а также создать и развернуть обновленную версию интерфейса, находящегося на стороне службы. Это ограничивает работу с обновлениями и требует много времени.

Монолиты удобно использовать на начальных этапах проектов, чтобы облегчить развертывание и не тратить слишком много умственных усилий при управлении кодом. Это позволяет сразу выпускать все, что есть в монолитном приложении.

Преимущества:

Простое развертывание. Использование одного исполняемого файла или каталога упрощает развертывание.

Разработка. Приложение легче разрабатывать, когда оно создано с использованием одной базы кода.

Производительность. В централизованной базе кода и репозитории один интерфейс API часто может выполнять ту функцию, которую при работе с микросервисами выполняют многочисленные API.

Упрощенное тестирование. Монолитное приложение представляет собой единый централизованный модуль, поэтому сквозное тестирование можно проводить быстрее, чем при использовании распределенного приложения.

Удобная отладка. Весь код находится в одном месте, благодаря чему становится легче выполнять запросы и находить проблемы.

Недостатки:

Снижение скорости разработки. Большое монолитное приложение усложняет и замедляет разработку.

Масштабируемость. Невозможно масштабировать отдельные компоненты.

Надежность. Ошибка в одном модуле может повлиять на доступность всего приложения.

Препятствия для внедрения технологий. Любые изменения в инфраструктуре или языке разработки влияют на приложение целиком, что зачастую приводит к увеличению стоимости и временных затрат.

Недостаточная гибкость. Возможности монолитных приложений ограничены используемыми технологиями.

Развертывание. При внесении небольшого изменения потребуется повторное развертывание всего монолитного приложения.