Клиент-серверная архитектура

Веб-приложение устроено по определённым принципам. Они определяют, из каких элементов состоит приложение и как они связаны.

Такие принципы называют ***архитектурой***. Это способ организации работы приложения.

Самая распространённая архитектура веб-приложений — ***клиент-серверная***.

Клиент, сервер и интернет — элементы клиент-серверной архитектуры. Клиент и сервер работают по отдельности: клиент отвечает за взаимодействие с пользователем, сервер — за логические операции, вычисления и хранение данных, а интернет, или сеть — за связь клиента и сервера.

Клиент — это система, которая связывается с сервером и запрашивает нужную пользователю информацию.

Сервер — система, которая обрабатывает запросы клиента и формирует ответ. Например, сохраняет заказ или передаёт информацию о цене.
Клиент и сервер «общаются» по сети.

Интернет, или ***Сеть*** — система связанных между собой устройств, которая помогает клиенту и серверу обмениваться данными.

Архитектура веб-приложения в основном представляет отношения и взаимодействия между такими компонентами, как пользовательские интерфейсы, мониторы обработки транзакций, базы данных и другие. ***Основная цель*** - убедиться, что все элементы правильно работают вместе.

Логика довольно проста: когда пользователь вводит URL-адрес в браузере и нажимает «ввод», браузер делает запрос к серверу. Сервер отвечает, а затем показывает требуемую веб-страницу. Все эти компоненты создают архитектуру веб-приложения.

Работа системной архитектуры для веб-приложений

Все приложения состоят из двух частей - клиентской (front-end) и серверной (back-end).

Интерфейс - это визуальная часть приложения. Пользователи могут видеть интерфейс и взаимодействовать с ним. Клиентский код реагирует на действия пользователей. Серверная часть не визуальна для пользователей, но заставляет их запросы работать. Он обрабатывает бизнес-логику и отвечает на HTTP-запросы.

Поэтому, когда вы вводите свои учетные данные в регистрационную форму, вы имеете дело с внешним интерфейсом, но как только вы нажимаете «ввод» и регистрируетесь - это серверная часть заставляет его работать.

При правильной работе клиентская и серверная стороны составляют архитектуру программного обеспечения веб-приложения.

Слои и компоненты архитектуры веб-приложений

Чтобы лучше понять архитектуру веб-приложения, следует погрузиться в его компоненты и уровни. Веб-приложения разделяют свои основные функции на уровни. Это позволяет заменять или обновлять каждый слой независимо.

Базовые компоненты архитектуры веб-приложений

Веб-архитектура имеет компоненты пользовательского интерфейса и структурные компоненты. Последние также делятся на клиентские и серверные.

Компоненты пользовательского интерфейса

Компоненты пользовательского интерфейса обозначают все элементы интерфейса, такие как журналы активности, информационные панели, уведомления, настройки и многое другое. Они являются частью макета интерфейса веб-приложения.

Структурные компоненты состоят из клиентской и серверной сторон:

Клиентский компонент разработан с HTML, CSS или JavaScript. Веб-браузеры запускают код и преобразуют его в интерфейс, поэтому нет необходимости в настройке операционной системы.

Что касается серверного компонента, он построен на Java, .Net, Node.JS, Python и других языках программирования. ***Сервер состоит из двух частей*** - логики приложения и базы данных. ***Логика приложения*** - это центр управления веб-приложением. База данных отвечает за хранение информации (например, учетных данных).

Уровни архитектуры веб-приложений

Существует четыре общих уровня веб-приложений:

- Уровень представления (PL)
- Уровень обслуживания данных (DSL)
- Уровень бизнес-логики (BLL)
- Уровень доступа к данным (DAL)

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.

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.

Уровень службы данных

DSL передает данные, обработанные уровнем бизнес-логики, на уровень представления. Этот уровень гарантирует безопасность данных, изолируя бизнес-логику со стороны клиента.

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

Можно выделить несколько типов архитектуры веб-приложений, в зависимости от того, как логика приложения распределяется между клиентской и серверной сторонами. Наиболее распространенные архитектуры веб-приложений:

- Одностраничные веб-приложения
- Многостраничные веб-приложения
- Архитектура микросервисов
- Бессерверная архитектура
- Прогрессивные веб-приложения

Одностраничное приложение или SPA

SPA - это веб-сайт или веб-приложение, которое загружает всю необходимую информацию при входе на страницу. Одностраничные приложения имеют одно существенное преимущество - они обеспечивают потрясающий пользовательский интерфейс, поскольку пользователи не испытывают перезагрузки веб-страниц. Одностраничные веб-приложения часто разрабатываются с использованием фреймворков JavaScript, таких как Angular, React и других.

Известные СПА : Gmail, Facebook, Twitter, Slack.

Многостраничное приложение или MPA

Многостраничные приложения более популярны в Интернете, так как в прошлом все веб-сайты были MPA. В наши дни компании выбирают MPA, если их веб-сайт довольно большой (например, eBay). Такие решения перезагружают веб-страницу для загрузки или отправки информации с / на сервер через браузеры пользователей.

Известные MPA: eBay, Amazon.

Архитектура микросервисов

Чтобы понять архитектуру микросервисов, лучше сравнить ее с монолитной моделью.

Традиционная монолитная архитектура веб-приложения состоит из трех частей - базы данных, клиентской и серверной сторон. Это означает, что внутренняя и внешняя логика, как и другие фоновые задачи, генерируются в одной кодовой базе. Чтобы изменить или обновить компонент приложения, разработчики программного обеспечения должны переписать все приложение.

Что касается микросервисов, этот подход позволяет разработчикам создавать веб-приложение из набора небольших сервисов. Разработчики создают и развертывают каждый компонент отдельно.

Архитектура микросервисов выгодна для больших и сложных проектов, поскольку каждый сервис может быть изменен без ущерба для других блоков. Поэтому, если вам нужно обновить логику оплаты, вам не придется на время останавливать работу сайта.

Известные проекты: Netflix, Uber, Spotify, PayPal.

Типы архитектуры веб-приложений

- Монолитные и микросервисы
- Бессерверная архитектура

Этот тип архитектуры веб-приложений заставляет разработчиков использовать облачную инфраструктуру сторонних поставщиков услуг, таких как Amazon и Microsoft.

Чтобы сохранить веб-приложение в Интернете, разработчики должны управлять серверной инфраструктурой (виртуальной или физической), операционной системой и другими процессами хостинга, связанными с сервером. Поставщики облачных услуг, такие как Amazon или Microsoft, предлагают виртуальные серверы, которые динамически управляют распределением машинных ресурсов. Другими словами, если приложение испытывает огромный всплеск трафика, к которому серверы не готовы, приложение не будет отключено.

Прогрессивные веб-приложения или PWA

Одна из основных тенденций в разработке веб-приложений последних лет - это прогрессивные веб-приложения. Это веб-решения, которые работают как собственные приложения на мобильных устройствах. PWA предлагают push-уведомления, автономный доступ и возможность установить приложение на домашний экран.

Для создания PWA разработчики используют «языки веб-программирования», такие как HTML, CSS и JavaScript. Если приложению требуется доступ к функциям устройств, разработчики используют дополнительные API - NFC API, API геолокации, Bluetooth API и другие.

Известные PWA: Uber, Starbucks, Pinterest.

Разработка архитектуры для веб-приложения

Качественная архитектура веб-приложения делает процесс разработки более эффективным и простым. Веб-приложение с продуманной архитектурой легче масштабировать, изменять, тестировать и отлаживать.

Есть несколько общих критериев для хорошо построенной архитектуры веб-приложения:

- Эффективность
- Гибкость
- Расширяемость
- Соблюдение принципа открыто-закрыто
- Масштабируемость процесса разработки
- Легко проверить
- Возможность повторного использования
- Хорошо структурированный и читаемый код
- Нижняя граница