УДК 004.65

**А.Р. БОЩЕНКО**

Рязанский государственный радиотехнический университет

имени В.Ф. Уткина.

**ТИПЫ И КОМПОНЕНТЫ АРХИТЕКТУР СОВРЕМЕННЫХ ВЕБ-ПРИЛОЖЕНИЙ**

*Рассматриваются перспективные разновидности к подходу организации архитектуры веб-приложений. Описывается структура и все её компоненты с примерами различных технологий разработки.*

Веб-приложение – это клиент-серверное приложение с веб-интерфейсом, в котором есть браузер (клиент) и веб-сервер. Логика веб-приложения распределена между сервером и клиентом, есть канал для обмена информацией и хранилище данных, расположенное локально или в облаке. Удобство такого приложения обусловлено тем, что такую системы легче администрировать, реализовывать в ней хранение данных, управлять доступами к той или иной информации. Пользователи приложения могут подключиться с любой точки мира через сеть интернет, можно создать закрытую систему только для пользователей, подключенный к локальной сети. Отличительной особенностью веб-приложения от веб-сайта являются аутентификация, интеграция и интерактивность.

Каким стеком технологий нужно обладать, чтобы написать свое веб-приложение? Первое, что нужно изучить, это технологии front-end разработки. Язык гипертекстовой разметки HTML, именно его понимают браузеры интерпретируя в интерфейс. Каскадная таблица стилей CSS как дополнение к коду HTML, имеет тот же синтаксис, но предназначена для красивого оформления нашей страницы.

Язык программирования JavaScript, это специально созданный язык для реализации действий (скриптов) на HTML-страницах. В эпоху, когда было уже недостаточно просто статичной текстовой страницы в сети интернет, которую можно было просто почитать, сообщество программистов решило, что нужен инструмент, который будет слушать действия пользователя и реагировать на него, изменяя каким-либо образом внешний вид сайта. В последние года набирает популярность надстройка над языком JavaScript – TypeScript. Этот язык был разработан корпораций Microsoft в 2012 году. Он расширяет возможности JavaScript добавлением в язык явной статической типизации, в нем объекты имеют вид классических объектов объектно-ориентированного программирования, также появилась поддержка подключения модулей. React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

Знание сетевого протокола передачи данных в сети-интернет HTTP или защищенный протокол передачи данных HTTPS. Знание методов запроса к ресурсу таких, как GET-запрос, PUT-запрос, DELETE-запрос, POST-запрос.

Знание интерфейса API. API – протокол договоренностей о том, как между собой будут общаться программы. Содержит в себе протокол передачи данных, формат данных и модель данных. Есть несколько типов построения API:

* SOAP (Simple Object Access Protocol) – обмен структурированными сообщениями между веб-сервисами, формат данных XML;
* RPC (Remote Procedure Call) – метод взаимодействия между компонентами распределённой системы, позволяющий вызывать функции на сервере, как будто они являются локальными, формат данных XML, JSON и бинарные;
* WebSocket API – двунаправленный протокол связи, позволяет установить постоянное соединение между клиентом и сервером для обмена данными, формат данных может быть текстовым или бинарным, в том числе XML и JSON;
* Rest (Representational State Transfer) – одна из самых популярных в последние годы подходов к написанию запросов. Работает поверх HTTPS, эффективно используя все его свойства.

Для блока серверной части приложения нужно знать любой язык программирования и фреймворки, которые могут облегчить сборку и запуск нашего приложения на сервере. Рассмотрим более подробно Java-технологии. Конечно, любой разработчик должен отлично знать Java Core. Также важно владеть и понимать спецификацию, предложенную комьюнити Java, под названием Servlet API. Данная спецификация основана на Java-технологии, если точнее, это Java-класс, скомпилированный в байт-код. Servlet API состоит из 2 пакетов, а именно, javax.servlet javax.servlet.http. Стек фреймворка Spring, то есть все его модули, такие как Core, Web, Data Access, Test. А также проекты Spring Data, Spring Boot, Spring Batch. Данный фреймворк по сути отвечает за внедрение зависимостей с некоторыми удобными фишками, как взаимодействие с СУБД, прокси, аспектно-ориентированное программирование, веб-инфраструктура MVC)

Необходимо понимание работы СУБД, знание языка SQL, необходимо иметь представление о технологии JDBC для доступа к базам данных из программ на языке Java.

Архитектура веб-приложения – это высокоуровневая структура, описывающая, как различные компоненты веб-приложения взаимодействуют друг с другом с целью повышения функциональности и результативности продукта для бизнеса. На этапе выбора архитектуры важно не потеряться в огромном разнообразии их типов, ведь неправильный выбор может нанести ущерб бизнесу.

Современные веб-приложения используют концепцию [трехуровневой](https://www.ibm.com/cloud/learn/three-tier-architecture) архитектуры, которая разделяет приложения на уровень представления, уровень приложений и уровень данных. Трехуровневая схема архитектуры веб-приложения более безопасна, поскольку клиент не имеет прямого доступа к данным. В рамках трехуровневой архитектуры веб-приложения каждый уровень работает на своей собственной инфраструктуре и может разрабатываться параллельно разными командами. Такая структура позволяет обновлять и масштабировать каждый уровень по мере необходимости, не затрагивая другие уровни.

Типы архитектур веб-приложений.

Рендеринг на стороне сервера (Server-Side Rendering) – это метод отрисовки веб-страницы на сервере, а не в браузере. Сервер отдает сгенерированную HTML-страницу пользователю, пока основное веб-приложение загружается и запускается. Рендеринг на стороне сервера Vue.js Приложение также можно считать "изоморфным" или "универсальным" в том смысле, что большая часть кода вашего приложения выполняется как на сервере, так и на клиенте. Статическая веб-страница, используя JavaScript, отправляет запрос серверу (API), сервер возвращает клиенту HTML-страницу. В случае если на новой странице есть только несколько элементов, отличных от текущей, браузер всё равно запросит всю страницу. Для кэширования страницы используется NGINX. Этот способ позволяет значительно ускорить процесс загрузки страницы, не обращаясь к бэкэнду.

Если данный тип сравнивать с SPA (клиентское одностраничное приложение), то у него есть ряд преимуществ:

* время обработки контента более низкое, данное свойство хорошо заметно, когда очень медленный доступ в интернет или медленное устройство;
* унифицированная ментальная модель данных. То есть не нужно будет метаться от языка к языку, на котором устроен ваш внешний интерфейс и внутренний сервер;
* Улучшение SEO: поисковые роботы будут видеть полностью отрисованную страницу напрямую.

Рендеринг на стороне клиента (Client-Side Rendering) – страница отображается на компьютере в браузере. При рендеринге на стороне клиента (CSR) получается простой HTML-документ с информацией о стилях CSS и файлом JavaScript, который будет отображать остальную часть сайта с помощью браузера. Это радикально отличается от использования SSR. Сервер отвечает только за загрузку HTML и CSS, всё остальное обрабатывается библиотекой JavaScript на стороне клиента. Имейте в виду, что CSR может повлиять на SEO. Некоторые поисковые роботы могут не выполнять JavaScript и, следовательно, видеть только начальное состояние пустоты или загрузки вашего приложения. Это также может привести к проблемам с производительностью у пользователей с более медленными интернет-соединениями или устройствами, поскольку им нужно дождаться загрузки и запуска всего JavaScript, прежде чем они смогут увидеть всю страницу целиком.

Статическая генерация сайта (Static Site Generation) – предоставляет пользователю через API по запросу сгенерированную HTML-страницу, расположенную на [CDN](https://en.wikipedia.org/wiki/Content_delivery_network) или любом другом сервере. Не требует пересоздания страницы, т.к. используется статический шаблон. Генерирует сайт на основе необработанных данных и набора шаблонов. Т.к. страницы заранее созданы, это приводит к значительному ускорению загрузки в браузере.

К плюсам можно отнести высокую производительность, разгруженный сервер, т.к. статические страницы не занимают много памяти. К минусам явно можно определить такие факты, как наличие только нескольких шаблонов (вообще отсутствие такового), тесть разработчику придется потратить время вначале на создание страниц, со стороны пользователя сложнее публиковать контент с помощью генератора статических сайтов, скорее всего придется прибегать к помощи разработчиков.

Одностраничное приложение (Single Page Application) – при работе загружает только одну страницу, а когда нужно поменять содержимое обращается к API JavaScript для отображения нового содержимого. В данном типе проектирования весь код, состоящий из HTML, CSS, JS, либо добавляется по мере необходимости, либо извлекается браузером. Это позволяет пользователю взаимодействовать с веб-приложением не загружая целые новые страницы с сервера, что потенциально повышает производительность.

Прогрессивное веб-приложение (Progressive Web App) – строится на логике одностраничного веб-приложения, но с сервисами, которые запускаются в браузере. В данном типе архитектуры нужно учитывать, что браузер и операционная система поддерживает набор указанных стандартов. Со стороны пользователя такое веб-приложение просит добавить себя на экран операционной системы. Приложение автоматически добавляется на устройство, когда пользователь дает свое согласие в сплывающем окне браузера. За счет такого подхода обеспечивается автономность приложения, фоновая синхронизация и допуск push-уведомлений.

Интерфейс MICRO (micro front-end) – технология подразумевает разделение монолитного веб-приложения на микросервисы, которые запущены на отдельных серверах. Каждое мини-приложение отвечает за определенный функционал. Благодаря такому подходу реализовывается распределение нагрузки. Микросервисы могут обращаться друг к другу посредством HTTP-запросов. Для пользователя все они находятся на одной страницы.

Описание структуры современного веб-приложения.

Уровень представления (Front-end) – всё с чем взаимодействует пользователь приложения. В современном мире есть два основных вида фронтенда: веб-интерфейсы и мобильные приложения. Первый вид реализуются в основном с использованием языка программирования JavaScript и фреймворка React. Мобильные приложения под iOS пишутся на языке Swift, а под Android в основном используется язык Kotlin.

Язык Swift был специально создать для устройств Apple. Поддерживает и функциональное программирование и объектно-ориентированное программирование, защищен от ошибок строгой типизаций (в отличие от его предшественника Objective-C), работа с памятью, когда часть кода (объект) перестает быть нужным, Swift автоматически освобождает память.

Язык Kotlin разработанный компанией JetBrains на базе JVM с полной компиляций в Java-байт код (с недавнего времени также компилируется в JavaScript). Достаточно прост в использовании и изучении Java-разработчикам. Используется на данный момент в разработке приложений под устройства Android вместо Java, но, конечно, не только в данной сфере разработки.

Уровень приложения (Back-end) – приложение запускается на сервере и является ключевым компонентом схемы архитектуры веб-приложения, который получает запросы пользователей, выполняет бизнес-логику и доставляет данные в СУБД. Бекенд может реализовываться на разных языках программирования, например, программная платформа Node.js, основанная на движке V8 (компилирующем JavaScript в машинный код); Java и фреймворки Spring, Python, PHP Laravel, Go, платформа .NET, Ruby.

Для того, чтобы обеспечить надежность большой системы, применяют технологию микросервисов, а также технологию Replication (запуск каждого из микро-приложений на нескольких отдельных серверах). Микросервисы обрели свою популярность, когда проблема развития большого монолитного приложения стала слишком велика, а также стало рискованными обновлять приложение, т.к. в монолитной структуре есть вероятность возникновения проблем в других местах. Внедрение новых функций тоже становится проблематичным, т.к. придется переписывать весь код, а это замедлит процесс разработки. Переход на микросервисную структуру решают эти и многие другие проблемы. Преимущества этого подхода:

* микросервисы состоят из отдельных компонентов, которые не влияют на другие, что предоставляет разработчик гибкость в управлении и развитии приложения, также есть возможность выбора технологий более подходящих для каждого отдельного сервиса.
* каждый микросервис может масштабироваться отдельно от других, позволяет эффективно управлять нагрузками;
* каждый микросервис представляет собой отдельную единицу, и его обновление может быть проведено независимо от других служб. Это означает, что развертывание новых версий службы не требует остановки всего приложения. Это минимизирует время простоя системы и улучшает общую доступность приложения.

При внедрении данного типа архитектуры используют кросс-платформенное программное обеспечение для размещения и поддержки веб-сервера. Одно из самых популярных это веб-сервер Apache. Благодаря ему устанавливается соединение между пользователем и сервером. Он ловит данные, отправляемые пользователем, направляет их на сервер, работает с динамическими PHP страницами, отвечает за распределение нагрузки сервера, повышает отказоустойчивость. Соответственно также работает в обратную сторону, от сервера клиенту.

Помимо популярного Apache существуют и другие решения, их достаточно много, но можно выделить самых значимых конкурентов. Рассмотрим программу для балансировки нагрузки (Load Balancer), под названием NGINX. Его главное преимущество перед Apache является преодоление барьера в 10 000 одновременных соединений. В ней можно зарегистрировать все компьютеры реплики для определенного сервиса, в момент, когда приходит запрос от пользователя в конкретный микросервис, load balancer будет принимать решение на какую конкретно машину-сервер будет направлен запрос, а также может самостоятельно равномерно распределять нагрузку. Внутри есть информация под какими IP-адресами в интернете находятся те или иные сервера. Таким образом фронтенд обращается теперь только к балансировщику. Некоторые хостинги используют на серверах связку Nginx+Apache. Nginx отдает статический контент, а Apache подключается позже и отвечает за обработку динамического контента.

Еще один конкурент – Tomcat, разработанный компанией, которая выпустила Apache, это контейнер сервлетов с открытым исходным кодом, который также выполняет функцию веб-сервера. Он используется для работы приложений, написанных на Java.

Для обеспечения более корректной работы между микросервисами применяют программный компонент под названием брокер сообщений. В его работе используются две основные сущности: producer (отправитель) и consumer (потребитель/подписчик). Каждый компонент системы, который хочет отправить или получить данные, подписывается на топик, используя специальный протокол (MQTT). В зависимости от используемого протокола обмена данными, брокеры могут поддерживать различные типы сообщений, такие как текстовые, бинарные данные или даже графические изображения. Когда отправитель передает сообщение, его принимает брокер и размещает в топике. Затем программный компонент сообщает о наличии нового письма всем подписчикам данного топика. Каждый подписчик получает его копию, которую может обработать или проигнорировать в зависимости от потребностей. Этот процесс называется публикацией-подпиской и используется в различных системах, таких как интернет-чаты, социальные сети, системы мониторинга и управления ресурсами и т.д. Брокер сообщений гарантирует, что сообщение, которое ему отправили, не будет утеряно. Например, ApacheKafka.

Прикладной уровень (API) – интерфейс программирования приложения. Инструмент, который позволяет приложениям взаимодействовать друг с другом. Веб API доступен через интернет по протоколу HTTP. Создается с использованием либо Java, либо .NET.

Уровень данных – базы данных, нужны для сохранения и структурирования любых данных, чтобы позже по запросу быстро выдать ответ серверу, а он в свою очередь клиенту. Приложение на сервере в свою очередь должно уметь обращаться к базам данных. Самыми популярными на данный момент в мире являются SQL-базы данных. Это базы данных, которые используют язык запросов SQL для формирования команд к ним на вставку, получение, обновление, удаление данных и тд. БД множество разнообразных, например, PostgreSQL, MySQL, Oracle DB, Microsoft SQL Server и тд.

Кроме баз данных используется кэш, отдельное хранилище, в котором находятся данные, которые редко меняются, к которым часто обращается приложение и данные, не относящиеся к критически важным, которые часто меняются. Такой подход эффективно использует скорость работы с данными, характерную для памяти, и смягчает нагрузку центральной базы данных приложения, связанную с обработкой разного вида данных. Например такие базы данных, как Redis или Memcached.

Redis (Remote Dictionary Service) — это опенсорсный сервер баз данных типа ключ-значение. Можно сказать, что Redis – это сервер структур данных. Уникальные особенности сервера Redis стали основной причиной его популярности и того, что он применяется во множестве реальных проектов. Первое время Redis использовали практически так же, как Memcached. Но, по мере развития Redis, эта система управления базами данных (СУБД) нашла применение и во многих других ситуациях. В частности — в реализациях механизма издатель/подписчик, в задачах потоковой обработки данных, в системах, где нужно работать с очередями. Redis поддерживает такие типы данных как: строка (String), битовый массив (Bitmap), битовое поле (Bitfield), хеш-таблица (Hash), список (List), множество (Set), упорядоченное множество (Sorted set), геопространственные данные (Geospatial), структура HyperLogLog (HyperLogLog), поток (Stream). Ещё одна важная особенность Redis заключается в том, что эта СУБД размывает границы между кешем и хранилищем данных. Тут важно понять то, что чтение данных из памяти и работа с данными, находящимися в памяти, гораздо быстрее чем те же операции, выполняемые традиционными СУБД, использующими обычные жёсткие диски (HDD) или твердотельные накопители (SSD).

В случае, если приложение подразумевает постоянный поток большого количества данных, и есть вероятность, что на машине, где запущена SQL-БД закончится память на жестком диске, используют NoSQL-базы данных. NoSQL применяется к системам, в которых делается попытка решить проблемы масштабируемости и доступности за счёт полного или частичного отказа от требований атомарности и согласованности данных. Такие базы данных позволяют легко распределяют данные по нескольким компьютерам, объединенных в кластер. Также они не предоставляют ACID-гарантии, только гарантии BASE (Basically Available, Soft State, Eventual Consistency). Например, MongoDB, Apache Cassandra, Neo4j и тд.

Для хранения файлов любого типа и объема используются object storage хранилища. Объектное хранилище, часто называемое объектно-ориентированным хранилищем, представляет собой архитектуру хранилища данных, идеально подходящую для хранения, архивирования, резервного копирования больших объемов статических неструктурированных данных и управления ими — надежно, эффективно и по доступной цене. Они предоставляют высокий уровень надежности и доступности. Принцип работы с S3-хранилищем сводится к созданию контейнеров и добавлению туда необходимых файлов, которые представляются в виде объектов. Всё, что попадает в контейнер, можно просматривать, перемещать или удалять.

Поиск ошибок и сбоев в работе системы, выявление вредоносной активности, сбор статистики посещения веб-ресурса выполняется с помощью сбора Log-ов. Для более удобной работы с логами существует такая база данных, как Elasticsearch. Удобный пользовательский интерфейс для работы с этой базой данных предоставляет Kibana.

Elasticsearch – это распределённая RESTful-система на основе JSON, которая сочетает в себе функции NoSQL-базы данных, поисковой системы и аналитической системы. Elasticsearch может использоваться для сбора логов и аналитики журналов, так как он обладает следующими возможностями:

* быстро обрабатывать большие объёмы журналов;
* индексировать системные логи по мере поступления;
* выполнять запросы к ним в режиме реального времени.

Все компоненты, перечисленные выше, составляют архитектуру веб-приложения. Реализовать такую структуру можно на большом количестве серверов, которые нужно приобрести, разместить и настроить, а можно воспользоваться сервисом по аренде серверов, например, AWS (Amazon Web Services), Google Cloud, Heroku, Microsoft Azure и тд.

Amazon Web Services — это набор облачных сервисов от компании Amazon. На единой платформе пользователи могут заказать вычислительные ресурсы, хранилище, инфраструктуру, сервисы с готовыми для использования инструментами. Если серверы Амазон перестанут работать, это сразу станет заметно. Услугами платформы пользуются крупнейшие корпорации. Согласно результатам исследования Synergy Research Group, AWS занимает треть мирового рынка облачных решений. На второй строчке рейтинга — Microsoft Azure с долей в 19%. Но конкурентов много: Google Cloud, IBM Cloud, timeweb.cloud и другие площадки тоже предлагают клиентам удобные решения для бизнеса.

Управлять сервисами можно тремя способами: AWS Console в браузере; AWS CLI — утилита для управления через командную строку. Отличное решение для автоматизации рутины; SDK — комплект приложения разработчика. SDK доступны для разных языков программирования, от C++ и Java до Python и PHP. Даже если нужного языка нет в списке, можно найти библиотеку, которую поддерживает сообщество. Или написать свой набор инструментов. Выбор способа зависит от квалификации администратора и задач, которые нужно решить.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Разработка веб-приложений на платформе Spring : учебно-методическое пособие / В.И. Суханов ; М-во науки и высшего образования РФ.— Екатеринбург : Изд-во Урал. ун-та, 2023.— 180 с. — ISBN 978-5-7996-3735-4. — Текст : непосредственный.
2. Архитектура и фреймворки веб-приложений : учебное электронное пособие / К. А. Кулаков, В. М. Димитров ; М-во науки и высшего образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования Петрозавод. гос. ун-т. — Электрон. дан.— Петрозаводск : Издательство ПетрГУ, 2020. — 1 CD-ROM. — Систем. требования : PC, MAC с процессором Intel 1,3 ГГц и выше ; Microsoft Windows, MAC OSX ; 256 Мб (RAM); Adobe Reader ; дисковод CD-ROM. — Загл. с титул. экрана. — Текст : электронный.