МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И
ТЕХНОЛОГИЧЕСКОГО ОБРАЗОВАНИЯ**
Кафедра информационных технологий и электронного обучения

Основная профессиональная образовательная программа
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль) «Технологии разработки программного обеспечения»
форма обучения – очная

ОБЗОР ИСТОЧНИКОВ

по теме

«архитектура программного обеспечения для веб-приложений»

Обучающегося 4 курса
Чирцова Тимофея Александровича

Санкт-Петербург
2024

Оглавление

ВВЕДЕНИЕ

Веб-приложения занимают центральное место в современном цифровом мире, обеспечивая пользователям доступ к информации и сервисам через интернет. Архитектура программного обеспечения для веб-приложений играет ключевую роль в обеспечении их производительности, безопасности и масштабируемости. На протяжении последних десятилетий развитие архитектурных подходов к созданию веб-приложений значительно изменилось, что связано с ростом требований со стороны бизнеса и пользователей [1].

Основная цель анализа архитектуры веб-приложений заключается в изучении её ключевых компонентов, подходов к проектированию и технологий, которые позволяют создавать устойчивые и функциональные системы. Понимание архитектурных принципов критически важно для разработчиков, архитекторов программного обеспечения и других специалистов, занимающихся созданием и поддержкой веб-приложений. Анализируя различные источники, включая научную и учебную литературу, статьи, а также профессиональные интернет-ресурсы [2], можно выделить основные направления развития архитектуры программного обеспечения и определить, какие подходы наиболее эффективны в различных сценариях использования.

Современные веб-приложения требуют высокой производительности, доступности и устойчивости к сбоям, что подчеркивает необходимость выбора правильной архитектуры. Например, микросервисная архитектура, ставшая популярной в последние годы, позволяет создавать масштабируемые системы с высокой степенью модульности [3]. Наряду с этим, существуют и другие подходы, такие как монолитная архитектура и серверлесс-архитектура, каждая из которых имеет свои преимущества и ограничения в зависимости от задач и контекста.

Принципы построения архитектуры бэкенда

Современная архитектура бэкенда основывается на ключевых принципах, которые обеспечивают её надёжность, производительность и гибкость. Эти принципы являются фундаментом для разработки эффективных и устойчивых систем.

1. Модульность и разделение обязанностей
   Бэкенд следует проектировать с учётом модульности, где каждая часть системы отвечает за конкретную задачу. Это упрощает масштабирование, тестирование и сопровождение кода. Такой подход основан на принципе единой ответственности (Single Responsibility Principle), который широко применяется в объектно-ориентированном проектировании [3].
2. Масштабируемость
   Архитектура должна быть способна адаптироваться к растущему числу пользователей и объёму данных. Это достигается за счёт горизонтального и вертикального масштабирования, а также использования технологий, таких как микросервисы и кластеризация
3. Устойчивость к сбоям (Fault Tolerance)
   Надёжность системы обеспечивается механизмами резервирования, автоматического восстановления и распределённой обработки данных. Например,

использование архитектуры с несколькими узлами и репликацией данных помогает минимизировать последствия отказов

4. Производительность и низкие задержки
   Важным требованием является минимизация задержек при обработке запросов. Для этого используются оптимизированные алгоритмы, кэширование и асинхронная обработка [6].
5. Безопасность
   Архитектура бэкенда должна защищать данные пользователей и предотвращать несанкционированный доступ. Это включает шифрование данных, использование токенов для аутентификации и контроль доступа на уровне API
6. Гибкость и расширяемость
   Возможность адаптации архитектуры под новые требования достигается за счёт использования стандартов и протоколов, таких как REST или gRPC, а также применения гибких технологий, например, серверлесс-архитектуры
7. Логирование и мониторинг
   Системы бэкенда должны предоставлять информацию о своём состоянии через логирование и мониторинг. Это позволяет выявлять узкие места и обеспечивать стабильность работы приложения [7].

Виды архитектуры бэкенда и современные подходы

Эволюция архитектуры бэкенда прошла через несколько этапов, каждый из которых вносил свои особенности и ограничения. Современные подходы направлены на обеспечение высокой производительности, гибкости и масштабируемости систем. Рассмотрим основные типы архитектуры, использовавшиеся в разработке бэкенда, и определим, какой подход наиболее эффективен сегодня.

На ранних этапах разработки веб-приложений основным подходом была монолитная архитектура. Она представляет собой единую структуру, где все компоненты приложения, включая пользовательский интерфейс, бизнес-логику и работу с базой данных, интегрированы в один кодовый блок. Такой подход характеризуется простотой разработки и развертывания, а также единым тестированием и управлением зависимостями. Однако он имеет свои ограничения, включая трудности масштабирования, так как изменения в одном компоненте требуют повторной сборки и развертывания всего приложения, а также ограничения гибкости, что затрудняет адаптацию к изменяющимся требованиям [8].

С развитием технологий и увеличением сложности приложений появилась сервис-ориентированная архитектура (SOA). Этот подход позволяет разделить систему на независимые сервисы, которые взаимодействуют через стандартизированные интерфейсы, такие как SOAP или REST. Сервис-ориентированная архитектура предлагает более высокую модульность по сравнению с монолитной, а также упрощённую интеграцию с другими системами. Однако она имеет сложности в управлении зависимостями и оркестрацией, что делает её реализацию более сложной [9].

Микросервисная архитектура стала дальнейшим развитием идей SOA. В этом подходе каждый сервис является полностью независимым и выполняет одну конкретную функцию, взаимодействуя с другими сервисами через лёгкие протоколы, такие как HTTP или gRPC. Микросервисы обеспечивают высокую масштабируемость, так как отдельные сервисы можно масштабировать независимо. Кроме того, проблемы в одном сервисе обычно не влияют на работу других, что повышает устойчивость системы. Гибкость

архитектуры позволяет использовать различные технологии для каждого сервиса. Однако микросервисы увеличивают сложность разработки и эксплуатации и требуют использования продвинутых инструментов для управления распределённой системой.

Серверлесс-архитектура, популярность которой возросла благодаря облачным платформам, таким как AWS Lambda, Azure Functions и Google Cloud Functions, предоставляет другой подход. Разработчики сосредотачиваются на бизнес-логике, а управление серверами и инфраструктурой берёт на себя провайдер. Серверлесс-архитектура снижает операционные затраты, обеспечивает автоматическое масштабирование и ускоряет развертывание. Однако она имеет ограничения, включая зависимость от конкретного провайдера и ограничения по времени выполнения функций.

На сегодняшний день наиболее распространённым и эффективным подходом является микросервисная архитектура. Она обеспечивает высокую гибкость и масштабируемость, хотя и требует квалифицированной команды и хорошо продуманной стратегии. Серверлесс-архитектура подходит для небольших проектов или стартапов, где важна простота и снижение затрат на инфраструктуру. Выбор архитектуры зависит от требований конкретного проекта. Монолитные решения всё ещё применяются для небольших приложений, где простота разработки является приоритетом. В условиях высокой нагрузки и необходимости быстрой адаптации к изменениям предпочтение отдаётся микросервисам или гибридным подходам, сочетающим лучшие стороны нескольких архитектур.

Требования к архитектуре фронтенда

Архитектура фронтенда играет ключевую роль в обеспечении качественного пользовательского опыта. Она должна учитывать требования к производительности, удобству разработки, безопасности и масштабируемости, чтобы соответствовать современным стандартам веб-приложений.

1. Производительность и оптимизация

Фронтенд-приложения должны быть максимально быстрыми, с минимальной задержкой при загрузке. Это достигается за счёт оптимизации кода, использования CDN для доставки статического контента, а также внедрения современных технологий, таких как lazy loading и предварительная загрузка данных. Эффективное управление ресурсами, например, кэширование и минимизация HTTP-запросов, также являются важными аспектами.

2. Масштабируемость

Современные фронтенд-приложения должны быть легко масштабируемыми. Это означает возможность добавления новых функций без значительных изменений в существующей архитектуре. Применение модульных подходов, таких как использование компонентов в React или Vue.js, а также организация кода в соответствии с принципами разделения обязанностей, помогают достичь этой цели.

3. Удобство разработки

Простота и удобство разработки обеспечиваются использованием современных инструментов и фреймворков. Например, использование TypeScript позволяет упростить работу с типизацией, а системы управления состоянием, такие как Redux или Zustand, помогают организовать взаимодействие между компонентами. Интеграция с системами сборки, такими как Webpack или Vite, способствует ускорению процесса разработки и тестирования.

4. Безопасность

Безопасность фронтенд-приложений требует внимания к защите от атак, таких как XSS и CSRF. Использование Content Security Policy (CSP), проверка данных, передаваемых пользователем, и минимизация доверия к сторонним библиотекам помогают снизить риски. Важным аспектом является также управление доступом и аутентификацией, что может быть реализовано через OAuth или JWT.

5. Кроссплатформенность и адаптивность

Фронтенд должен обеспечивать корректное отображение и функциональность на различных устройствах и платформах. Это достигается через адаптивный дизайн с использованием CSS-фреймворков, таких как Tailwind CSS или Bootstrap, а также тестирование на множестве устройств. Кроссбраузерная совместимость также является важным требованием, которое может быть обеспечено через инструментальные библиотеки, такие как Babel.

6. Поддержка и масштабируемость команды

Фронтенд-архитектура должна быть понятной и доступной для новой команды разработчиков. Это достигается за счёт использования стандартных соглашений, написания документации и внедрения CI/CD-процессов. Автоматизация тестирования с помощью Jest или Cypress помогает поддерживать качество кода.

Типы архитектуры фронтенда и их подходы к решению задач

Архитектура фронтенда может быть организована различными способами, включая плоскую, модульную и архитектуру на основе Feature-Sliced Design (FSD). Каждая из них предназначена для решения задач производительности, масштабируемости, удобства разработки и поддержки проекта.

Плоская архитектура

Плоская архитектура предполагает минимальную организацию структуры проекта, где все файлы находятся на одном уровне или разделены только по типу (например, компоненты, стили, скрипты). Такой подход часто используется в небольших проектах или на начальных этапах разработки.

Этот подход обеспечивает простоту и быструю скорость начальной разработки, так как не требует сложной структуры. Однако с увеличением размера проекта плоская архитектура

становится трудной для поддержки. Отсутствие чёткой структуры затрудняет поиск и изменение компонентов, что делает масштабирование проекта сложной задачей.

Модульная архитектура

Модульная архитектура разделяет проект на отдельные модули, каждый из которых отвечает за конкретную функциональность. Модули могут быть организованы по страницам, функциональным блокам или другим логическим группам. Например, каждая страница приложения может быть отдельным модулем, содержащим свои компоненты, стили и тесты.

Модульный подход решает задачу масштабируемости и удобства разработки. Разделение проекта на модули упрощает добавление новых функций и поддержку существующих. Каждый модуль можно разрабатывать и тестировать независимо, что сокращает риски ошибок при внесении изменений. Однако модульная архитектура может усложнить начальный этап разработки из-за необходимости продумывания структуры и связей между модулями.

Feature-Sliced Design (FSD)

FSD представляет собой современный подход к организации фронтенда, где проект структурируется по функциональным областям или фичам. Каждая фича включает в себя все необходимые элементы: компоненты, стили, состояния и логику. FSD также предполагает разделение на уровни (например, entities, features, shared), что помогает отделить высокоуровневую бизнес-логику от инфраструктурных компонентов.

Feature-Sliced Design решает задачи масштабируемости и удобства поддержки больших проектов. Чёткое разделение на фичи позволяет командам работать независимо друг от друга, минимизируя конфликты. Кроме того, FSD делает проект более гибким и адаптивным к изменениям. Однако внедрение этой архитектуры требует тщательного планирования и знаний, что может усложнить процесс для начинающих команд.

Таким образом, плоская архитектура подойдёт для небольших проектов с ограниченными ресурсами. Модульная архитектура является оптимальным выбором для средних по масштабу приложений. Для крупных проектов с высокой степенью сложности и требованием к независимой разработке отдельных частей FSD становится предпочтительным решением, обеспечивая баланс между гибкостью, масштабируемостью и удобством поддержки.

Заключение

В этой работе был проведён анализ требований к архитектуре серверных и клиентских веб-приложений. Рассмотренные подходы включают архитектурные решения, применимые как для бэкенда, так и для фронтенда, с акцентом на их соответствие современным требованиям производительности, масштабируемости, безопасности и удобства поддержки.

Среди серверных архитектур наиболее популярными на сегодняшний день являются микросервисная и серверлесс-архитектуры. Микросервисы позволяют создавать

масштабируемые и гибкие системы за счёт независимого развертывания и масштабирования отдельных компонентов. Серверлесс-архитектура, в свою очередь, минимизирует затраты на инфраструктуру и автоматизирует управление ресурсами, делая её идеальным выбором для небольших проектов или стартапов.

Для фронтенда наиболее значимыми являются модульная структура и Feature-Sliced Design (FSD). Модульная архитектура помогает упростить разработку и поддержку приложений среднего размера за счёт логического разделения кода. Feature-Sliced Design оптимизирует работу с большими и сложными проектами, предоставляя чёткое разделение функциональности и упрощая взаимодействие между командами.

Анализ показал, что выбор архитектурного подхода зависит от множества факторов, включая размер проекта, требования к функциональности и распределение обязанностей внутри команды. Современные архитектурные решения помогают эффективно справляться с задачами различной сложности, обеспечивая устойчивость и гибкость веб-приложений в условиях постоянно растущих нагрузок и меняющихся требований.

Таким образом, проведённое исследование позволяет выделить ключевые тенденции и преимущества современных архитектурных решений, что делает их незаменимыми инструментами для успешного проектирования и реализации веб-приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Web Application Architecture: The Latest Guide 2025 // ClickIT URL: https://www.clickittech.com/devops/web-application-architecture/ (дата обращения: 26.12.24).
2. Web Application Architecture: Choosing the Best for Your Product // MobiDev URL: https://mobidev.biz/blog/web-application-architecture-types (дата обращения: 26.12.24).
3. Clean Architecture: A Craftsman's Guide to Software Structure and Design // Robert C. Martin. URL: https://www.pearson.com (дата обращения: 26.12.24).
4. Microservices Patterns: With examples in Java // Chris Richardson. URL: https://www.microservices.io (дата обращения: 26.12.24).
5. OWASP API Security Top 10 // OWASP Foundation. URL: https://owasp.org (дата обращения: 26.12.24).
6. Distributed Systems Observability // Cindy Sridharan. URL: https://www.oreilly.com (дата обращения: 26.12.24).
7. Building Microservices: Designing Fine-Grained Systems // Sam Newman. URL: https://www.oreilly.com/library/view/building-microservices/9781491950340/ (дата обращения: 26.12.24).
8. Monolithic vs Microservices Architecture // IBM Cloud Education. URL: https://www.ibm.com/cloud/learn/monolithic-vs-microservices (дата обращения: 26.12.24).