Программные системы и вычислительные методы
Правильная ссылка на статью:
Федулов А.А. Проектирование архитектуры протокола клиент-серверного взаимодействия web-приложений на основе websocket // Программные системы и вычислительные методы. 2025. № 3. DOI: 10.7256/2454-0714.2025.3.75392 EDN: HGIBMU URL: https://nbpublish.com/library_read_article.php?id=75392

Проектирование архитектуры протокола клиент-серверного взаимодействия web-приложений на основе websocket

Федулов Алексей Алексеевич

ORCID: 0009-0003-8082-4316

аспирант; Федеральное государственное бюджетное образовательное учреждение высшего образования Национальный исследовательский университет «МЭИ»; Национальный исследовательский университет "МЭИ"

111250, Россия, г. Москва, р-н Лефортово, ул. Красноказарменная, д. 14 стр. 1

afed2000@yandex.ru

Статья из рубрики "Сетевые протоколы"

DOI:

10.7256/2454-0714.2025.3.75392

EDN:

HGIBMU

Дата направления статьи в редакцию:

30-07-2025

Дата публикации:

06-08-2025

Аннотация: Предметом исследования является проектирование архитектуры протокола клиент–серверного взаимодействия web-приложений на основе технологии WebSocket. Объектом исследования выступают механизмы двустороннего обмена данными в реальном времени и существующие решения на базе HTTP/HTTPS, SSE, SignalR, gRPC-Web, Socket.IO и WS. Автор подробно рассматривает такие аспекты темы, как формализация и унификация структуры логических каналов, маршрутизация сообщений, контроль активности соединений и автоматическое восстановление сессий. Особое внимание уделяется анализу преимуществ и ограничений каждого подхода с целью выработки требований к новой архитектуре протокола. В рамках работы выполнен

сравнительный анализ существующих библиотек и технологий, что позволило выявить ключевые параметры для эффективной, масштабируемой и отказоустойчивой реализации двустороннего взаимодействия. В качестве метода проектирования использован шаблон MVCS (Модель–Представление–Контроллер–Сервис), дополненный модульной организацией библиотеки SingleSocket и аналитическим сравнением существующих технологических решений в области клиент–серверного взаимодействия. Основными выводами проведенного исследования являются разработка и реализация архитектуры протокола, обеспечивающей сбалансированный и технически обоснованный подход к двустороннему обмену данными в режиме реального времени. Особым вкладом автора в исследование темы является формализация декларативной структуры каналов с привязкой к контроллерам, внедрение конфигурируемого механизма контроля активности соединений на сервере с автоматическим завершением неактивных сессий и встроенная логика восстановления соединения на клиенте. Новизна исследования заключается в применении архитектурной модели MVCS для повышения структурированности, использовании универсального JSON-формата с маршрутизацией по каналам и модульной реализации компонентов для повторного использования и упрощенной интеграции. Предложенная архитектура служит прочной основой для создания масштабируемых, надежных и гибких современных информационных систем.

Ключевые слова:

проектирование архитектуры протокола, технология WebSocket, двусторонняя коммуникация, реальное время, маршрутизация сетевых сообщений, управление активностью соединений, механизмы восстановления сессий, MVCS-архитектура, клиент-серверное взаимодействие, модульная архитектура

Введение

Современные web-приложения требуют обеспечения эффективного обмена данными в режиме реального времени для поддержания высокой интерактивности и быстрого отклика пользовательского интерфейса. Традиционные протоколы взаимодействия на основе HTTP, несмотря на широкое распространение, обладают ограничениями, связанными с высокой задержкой и невозможностью полноценной двусторонней коммуникации. В этой связи протокол WebSocket, обеспечивающий установление постоянного двунаправленного канала связи между клиентом и сервером, становится все более востребованным решением [1, с. 134]. В исследовании, проведённом Murley и соавторами в 2021 году, был выполнен масштабный анализ более 71000 WebSocket-соединений, в ходе которого было обнаружено использование протокола WebSocket на 55805 веб-сайтах, что составляет 6,3% от общего числа проанализированных ресурсов. Это представляет собой значительный рост по сравнению с исследованием 2018 года, в котором доля сайтов с использованием WebSocket составляла лишь 1,6–2,5%. Полученные данные подтверждают устойчивую тенденцию к увеличению распространённости WebSocket как базовой технологии для двунаправленного обмена данными в реальном времени [2, с. 1195].

Целью настоящего исследования является разработка архитектуры протокола клиент-серверного взаимодействия на основе технологии WebSocket, ориентированной на применение данного протокола в web-приложениях не только в рамках задач обмена сообщениями, но и в более широком спектре прикладных сценариев. Кроме того,

исследование направлено на формализацию и стандартизацию подходов к проектированию приложений и их архитектур с учётом специфики WebSocket, поскольку в настоящее время отсутствуют чётко определённые методологические рекомендации по разработке подобных систем. Научная новизна работы заключается в комплексном подходе к проектированию архитектуры, включающем многослойную структуру обработки сообщений, эффективные механизмы управления сессиями и устойчивые методы обработки ошибок, что в совокупности обеспечивает высокую производительность и надежность системы. Практическая значимость исследования состоит в возможности применения предложенной архитектуры для разработки современных web-приложений с требованиями к реальному времени, а также в перспективе интеграции с существующими стандартами и протоколами для повышения совместимости и безопасности.

Анализ существующих решений

Разработка эффективного протокола клиент-серверного взаимодействия в web-приложениях требует глубокого анализа существующих решений. В данном разделе рассмотрены наиболее распространённые технологии: HTTP/HTTPS, WebSocket, Server-Sent Events (SSE), SignalR, gRPC-Web и Socket.IO.

Протоколы HTTP и его защищённая версия HTTPS являются основой классического клиент-серверного взаимодействия. Они построены на парадигме запрос-ответ, где клиент инициирует запрос, а сервер возвращает ответ. Преимуществами являются широкая поддержка браузерами и инструментами, простота реализации и надёжность. Однако основным недостатком является отсутствие постоянного соединения и невозможность сервера инициировать обмен данными без предварительного запроса клиента [3, с. 2-4].

WebSocket – это протокол, позволяющий установить постоянное двустороннее соединение между клиентом и сервером по единому TCP-соединению. После начального рукопожатия по HTTP происходит переключение на WebSocket, что обеспечивает низкую задержку, экономию ресурсов на повторных соединениях и поддержку обмена данными в реальном времени [4, с. 41]. WebSocket требует более сложной реализации и обработки состояния соединения, но в современных условиях это оправдано преимуществами производительности и гибкости.

SSE – это технология одностороннего соединения от сервера к клиенту на основе HTTP. SSE обеспечивает стабильную и надёжную доставку событий клиенту в режиме реального времени. Она поддерживается большинством современных браузеров и использует простой текстовый формат данных. Однако SSE не поддерживает двустороннюю связь: клиент может лишь инициировать HTTP-запросы, а сервер — только отправлять события. Также протокол работает исключительно по HTTP, без поддержки бинарных данных и ограниченной поддержки масштабируемости через прокси и балансировщики [5, с. 7-8].

SignalR – библиотека от Microsoft, предоставляющая абстракцию над различными технологиями связи: WebSocket, SSE и Long Polling. Она автоматически выбирает оптимальный способ коммуникации между клиентом и сервером, обеспечивая поддержку в режиме реального времени [1, с. 130-131]. SignalR удобно использовать в .NET-приложениях, поскольку она глубоко интегрирована в ASP.NET Core. Среди недостатков – зависимость от экосистемы Microsoft и возможные избыточные накладные расходы при использовании в лёгких или высоконагруженных системах, где необходим полный контроль над соединениями и форматами сообщений.

gRPC-Web – это адаптация протокола gRPC (построенного на HTTP/2 и Protocol Buffers) для использования в браузере. Он поддерживает асинхронные вызовы и бинарный формат передачи данных, обеспечивая высокую производительность и строгое определение интерфейсов. Однако в браузерной среде gRPC-Web ограничен – отсутствует полноценная поддержка двустороннего соединения, как в оригинальном gRPC, и требуется наличие gRPC-прокси[6, с. 572]. gRPC-Web подходит для корпоративных приложений и микросервисных архитектур, но менее удобен для обмена данными в реальном времени, по сравнению с WebSocket.

Socket.IO – это библиотека, которая строится на WebSocket и добавляет дополнительные функции: автоматическое переключение на другие виды соединений, обнаружение потери связи, подключения с разными именами и комнаты. Socket.IO популярен в экосистеме Node.js и используется во множестве приложений с обменом данных в реальном времени [7, с. 82; 8, с. 12]. Основные преимущества – простота интеграции, высокая гибкость, кроссплатформенность. Недостатки – относительно высокая сложность внутренней архитектуры, отсутствие стандартизации (не является чистым WebSocket-протоколом), зависимость от конкретной библиотеки на клиенте и сервере.

Анализ показывает, что WebSocket представляет собой оптимальную технологическую основу для проектирования протоколов двустороннего взаимодействия в web-приложениях с высокими требованиями к производительности и интерактивности. По сравнению с альтернативами, WebSocket обеспечивает прямой и стандартизированный доступ к низкоуровневой двусторонней передаче данных, позволяя разработчику реализовать собственную логику обмена без ограничения архитектурой сторонних библиотек.

Требования к архитектуре протокола

Проектирование нового протокола клиент-серверного взаимодействия – важный этап в развитии технологий веб-коммуникаций, направленный на создание эффективного механизма постоянного соединения, способного обеспечивать двусторонний обмен данными в режиме реального времени. Однако широкое распространение подобных решений сдерживается отсутствием универсального стандарта прикладного уровня для форматирования и логики обмена сообщениями, а также необходимостью значительной ручной доработки при их внедрении.

В процессе разработки нового протокола необходимо предусмотреть создание специализированных библиотек как для клиентской части, так и для серверной, что обеспечит удобство интеграции, унификацию интерфейсов и гибкость использования в различных веб-приложениях. Особое внимание следует уделить формализации и стандартизации протокола, что позволит не только обеспечить корректность и предсказуемость взаимодействия, но и создать основу для развития экосистемы приложений, использующих данный протокол. Стандартизация является критическим фактором для обеспечения совместимости, расширяемости и долгосрочной поддержки решений, основанных на новой архитектуре.

Архитектурный дизайн протокола охватывает широкий спектр вопросов, связанных не только с техническими аспектами передачи и обработки данных, но и с методологическими проблемами проектирования и разработки клиентских и серверных приложений. В частности, разработка должна учитывать требования к управлению сессиями, обработке ошибок, масштабируемости, безопасности и отказоустойчивости, что предполагает комплексный подход и полное взаимодействие между компонентами

системы. В совокупности это позволяет создать инновационную и эффективную архитектуру, способную удовлетворить растущие требования современных web-приложений и заложить фундамент для дальнейших исследований и практических внедрений.

Предлагаемая архитектура протокола

Предлагаемая архитектура протокола клиент-серверного взаимодействия представляет собой оригинальный подход к организации обмена сообщениями в web-приложениях с использованием технологии WebSocket. Основу архитектуры составляет разработанная библиотека SingleSocket, реализующая модульный, конфигурируемый протокол взаимодействия с поддержкой событийно-ориентированной модели, гибкой маршрутизацией сообщений и встроенными механизмами контроля соединения.

Модульность обеспечивается в первую очередь благодаря чёткой архитектурной сегрегации на клиентскую и серверную части, которые функционируют автономно, обеспечивая независимость в развитии, настройке и внедрении. Такое структурное разделение способствует упрощению интеграционных процессов, облегчает тестирование отдельных компонентов и повышает масштабируемость системы в целом. Помимо этого, непосредственно внутри вышеупомянутых частей библиотеки реализована композиция независимых функциональных модулей, включающих механизмы гибкой маршрутизации сообщений, контроля состояния соединения, ведения логов, а также кеширование. Каждый модуль выполняет строго ограниченную задачу, что не только повышает степень гибкости и возможности повторного использования компонентов, но и облегчает сопровождение и развитие протокола без риска нарушения целостности всей системы.

Интеграция серверной части библиотеки в приложение осуществляется путем инициализации экземпляра сервера, реализованного непосредственно в библиотеке SingleSocket, с заданной конфигурацией. В конфигурации задаётся набор логических каналов, которые будут использоваться в общении клиента и сервера, время периодической проверки состояния всех активных клиентов, необходимость логирования событий соединения. Каждый из них ассоциируется с определённым обработчиком сообщений – контроллером, который будет задаваться в самом приложении. Эти контроллеры отвечают за обработку входящих данных, обеспечивая их маршрутизацию и корректную реакцию сервера на события. Такое разделение каналов позволяет чётко разграничивать области ответственности и упрощает расширение функциональности сервера путём добавления новых каналов с собственными контроллерами. При этом архитектура серверного приложения должна строится в соответствии с шаблоном проектирования MVCS (Модель-Представление-Контроллер-Сервис), где контроллеры выступают посредниками при обработке сообщений, сервисы реализуют основную прикладную логику, модели предоставляют доступ к данным, а представления выступают каналом передачи данных [9, с. 5-7]. Такой комплексный подход к использованию библиотеки обеспечивает структурированность, масштабируемость и удобство сопровождения серверной части приложения.

Со стороны клиента взаимодействие с библиотекой организуется путем инициализации экземпляра клиента, также реализованного непосредственно в библиотеки SingleSocket с заданной конфигурацией. В конфигурации задаётся набор логических каналов, которые будут использоваться в общении клиента и сервера. Клиентские сообщения формируются в унифицированном формате JSON, включающем обязательные поля, такие как идентификатор логического канала и полезная нагрузка. Такой формат

обеспечивает прозрачную и эффективную маршрутизацию сообщений по соответствующим каналам, а также поддерживает необходимые механизмы контроля доступа.

Механизм контроля соединений реализован в виде периодической проверки состояния всех активных клиентов. Каждые n секунд каждому клиенту отправляется ping, и при отсутствии ответа соединение принудительно закрывается. Этот механизм позволяет предотвратить утечки ресурсов, вызванные обрывами соединения или потерей активности на стороне клиента. Кроме того, клиентская часть протокола содержит встроенную логику контроля соединения: при обнаружении разрыва или отсутствия подтверждения активности производится автоматическая попытка повторного подключения. Это обеспечивает устойчивость к нестабильным сетевым условиям и повышает надёжность функционирования системы. На рисунке 1 представлена схема архитектуры приложения, использующего библиотеку для реализации клиент-серверного взаимодействия.

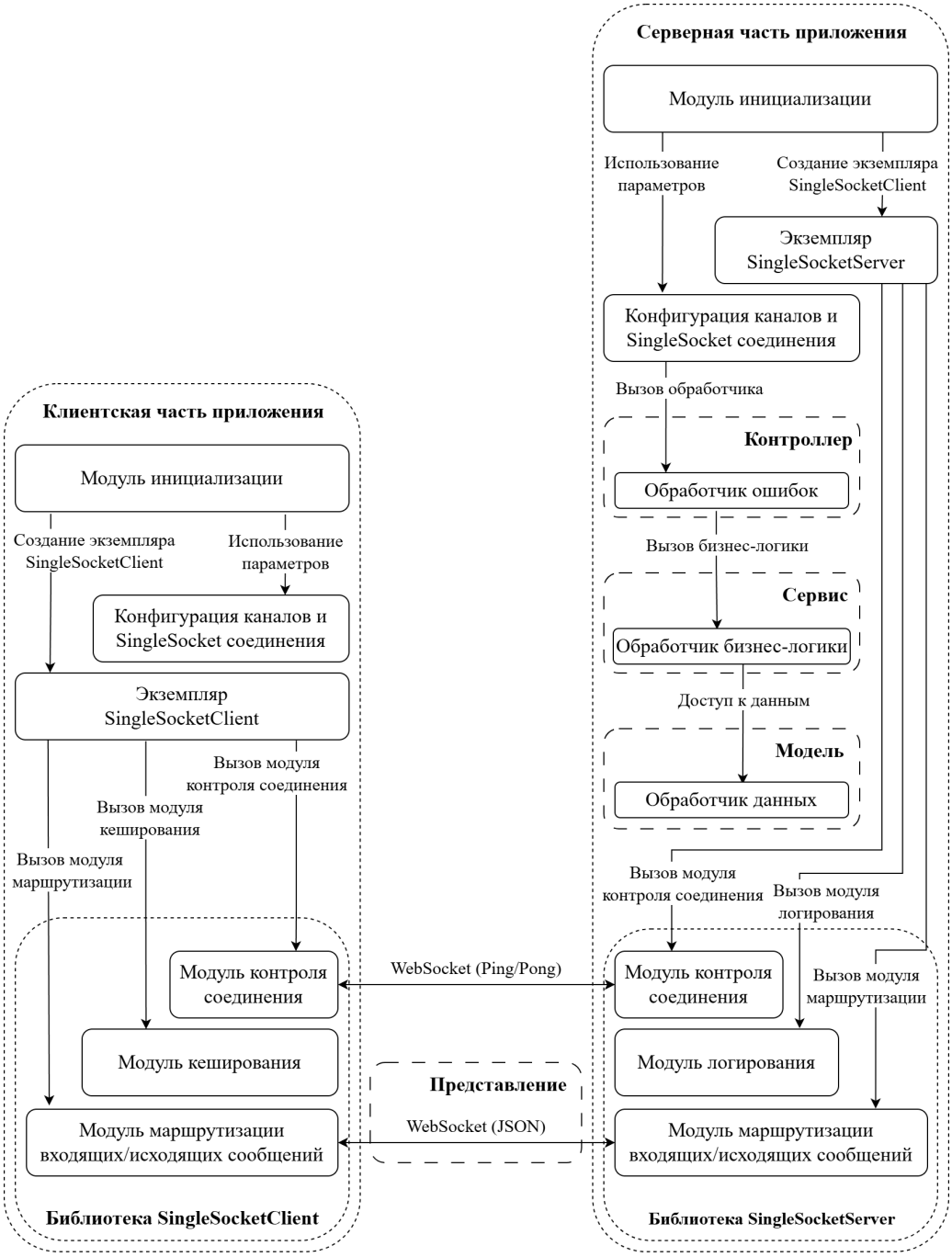

Рисунок 1 – схема архитектуры библиотеки и приложения, реализующих клиент-серверное взаимодействие.

Научная новизна предложенной архитектуры заключается в следующем:

1. формализация и унификация конфигурации протокола обмена в виде декларативной структуры каналов с привязкой к контроллерам;

2 . конфигурируемый и технически реализованный механизм контроля активности соединений на стороне сервера с автоматическим завершением неактивных сессий;

3. встроенная логика восстановления соединения на стороне клиента;

4 . применение архитектурной модели MVCS, повышающей структурированность и масштабируемость серверной логики;

5. использование универсального формата JSON с маршрутизацией по каналам;

6 . модульная реализация компонентов, обеспечивающая повторное использование и простоту интеграции в прикладные проекты.

Таким образом, предложенный протокол на базе библиотеки WebSocket представляет собой масштабируемую, отказоустойчивую и расширяемую платформу для построения web-приложений с поддержкой высокопроизводительного обмена данными в режиме реального времени. Разработанный подход ориентирован на практическое внедрение в информационные системы, которые хотят использовать протоколы нового поколения взаимодействия в системах реального времени, обеспечивая улучшенную производительность, минимизацию задержек и гибкость в адаптации к динамичным требованиям современных бизнес-процессов. Это делает его идеальным решением для организаций, стремящихся оптимизировать свои IT-архитектуры и интегрировать инновационные технологии для более эффективного взаимодействия с пользователями и внешними системами.

Сравнительный анализ

Сравнение разработанной архитектуры протокола с существующими решениями, такими как библиотеки Socket.IO и WS, позволяет выделить ключевые достоинства и слабые стороны каждого подхода. Это дает возможность объективно оценить предложенное в данной работе решение.

Библиотека Socket.IO является решением высокого уровня, надстроенным над WebSocket. Она использует событийно-ориентированную модель и предоставляет широкий набор дополнительных механизмов: автоматическое восстановление соединений, альтернативные каналы связи на случай недоступности WebSocket, возможность разделения клиентов по комнатам и поддержку кластерных решений [7, с. 82; 8, с. 12]. Эти возможности значительно упрощают разработку, особенно в приложениях с высокой интерактивностью. Однако избыточный уровень абстракции приводит к дополнительным накладным расходам и потенциальному увеличению задержек передачи сообщений, что может негативно сказаться на производительности.

WS – это малозатратная по ресурсам библиотека, реализующая базовые функции работы с WebSocket-соединением. Она ориентирована на прямую работу с протоколом, что обеспечивает высокую скорость и предсказуемость. В то же время отсутствие встроенных инструментов (для управления сессиями, обработки ошибок, повторного подключения и

пр.) требует значительных усилий со стороны разработчика и увеличивает риск ошибок при реализации логики приложения.

Предлагаемый протокол стремится к балансу между этими крайностями. Он предоставляет встроенные средства управления сессиями, маршрутизации сообщений, восстановления соединений и обработки ошибок, оставаясь при этом лёгким по архитектуре. Это обеспечивает высокую производительность, надёжность и удобство внедрения, минимизируя зависимость от внешней инфраструктуры и избегая излишней сложности реализации. В таблице 1 приведена сравнительная характеристика решений, использующих WebSocket.

Таблица 1 – сравнительные характеристики решений на основе WebSocket

Критерий	**WebSocket**	**Socket.IO**	**Предлагаемый протокол**
Тип связи	Двунаправленная	Двунаправленная	Двунаправленная
Постоянное соединение	Да	Да	Да
Мультиплексирование	Есть	Есть	Есть
Восстановление соединения	Нет	Автоматическое	Автоматическое
Управление сессиями	Не предусмотрено	Встроено	Встроено
Обработка ошибок	Базовая	Расширенная	Расширенная
Совместимость с другими протоколами	Высокая	Ограниченная	Высокая
Накладные расходы	Низкие	Повышенные	Оптимизированные
Гибкость настройки	Высокая	Ограничена	Высокая
Зависимость от инфраструктуры	Минимальная	Значительная	Средняя

Несмотря на преимущества, разработанная архитектура имеет ряд ограничений, которые необходимо учитывать при её использовании:

1 . Ограниченная совместимость с устаревшими браузерами и прокси-серверами. Некоторые корпоративные сети могут блокировать WebSocket-трафик или не поддерживать постоянные соединения, что затрудняет использование протокола в изолированных средах [10, с. 176]. Это связано с тем, что предлагаемый протокол – относительно новый и использует современные принципы связи, которые ещё не везде получили широкую поддержку.

2 . Зависимость от структуры MVCS. Архитектура сервера требует строго соблюдения разделения на контроллеры, сервисы и модели. Это облегчает работу с кодом, но может ограничивать гибкость в проектах с нестандартными архитектурными решениями. Однако стоит учитывать, что нестандартные архитектуры по своей природе предполагают нестандартные инструменты, поэтому подобная зависимость от структурной модели вполне обоснована и ожидаема.

3 . Увеличение объема служебной логики на клиенте. Несмотря на автоматическое восстановление соединений и маршрутизацию, клиентская часть становится значительно

более сложной по сравнению с традиционными REST-приложениями, что может потребовать дополнительных усилий по отладке. Однако подобная нагрузка на клиентскую сторону является общей чертой всех протоколов, использующих WebSocket.

4 . Риски при масштабировании соединений. При большом числе одновременно подключённых клиентов могут возникнуть проблемы с эффективной реализацией контроля соединений и мониторинга активности. Такие нагрузки характерны для всех современных решений, работающих в режиме постоянной связи. В предлагаемом протоколе уже предусмотрены базовые механизмы для работы с большим числом клиентов, но при дальнейшем масштабировании может потребоваться дополнительная настройка.

Таким образом, предлагаемая архитектура представляет собой компромисс между лёгкостью низкоуровневых решений и удобством сложных библиотек. Она обеспечивает высокий уровень надёжности и расширяемости при умеренных требованиях к ресурсам, однако требует осознанного подхода к проектированию и внедрению, особенно при построении масштабируемых распределённых систем. Своевременное выявление и устранение ограничений позволит расширить область применения протокола и повысить его универсальность.

Заключение

В ходе проведённого исследования рассмотрены современные подходы к организации клиент-серверного взаимодействия в web-приложениях, выявлены их преимущества и ограничения, а также обоснована необходимость разработки нового протокола, способного обеспечить постоянное двунаправленное соединение с высокой производительностью и масштабируемостью. Предложенная архитектура протокола сочетает в себе лучшие черты существующих решений, что позволяет создавать надёжные и отказоустойчивые системы.

Основные результаты работы заключаются в разработке концепции комплексной архитектуры, включающей клиентскую и серверную библиотеки, обеспечивающих обмен данными в режиме реального времени. Предложенное решение ориентировано на современные требования web-приложений и учитывает ограничения среды выполнения, что повышает его практическую значимость и перспективы внедрения.

Тем не менее, отдельные аспекты реализации подобного рода протоколов сопряжены с определёнными трудностями. Однако современный технологический прогресс и успешные примеры использования протоколов постоянного соединения в промышленных масштабах – например, в мессенджере Telegram.

В дальнейшем планируется реализация прототипа протокола, проведение интеграционных тестов и моделирование нагрузки для оценки производительности и устойчивости. Перспективным направлением развития является расширение функциональности, включая поддержку шифрования и качества обслуживания. Таким образом, разработка и внедрение данного протокола станет значительным шагом в эволюции клиент-серверного взаимодействия и откроет новые возможности для создания интерактивных и масштабируемых web-приложений.

Библиография

1. Sharma N., Agarwal R. HTTP, WebSocket, and SignalR: A Comparison of Real-Time Online Communication Protocols // Mining Intelligence and Knowledge Exploration: 9th International Conference, MIKE 2023, Proceedings. – Cham: Springer, 2023. – С. 128-135.

2. Murley P., Ma Z., Mason J., Bailey M., Kharraz A. Websocket adoption and the landscape of the real-time web // WWW '21: Proceedings of the Web Conference 2021. – New York: ACM, 2021. – С. 1192–1203.
3. Песошина Н.Т., Нуриев М.Г., Минязев Р.Ш. Разработка корпоративного веб-чата с использованием библиотеки SignalR // Международный научно-исследовательский журнал. – 2024. – № 11. – С. 1-15.
4. Kaminski L., Kozlowski M., Sporysz D., Wolska K., Zaniewski P., Roszczyk R. Comparative Review of Selected Internet Communication Protocols // Foundations of Computing and Decision Sciences. – 2023. – № 48(1). – С. 39-56.
5. Morchid A., Et-taibi B., Oughannou Z., Alami R.E., Qjidaa H., Jamil M.O., Boufounas E., Riduan M. Abid IoT-enabled smart agriculture for improving water management: A smart irrigation control using embedded systems and Server-Sent Events // Scientific African. – 2025. – № 27. – С. 1-17.
6. Price M.J. Apps and Services with .NET. – 2-е изд. – Birmingham: Packt Publishing, 2023. – 765 с.
7. Karam S.J., Abdulrahman B.F. Using Socket.io Approach for Many-to-Many Bi-Directional Video Conferencing // Al-Rafidain Journal of Computer Sciences and Mathematics. – 2021. – № 16(1). – С. 81-86.
8. Алпатов А.Н., Юров И.И. Алгоритм и программная реализация совместного редактирования графических схем в режиме реального времени с использованием библиотеки Socket.IO // Программные системы и вычислительные методы. 2024. № 1. С. 10-19.
9. Лясковский В.Л., Федулов А.А. Анализ применимости и потенциальной полезности использования шаблона проектирования модель-представление-контроллер-сервис при разработке автоматизированных систем сбора и обработки цифровых отчётных документов для промышленных предприятий // ИТ-Стандарт. – 2024. – № 2(39). – С. 4-13.
10. Василевский С.М., Шедльбауэр А.А., Морозов Д.А. Веб-приложения реального времени // Сборник материалов VII-й Всероссийской научно-практической конференции. – Курск: Юго-Западный государственный университет, 2023. – С. 174-178.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Представленная статья на тему «Проектирование архитектуры протокола клиент-серверного взаимодействия web-приложений на основе websocket» соответствует тематике журнала «Программные системы и вычислительные методы» и посвящена вопросу разработки архитектуры протокола клиент-серверного взаимодействия на основе технологии WebSocket, ориентированной на применение данного протокола в web-приложениях не только в рамках задач обмена сообщениями, но и в более широком спектре прикладных сценариев. Авторское исследование направлено на формализацию и стандартизацию подходов к проектированию приложений и их архитектур с учётом специфики WebSocket, поскольку в настоящее время отсутствуют чётко определённые методологические рекомендации по разработке подобных систем.
Авторами также сформулирована научная новизна работы, которая заключается в комплексном подходе к проектированию архитектуры, включающем многослойную структуру обработки сообщений, эффективные механизмы управления сессиями и устойчивые методы обработки ошибок, что в совокупности обеспечивает высокую

производительность и надежность системы.

Практическая значимость исследования четко сформулирована и заключается в возможности применения предложенной архитектуры для разработки современных web-приложений с требованиями к реальному времени, а также в перспективе интеграции с существующими стандартами и протоколами для повышения совместимости и безопасности.

В статье представлен анализ литературных российских и зарубежных источников по теме исследования. Список литературы включает десять источников, на все источники в тексте даны ссылки.

Авторами проведен анализ существующих решений для разработки эффективного протокола клиент-серверного взаимодействия в web-приложениях. В работе рассмотрены наиболее распространённые технологии: HTTP/HTTPS, WebSocket, Server-Sent Events (SSE), SignalR, gRPC-Web и Socket.IO.

Авторами проведен сравнительный анализ характеристик решений на основе WebSocket, результаты представлены в таблице.

Стиль и язык изложения материала является научным, материал изложен логично. Статья по объему соответствует рекомендуемому объему от 12 000 знаков. Статья достаточно структурирована – в наличии введение, заключение, внутреннее членение основной части (Анализ существующих решений, Требования к архитектуре протокола, Предлагаемая архитектура протокола, Сравнительный анализ).

В заключении авторы излагают, что предложенный ими протокол на базе библиотеки WebSocket представляет собой масштабируемую, отказоустойчивую и расширяемую платформу для построения web-приложений с поддержкой высокопроизводительного обмена данными в режиме реального времени. Разработанный подход ориентирован на практическое внедрение в информационные системы, которые хотят использовать протоколы нового поколения взаимодействия в системах реального времени, обеспечивая улучшенную производительность, минимизацию задержек и гибкость в адаптации к динамичным требованиям современных бизнес-процессов.

Статья «Проектирование архитектуры протокола клиент-серверного взаимодействия web-приложений на основе websocket» может быть рекомендована к публикации в журнале «Программные системы и вычислительные методы».