ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ/INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/IRJ.2024.149.126

РАЗРАБОТКА КОРПОРАТИВНОГО ВЕБ-ЧАТА С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ SIGNALR

Научная статья

Песошина Н.Т.[1,]*, Нуриев М.Г.[2], Минязев Р.Ш.[3]
[2] ORCID : 0009-0003-0741-1734;
[1, 3] Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, Казань, Российская Федерация
[2] Казанский национальный исследовательский технический университет им. А. Н. Туполева – КАИ, Казань, Российская Федерация

* Корреспондирующий автор (pesoshina_nataly[at]mail.ru)

Аннотация

В статье рассматривается разработка корпоративного веб-чата с использованием библиотеки SignalR, предназначенного для обеспечения эффективной и безопасной коммуникации внутри компании. Веб-чаты стали важным инструментом в современном бизнесе, позволяя сотрудникам быстро обмениваться информацией и координировать свои действия. Приведено сравнение популярных решений на рынке, таких как «Пачка», "VK Teams" и "Express", выявлены их сильные и слабые стороны. Основное внимание уделено созданию специализированного решения, которое отвечает специфическим требованиям бизнеса, включая интеграцию с корпоративными системами и обеспечение высокого уровня безопасности данных. В статье подробно описаны архитектура разработанного веб-чата, использование технологии SignalR, а также ключевые элементы, такие как WebSocket, Server-Sent Events и Long Polling. Представлены результаты нагрузочного тестирования, которые подтвердили стабильность и производительность приложения при высоких нагрузках. Предложенное решение обеспечивает надежную и эффективную платформу для коммуникации, адаптируемую под индивидуальные потребности компании.

Ключевые слова: корпоративный веб-чат, SignalR, коммуникация в реальном времени, WebSocket, Server-Sent Events, Long Polling, безопасность данных, нагрузочное тестирование, ASP.NET Core.

DEVELOPMENT OF CORPORATE WEB-CHAT USING SIGNALR LIBRARY

Research article

Pesoshina N.T.[1,]*, Nuriev M.G.[2], Minyazev R.S.[3]
[2] ORCID : 0009-0003-0741-1734;
[1, 3] Kazan National Research Technical University named after A.N. Tupolev – KAI, Kazan, Russian Federation
[2] Kazan National Research Technical University named after A. N. Tupolev – KAI, Kazan, Russian Federation

* Corresponding author (pesoshina_nataly[at]mail.ru)

Abstract

The article examines the development of a corporate web chat using the SignalR library, designed to ensure efficient and secure communication within the company. Web chats have become an important tool in modern business, allowing employees to quickly exchange information and coordinate their actions. A comparison of popular solutions on the market such as Tutu, VK Teams and Express is given, their strengths and weaknesses are identified. The main focus is on creating a specialized solution that meets specific business requirements, including integration with corporate systems and ensuring a high level of data security. The paper describes in detail the architecture of the developed web chat, the use of SignalR technology, as well as key elements such as WebSocket, Server-Sent Events and Long Polling. Load testing results are presented, which confirmed the stability and performance of the application under high loads. The proposed solution provides a reliable and efficient communication platform adaptable to individual company needs.

Keywords: corporate web chat, SignalR, real-time communication, WebSocket, Server-Sent Events, Long Polling, data security, load testing, ASP.NET Core.

Введение

В условиях современного бизнеса коммуникация играет ключевую роль в обеспечении эффективности рабочих процессов и взаимодействия между сотрудниками. В быстро меняющемся мире, где информация и скорость принятия решений имеют решающее значение, наличие надежных средств связи внутри компании становится одним из важнейших факторов, способствующих успеху и конкурентоспособности организации.

Одним из таких инструментов являются корпоративные веб-чаты, которые позволяют сотрудникам мгновенно обмениваться сообщениями, файлами и другими данными, что значительно ускоряет процессы координации и принятия решений. Эти чаты обеспечивают постоянное взаимодействие между членами команды, даже если они находятся на значительном расстоянии друг от друга. Однако стандартные общедоступные решения не всегда могут удовлетворить специфические потребности бизнеса, такие как интеграция с другими корпоративными системами, настройка под конкретные задачи компании и обеспечение высокого уровня безопасности данных [1], [2].

В современном мире корпоративные веб-чаты становятся неотъемлемой частью работы любой организации. Они обеспечивают эффективное взаимодействие между сотрудниками, способствуют улучшению координации и

повышению производительности. Существуют различные решения на рынке, каждое из которых имеет свои особенности, преимущества и недостатки.

Рассмотрим три популярных аналога корпоративных веб-чатов: «Пачка», "VK Teams" и "Express".

1. «Пачка» – это российский корпоративный мессенджер, который зарекомендовал себя как удобный и надежный инструмент для командной работы. Он поддерживается на всех популярных операционных системах и предлагает интуитивно понятный интерфейс, что делает его доступным для пользователей с разным уровнем подготовки. Основные функции «Пачка» включают создание чатов и каналов, отправку сообщений и файлов, а также организацию групповых звонков.

Одним из ключевых преимуществ «Пачка» является возможность интеграции с различными бизнес-приложениями через API. Это делает мессенджер гибким и пригодным для использования как в малых, так и в крупных компаниях. Кроме того, серверы «Пачка» расположены на территории Российской Федерации, что гарантирует высокий уровень безопасности данных, что особенно важно для государственных и крупных корпоративных клиентов.

Тем не менее, «Пачка» имеет и свои ограничения. В бесплатной версии мессенджера ограничено количество отправляемых сообщений, а отсутствие "коробочного решения" может затруднить его использование для компаний, предпочитающих размещать свои системы на собственных серверах.

2. "VK Teams" – это продукт, разработанный компанией "VK Tech", ориентированный на корпоративный сегмент. "VK Teams" предлагает полный набор инструментов для управления проектами и общения внутри команды, включая создание рабочих групп, назначение задач, обсуждение и отслеживание их выполнения.

Среди уникальных функций "VK Teams" стоит отметить возможность частичного цитирования сообщений, проведение опросов и оставление комментариев к сообщениям. Эти функции делают общение более структурированным и удобным для всех участников процесса. "VK Teams" также поддерживает интеграцию с другими сервисами VK, что расширяет функциональные возможности мессенджера.

Однако, несмотря на широкий функционал, "VK Teams" не лишен недостатков. Интерфейс приложения может потребовать времени на адаптацию, особенно для новых пользователей, а также иногда возникают технические сбои, такие как проблемы с синхронизацией между мобильной и десктопной версиями.

3. "Express" – это платформа, ориентированная на обеспечение корпоративной коммуникации, которая была впервые представлена в 2017 году. "Express" предлагает такие функции, как создание каналов и чатов, поддержка аудио- и видеозвонков, а также возможность использования стикеров и реакций для более интерактивного общения.

Одной из уникальных особенностей "Express" является режим повышенной конфиденциальности, который включает автоматическое удаление сообщений и ограничение возможности копирования и пересылки файлов. Это делает "Express" привлекательным для компаний, уделяющих особое внимание безопасности и конфиденциальности данных.

Однако пользователи "Express"отмечают проблемы со стабильностью работы приложения, включая отсутствие уведомлений и сбои при звонках. Эти недостатки могут существенно влиять на рабочий процесс, особенно в ситуациях, требующих быстрой реакции и постоянного обмена информацией.

Сравнение трех рассмотренных мессенджеров показывает, что каждый из них имеет свои сильные и слабые стороны. «Пачка» идеально подходит для небольших команд и обеспечивает высокий уровень безопасности данных. "VK Teams" предоставляет широкий функционал для командной работы, но может требовать времени на освоение. "Express", несмотря на свои уникальные функции конфиденциальности, может столкнуться с проблемами стабильности. Выбор подходящего корпоративного веб-чата зависит от специфических требований компании, её масштаба и приоритетов в области безопасности и функциональности.

Таким образом, разработка специализированного корпоративного веб-чата с использованием современных технологий становится актуальной задачей. В данной статье рассматривается создание такого чата с использованием библиотеки SignalR, которая является мощным инструментом для реализации приложений с функцией общения в режиме реального времени [3], [4].

Цель работы заключается в разработке корпоративного веб-чата, который обеспечит удобство общения сотрудников в группах, обеспечивая при этом надежность и безопасность передачи данных. Для достижения этой цели были поставлены следующие задачи: анализ предметной области, исследование существующих решений, выбор инструментов и технологий для разработки, проектирование и реализация серверной и клиентской частей приложения, а также тестирование разработанного решения.

Теоретические основы разработки веб-чата

В современном мире, где технологии развиваются с неимоверной скоростью и всегда ставится вопрос об улучшении процессов, фундаментальное понимание принципов взаимодействия в сети Интернет становится критически важным. Один из наиболее основных и распространенных механизмов обмена информацией между клиентом и сервером – это модель «запрос-ответ». Эта концепция лежит в основе большого количества веб-технологий и является стартовой точкой для понимания более сложных и эффективных методов общения, таких как предлагаемые в библиотеке SignalR.

В этом разделе статьи будет описано, как работает модель «запрос-ответ», ее преимущества и недостатки. Это поможет лучше понять, почему в некоторых случаях может потребоваться более продвинутый подход, такой как SignalR.

Большая часть веб-приложений основана на классическом подходе запрос-ответ, известном как HTTP (HyperText Transfer Protocol) или протокол передачи гипертекста. В этом случае клиент, представленный в виде веб-браузера, отправляет запрос на сервер. Сервер, в свою очередь, обрабатывает запрос и отправляет обратно ответ. Данный цикл повторяется каждый раз, когда клиенту необходимы новые данные [5].

Рассмотрим основные этапы классического принципа взаимодействия «запрос-ответ» более подробно:

1. Инициация запроса клиентом: процесс начинается с отправки клиентом HTTP-запроса на сервер. Запрос может иметь различный характер, включая получение данных (GET), передачу данных (POST), обновление данных (PUT) или удаление данных (DELETE). Запрос включает в себя различные компоненты, такие как URL, параметры строки запроса, содержание самого запроса или заголовки.

2. Обработка запроса на сервере: сервер, получив запрос, начинает его обработку. Этот процесс может включать обращение к базе данных, выполнение определенных расчетов или взаимодействие с другими сервисами.

3. Сервер готовит и отправляет ответ: обработав запрос, сервер отправляет ответ назад клиенту. Ответ может включать в себя данные (например, HTML-страницу или JSON-объект), статус (например, успешно или ошибка), а также дополнительные сведения (например, cookies или заголовки).

4. Обработка ответа клиентом: получив ответ от сервера, клиент начинает его обработку. Это может включать обновление визуального интерфейса пользователя, сохранение данных в локальном хранилище или выполнение других действий на основе полученной информации.

Пример работы системы модели «запрос-ответ» приведен на рисунке 1.

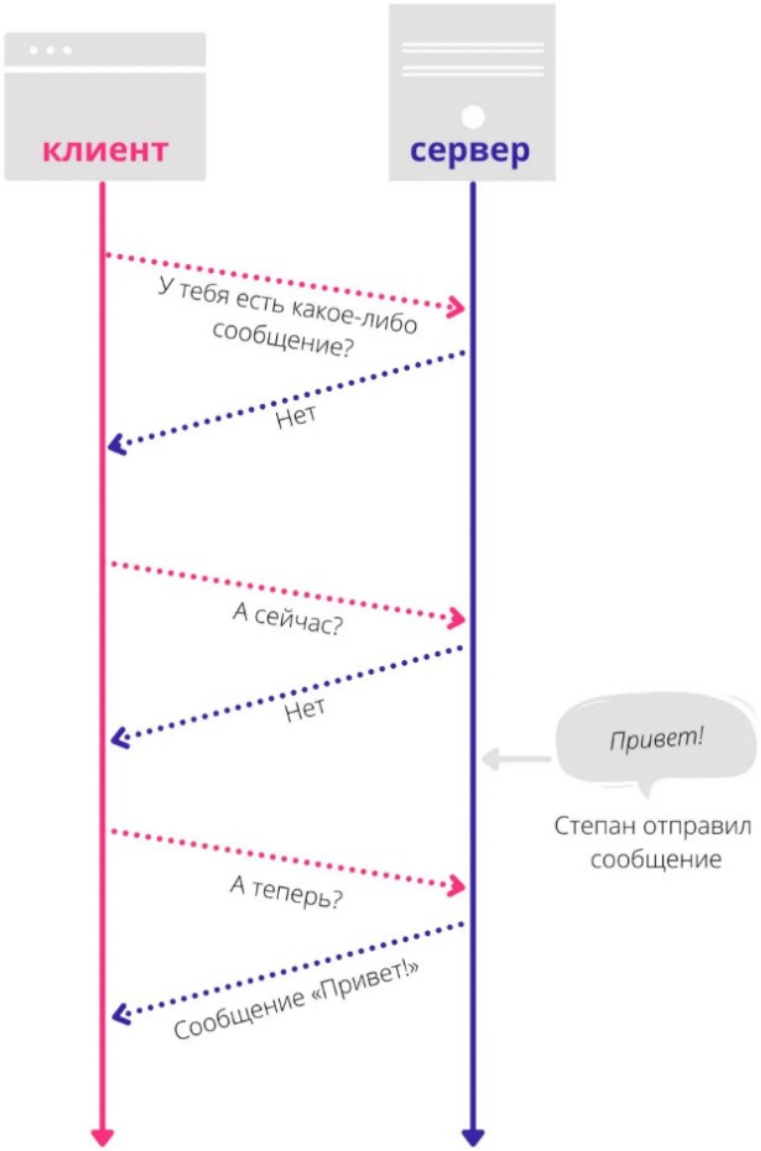

Рисунок 1 - Традиционная модель клиент-сервер
DOI: https://doi.org/10.60797/IRJ.2024.149.126.1

В проиллюстрированном примере на рисунке 1 сервер, по своей сути, находится в режиме ожидания, пока клиент не проинициализирует запрос на получения данных. Только именно в этом случае сервер отправляет обновления.

Несмотря на то, что традиционный подход запрос-ответ, основанный на протоколе HTTP, является фундаментом для большинства веб-приложений и является широко распространённым и универсальным, этот подход имеет ряд существенных недостатков, которые могут оказать негативное влияние на производительность и эффективность веб-приложений.

В качестве недостатков можно выделить следующие:

1. Низкая эффективность: каждый запрос и ответ требуют установки и разрыва соединения. Это может быть ресурсоемким, особенно в крупномасштабных веб-приложениях, где многочисленные пользователи могут одновременно отправлять запросы. Более того, каждый запрос занимает время на установку соединения, что усложняет отклик системы. Это также может привести к излишнему расходу ресурсов, поскольку каждый запрос требует создания нового соединения и его последующего закрытия после получения ответа.

2. Задержка в обновлении данных: сервер не имеет возможности самостоятельно инициировать взаимодействие с клиентом, клиент вынужден регулярно отправлять запросы на сервер с целью проверки наличия новых данных, что приводит к задержкам в обновлении информации. Это особенно критично для приложений, требующих мгновенной реакции на новую информацию, что способствует к созданию избыточного трафика, ведь клиент продолжает отправлять запросы, даже если новые данные еще не появились.

3. Нагрузка на сервер: постоянные запросы от множества клиентов могут создать значительную нагрузку на сервер. Это может снизить производительность сервера и ухудшить общее качество обслуживания.

4. Невозможность работы в реальном времени: традиционный подход запрос-ответ не поддерживает взаимодействие в реальном времени. Данные обновляются только при получении нового запроса от клиента, что может не соответствовать требованиям некоторых современных веб-приложений, где необходимо мгновенное обновление данных.

5. Ограниченная двусторонняя связь: в традиционном подходе запрос-ответ сервер не может самостоятельно связываться с клиентом. Все общение идет от клиента к серверу. Это ограничивает возможности для создания веб-приложений, требующих активного двустороннего обмена, таких как чаты или игры в реальном времени.

6. Неэффективное использование ресурсов: поскольку каждый запрос требует установки и разрыва соединения, это может привести к неэффективному использованию ресурсов, особенно при большом количестве одновременных пользователей.

Все эти недостатки делают традиционный подход запрос-ответ менее подходящим для современных веб-приложений, в которых требуются высокая производительность, мгновенное обновление данных и активная двунаправленная связь. В связи с этим, технологии Real Time Communication, такие как WebSockets и SignalR, предлагают более эффективные и отзывчивые способы взаимодействия между клиентом и сервером. Эти технологии позволяют преодолеть многие из недостатков традиционного подхода запроса-ответа, предлагая более современные и эффективные решения для веб-разработки [6], [7].

Веб-приложения, функционирующие в режиме коммуникации реального времени (RTC), представляют собой существенное отступление от традиционной модели «запрос-ответ», которая доминировала в Интернете практически с момента его появления на свет. Благодаря использованию таких передовых технологий, как WebSockets, Server-Sent Events (SSE) и Long Polling, современный приложения теперь способны поддерживать постоянно открытый канал связи между сервером и клиентом [3]. Это позволяет серверу немедленно отправлять обновления клиенту без необходимости ожидания явных запросов со стороны клиента. Этот подход к веб-разработке значительно упрощает процесс мгновенной передачи данных, в то же время создавая непрерывный и интерактивный опыт для пользователя.

Благодаря использованию технологии RTC в приложениях, таких как чаты, сообщения мгновенно становятся видны для всех пользователей, без необходимости обновления страницы. Схожая ситуация наблюдается и с инструментами для коллективной работы над документами – пользователи имеют возможность немедленно видеть, какие корректировки вносят другие участники. Это создает ощущение совместной работы и сотрудничества, усиливая чувство командной работы и эффективности.

В общем, технологии коммуникации в реальном времени предоставляют обширный набор преимуществ, что делает их идеальным решением для создания современных веб-приложений. Некоторые из этих преимуществ:

1. Мгновенная коммуникация: для приложений, которые требуют мгновенного отклика, таких как онлайн-игры, веб-чаты и видеоконференции, это особенно важно.

2. Двусторонняя связь: технологии RTC поддерживают двустороннюю связь, что означает, что данные могут передаваться одновременно в обе стороны. Это способствует улучшению интерактивности и уменьшению времени ожидания, что крайне важно в приложениях, работающих в реальном времени. Благодаря двусторонней связи, пользователи могут одновременно отправлять и получать данные, что ускоряет обмен информацией и делает взаимодействие более плавным и непрерывным.

3. Автоматическое управление соединениями: Real Time Communication автоматически управляет соединениями, обеспечивая надежную и стабильную связь. Без автоматического управления соединениями, пользователю пришлось бы вручную устанавливать и поддерживать соединение, что могло бы привести к проблемам стабильности и надежности. С RTC все эти задачи выполняются автоматически, что позволяет пользователям сосредоточиться на взаимодействии с приложением, а не на технических аспектах подключения.

4. Масштабируемость: технологии RTC легко масштабируются для обработки увеличивающегося трафика. Это делает их идеальным выбором для крупных веб-приложений и сервисов.

5. Поддержка различных типов данных: RTC поддерживает передачу различных типов данных, включая текст, аудио и видео. Это делает его идеальным выбором для мультимедийных приложений.

6. Низкая задержка: в отличие от традиционных методов передачи данных, RTC обеспечивает низкую задержку, что особенно важно для приложений, требующих быстрой обратной связи.

На основании приведенных выше преимуществ, становится ясно, что технология RTC представляет собой идеальное решение для создания современных веб-приложений, требующих мгновенной и надежной коммуникации.

SignalR – это мощная библиотека, созданная корпорацией Microsoft. Она работает на основе ASP.NET и разработана специально с целью создания приложений, работающих в режиме реального времени. В рамках экосистемы .NET Core, SignalR зарекомендовала себя как известная и надежная библиотека. Она иллюстрирует

значительные достижения в области упрощения процесса разработки веб-приложений, в которых требуется мгновенная передача данных. Одним из ключевых преимуществ SignalR является ее способность абстрагировать сложности, связанные с управлением соединениями в режиме реального времени. Это значительно облегчает задачу разработчиков, позволяя им сосредоточиться на основной функциональности приложения, в то время как SignalR берет на себя тяжелую работу по обработке соединений [4], [5].

Архитектура SignalR построена на основе клиент-серверной модели и включает несколько ключевых компонентов. Клиентская часть отвечает за подключение к серверу SignalR и за отправку и получение сообщений. Клиентом может выступать веб-браузер, мобильное приложение или любое другое приложение, поддерживающее протокол SignalR. Серверная часть управляет подключениями, маршрутизирует сообщения между клиентами и обрабатывает события подключения и отключения пользователей. Сервер SignalR выполняет функцию центрального узла для всех коммуникаций в режиме реального времени.

Концентраторы (Hubs) играют важную роль в архитектуре SignalR, представляя собой основные каналы связи между клиентами и сервером. Концентраторы абстрагируют базовый транспортный уровень и предоставляют клиентам высокоуровневый API для отправки и получения сообщений. Эти специальные классы обрабатывают взаимодействие клиент-сервера в SignalR и представляют собой точки вызова процедур, которые могут инициироваться как клиентами, так и сервером.

Транспорты отвечают за установление и поддержание соединения между клиентом и сервером. Соединение представляет собой постоянный канал связи между клиентом и сервером, что позволяет серверу отправлять сообщения определенным клиентам или транслировать их нескольким пользователям одновременно. При первоначальном подключении каждому клиенту присваивается уникальный идентификатор, который используется для его идентификации при последующем взаимодействии.

Важно также обратить внимание на одну из ее ключевых составляющих библиотеки SignalR – удаленные вызовы процедур (RPC)**.** Удаленные вызовы процедур – это концепция, которая используется в SignalR. В библиотеке используются API хабов, которые позволяют делать удаленные процедурные вызовы от сервера к подключенным клиентам и от клиентов к серверу. В коде сервера определяются методы, которые могут быть вызваны клиентами, и сервер может вызывать методы, которые выполняются на стороне клиента. Ниже на рисунке 2 приведена схема, на которой сначала сервер вызывает метод клиента, а затем клиент вызывает метод сервера.

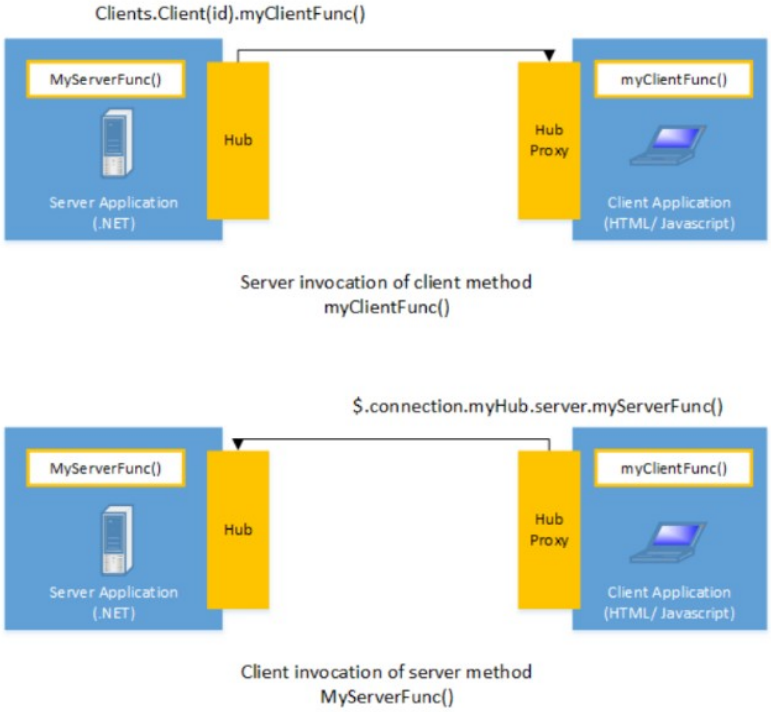

Рисунок 2 - SignalR: взаимодействие между клиентом и сервером
DOI: https://doi.org/10.60797/IRJ.2024.149.126.2

Взаимодействие между хабами и клиентами в SignalR является ключевым аспектом, обеспечивающим эффективную и надежную коммуникацию в реальном времени. Необходимо подробнее рассмотреть этот процесс. Пример простого кода взаимодействия между клиентом и сервером, где клиентский скрипт общается с серверным для обеспечения обмена сообщениями продемонстрирован на рисунок 3.

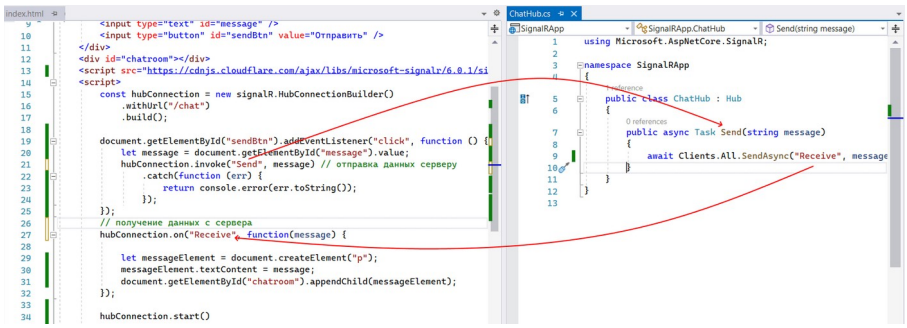

Рисунок 3 - SignalR: пример демонстрации RPC
DOI: https://doi.org/10.60797/IRJ.2024.149.126.3

На представленной иллюстрации (рис. 3) можно увидеть две ключевые части кода: серверную – правая часть рисунка, и клиентскую – левая часть. Основным элементом серверной части является класс ChatHub, в котором определен метод Send. Он выполняет функцию приема сообщения и дальнейшей его рассылки всем пользователям, которые подключены к ChatHub.

Клиенты могут активировать метод Send с помощью функции «hubConnection.invoke ("Send", message)», принимающей в качестве первого параметра имя метода сервера, которого нужно вызвать, а текст самого сообщения, являющийся входными данными метода сервера, передается как второй параметр.

В процессе выполнения метода Send, класса ChatHub, вызывается метод SendAsync. Этот метод принимает два параметра: имя метода "Receive", который должен быть реализован на клиентской стороне для получения сообщений от ChatHub, и данные, которые необходимо передать.

Для наглядного представления о взаимодействии этих методов между клиентской и серверной частью, на иллюстрации (рис. 3) приведены красные направленные линии. Эти линии показывают направление вызовов функций и передачи данных между серверной и клиентской частями кода.

Продолжая исследование технологии SignalR, необходимо обратить внимание на фундаментальные механизмы, которые обеспечивают функциональность в реальном времени – транспортные механизмы передачи данных. В основе своей работы SignalR использует различные протоколы для установления и поддержания соединений, а именно WebSockets, Server-Side Events (SSE) и Long Polling, каждый из которых имеет свои уникальные характеристики и варианты использования. Подробнее рассмотрим, как работает каждый из них и какие преимущества они предоставляют.

Первым в этом списке идет WebSocket, дуплексный протокол, который применяется в клиент-серверном канале связи и открывает новые горизонты для взаимодействия в реальном времени, обеспечивая полнодуплексное соединение между клиентом и сервером. Это означает, что данные могут передаваться в обоих направлениях одновременно, что является ключевым преимуществом для создания интерактивных веб-приложений.

Соединение, установленное с помощью WebSocket, будет сохранятся, пока его не прервет любой из участников. Данный вид соединения снижает накладные расходы, повышает эффективность и обеспечивает превосходные возможности для конечных пользователей и особенно полезен для приложений, таких как онлайн-игры, чат-приложения и платформы для торговли акциями, где важна связь с малой задержкой. Протокол WebSocket значительно повышает производительность сетевых коммуникаций, что делает его жизненно важным компонентом в современной веб-разработки. Схема соединения представлена на рисунке 4.

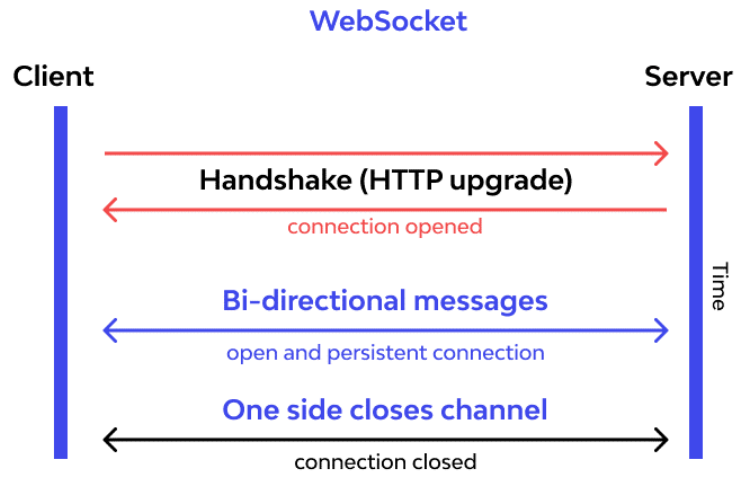

Рисунок 4 - Соединение с использованием WebSocket
DOI: https://doi.org/10.60797/IRJ.2024.149.126.4

Протокол WebSocket работает следующим образом:

1. Инициализация соединения: в начале процесса клиент отправляет HTTP-запрос на сервер с предложением переключиться на протокол WebSocket. Этот этап называется «рукопожатием».

2. Обмен данными: если сервер поддерживает протокол WebSocket, он соглашается на предложение клиента, и после успешного «рукопожатия» устанавливается соединение WebSocket. Теперь клиент и сервер могут обмениваться данными в режиме реального времени без необходимости повторного установления соединения.

3. Завершение соединения: по завершении обмена данными соединение может быть закрыто как клиентом, так и сервером.

В ходе исследования транспортных механизмов SignalR перейдем к рассмотрению следующего ключевого элемента – Server-Sent Events. Server-Sent Events или сокращенно SSE (Server-Sent Events) представляет технологию взаимодействия клиента и сервера, которая позволяет серверу отправлять сообщению клиенту. Главное отличие Server-Sent Events от WebSockets заключается в том, что коммуникация через SSE является однонаправленной т.е. сообщения, доставляются только в одном направлении – от сервера к клиенту. SSE является отличным выбором, когда нет необходимости осуществлять отправку данных от клиента к серверу. Данный вид соединения представлен на рисунке 5.

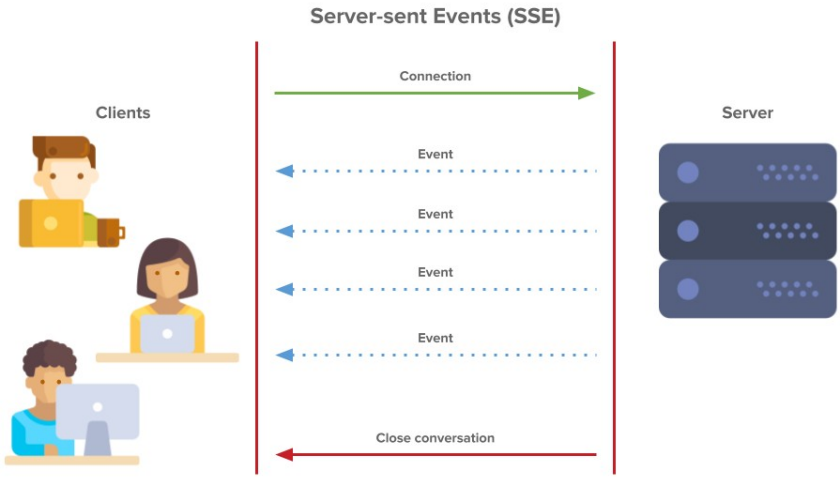

Рисунок 5 - Соединение с использованием Server-Sent Events
DOI: https://doi.org/10.60797/IRJ.2024.149.126.5

Server-Sent Events работает по следующему алгоритму:

1. Установление соединения: клиент отправляет HTTP-запрос на сервер, указывая, что он хочет получать обновления через SSE. Если сервер поддерживает SSE, он устанавливает постоянное соединение с клиентом.

2. Отправка данных: сервер может отправлять данные на клиента в любое время, используя это открытое соединение.

3. Обработка данных: клиент слушает события, отправленные сервером, и обрабатывает их соответствующим образом. Это может включать в себя обновление пользовательского интерфейса, запуск определенных функций или обработку данных.

4. Закрытие соединения: соединение может быть закрыто либо сервером, либо клиентом. Если соединение прерывается, клиент обычно пытается автоматически восстановить его.

Далее обратим внимание на протокол Long Polling. Несмотря на то, что он может казаться менее современным по сравнению с WebSocket или Server-Sent Events, Long Polling играет критически важную роль в обеспечении взаимодействия в реальном времени. Long Polling – это метод, используемый в веб-приложениях для обеспечения ответа на сообщения сервера с низкой задержкой без затрат процессора или трафика, связанных с высокочастотным опросом. Клиент делает запрос к веб-серверу, но вместо того, чтобы ответить сразу же сервер удерживает соединение, в течение потенциально длительного времени и отвечает только тогда, когда данные доступны. Клиент отреагирует на эти данные, а затем перезапустит опрос и дождётся получения данных, создавая иллюзию постоянного соединения между клиентом и сервером. Несмотря на то, что фактически соединение переустанавливается после каждого обмена данными, для пользователя это выглядит так, как будто сервер постоянно отправляет обновления. Данный вид соединения представлен на рисунке 6.

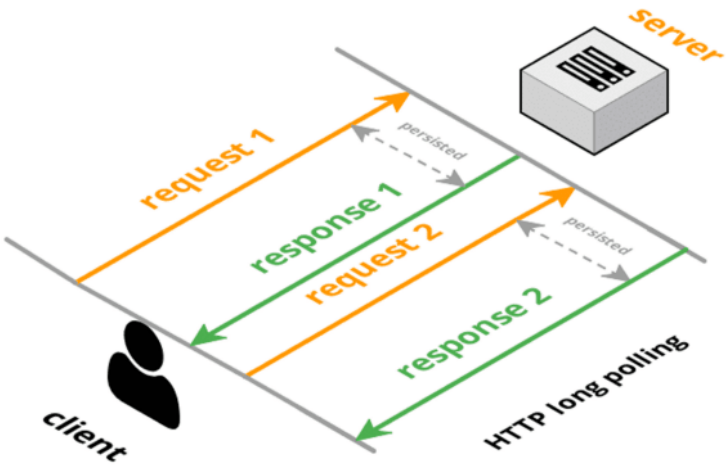

Рисунок 6 - Соединение с использованием Long Polling
DOI: https://doi.org/10.60797/IRJ.2024.149.126.6

SignalR автоматически выбирает наиболее подходящий метод передачи данных в зависимости от версии браузера, используемого веб-приложением. Это позволяет разработчикам сосредоточиться на бизнес-требованиях своих приложений, не беспокоясь о том, какой транспорт будет использоваться для отправки и получения сообщений между клиентом и сервером. Алгоритм выбора метода передачи данных, реализованный в SignalR, начинается с определения возможностей клиента и сервера, проверяя поддержку таких технологий, как WebSocket, Server-Sent Events и Long Polling. После этого SignalR выбирает наилучший доступный метод передачи данных. Если клиент и сервер поддерживают WebSocket, именно этот метод будет использован, так как он обеспечивает наиболее эффективную и с минимальными задержками передачу данных. В случае, если WebSocket недоступен, SignalR переключается на другой метод передачи данных, например, на Server-Sent Events. Если и этот метод не поддерживается, SignalR прибегает к Long Polling в качестве последнего варианта. Важно отметить, что SignalR автоматически переключается на другой метод передачи данных, если текущий метод становится недоступным во время сессии, что обеспечивает непрерывность связи. Таким образом, процесс выбора метода передачи данных в SignalR гарантирует, что приложение всегда использует наиболее эффективный доступный метод, что делает SignalR идеальным выбором для высокопроизводительных приложений, работающих в режиме реального времени. Эта гибкость позволяет SignalR динамически адаптироваться к условиям сети и возможностям клиента, обеспечивая максимальную производительность и надежность.

Как и в случае с любой технологией, связанной с передачей и обработкой данных, безопасность является важным фактором при использовании SignalR. SignalR поддерживает различные методы проверки подлинности, такие как ASP.NET Forms, проверка подлинности Windows и OAuth, которые можно использовать для ограничения доступа к центру SignalR только авторизованным пользователям. Кроме того, SignalR позволяет авторизовывать определенных клиентов и группы клиентов, что помогает гарантировать, что только авторизованные пользователи могут получать определенные сообщения. Безопасность веб-приложений также зависит от проверки входных данных, и SignalR предоставляет возможность проверять сообщения, чтобы убедиться, что они соответствуют ожидаемым критериям, предотвращая вредоносный ввод. Дополнительно, сообщения, передаваемые через SignalR, могут быть зашифрованы с использованием стандартного шифрования SSL/TLS, что обеспечивает безопасную связь между клиентом и сервером. SignalR поддерживает защиту от подделки межсайтовых запросов (CSRF) по умолчанию, что помогает предотвратить отправку вредоносных запросов на сервер. Кроме того, метод ограничения скорости может быть

использован в SignalR для предотвращения атак типа «отказ в обслуживании», что снижает риск перегрузки сервера. Использование безопасных транспортных протоколов, таких как HTTPS, также поддерживается SignalR, что обеспечивает дополнительную защиту данных, передаваемых между клиентом и сервером. Таким образом, безопасность является неотъемлемой частью использования SignalR в веб-приложениях. Реализуя аутентификацию, авторизацию, проверку данных, шифрование, защиту CSRF, ограничение скорости и безопасные транспортные протоколы, разработчики могут гарантировать, что их реализация SignalR будет надежно защищена от различных типов атак.

Программная реализация веб-чата

Предлагаемое решение реализует клиент-серверную архитектуру. Эта архитектура предполагает разделение системы на два или более логических модуля. Один из них функционирует в роли сервера, обеспечивающего услуги, в то время как другой выполняет роль клиента, заказчика услуг сервера.

Используется двухуровневая архитектура взаимодействия «клиент-сервер». В этом контексте процесс, выполняющийся на стороне клиента, отвечает за взаимодействие с пользователем. Сервер же управляет всеми ресурсами базы данных. С помощью специального разработанного набора команда также известного как API происходит циркуляция информация между клиентом и сервером. Этот набор команд определяется самим разработчиком, что позволяет обеспечить полный контроль над доступом к данным и установить допустимый список операций, которым разрешено взаимодействовать с ними. Для наглядного преставления архитектура веб-чата приведена рисунке 7.

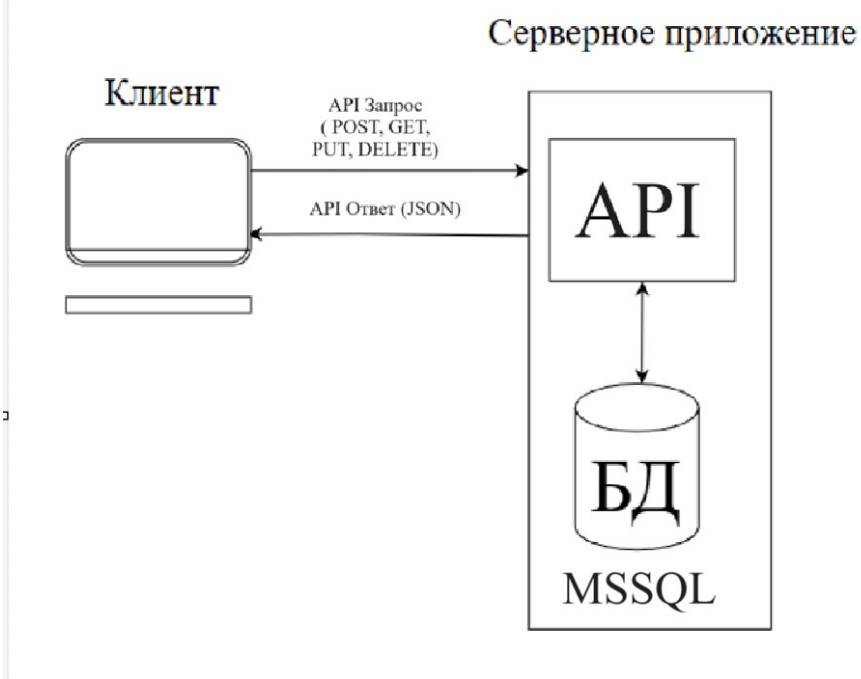

Рисунок 7 - Общая архитектура проекта
DOI: https://doi.org/10.60797/IRJ.2024.149.126.7

На рисунке 7 показано, что веб-чат будет разделен на две логические составляющие – клиентскую и серверную часть. Клиентская часть, выступающая в виде браузера, несет ответственность за взаимодействие с пользователем и коммуникацию с сервером, именное её будут видеть клиенты во время работы приложения. Серверная часть в свою очередь занимается обработкой и хранением информации, которую предоставляют ей клиенты. В число этой информации входят данные о самих пользователях, комнатах чатов и отправленные, полученные ими сообщения.

Рисунок 8 представляет собой UML-диаграмму развертывания, которая иллюстрирует физическую конфигурацию программного обеспечения и аппаратного обеспечения корпоративного веб-чата. Диаграмма показывает, как компоненты системы распределены по различным узлам и как они взаимодействуют друг с другом.

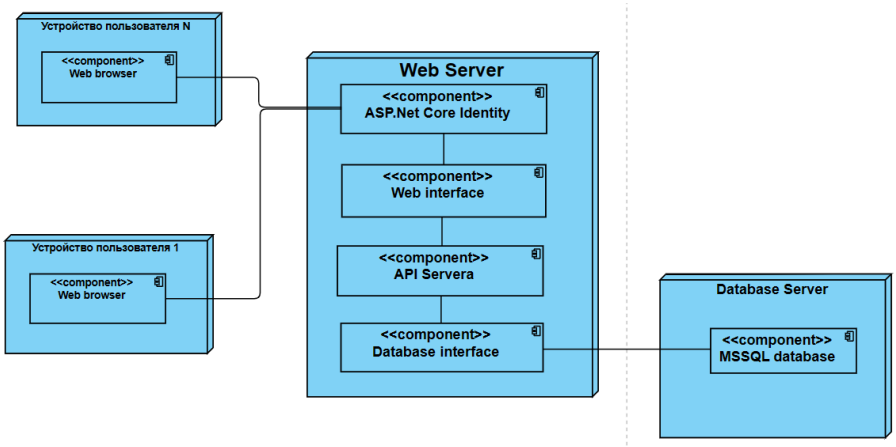

Рисунок 8 - UML-диаграмма развертывания
DOI: https://doi.org/10.60797/IRJ.2024.149.126.8

Данная диаграмма показывает, что устройства пользователей подключаются к веб-серверу чата и проводят с ним взаимодействие с помощью веб-браузера на устройствах клиента. Сервер в свою очередь состоит из таких компонентов как:

1) ASP.Net Core Indentity: отвечает за регистрацию и аутентификацию;
2) веб-интерфейс: его видят пользователи в своих браузерах;
3) API Servera: обеспечивает функционал сервера;
4) database Interface: для получения информации с сервера базы данных.

В проекте веб-чата используется база данных Microsoft SQL Server 2022 [8]. Диаграмма баз данных приведена на рисунке 9.

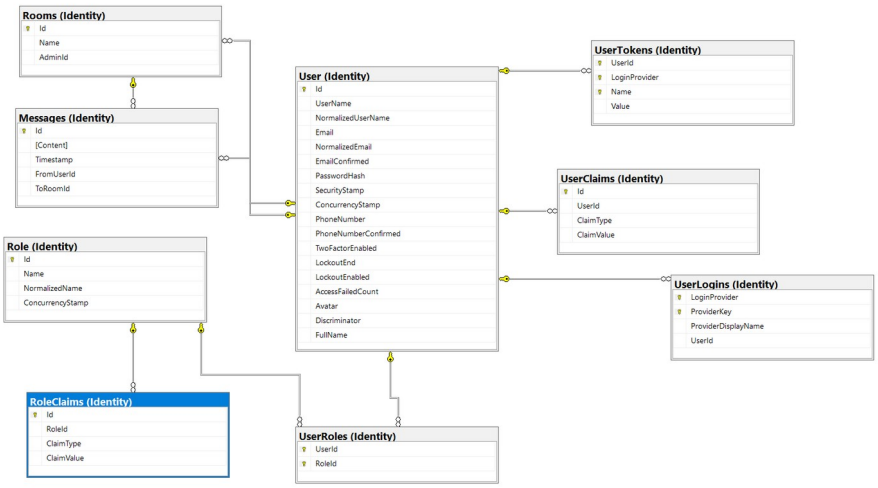

Рисунок 9 - Диаграмма БД
DOI: https://doi.org/10.60797/IRJ.2024.149.126.9

База данных проекта состоит из 9 таблиц:

1. User (Indentity): это основная таблица, которая хранит информацию о пользователях. Она включает в себя основные поля, такие как Id, UserName (имя пользователя), Email (электронная почта), Password-Hash (хэш пароля) и PhoneNumber (номер телефона). Также могут быть дополнительные поля для обеспечения безопасности и отслеживания, такие как SecurityStamp, ConcurrencyStamp, TwoFactorEnabled и LockoutEnd.
2. Role (Indentity): эта таблица хранит информацию о ролях. Роли используются для управления доступом пользователей в приложении.
3. UserClaims (Indentity): эта таблица представляет утверждения, которыми обладает пользователь. Утверждение – это утверждение, которое субъект (пользователь или другое приложение) делает о себе.
4. UserTokens (Indentity): эта таблица представляет токен аутентификации для пользователя. Токены могут быть созданы из любых данных пользователя или идентификации, которые могут быть выданы с помощью доверенного поставщика идентификации или ASP.NET Core Identity.

5. UserLogins (Indentity): эта таблица связывает пользователя с входом в систему. В системе Identity Asp.net используется таблица AspNetUser-Logins для хранения информации о входах из сторонних/внешних источников.

6. RoleClaims **(**Indentity): эта таблица представляет утверждение, которое предоставляется всем пользователям в рамках роли. Утверждение – это утверждение, которое субъект (пользователь или другое приложение) делает о себе.

7. UserRole **(**Indentity): это связующая сущность, которая связывает пользователей и роли. Это отношение многие ко многим, которое требует связующей таблицы в базе данных.

8. Rooms: представляет собой сущность, которая хранит информацию о комнатах чата веб-приложения.

9. Messages: представляется основным компонентом в структуре базы данных корпоративного веб-чата. Она служит центральным хранилищем для информации о сообщениях, отправленных пользователями в комнатах чата.

На рисунке 10 представлена главная страница пользовательского интерфейса корпоративного веб-чата. На этом изображении отображены основные элементы интерфейса, включая окно чата, список контактов и форму ввода сообщений. Этот рисунок демонстрирует, как пользователи взаимодействуют с системой и как они могут использовать различные функции чата.

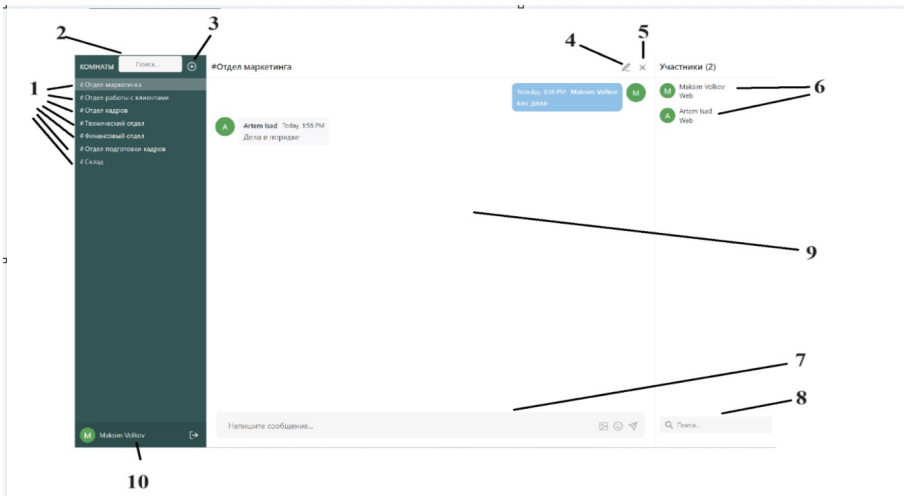

Рисунок 10 - Страница чата:
1 – список всех комнат чата пользователя; 2 – поля для поиска нужной комнаты чата; 3 – кнопка для создания нового чата; 4 – кнопка для редактирования имени группы (доступна только для администратора группы); 5 – кнопка для удаления чата группы (доступна только для администратора группы); 6 – список участников группы; 7 – поля для ввода и отправки сообщения в группу (к сообщению можно прикрепить изображение и файлы, а также эмодзи); 8 – поля для поиска нужного участника; 9 – поле, в котором отображаются все сообщения, отправленные в группу; 10 – кнопка для перехода в профиль пользователя

DOI: https://doi.org/10.60797/IRJ.2024.149.126.10

Тестирование веб-чата

Нагрузочное тестирование веб-приложений является неотъемлемым этапом оценки их работоспособности и стабильности. Этот процесс позволяет выявить потенциал системы, ее узкие места, а также определить, насколько она готова к работе в условиях высоких нагрузок. В контексте разработки корпоративного веб-чата нагрузочное тестирование играет ключевую роль, так как необходимо обеспечить корректную работу системы при значительном числе одновременно подключенных пользователей.

Для выполнения нагрузочного тестирования применялся инструмент Apache JMeter – широко используемая платформа с открытым исходным кодом, предоставляющая обширные возможности для моделирования и анализа работы веб-приложений. Этот инструмент зарекомендовал себя как надежное средство для симуляции пользовательского трафика, поиска и устранения проблем в приложениях, а также формирования отчетов о результатах тестирования. Многолетняя история развития и активная поддержка со стороны мирового сообщества разработчиков подтверждают высокую востребованность и эффективность Apache JMeter [9], [10].

Ключевые преимущества Apache JMeter включают в себя простоту интерфейса, что снижает порог вхождения для новых пользователей, а также наличие множества готовых плагинов, расширяющих стандартные возможности инструмента. Плагины могут быть использованы для решения специфических задач, а при необходимости разработчики могут создавать свои собственные расширения. Еще одной важной особенностью является возможность выполнения автоматизированных тестов API без необходимости написания кода, что упрощает процесс тестирования. Также инструмент поддерживает парсинг различных форматов данных, таких как JSON, YAML и CSV, что позволяет использовать данные из этих файлов при построении сценариев тестирования. После проведения тестирования Apache JMeter генерирует персонализированные отчеты, что облегчает анализ производительности системы и выявление проблемных зон.

Прежде чем приступить к нагрузочному тестированию, крайне важно тщательно разработать сценарии, которые будут моделировать поведение пользователей системы в реальных условиях эксплуатации. Сценарии нагрузочного

тестирования позволяют воспроизвести нагрузку на систему, аналогичную реальной, что необходимо для точного измерения производительности и стабильности веб-приложения. При разработке сценариев учитываются такие параметры, как количество пользователей, их типовое поведение, частота выполнения определенных действий, а также продолжительность сеансов работы с системой. Реалистичность таких сценариев имеет первостепенное значение для успешного проведения тестирования.

Использование сценариев нагрузочного тестирования помогает определить пределы системы: как она будет функционировать при увеличении числа пользователей, какие действия приводят к замедлению работы, и на каком этапе система начинает терять производительность. Важно не только выявить узкие места в системе, но и понять, при каких нагрузках система теряет свою стабильность, что позволяет внести необходимые изменения в архитектуру или настройки серверной части.

Для выполнения нагрузочного тестирования корпоративного веб-чата был разработан сценарий, предполагающий участие 150 виртуальных пользователей. Эти пользователи осуществляли следующие действия: переход по ссылке к чату, регистрация нового пользователя через заполнение соответствующей формы, отправка данных на сервер, перенаправление на главную страницу чата, присоединение к комнате чата, получение всех сообщений, находящихся в комнате, и отправка собственного сообщения. Данный сценарий позволил воспроизвести ключевые действия, выполняемые пользователями в реальной системе, и оценить, как сервер справляется с такими операциями при массовом подключении.

Результаты нагрузочного тестирования были представлены в виде ряда графиков, первый из которых демонстрирует количество активных пользователей во время выполнения теста (рисунок 11). Этот график позволяет наглядно оценить динамику нагрузки на сервер и выявить моменты пиковых значений, что является важным фактором при анализе стабильности работы системы.

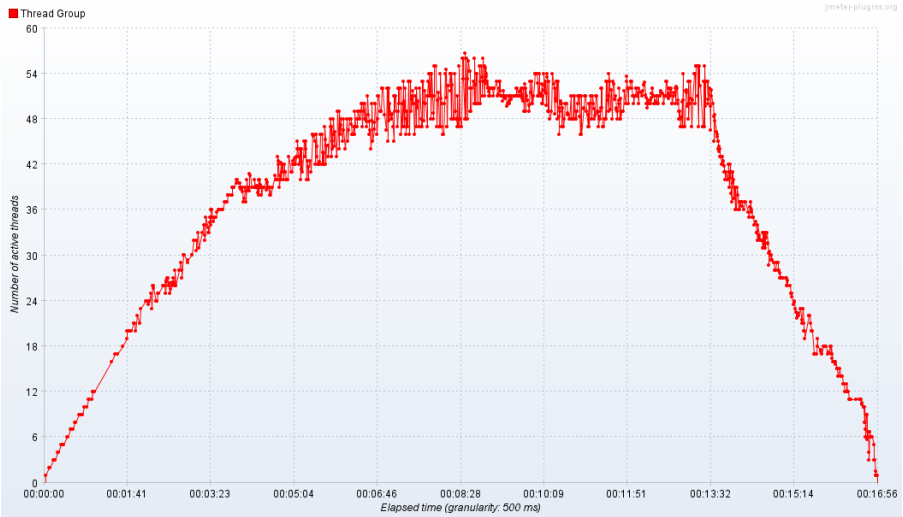

Рисунок 11 - График активных пользователей
DOI: https://doi.org/10.60797/IRJ.2024.149.126.11

Этот график (рисунок 11) является отображением количества активных пользователей, которые в период проведения теста взаимодействовали с веб-чатом. Под «активными» пользователями понимаются те, которые не просто зашли в веб-чат, но и выполняют определенные действия. Данный параметр очень важен для понимания того, как система справляется с реальной нагрузкой. Благодаря этим данным можно увидеть, насколько система эффективно отвечает на запросы пользователей, не теряет ли она в производительности при увеличении числа активных пользователей.

Второй график, приведенный на рисунке 12 иллюстрирует количество отправленных пользователями HTTP-запросов веб-серверу в секунду.

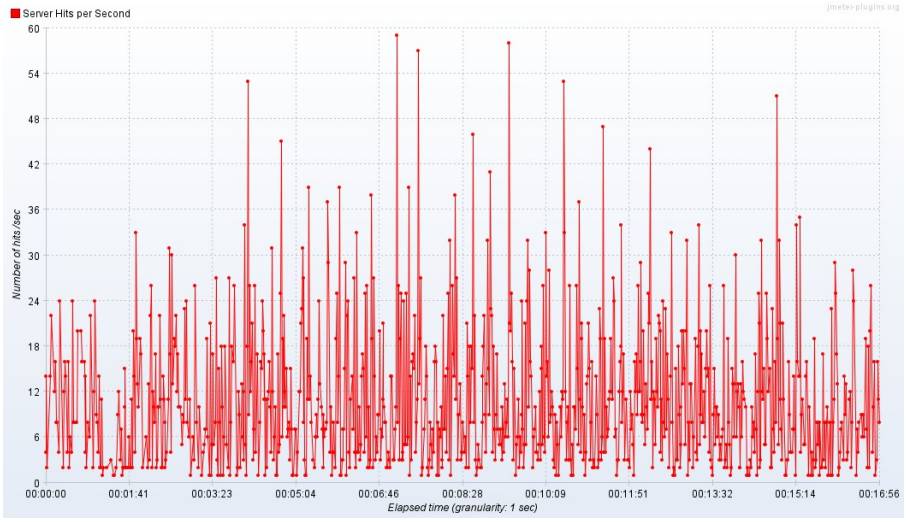

Рисунок 12 - Количество запросов, отправленных пользователями серверу
DOI: https://doi.org/10.60797/IRJ.2024.149.126.12

С помощью графика, приведенного на рисунке 12, можно увидеть, насколько сервер используется в определенные периоды времени. Не только количество запросов, но и их частота важны для анализа производительности сервера, поскольку они вместе определяют нагрузку на сервер.

Следующие 2 графика, приведенные на рисунках 13 и 14 иллюстрируют время ответа сервера на запросы от пользователей.

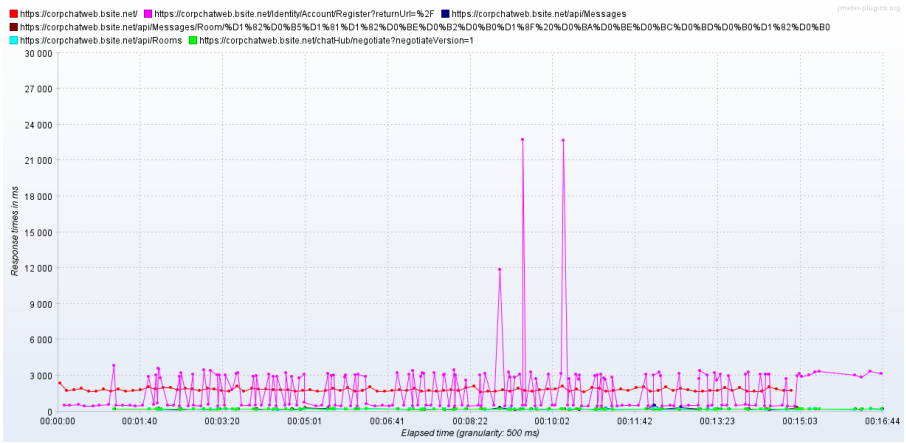

Рисунок 13 - Показатели времени ответа на запросы к серверу
DOI: https://doi.org/10.60797/IRJ.2024.149.126.13

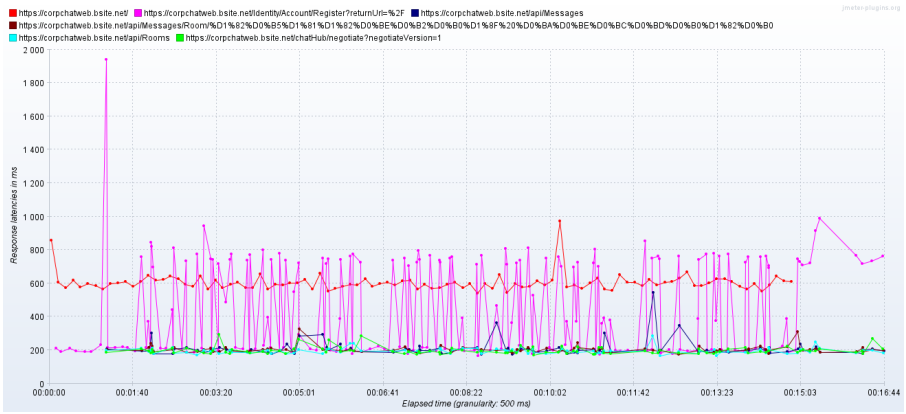

Рисунок 14 - Средние показатели времени ответа на запросы к серверу
DOI: https://doi.org/10.60797/IRJ.2024.149.126.14

В ходе анализа всех полученных графиков при нагрузке 150 пользователей, можно сделать выводы о том, что результаты проведенного тестирования оказались весьма обнадеживающими: не было обнаружено никаких слабых мест в работе веб-чата.

Заключение

В данной статье была представлена разработка корпоративного веб-чата с использованием библиотеки SignalR. Веб-чаты становятся неотъемлемой частью современного бизнеса, обеспечивая эффективную коммуникацию внутри компании. Представленный проект направлен на создание надежного и функционального решения, способного удовлетворить потребности современных организаций в области внутреннего общения.

На основе анализа существующих решений, таких как «Пачка», "VK Teams" и "Express", было выявлено, что несмотря на широкий функционал доступных на рынке мессенджеров, они имеют свои ограничения и не всегда могут удовлетворить специфические потребности бизнеса. Это делает актуальной задачу разработки специализированного корпоративного веб-чата, способного интегрироваться с другими системами компании и обеспечивать высокий уровень безопасности данных.

Проектируемое приложение построено на основе архитектуры клиент-сервер, с использованием технологии SignalR, которая позволяет реализовать общение в режиме реального времени. В статье подробно рассмотрены механизмы работы библиотеки, включая поддержку WebSocket, Server-Sent Events и Long Polling, что делает разработку максимально гибкой и адаптивной к различным условиям эксплуатации.

Проведенное нагрузочное тестирование веб-чата с использованием Apache JMeter показало, что разработанное приложение успешно справляется с высокими нагрузками, обеспечивая стабильную работу при одновременном подключении большого числа пользователей. Это подтверждает эффективность выбранных технологий и правильность архитектурных решений [11], [12], [13].

Таким образом, предложенное решение представляет собой надежный и функциональный инструмент для корпоративной коммуникации, который может быть легко адаптирован под нужды конкретной компании. Его использование позволит существенно улучшить внутренние процессы общения и координации сотрудников, что в конечном итоге будет способствовать повышению эффективности и конкурентоспособности организации.

Конфликт интересов

Не указан.

Рецензия

Кацко С.Ю., Сибирский государственный университет геосистем и технологий, Новосибирск Российская Федерация

DOI: https://doi.org/10.60797/IRJ.2024.149.126.15

Conflict of Interest

None declared.

Review

Katsko S.Y., Siberian State University of Geosystems and Technologies, Novosibirsk Russian Federation

DOI: https://doi.org/10.60797/IRJ.2024.149.126.15

Список литературы / References

1. Лизин С. Управление данными в корпоративных системах / С. Лизин // Открытые системы. СУБД. — 2010. — № 8. — С. 31.

2. Логинова М. А. Защита информации компьютерных сетей от внутренних и внешних пользователей / М. А. Логинова // Молодежный научно-технический вестник. — 2012. — № 8. — С. 16.

3. Волков М. Р. Корпоративный web-чат с использованием библиотеки SignalR / М. Р. Волков // Развитие современной науки и технологий в условиях трансформационных процессов (шифр – МКНТ) : Сборник материалов XIX Международной научно-практической конференции, Москва, 21 марта 2024 года. — Москва : Академическая среда, 2024. — С. 66–76.

4. Джангаров А. И. Современные инструменты веб-разработки / А. И. Джангаров, Д. Ш. Калхиташвили, Д. М. Магамедова // Тенденции развития науки и образования. — 2021. — № 80-2. — С. 96–98. DOI: 10.18411/trnio-12-2021-80.

5. Натальченко И. А. Анализ механизмов передачи крупных массивов данных через сеть интернет с помощью технологии веб-сервиса / И. А. Натальченко // Инженерный вестник Дона. — 2008. — № 4(6). — С. 11–15.

6. Фролов В. А. Сравнение технологий клиент-серверного взаимодействия в реальном времени при их использовании в веб-приложении / В. А. Фролов, Е. В. Вершинин // Электронный журнал: наука, техника и образование. — 2022. — № 3(38). — С. 20–27.

7. Черняк Л. WebRTC: веб-коммуникации в реальном времени / Л. Черняк // Открытые системы. СУБД. — 2014. — № 5. — С. 14–16.

8. Гринченко Н. Н. Базы данных. Разработка клиентских приложений на платформе Net : Учебное пособие / Н. Н. Гринченко, А. Ю. Громов, А. В. Благодаров. — Москва : КУРС, 2018. — 288 с.

9. Федоров М. В. Сравнительные характеристики веб-приложений для тестирования нагрузок / М. В. Федоров // Наука сегодня: технические и естественные науки : Сборник материалов XXXVI-ой международной очно-заочной научно-практической конференции. В 3-х томах, Москва, 09 октября 2023 года. — Москва : Империя, 2023. — С. 42–45.

10. Редкин П. А. Программный комплекс распределенного тестирования веб-приложений / П. А. Редкин, А. С. Алешкин // International Journal of Open Information Technologies. — 2024. — Т. 12. — № 4. — С. 125–132.

11. Макиенко К. А. Один из подходов к тестированию web-приложений / К. А. Макиенко, О. И. Синельникова // Восточно-Европейский журнал передовых технологий. — 2012. — Т. 1. — № 2(55). — С. 39–41.

12. Утеев Г. Разработка децентрализованной системы идентификации личности по биометрическим данным с помощью технологии блокчейн и компьютерного зрения / Г. Утеев, Р. Ф. Гибадуллин // Международный научно-исследовательский журнал. — 2024. — № 4(142). DOI: 10.23670/IRJ.2024.142.6.

13. Zaripova R. Principles of autonomous testing of high-performance .NET application / R. Zaripova, M. Kuznetsov, V. Kosulin [et al.] // E3S Web of Conferences. — 2024. — Vol. 531. — P. 03014. DOI: 10.1051/e3sconf/202453103014.

Список литературы на английском языке / References in English

1. Lizin S. Upravlenie dannymi v korporativnyh sistemah [Data management in corporate systems] / S. Lizin // Otkrytye sistemy. SUBD [Open Systems. DBMS]. — 2010. — No. 8. — P. 31. [in Russian]

2. Loginova M. A. Zashhita informacii komp'juternyh setej ot vnutrennih i vneshnih pol'zovatelej [Protection of computer network information from internal and external users] / M. A. Loginova // Molodezhnyj nauchno-tehnicheskij vestnik [Youth Scientific and Technical Bulletin]. — 2012. — No. 8. — P. 16. [in Russian]

3. Volkov M. R. Korporativnyj web-chat s ispol'zovaniem biblioteki SignalR [Corporate web-chat using the SignalR library] / M. R. Volkov // Razvitie sovremennoj nauki i tehnologij v uslovijah transformacionnyh processov (shifr - MKNT) [Development of modern science and technology in conditions of transformational processes (cipher – MCNT)] : Collection of materials of the XIX International Scientific and Practical Conference, Moscow, March 21, 2024. — Moscow : Academic Environment, 2024. — P. 66–76. [in Russian]

4. Dzhangarov A. I. Sovremennye instrumenty veb-razrabotki [Modern web development tools] / A. I. Dzhangarov, D. Sh. Kalkhitashvili, D. M. Magamedova // Tendencii razvitija nauki i obrazovanija [Trends in the Development of Science and Education]. — 2021. — No. 80-2. — P. 96–98. DOI: 10.18411/trnio-12-2021-80. [in Russian]

5. Natalchenko I. A. Analiz mehanizmov peredachi krupnyh massivov dannyh cherez set' internet s pomoshh'ju tehnologii veb-servisa [Analysis of the mechanisms of transmission of large amounts of data over the Internet using web service technology] / I. A. Natalchenko // Inzhenernyj vestnik Dona [Engineering Bulletin of the Don]. — 2008. — № 4(6). — P. 11–15. [in Russian]

6. Frolov V. A. Sravnenie tehnologij klient-servernogo vzaimodejstvija v real'nom vremeni pri ih ispol'zovanii v veb-prilozhenii [Comparison of technologies of client-server interaction in real time when using them in a web application] / V. A. Frolov, E. V. Vershinin // Jelektronnyj zhurnal: nauka, tehnika i obrazovanie [Electronic Journal: Science, Technology and Education]. — 2022. — № 3(38). — P. 20–27. [in Russian]

7. Chernyak L. WebRTC: veb-kommunikacii v real'nom vremeni [WebRTC: Real-time web communications] / L. Chernyak // Otkrytye sistemy. SUBD [Open systems. DBMS]. — 2014. — No. 5. — P. 14–16. [in Russian]

8. Grinchenko N. N. Bazy dannyh. Razrabotka klientskih prilozhenij na platforme Net [Databases. Development of client applications on the Net platform] : textbook / N. N. Grinchenko, A. Y. Gromov, A.V. Blagodarov. — Moscow : KURS, 2018. — 288 p. [in Russian]

9. Fedorov M. V. Sravnitel'nye harakteristiki veb-prilozhenij dlja testirovanija nagruzok [Comparative characteristics of web applications for load testing] / M. V. Fedorov // Nauka segodnja: tehnicheskie i estestvennye nauki [Science Today: technical and Natural Sciences] : Collection of materials of the XXXVI-th International Intramural Scientific and Practical Conference. In 3 volumes, Moscow, October 09, 2023. — Moscow : Empire, 2023. — P. 42–45. [in Russian]

10. Redkin P. A. Programmnyj kompleks raspredelennogo testirovanija veb-prilozhenij [Software package for distributed testing of web applications] / P. A. Redkin, A. S. Aleshkin // International Journal of Open Information Technologies. — 2024. — Vol. 12. — No. 4. — P. 125–132. [in Russian]

11. Makienko K. A. Odin iz podhodov k testirovaniju web-prilozhenij [One of the approaches to testing web applications] / K. A. Makienko, O. I. Sinelnikova // Vostochno-Evropejskij zhurnal peredovyh tehnologij [Eastern European Journal of Advanced Technologies]. — 2012. — Vol. 1. — No. 2(55). — P. 39–41. [in Russian]

12. Uteev G. Razrabotka decentralizovannoj sistemy identifikacii lichnosti po biometricheskim dannym s pomoshh'ju tehnologii blokchejn i komp'juternogo zrenija [Development of a decentralized identity identification system based on biometric data using blockchain technology and computer vision] / G. Uteev, R. F. Gibadullin // Mezhdunarodnyj nauchno-issledovatel'skij zhurnal [International Scientific Research Journal]. — 2024. — № 4(142). DOI: 10.23670/IRJ.2024.142.6. [in Russian]

13. Zaripova R. Principles of autonomous testing of high-performance .NET application / R. Zaripova, M. Kuznetsov, V. Kosulin [et al.] // E3S Web of Conferences. — 2024. — Vol. 531. — P. 03014. DOI: 10.1051/e3sconf/202453103014.