Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА ВЕБ-ЧАТА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ «WEBSOCKET»

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Матвеева Сергея Сергеевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С появлением сети Интернет человечество получило доступ к неограниченному библиотекой количеству информации. С ее развитием людям больше не приходится писать письма и отправлять в другой город или страну, через почтовые отделения — теперь все это возможно в сети и доступно практически каждому человеку в любом уголке планеты.

Существуют разные способы передачи данных между сервером и приложениями или обратно. Правила таких способов описаны в специальных протоколах. Часть таковых применяется там, где не важна скорость обмена данными, другие, наоборот, ускоряют процесс обмена. Часть из них не требует безопасной пересылки. Одним из протоколов ускоряющих обмен между сервером и `Web`-приложением является протокол `WebSocket` [1], о котором в дальнейшем пойдет речь.

Целью работы является разработка `Web`-чата для использования в коммерческой компании. При разработке были поставлены следующие задачи:

1. Реализация классов для установления соединения по протоколу `WebSocket`;
2. Разработка методов получения, отправки и обработки сообщений.

Актуальность внедрения обусловлена желанием заказчика иметь возможность общения с клиентами не переходя на сторонние ресурсы.

1 Основные положения

1.1 История возникновения идеи

Идея состояла в том, чтобы создать и использовать технологию динамического обращения к серверу «на лету», то есть без перезагрузки страницы, используя динамическое создание дочерних фреймов. Такой подход был назван AJAX и его методы ускоряли работу со страницами браузера, полная перезагрузка страницы уходила на второй план и снижалась нагрузка на сервер. Конечно у AJAX были и свои недостатки. Разрабатывавшиеся далее методы, основанные на идее AJAX, увеличивали время передачи информации, так как использовали большие HTTP запросы.

В связи с этим назревал вопрос об ускорении процесса передачи информации, используя протокол постоянного соединения с минимальными задержками. Таким решением стал протокол WebSocket. Первая версия протокола была создана в 2008 году Майклом Картером, далее в 2009 году Ян Хиксон включил WebSocket в спецификацию HTML5. Такой технологический прорыв не мог остаться незамеченным и в конце этого же года Chrome 4 стал первым браузером, в котором WebSocket был включен по умолчанию [1, 2].

1.2 Почему именно WebSocket

Почему не использовать AJAX? Если вернуться к истории, то AJAX запросы при создании чатов будут нагружать сервер большими HTTP запросами, из-за чего время отображения будет стремиться к бесконечности с каждым новым сообщением. Поэтому выгодно будет использовать технологию двунаправленного соединения, без лишних пересылок head тегов.

Для передачи информации по сети используя постоянное соединение существует несколько способов. Ротрим наиболее распространенные из них — WebSocket и Socket.IO.

Как говорилось ранее, WebSocket это протокол веб-коммуникации, позволяющий установить связь между сервером и клиентом для использования всех преимуществ живого общения. Протокол использует расширение TCP-соединения для поддержания соединения открытым сколь угодно долго. Socket.IO является библиотекой языка JavaScript. Технология основана на WebSocket и нескольких других, но использует WebSocket только когда они доступны. В остальных же случаях могут использоваться такие технологии как Flash Socket

или AJAX Multipart Stream. [3]

Казалось бы, Socket.IO является более изящным решением написания приложений обмена данными в реальном времени. Библиотека дает возможности соединений «из коробки», в то время как WebSocket соединение должно быть правильно спроектировано и описано. Но все плюсы JavaScript библиотеки нивелируются лишь тем, что для общения и установки соединения библиотека использует намного больше драгоценного интернет трафика. Для установления соединения WebSocket использует два запроса — получения страницы и запрос на обновление протокола.

Минимизация исходящего и входящего трафика является одним из принципов построения хороших веб-приложений, по этой причине мной был выбран именно протокол WebSocket.

1.3 Принципы работы

Веб-сокет создает постоянное соединение между сервером и клиентом, которым для отправки сообщений могут пользоваться обе стороны. Браузер устанавливает такое соединение методом рукопожатия, то есть с согласия обеих сторон. Отправляется HTTP запрос с включенным в него заголовком Upgrade, который сообщает серверу о желании установления соединения по веб-сокету [4]. Если сервер поддерживает протокол WebSocket, он сообщает свою готовность с заголовком Upgrade в ответе. После того, как рукопожатие выполнено, HTTP соединение заменяется на соединение по веб-сокету. После этого любая из сторон может начинать отправку сообщений.

Благодаря использованию WebSocket мы можем отправлять любой объем данных, не подвергая систему ненужной нагрузке, которую создают HTTP-запросы. Данные передаются по WebSocket соединению в виде сообщений, состоящих из одного или нескольких фреймов с информацией об отправляемых данных. Для того чтобы правильно собрать исходные данные, у каждого фрейма есть свой префикс, содержащий данные о полезной нагрузке. Использование системы обмена сообщениями основываясь на фреймах сокращает количество передаваемой служебной информации, что значительно уменьшает задержки при передаче сообщений [5].

1.4 Различные URL протоколы

Для протокола WebSocket могут быть использованы две схемы URL это ws:// и wss://. Схема ws:// работает только с HTTP, а HTTP принимает только ws://. Также для использования

Так же работает и с wss://. Также для использования wss:// должен быть активирован SSL сертификат.

2 Описание рабочей программы

Цель разработки состояла в написании чата для коммерческой CRM-системы, чтобы клиент, обращаясь за помощью в компанию в формате вопрос-ответ мог получить консультацию специалиста. Функции доработки веб-приложения были написаны на языке C#, с использованием среды разработки Microsoft Visual Studio.

Для реализации сервера был создан сервис с использованием библиотеки Websockets. В нем были определены основные классы для работы с входящими и исходящими данными.

2.1 Описание библиотеки WebSocket

Класс библиотеки WebSocket содержит в себе методы и свойства, позволяющие приложениям получать и отправлять данные после завершения обновления протокола WebSocket [6, 7].

2.2 Описание методов сервиса

2.2.1 Класс WebSocketHandler

Класс для работы с самим WebSoket общением. Данный класс реализует структуру работы с технологией WebSocket. В него включено множество методов для реализации клиент-серверного общения. Класс включает в себя [8, 9]:

1. экземпляр WebSocket;
2. экземпляр интерфейса ILogger для записи и вывода в консоль сообщений об ошибках;
3. экземпляр класса TaskCompletionSource для обращения к внешнему асинхронному запросу;
4. конструктор класса WebSocketHandler;

. Также класс включает в себя методы получения и обработки сообщений.

Данный класс является основополагающим классом всего сервиса. На его основе в дальнейшем будет произведена вся работа с данными [10].

2.2.2 Класс ChatHandler

Класс для работы с чатами. Его реализация подразумевает под собой реализацию постоянно открытого «порта» в котором предварительно будут обра-

ботаны входящие сообщения. Класс является наследником класса `WebSocketHandler`, а так же включает в себя методы и свойства для работы с чатами.

2.2.3 Класс OnlineUsersManager

Класс хранит в себе информацию о подключенных `WebSocket` соединениях и пользователях. Реализует методы записи и удаления пары пользователь-вебсокет.

Так как к этой информации может быть получен доступ не одного экземпляра класса, то используется `LockSlim` [11].

Так же в `OnlineUsersManager` реализованы методы поиска пользователя по вебсокет соединению и поиска вебсокет соединений для одного или нескольких пользователей.

2.2.4 Класс ChatManager

Класс используемый для работы с `ChatHandler` . Содержит в себе конструктор для инициализации основных свойств, а так же методы для запуска хендлера `ChatHandler` и для отправки сообщений.

2.2.5 Интерфейс IChatCallback

Интерфейс для получения информации о чате . В каркасе потомка должны быть реализованы методы сохранения сообщения, получения пользователей, отправки уведомлений и получения ссылки на файлы.

2.2.6 Интерфейс ITokenProcessor

Интерфейс для получения токена пользователя.

2.2.7 Модель Attachment

Данная модель представляет собой класс с набором свойств, однозначно сохраняющим файл в базе данных, а так же отражая все основыне свойства файла.

Модель Attachment будет использована при преобразовании сообщения в нужный тип (текст или файл) на этапе обработки методами класса `ChatManager`.

2.2.8 Модель ChatMessage

Данная модель представляет собой класс с набором свойств, используемым для однозначной записи в базе данных, а так же для хранения в оперативной памяти информации о сообщении и предачи его в сеть.

Модель ChatMessage будет использована при преобразовании сообщения в нужный тип (текст или файл) на этапе обработки методами класса `ChatManager`.

2.3 Описание методов API

Для того чтобы наш чат начал свою работу, необходимо запустить IIS с подключенной к нему библиотекой WebSockets и базой данных. Определим модель пользователя RtcServicesManager для обращения к API. Данная модель содержит следующие свойства и методы для отбора пользователей, манипулирования данными для отображения пользователей на стороне клиента.

Для обращения к выше описанным методам сервиса воспользуемся классом RtcServicesManager, в котором явно определим используемый ChatManager с описанной нами моделью пользователя API.

Так же определим в классе новый объект SignalingManager. Принцип работы класса схож с ChatManager, отличие состоит в том, что SignalingManager реализует задачу рассылки уведомлений о том, что в чате появилось новое сообщение.

Далее воспользуемся MVC технологией и для API создадим контроллер, который будет отвечать за принятие запросов из вне. Определим на нем метод Connect, реализующий получение WebSockets запросов. Также создадим метод GetChatMessages возвращающий на сторону клиента сообщения по запрошенному чату [12]. Затем определим метод загрузки документа(UploadDocument) и метод получения всех членов чата (GetChatMembers).

Теперь сервер готов принимать сообщения.

2.4 Работа с WebSocket на стороне браузера

Теперь, когда мы можем получать сообщения, можно начинать обмен ими. Для этого воспользуемся функциями `Json` и библиотекой `knockout.js`. Для вывода сообщений на страницу создадим наблюдаемый элемент `PageModel` отвечающий за отображение всех чатов, доступных пользователю. Для этого элемента определим несколько наблюдаемых свойств, таких как новое сообщение, выбранный пользователь и подобные им. Данные содержащиеся этих объектах будут выведены на страницу автоматически, так как мы используем `knockout.js`.

Так же добавим метод создания нового чата и несколько методов навигации. Не забудем про методы обновления текущего чата и отправки нового сообщения. Текущий чат будет обновляться как только в переменных модели страницы будут происходить изменения. Новое сообщение будет считано и отправлено так же по средствам обращения к наблюдаемой переменной библиотеки `knockout.js`. Для добавления сообщения в чат будем отправлять строку в формате `JSON` на сервер.

При работе с чатом текст указанный в поле ввода при помощи запроса будет направлен в контроллер, где он будет обработан и отправлен на клинеты других участников.

ЗАКЛЮЧЕНИЕ

В ходе работы была выполнена задача по разработке и внедрению в CRM систему коммерческой организации, а именно

1. Были реализованы классы для установления соединения по протоколу WebSocket;
2. Были разработаны методы получения, отправки и обработки сообщений.

. Произведена разработка программного средства реализующего в себе Web-чаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 WebSocket — blog.skillfactory.ru [Электронный ресурс]. — URL: https://blog.skillfactory.ru/glossary/websocket/ (Дата обращения 17.05.2022). Загл. с экр. Яз. рус.

2 Как работает JS: WebSocket и HTTP/2+SSE. — habr.com [Электронный ресурс]. — URL: https://habr.com/ru/company/ruvds/blog/342346/ (Дата обращения 05.05.2022). Загл. с экр. Яз. рус.

3 The Socket instance (client-side) — socket.io [Электронный ресурс]. — URL: https://socket.io/docs/v4/client-socket-instance/ (Дата обращения 21.05.2022). Загл. с экр. Яз. анг.

4 Understanding WebSockets with ASP.NET Core — sahansera.dev [Электронный ресурс]. — URL: https://sahansera.dev/understanding-websockets-with-aspnetcore-5/ (Дата обращения 29.04.2022). Загл. с экр. Яз. рус.

5 Глубокое погружение в WebSockets — webformyself.com [Электронный ресурс]. — URL: https://webformyself.com/glubokoe-pogruzhenie-v-websockets/ (Дата обращения 20.05.2022). Загл. с экр. Яз. анг.

6 WebSocket.SendAsync Метод — docs.microsoft.com [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.net.websockets.websocket.sendasync?view=net-6.0 (Дата обращения 13.05.2022). Загл. с экр. Яз. рус.

7 WebSockets — docs.microsoft.com [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/windows/uwp/networking/websockets (Дата обращения 12.05.2022). Загл. с экр. Яз. рус.

8 Поддержка WebSockets в ASP.NET Core — docs.microsoft.com [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/websockets?view=aspnetcore-6.0 (Дата обращения 16.05.2022). Загл. с экр. Яз. рус.

9 *Boner, J.* Reactive Microservices Architecture / J. Boner. — O'Reilly Media, Inc., 2016.

10 *Lombardi, A.* WebSocket. LIGHTWEIGHT CLIENT-SERVER COMMUNICATIONS / A. Lombardi. — Mystic Coders, 2015.

11 ReaderWriterLockSlim Класс — docs.microsoft.com [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-Ru/dotnet/api/system.threading.readerwriterlockslim?view=net-5.0&viewFallbackFrom=windowsdesktop-6.0 (Дата обращения 10.05.2022). Загл. с экр. Яз. рус.

12 Task Класс — docs.microsoft.com [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task?view=net-5.0 (Дата обращения 13.05.2022). Загл. с экр. Яз. рус.