Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

«Клиент-серверное приложение «Чат»

c поддержкой файлового обмена»

БГУИР КР 1-40 01 01 225 ПЗ

Студент: гр. 951002 Протасеня В.Н.

Руководитель:

асс. Мельникова Е.В.

Минск 2021

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2021 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту   Протасене Валентине Николаевне

1. Тема работы   Клиент-серверное приложение “чат” с поддержкой файлового обмена

2. Срок сдачи студентом законченной работы––01.06.2021 г.–––

3. Исходные данные к работе язык программирования Java, JavaScript, библиотека Spring

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Клиент-серверное приложение «Чат», с поддержкой файлового обмена", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Мельникова Е.В.

7. Дата выдачи задания 10.02.2021

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 01.03.2021 – 15 % готовности работы;

разделы 2, 3 к 15.03.2021 – 30 % готовности работы;

разделы 4, 5 к 15.04.2021 – 60 % готовности работы;

раздел 6 к 15.05.2021 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 20.05.2021s – 100 % готовности работы.

Защита курсовой работы с 01.06.2021 по 09.06.2021 г.–––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Е.В.Мельникова

(подпись)

Задание принял к исполнению 110.02.2020

(дата и подпись студента)

# **СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 4](#_Toc71465877)

[ВВЕДЕНИЕ 5](#_Toc71465878)

[1 АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ 6](#_Toc71465879)

[1.1 Анализ существующих прототипов 6](#_Toc71465880)

[1.2 Постановка задачи 10](#_Toc71465881)

[2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 12](#_Toc71465882)

[2.1 Описание функциональности ПС 12](#_Toc71465883)

[2.2 Спецификация функциональных требований 13](#_Toc71465884)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 14](#_Toc71465885)

[3.1 Описание архитектуры программного средства 14](#_Toc71465886)

[3.2 Разработка алгоритма подключения пользователя 16](#_Toc71465887)

[3.3 Разработка алгоритма отправки сообщений 17](#_Toc71465888)

[3.4 Разработка алгоритма отправки файлов 18](#_Toc71465889)

[3.5 Разработка алгоритма отключения пользователя 19](#_Toc71465890)

[4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 20](#_Toc71465891)

[4.1 Проектирование серверной части 20](#_Toc71465892)

[4.2 Проектирование клиентской части 23](#_Toc71465893)

[5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 26](#_Toc71465894)

[6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ 28](#_Toc71465895)

[ЗАКЛЮЧЕНИЕ 32](#_Toc71465896)

[СПИСОК ЛИТЕРАТУРЫ 33](#_Toc71465897)

[ПРИЛОЖЕНИЕ 1 34](#_Toc71465898)

[ПРИЛОЖЕНИЕ 2 35](#_Toc71465899)

# **ВВЕДЕНИЕ**

В настоящее время в мире яовольно много различных средств и способов общения, и колоссальную роль в этом процессе играет Интернет. Ведь общение – это способ взаимодействия людей друг с другом, а с появлением всемирной паутины это общение вышло за рамки одного дома, города и даже одной страны.

Электронная почта, всевозможные форумы и журналы, организованные в сетевом пространстве, многочисленные Интернет-журналы и Интернет-газеты и так далее для многих стали не менее важным аспектом повседневности, чем телевизор или телефон.

Одним из средств обмена сообщениями по компьютерной сети в режиме реального времени является чат. Характерной особенностью является коммуникация именно в реальном времени или близкая к этому, что отличает чат от форумов и других «медленных» средств.

Под словом чат обычно понимается групповое общение, хотя к ним можно отнести и обмен текстом «один на один» посредством программ мгновенного обмена сообщениями, например, ICQ или даже SMS.

Целью данной курсовой работы является разработка программного средства, которое позволит пользователям обмениваться сообщениями и файлами в режиме реального времени.

Данная пояснительная записка содержит следующие основные разделы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ**
   1. **Анализ** **существующих прототипов**

На данный момент существует множество разных форматов приложений для общения в интернете. Поэтому в качестве прототипов будут использованы самые известные и широко используемые программы.

**«WeСhat» —** китайский мессенджер для различных платформ, который позволяет обмениваться текстовыми и голосовыми сообщениями, а также совершать видеозвонки в том числе и групповые.

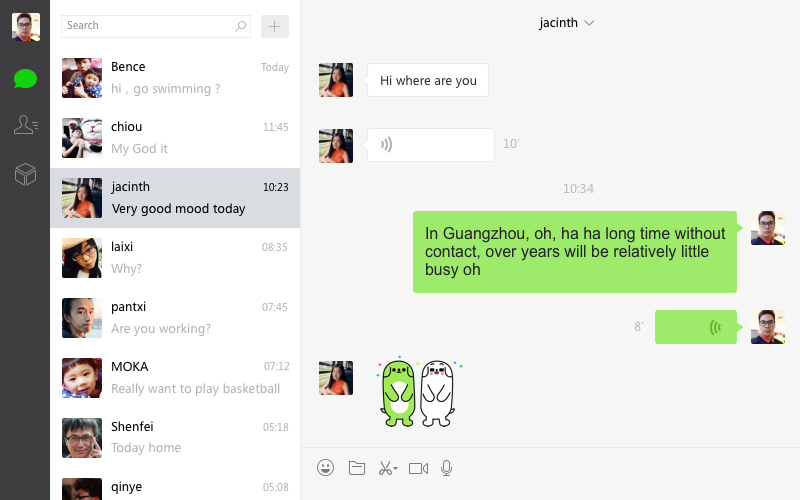


Рисунок 1.2 – Интерфейс «WeChat»

Плюсы:

* приятный и качественный дизайн приложения;
* удобный и понятный пользовательский интерфейс;
* возможность создавать каналы, которых существует три варианта: корпоративный, сервисный и для подписки;
* возможна интеграция с такой популярной соцсетью, как Facebook;
* полная бесплатная доступность;
* к фотографиям можно подставлять надписи, накладывать на снимки фильтры;
* доступна массовая рассылка, аудио и видеозвонки;
* можно создавать групповые чаты.

Минусы:

* низкая производительность;
* наличие спамных звонков.
* поддержка осуществляется от лица сообщества, то есть, она не является официальной.

**«Telegram»** — облачная платформа по учету финансов, интегрируемая в любой интернет-банк и сайт в интернете, позволяющая видеть конечному пользователю всю его финансовую информацию из разных источников в одном месте.

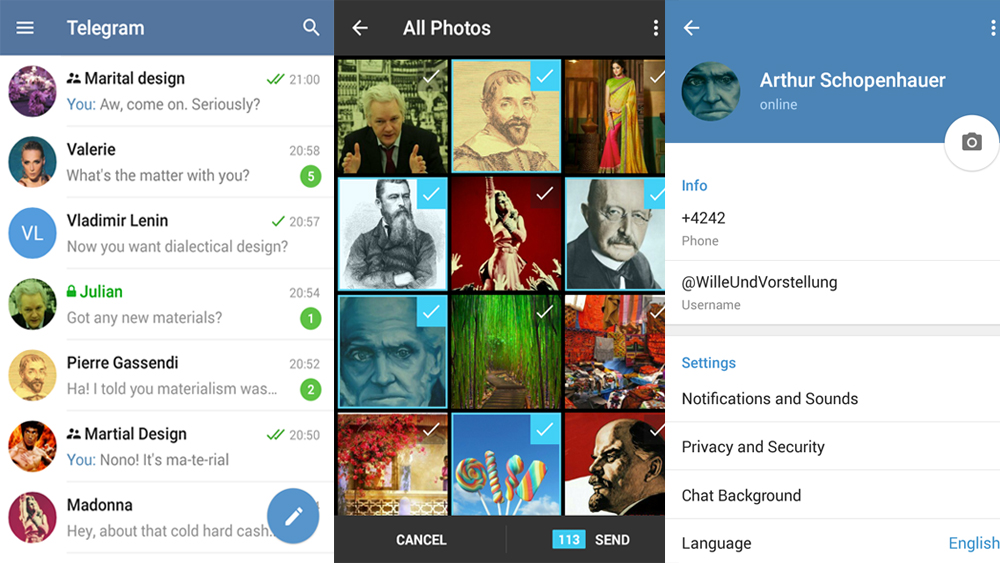


Рисунок 1.3 – Интерфейс «Telegram»

Плюсы:

* бесплатная версия;
* надежная защита от несанкционированного доступа сторонних особ, благодаря MTProto протоколу;
* адаптивность;
* отсутствие рекламы;
* cекретные чаты. Пользователи общаются на основе end-to-end шифрования, история такой переписки не сохраняется на серверах.
* таймер автоуничтожения с выставленным промежутком времени.
* высокая скорость работы, передачи данных адресату;
* возможность отправки файлов больших размеров;
* аудиозвонки;
* видеозвонки;
* видеоконференции;
* веб-версия.

Минусы:

* привязка к номеру телефона;
* ограниченность десктопной и веб-версий. В веб-версии не всегда удаляется история переписки, а создать секретный чат на любой ПК версии просто невозможно;
* сами сообщения не зашифрованы. Они передаются по зашифрованным протоколам, но на серверах сообщения хранятся в открытом виде.

**«WhatsApp»** — cамый популярный мессенджер с аудиторией в 30 млн человек, который предоставляет возможности для общения - чаты, группы (до 256 участников) и звонки.

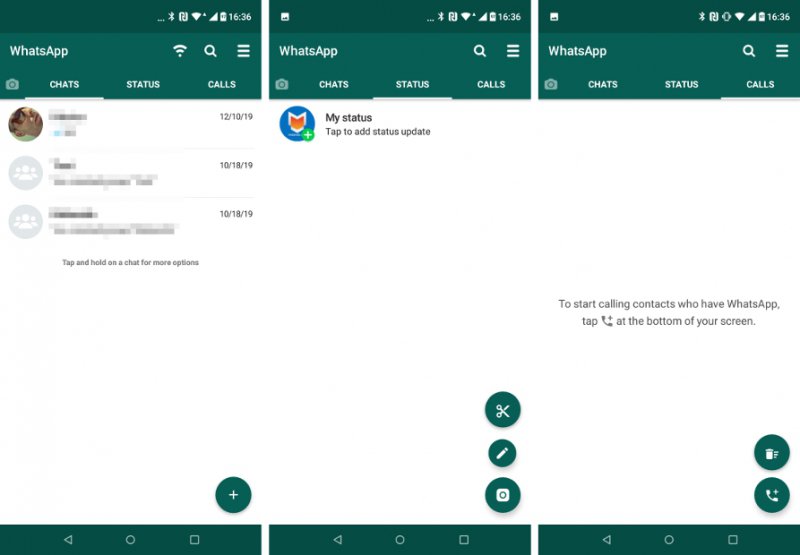


Рисунок 1.4 – Интерфейс «WhatsUp»

Плюсы:

* поддержка разных форматов файлов;
* кроссплатформенность;
* синхронизация списка контактов с телефонной книгой;
* групповые чаты для общения по интересам;
* бесплатные аудио- и видеозвонки;
* фото, аудио и видеоматериалы вначале отправляются на специальный HTTP-сервер, а потом уже передаются в уменьшенном варианте конечному получателю. Это позволяет экономить интернет-трафик;
* программа полностью бесплатна.

Минусы:

* отсутствие приватных чатов, каналов и стикеров;
* нет официальной версии для iPad;
* ограничение на отправку файлов – размер не должен превышать 100 Мб.

**Вывод**

Однозначного ответа на вопрос какое из существующих приложений самое лучшее с точки зрения интерфейса, быстроты работы и функциональных возможностей нет. Установлено, что при использовании программ на мощных компьютерах разница в быстродействии практически незаметна. Каждое из протестированных приложений имеет свои неоспоримые плюсы и минусы, по-своему удобно и найдет своих почитателей.

## Постановка задачи

Для максимально удобного взаимодействия пользователя с проектируемым программным средством должен быть создан информативный и интуитивно понятный пользовательский интерфейс. Также программное средство должно иметь достаточный функционал для обеспечения удобной работы.

В проектируемом программном средстве должны быть реализованы следующие функции:

* регистрация нового пользователя;
* добавление неограниченного числа пользователей в чат;
* отправка текстовых сообщений;
* отправка файлов;
* указание даты и времени отправки сообщения;
* оповещение о новых и вышедших из чата пользователях;
* отображение всех пользователей в чате.

В качестве основной архитектуры для разработки программного средства был выбран WebSocket API. Это протокол полнодуплексной двунаправленной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

Протокол WebSocket является довольно низкоуровневым. Он определяет, как поток байт преобразуется в кадры. Это значит, что в WebSocket просто недостаточно информации во входящем сообщении, чтобы фреймворк или контейнер знали, как его маршрутизировать или обрабатывать. К счастью, спецификация WebSocket позволяет использовать субпротоколы, которые работают на более высоком, прикладном уровне. Одним из них, поддерживаемым Spring Framework, является STOMP.

STOMP — это простой текстовый протокол обмена сообщениями. **Он предоставляет слой форматирования строки на основе кадра поверх WebSocket для определения семантики сообщений.**

Для реализации програмного средства “чат” выбранные протоколы обладают следующими преимуществами:

* **скорость и эффективность**передачи данных. TCP соединение уже готово и нет необходимости тратить время и трафик на его установку;
* WebSockets не имеют ограничений на время жизни в неактивном состоянии. Это значит, что больше не надо периодически обновлять соединение, т.к. его не вправе закрывать прокси-сервера. Соединение может висеть в неактивном виде и не требовать ресурсов;
* в WebSocket нет ограничений на число одновременно открытых сессий к одному серверу, в отличие от HTTP.

# **АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## 2.1 Описание функциональности ПС

Программное средство представляет собой чат в режиме реального времени, которое позволяет неограниченному количеству пользователей отправлять текстовые сообщения и файлы.

Для подключения необходимо в адресной строке веб-клиента, так же называемом браузер, указать URL чата. Далее откроется веб-страница с формой для регистрации.

После корректного ввода имени пользователя и нажатия на кнопку «Start Chatting» или же Enter, пользователь переходит непосредственно к веб-странице чата, при этом на ней отображается, что данный пользователь подключился к чату.

Страница чата представляет собой поле оправленных сообщений и форму для ввода нового сообщения. Для отправки сообщения пользователю необходимо заполнить форму, нажать кнопку «Send» или Enter, после чего оно отобразится в поле отправленных сообщений.

Для идентификации пользователей в чате каждому новому пользователю после подключения назначается аватарка: цветной круг с первой буквой имени в центре.

Сообщение же состоит из аватарки, имени пользователя, которому принадлежит сообщение, текста сообщения, даты и времени отправки.



Рисунок 2.1 – пример сообщения

При отключении пользователя в чате появляется соответствующее сообщение с указанием конкретного имени пользователя, который покинул чат.

## Спецификация функциональных требований

Среди функциональных требований есть «Отправка файлов». Пользователи могут обмениваться файлами в формате GIF, JPEG, PNG и WebP. Для отправки изображения необходимо на кликнуть кнопку с надписью “File”.

Shape, rectangle

Description automatically generated

Рисунок 2.2 – пример отправленного изображения

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 3.1 Описание архитектуры программного средства

Как упоминалось ранее программное средство реализовано на основе протоколов WebSocket и STOMP. Отношение между **ними похоже на отношения между TCP и HTTP.**

**Для наибольшей удобности использовалась библиотека SockJS,** которая обеспечивает двусторонний междоменный канал связи между клиентом и сервером. Другими словами SockJS имитирует WebSocket API и служит своеобразной оберткой вокруг него.

Под капотом SockJS сначала пытается использовать нативную реализацию WebSocket API. Если это не удается, используются различные транспортные протоколы, специфичные для браузера, и представляет их через абстракции, подобные WebSocket.

Переключение на WebSocket происходит после HTTP-запроса от клиента. Браузер отправляет GET-запрос со специальными заголовками, одним из которых являться Upgrade: websocket.

Таким образом клиент спрашивает поддерживает ли сервер данный протокол. Если сервер согласен переключиться на WebSocket, то он должен отправить в ответ код 101. После этого открывается двунаправленное соединение и данные передаются по протоколу WebSocket.

Diagram

Description automatically generated

Рисунок 3.1 – схема переключения на протокол WebSocket

Так же важной составляющей архитектуры разрабатываемого приложения является брокер сообщений (Message Broker). Брокер сообщений — это промежуточная программа, которая принимает отправленные сообщения до того, как перенаправить к необходимым адресам. В данном программном средстве используется встроенный STOMP Message Broker.

Diagram

Description automatically generated

Рисунок 3.2 – схема работы брокера сообщений

Для начала работы с брокером сообщений необходимо его настроить. Сперва выставляется конечная точка, благодаря которой клиент может контактировать и формировать соединение.

В то же время брокер сообщений так же выставляет два пункта назначения. Первый пункт назначения — это тема (topic), на которую клиент может подписаться. Когда тема получает новые сообщения, они отправляются всем клиентам, которые подписались на данную тему. Второй — это место (app), где клиент может отправить сообщения на сервер. Далее сообщения должны быть направлены в методы, занимающиеся их обработкой.

## 3.2 Разработка алгоритма подключения пользователя

При добавлении в чат нового пользователя сначала необходимо подключиться к брокеру сообщения. Для подключения клиент посылает кадр CONNECT с двумя обязательными заголовками:

* accept-version — версии протокола STOMP, которые поддерживает клиент;
* host — имя виртуального хоста, к которому клиент хочет подключиться.

Для подтверждения соединения брокер посылает клиенту кадр CONNECTED с обязательным заголовком:

* version — версия протокола STOMP, которую будет использовать текущая сессия.

Diagram

Description automatically generated

Рисунок 3.3 – схема алгоритма подключения клиента к брокеру сообщений

Далее клиенту необходимо подписаться на рассылку сообщений от брокера. Чтобы подписаться на пункт назначения, клиент отправляет кадр SUBSCRIBE с двумя обязательными заголовками:

* destination — пункт назначения, на который клиент хочет подписаться;
* id — уникальный идентификатор подписки.

После подписки клиент может приступать к обмену сообщениями.

Diagram

Description automatically generated

Рисунок 3.4 – схема алгоритма подписки клиента на тему

## 3.3 Разработка алгоритма отправки сообщений

Чтобы отправить сообщение в пункт назначения, клиент посылает кадр SEND с обязательны заголовками:

* destination — пункт назначения, в который клиент хочет отправить сообщение;
* content-length – количество знаков в теле сообщения.

Тело кадра представляет собой JSON-объект. Это облегченный формат для передачи читаемых человеком данных между компьютерами. Структура JSON состоит из пар ключ-значение. В данном приложении объект содержит имя пользователя, которому принадлежит сообщение, само сообщение, время отправки и тип сообщения.

Diagram

Description automatically generated

Рисунок 3.5 – схема алгоритма отправки сообщения брокеру

Далее брокер сообщений передает его всем клиентам, которые подписаны на данный пункт назначения. Для передачи сообщений от подписок клиенту сервер посылает кадр MESSAGE с тремя обязательными заголовками:

* destination — место назначения, куда было отправлено сообщение;
* subscription — идентификатор подписки, которая получает сообщение;
* message-id — уникальный идентификатор для данного сообщения.

Так же кадр содержит 2 дополнительных заголовка:

* content-type — тип передаваемого сообщения;
* content-length.

## 3.4 Разработка алгоритма отправки файлов

Протокол STOMP, который является веткой WebSocket, предназначен для обмена данными в виде текстовых сообщений. Следовательно, нет возможности передавать бинарные файлы в чистом виде. Решением данной проблемы послужила кодировка Base64.

Base64 - это алгоритм кодирования двоичных данных при помощи только 64 символов ASCII. Алфавит кодирования содержит текстово-цифровые латинские символы A-Z, a-z, цифры от 0 до 9 и 2 дополнительных символа “+” и “/”. Каждые 3 исходных байта кодируются 4 символами. Благодаря этому алгоритму даже китайские иероглифы, эмоджи и изображения могут быть преобразованы в “читаемую” строку.

При отправке пользователем выбранного графического файла, копируется его URL-адрес. Далее он конвертируется в Base64 строку. Закодированное изображение помещается в JSON-объект, который затом помещается в тело заголовка для передачи сообщения серверу.

При получении сообщения клиент извлекает закодированный URL-адрес изображения, декодирует и вставляет его в атрибут src тега img языка гипертекстовой разметки HTML для отображения на веб-странице.

## 3.5 Разработка алгоритма отключения пользователя

Клиент может отсоединиться от брокера в любое время, закрыв сокет, но при этом нет гарантии, что ранее отправленные кадры были получены брокером. Для правильного разъединения, когда клиент уверен, что все предыдущие кадры были получены брокером, клиент должен:

1. Отправить кадр DISCONNECT с заголовком receipt;
2. Получить кадр RECEIPT;
3. Закрыть сокет.

Diagram

Description automatically generated

Рисунок 3.6 – схема алгоритма отключения пользователя

Так же клиенты не должны посылать больше никаких кадров после отправки кадра DISCONNECT.

# **КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 4.1 Проектирование серверной части

Данное программное средство не является привычным клиент-серверным приложением. Протокол WebSocket не является протоколом запрос – ответ, таким как HTTP, где только клиент может отправить запрос к серверу. Когда устанавливается соединение с протоколом WebSocket, клиент и сервер могут отправлять данные друг другу, до тех пор пока соединение на уровне ниже как TCP не закроется.

Сервер WebSocket — это простая программа, которая может обрабатывать события и действия WebSocket. Обычно он предоставляет методы, аналогичные API-интерфейсу клиента WebSocket. Сервер реагирует на события и выполняет действия при необходимости.

Первым делом на сервере необходимо настроить конечную точку и брокер сообщений.  Все это сделано в классе конфигурации WebSocketConfig в пакете config, который наследуется от класса WebSocketMessageBrokerConfigurer.

Класс WebSocketConfig содержит аннотацию @Configuration, чтобы указать, что это класс конфигурации библиотеки Spring. Он также имеет аннотацию @EnableWebSocketMessageBroker. Как следует из названия, она включает обработку сообщений, передаваемых брокером сообщений.

Описание основных подпрограмм, описанных в классе WebSocketConfig приведено в таблице 4.1.

Таблица 4.1. – основные подпрограммы класса WebSocketConfig

|  |  |
| --- | --- |
| **Имя подпрограммы** | **Описание** |
| registerStompEndpoints | Регистрация конечной точки, которые используют клиенты, чтобы подключится к серверу |
| configureMessageBroker | Настройка брокера сообщений, который используется для направления сообщений от одного клиента к другому |
| configureWebSocketTransport | Настройка параметров, связанных с обработкой сообщений, полученных от клиентов WebSocket и отправленных им |

Далее на сервере необходимо настроить добавление клиента, получение сообщения от одного клиента и трансляцию его всем остальным. Данные функции реализованы в пакете controller класса ChatController.

Все методы в данном классе снабжены двумя аннотациями: @MessageMapping и @SendTo. В классе конфигурации указано, что все сообщения от клиентов, направленные по определенному адресу, будут перенаправлены в соответствующие методы. Имелись в виду как раз методы, аннотированные @MessageMapping.

Аннотация @SendTo рассылает возвращаемое значение всем подписчикам на указанный пункт назначения. Ниже приведены основные подпрограммы класса ChatController.

Таблица 4.2. – основные подпрограммы класса ChatController

|  |  |
| --- | --- |
| **Имя подпрограммы** | **Описание** |
| sendMessage | Отправка сообщения |
| addUser | Добавление пользователя |

Так же пакет controller содержит класс WebSocketEventListener, в котором сервер слушает события соединения и отсоединения клиентов. Это нужно для того, чтобы логировать новые подключения и передавать в чат на всеобщее обозрение. Так все пользователи видят, когда кто-то заходит в чат или выходит из него.

Описание основных подпрограмм, описанных в классе WebSocketEventListener приведено в таблице 4.3.

Таблица 4.3. – основные подпрограммы класса ChatController

|  |  |
| --- | --- |
| **Имя подпрограммы** | **Описание** |
| handleWebSocketConnectListener | Прослушивание новых подключений к серверу |
| handleWebSocketDisconnectListener | Прослушивание отключений от сервера |

Модель сообщений находится в пакете model в классе ChatMessage. Данный класс содержит четыре приватных поля: type, content, sender и timestamp. Поле type относится к классу MessageType, который представляет собой тип перечисления.

Для идентификации сообщений используется четыре типа: chat, join, leave, file. Благодаря данной классификации как клиент, так и сервер могу различать какие данные пришли и в каком виде отображать на веб-станице.

Работу серверной части программного средства можно посмотреть и оценить, открыв в браузере инспектор кода. В консоли отображаются все кадры, которые сервер и брокер сообщений отправляют клиентам. Пример работы сервера приведен на рисунке 4.1.

Text

Description automatically generated

Рисунок 4.1 – пример работы сервера

## 4.2 Проектирование клиентской части

Клиентская часть содержится в пакете static и состоит из трёx файлов: main.js, main.css и index.html.

В файле index.html определен пользовательский интерфейс приложения на языке гипертекстовой разметки HTML. Также в нем используются библиотеки SockJS и STOMP. Пользовательский интерфейс можно разделить на две части: форма идентификации нового пользователя и страница чата.

Файл main.css содержит стили для веб-страницы, которые обеспечивают удобный, минималистичный и интуитивно понятный дизайн проектируемого средства.

Graphical user interface

Description automatically generated with medium confidence

Рисунок 4.2 – форма идентификации нового пользователя

Graphical user interface, text, application

Description automatically generated

Рисунок 4.3 – страница чата

Файл main.js является основой клиентской части, так как он необходим для соединения с конечной точкой и отправки/получения сообщений. Для реализации логики используется библиотека STOMP.js — это stomp-клиент для JavaScript.

Новый пользователь подключается к конечной точке /ws, которая настроена на сервере, при помощи функции connect. После успешного подключения, клиент подписывается на адрес /chat/messaging и сообщает серверу имя пользователя по адресу /app/chat.addUser.

Функция stompClient.subscribe принимает аргументом callback-функцию onMessageReceived, которая вызывается каждый раз, когда в тему приходит сообщение.

В функции onMessageReceived при помощи метода JSON.parse полученная строка преобразовывается в JSON-объект. Далее в зависимости от типа сообщения динамически добавляются HTML элементы в DOM-дерево веб-страницы с соответствующими стилями. Ниже приведены основные подпрограммы файла main.js.

Таблица 4.4. – основные подпрограммы файла main.js

|  |  |
| --- | --- |
| **Имя подпрограммы** | **Описание** |
| connect | Подключение нового пользователя к конечной точке |
| onConnected | Функция, которая вызывается после успешного подключения. Подписка клиента на пункт назначения и вывод сообщения о добавлении нового пользователя |
| onError | Вывод ошибки подключения к серверу |
| sendMessage | Отправка сообщения в пункт назначения |
| sendFile | Отправка файла в пункт назначения |
| onMessageReceived | Callback-функция, которая вызывается каждый раз, когда в тему приходит сообщение |
| outputJoinMessage | Добавление в DOM-дерево сообщения о подключении нового пользователя |
| outputLeaveMessage | Добавление в DOM-дерево сообщения об отключении нового пользователя |
| outputChatMessage | Добавление в DOM-дерево текста сообщения нового пользователя |

Продолжение таблицы 4.4

|  |  |
| --- | --- |
| outputFileMessage | Добавление в DOM-дерево файла |
| getAvatarColor | Получение цвета иконки пользователя |
| getMessageTime | Получение времени отправки сообщения |

# **ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Тестирование играет важную роль в разработке программного обеспечения и является неотъемлемой частью жизненного цикла его разработки, так как:

* повышает надежность, качество и производительность программного средства;
* помогает проверить, правильно ли работает программное средство, убедиться, что программа выполняет то, для чего оно предназначена;
* дает возможность сравнить реальное и ожидаемое поведение программы.

Данное тестирование проводилось на персональном компьютере с установленной операционной системой MacOS.

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Запуск и отображение веб-страницы приложения | 1.Открыть веб-браузер  2.В поисковой строке ввести URL-адрес приложения  3.Нажать кнопку поиск | Отображение главной веб-страница приложения | Тест пройден |
| 1 | Подключение нового пользователя в чат | 1.Ввести имя пользователя  2.Кликнуть на кнопку «Start Chatting» или Enter | Открытие страницы чата и вывод сообщения о добавлении нового пользователя | Тест пройден |
| 2 | Отправка сообщения | 1.Ввести сообщение в поле ввода  2.Кликнуть на кнопку «Send» или Enter | Отображения отправленного сообщения, времени отправки и имени отправителя на веб-странице | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 3 | Отправка файла | 1.Кликнуть на кнопку «File»  2.В диалоговом окне выбрать нужный файл  3. Кликнуть на кнопку «Choose for Upload» | Отображение выбранного файла, времени отправки и имени отправителя на. Веб-странице | Тест пройден |
| 4 | Отключение пользователя от чата | 1.Закрыть вкладку чата | Вывод сообщения в чат об отключении пользователя | Тест пройден |

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к пользовательскому интерфейсу не имеется. Все поставленные задачи и требования выполнены в полном объёме.

# **РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

Чат — средство обмена сообщениями по компьютерной сети в режиме реального времени, а также программное обеспечение, позволяющее организовывать такое общение. Характерной особенностью является коммуникация именно в реальном времени или близкая к этому, что отличает чат от форумов и других «медленных» средств.

Сетевой чат должен включать в себя не только информацию, передаваемую между пользователями, но и иметь доступное управление. Сетевой чат призван автоматизировать и сократить трату времени во время работы.

Данное программное средство разработано для использования на любой операционной системе.

Для того, чтобы запустить программу, нужно открыть веб-браузер и в поисковой строке ввести URL-адрес приложения. После ввода и поиска адреса появиться веб-страница с полем для идентификации нового пользователя, показанная на рисунке 6.1.

Graphical user interface

Description automatically generated with low confidence

Рисунок 6.1 – страница идентификации нового пользователя

Далее пользователь попадает на веб-страницу с чатом и выводится оповещение о его подключении к переписке.

Для отправки сообщения необходимо ввести текст в поле ввода сообщений, а затем кликнуть на кнопку «Send» или Enter. Введенное сообщение будет получено сервером и разослано всем подключенным клиентам. Данная функция приложения продемонстрирована на рисунках 6.2.-6.3.

Graphical user interface, text, application

Description automatically generated

Рисунок 6.2 –подключение пользователя

Graphical user interface, application

Description automatically generated

Рисунок 6.3 – отправка сообщения

После подключения к чату можно общаться с другими пользователями. При этом каждый пользователь имеет свою иконку, для удобства идентификации пользователей и обмена сообщениями. Данный функционал представлен на рисунке 6.4.

Graphical user interface, application, Teams

Description automatically generated

Рисунок 6.4 – общение в чате нескольких пользователей

Отправка файловых данных в чат осуществляется при клике на кнопку «File». Затем откроется диалоговое окно для выбора необходимого графического файла. После того, как пользователь выбрал нужный файл, он отправляется на сервер, а уже с сервера всем клиентам и отображается на веб-странице чата.

Пользователь так же имеет возможность передумать отправлять файл, для этого необходимо просто закрыть диалоговое окно. Демонстрация данного функционала отражена на рисунках 6.5-6.6. Общая схема работы программы представлена в приложении 1.

A screenshot of a cell phone

Description automatically generated with low confidence

Рисунок 6.5 – диалоговое окно выбора файла для отправки

Graphical user interface, application, Teams

Description automatically generated

Рисунок 6.6 – отображение отправленного файла в чате

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги, выявлены их преимущества и недостатки.

Было создано программное средство «Чат», которое предоставляет пользователю возможность обмениваться сообщениями и файлами с подключенными пользователями в режиме реального времени.

На этапе проектирования был получен опыт разработки в парадигме ООП, изучены протоколы WebSocket и STOMP, а так же библиотека Spring. Был применен шаблон MVC для разделения программы на модули.

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства. Разработано руководство пользователя.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания полностью выполнены.

# **СПИСОК ЛИТЕРАТУРЫ**

[1] CodeLessons [Электронный ресурс] – WebSockets — Краткое руководство – Режим доступа: <https://coderlessons.com/tutorials/veb-razrabotka/izuchite-veb-sokety/websockets-kratkoe-rukovodstvo>

[2] Start it up [Электронный ресурс] – WebSockets With Spring – Режим доступа: [https://medium.com/swlh/websockets-with-spring-part-3-stomp-over-websocket-3dab4a21f397](https://coderlessons.com/tutorials/veb-razrabotka/izuchite-veb-sokety/websockets-kratkoe-rukovodstvo)

[3] Wikipedia [Электронный ресурс] – STOMP – Режим доступа: <https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol>

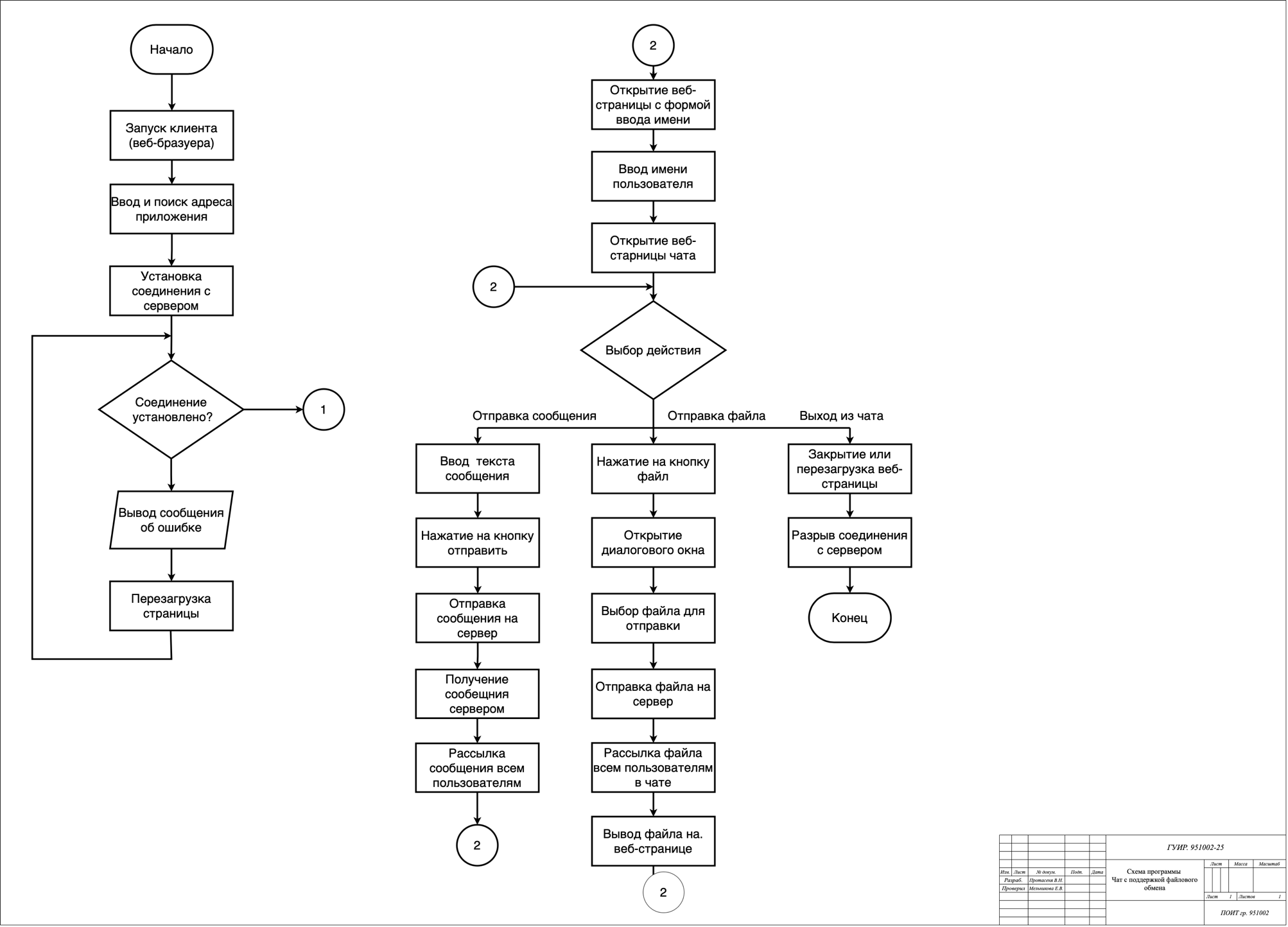
[4] Wikipedia [Электронный ресурс] – Base64 – Режим доступа: <https://ru.wikipedia.org/wiki/Base64>

[5] Э.Таненбаум, Д.Уэзеролл “Компьютерные сети” 5-е изд. (2016)

[6] NLPx [Электронный ресурс] – Немного о брокерах сообщений – Режим доступа: <http://nlpx.net/archives/566>

# **ПРИЛОЖЕНИЕ 1**

**Схема алгоритма работы программы**

****

# **ПРИЛОЖЕНИЕ 2**

**Исходный код программы**

**Пакет config**

package com.example.chat.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.messaging.simp.config.MessageBrokerRegistry;

import org.springframework.web.socket.config.annotation.\*;

/\*

Настройки WebSocket

\*/

@Configuration // конфигурационный Spring класс

@EnableWebSocketMessageBroker // включение обработки сообщений по WebSocket, возвращаемых брокером сообщений

public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

// подключение конечного адреса, по которому происходит прослушивание и передача сообщения

@Override

public void registerStompEndpoints(StompEndpointRegistry registry) {

registry.addEndpoint("/ws").withSockJS();

}

@Override

public void configureMessageBroker(MessageBrokerRegistry registry) {

registry.setApplicationDestinationPrefixes("/app");

registry.enableSimpleBroker("/chat");

}

@Override

public void configureWebSocketTransport(WebSocketTransportRegistration registration) {

registration.setMessageSizeLimit(200000000);

registration.setSendBufferSizeLimit(3 \* 1024 \* 1024);

registration.setSendTimeLimit(20000);

}

}

**Пакет controller**

package com.example.chat.controller;

import com.example.chat.model.ChatMessage;

import org.springframework.messaging.handler.annotation.MessageMapping;

import org.springframework.messaging.handler.annotation.Payload;

import org.springframework.messaging.handler.annotation.SendTo;

import org.springframework.messaging.simp.SimpMessageHeaderAccessor;

import org.springframework.stereotype.Controller;

/\*

Контроллер который прослушивает входящие сообщения и посылает исходящие

\*/

@Controller

public class ChatController {

@MessageMapping("/chat.sendMessage") // url по которому слушает

@SendTo("/chat/messaging") // url на который отправляет

public ChatMessage sendMessage(@Payload ChatMessage message) {

return message;

}

@MessageMapping("/chat.addUser")

@SendTo("/chat/messaging")

public ChatMessage addUser(@Payload ChatMessage message,

SimpMessageHeaderAccessor headerAccessor) {

headerAccessor.getSessionAttributes().put("username", message.getSender());

return message;

}

}

package com.example.chat.controller;

import com.example.chat.model.ChatMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.event.EventListener;

import org.springframework.messaging.simp.SimpMessageSendingOperations;

import org.springframework.messaging.simp.stomp.StompHeaderAccessor;

import org.springframework.stereotype.Component;

import org.springframework.web.socket.messaging.SessionConnectedEvent;

import org.springframework.web.socket.messaging.SessionDisconnectEvent;

@Component

public class WebSocketEventListener {

private static final Logger logger = LoggerFactory.getLogger(WebSocketEventListener.class);

@Autowired

private SimpMessageSendingOperations messagingTemplate;

@EventListener

public void handleWebSocketConnectListener(SessionConnectedEvent event) {

logger.info("Received a new web socket connection");

}

@EventListener

public void handleWebSocketDisconnectListener(SessionDisconnectEvent event) {

StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());

String username = (String) headerAccessor.getSessionAttributes().get("username");

if(username != null) {

logger.info("User Disconnected : " + username);

ChatMessage message = new ChatMessage();

message.setType(ChatMessage.MessageType.LEAVE);

message.setSender(username);

messagingTemplate.convertAndSend("/chat/messaging", message);

}

}

}

**Пакет model**

package com.example.chat.model;

public class ChatMessage {

private MessageType type;

private String content;

private String sender;

private String timestamp;

public enum MessageType {

CHAT,

JOIN,

LEAVE,

FILE

}

public MessageType getType() {

return type;

}

public void setType(MessageType type) {

this.type = type;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

public String getSender() {

return sender;

}

public void setSender(String sender) {

this.sender = sender;

}

public void setTimestamp(String timestamp) {

this.timestamp = timestamp;

}

public String getTimestamp() {

return timestamp;

}

}

package com.example.chat;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ChatApplication {

public static void main(String[] args) {

SpringApplication.run(ChatApplication.class, args);

}

}

**Пакет static**

**index.html**

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0">

<meta charset="utf-8">

<title>Chat</title>

<link rel="stylesheet" href="/css/main.css" />

</head>

<body>

<noscript>

<h2>Sorry! Your browser doesn't support Javascript</h2>

</noscript>

<div id="username-page">

<div class="username-page-container">

<h1 class="title">Type your username</h1>

<form id="usernameForm" name="usernameForm">

<div class="form-group">

<input type="text" id="name" placeholder="Username" autocomplete="off" class="form-control" />

</div>

<div class="form-group">

<button type="submit" class="accent username-submit">Start Chatting</button>

</div>

</form>

</div>

</div>

<div id="chat-page" class="hidden">

<div class="chat-container">

<div class="chat-header">

<h2>Chat</h2>

</div>

<div class="connecting">

Connecting...

</div>

<ul id="messageArea">

</ul>

<form id="messageForm" name="messageForm" nameForm="messageForm">

<div class="form-group">

<div class="input-group clearfix">

<input type="text" id="message" placeholder="Type a message..." autocomplete="off" class="form-control"/>

<input type="file" name="file" id="file" class="input-file" onchange="sendFile(this)" accept="image/\*"/>

<label for="file">File</label>

<button type="submit" class="primary">Send</button>

</div>

</div>

</form>

</div>

</div>

<script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.1.4/sockjs.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js"></script>

<script src="/js/main.js"></script>

</body>

</html>

**main.css**

\* {

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

box-sizing: border-box;

}

html,body {

height: 100%;

overflow: hidden;

}

body {

margin: 0;

padding: 0;

font-weight: 400;

font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;

font-size: 1rem;

line-height: 1.58;

color: #333;

background: linear-gradient(90deg, #b9deed, #efefef);

/\*background: linear-gradient(90deg, #aea4e3, #d3ffe8);\*/

height: 100%;

}

.clearfix:after {

display: block;

content: "";

clear: both;

}

.hidden {

display: none;

}

.form-control {

width: 100%;

min-height: 38px;

font-size: 15px;

border: 1px solid #c8c8c8;

}

.form-group {

margin-bottom: 15px;

}

input {

padding-left: 10px;

outline: none;

border-radius: 8px;

}

h1, h2, h3, h4, h5, h6 {

margin-top: 20px;

margin-bottom: 20px;

}

h1 {

font-size: 1.7em;

}

a {

color: #128ff2;

}

/\* buttons \*/

button {

box-shadow: none;

border: 1px solid transparent;

font-size: 14px;

outline: none;

line-height: 100%;

white-space: nowrap;

vertical-align: middle;

padding: 0.6rem 1rem;

border-radius: 8px;

transition: all 0.2s ease-in-out;

cursor: pointer;

min-height: 38px;

}

button.default {

background-color: #e8e8e8;

color: #333;

box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.12);

}

button.primary {

background-color: #128ff2;

box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.12);

color: #fff;

}

button.primary:hover {

background-color: #46c9ff;

}

button.accent {

background-color: #128ff2;

box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.12);

color: #fff;

}

button.accent:hover {

background-color: #46c9ff;

}

/\* username page\*/

#username-page {

text-align: center;

}

.username-page-container {

background: #fff;

box-shadow: 0 1px 11px rgba(0, 0, 0, 0.27);

border-radius: 30px;

width: 100%;

max-width: 500px;

display: inline-block;

margin-top: 42px;

vertical-align: middle;

position: relative;

padding: 35px 55px 35px;

min-height: 250px;

position: absolute;

top: 50%;

left: 0;

right: 0;

margin: 0 auto;

margin-top: -160px;

}

.username-page-container .username-submit {

margin-top: 10px;

}

/\* chat page \*/

#chat-page {

position: relative;

height: 100%;

}

.chat-container {

max-width: 800px;

margin-left: auto;

margin-right: auto;

background-color: #fff;

box-shadow: 0 1px 11px rgba(0, 0, 0, 0.27);

margin-top: 70px;

height: calc(100% - 60px);

max-height: 700px;

position: relative;

border-radius: 30px;

}

#chat-page ul {

list-style-type: none;

background-color: #FFF;

margin: 0;

overflow: auto;

overflow-y: scroll;

padding: 0 20px 0px 20px;

height: calc(100% - 150px);

}

#chat-page #messageForm {

padding: 20px;

}

#chat-page ul li {

line-height: 1.5rem;

padding: 10px 20px;

margin: 0;

border-bottom: 1px solid #f4f4f4;

}

#chat-page ul li p {

margin: 0;

}

#chat-page .event-message {

width: 100%;

text-align: center;

clear: both;

}

#chat-page .event-message p {

color: #777;

font-size: 14px;

word-wrap: break-word;

}

#chat-page .chat-message {

padding-left: 68px;

position: relative;

}

#chat-page .chat-message i {

position: absolute;

width: 42px;

height: 42px;

overflow: hidden;

left: 10px;

display: inline-block;

vertical-align: middle;

font-size: 18px;

line-height: 42px;

color: #fff;

text-align: center;

border-radius: 50%;

font-style: normal;

text-transform: uppercase;

}

#chat-page .chat-message span {

color: #333;

font-weight: 600;

}

#chat-page .chat-message p {

color: #43464b;

}

/\* message form \*/

#messageForm .input-group input {

float: left;

width: calc(100% - 160px);

}

#messageForm .input-group button {

float: left;

width: 80px;

height: 38px;

margin-left: 5px;

}

.chat-header {

text-align: center;

padding: 15px;

border-bottom: 1px solid #ececec;

}

.chat-header h2 {

margin: 0;

font-weight: 500;

}

.connecting {

padding-top: 5px;

text-align: center;

color: #777;

position: absolute;

top: 65px;

width: 100%;

}

/\* time string \*/

.time {

font-size: 13px;

color: #73777d;

text-align: end;

}

.send-img {

max-width: 100%;

max-height: 300px;

}

/\* input file \*/

.input-file {

width: 0.1px;

height: 0.1px;

opacity: 0;

overflow: hidden;

position: absolute;

z-index: -1;

}

.input-file + label {

color: white;

display: inline-block;

background-color: #128ff2;

box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.12);

border: 1px solid transparent;

font-size: 14px;

outline: none;

line-height: 100%;

white-space: nowrap;

vertical-align: middle;

padding: 0.6rem 1rem;

border-radius: 8px;

transition: all 0.2s ease-in-out;

cursor: pointer;

min-height: 38px;

min-width: 70px;

}

.input-file:focus + label,

.input-file + label:hover {

background-color: #ff4743;

}

.input-file + label {

cursor: pointer;

float: right;

text-align: center;

}

.input-file:focus + label {

outline: 1px dotted #000;

outline: -webkit-focus-ring-color auto 5px;

}

.input-file + label:hover {

background-color: #46c9ff;

}

/\* media \*/

@media screen and (max-width: 730px) {

.chat-container {

margin-left: 10px;

margin-right: 10px;

margin-top: 10px;

}

}

@media screen and (max-width: 480px) {

.chat-container {

height: calc(100% - 30px);

}

.username-page-container {

width: auto;

margin-left: 15px;

margin-right: 15px;

padding: 25px;

}

#chat-page ul {

height: calc(100% - 120px);

}

#messageForm .input-group button {

width: 65px;

}

#messageForm .input-group input {

width: calc(100% - 70px);

}

.chat-header {

padding: 10px;

}

.connecting {

top: 60px;

}

.chat-header h2 {

font-size: 1.1em;

}

}

**main.js**

'use strict';

const usernamePage = document.querySelector('#username-page');

const chatPage = document.querySelector('#chat-page');

const usernameForm = document.querySelector('#usernameForm');

const messageForm = document.querySelector('#messageForm');

const messageInput = document.querySelector('#message');

const messageArea = document.querySelector('#messageArea');

const connectingElement = document.querySelector('.connecting');

const onlineUsersArea = document.querySelector('#online-users');

var stompClient = null;

var username = null;

var avatarColors = [

'#2196F3', '#32c787',

'#00BCD4', '#ff5652',

'#ffc107', '#ff85af',

'#FF9800', '#39bbb0',

'#97b1f3', '#f782e7',

'#addfee', '#b967ff',

'#ffdc73', '#eabec8',

];

function connect(event) {

// получение введенного имени из формы

username = document.querySelector('#name').value.trim();

if(username) {

usernamePage.classList.add('hidden');

chatPage.classList.remove('hidden');

// создание экзмепляра для подключения к веб сокетам на сервере

var socket = new SockJS('/ws');

stompClient = Stomp.over(socket);

stompClient.connect({}, onConnected, onError);

}

event.preventDefault();

}

function onConnected() {

// подписка клиента на url /chat/messages

// таким образом он будет слушать все, что придет по этому адресу без перезагрузки страницы

stompClient.subscribe('/chat/messaging', onMessageReceived);

stompClient.send("/app/chat.addUser",

{},

JSON.stringify({sender: username, type: 'JOIN'})

)

connectingElement.classList.add('hidden');

}

function onError(error) {

connectingElement.textContent = 'Could not connect to WebSocket server. Please refresh this page to try again!';

connectingElement.style.color = 'red';

}

function sendMessage(event) {

var messageContent = messageInput.value.trim();

if(messageContent && stompClient) {

var chatMessage = {

sender: username,

content: messageInput.value,

timestamp: getMessageTime(),

type: 'CHAT',

};

stompClient.send("/app/chat.sendMessage", {}, JSON.stringify(chatMessage));

messageInput.value = '';

}

// отмена действия браузера по умолчанию

event.preventDefault();

}

function sendFile(fileInput) {

var reader = new FileReader();

reader.readAsDataURL(fileInput.files[0]);

reader.onload = function () {

console.log(reader.result);

var chatMessage = {

sender: username,

content: reader.result,

timestamp: getMessageTime(),

type: 'FILE',

};

console.log(chatMessage);

stompClient.send("/app/chat.sendMessage", {}, JSON.stringify(chatMessage));

};

// отмена действия браузера по умолчанию

event.preventDefault();

}

function onMessageReceived(payload) {

var message = JSON.parse(payload.body);

switch (message.type) {

case 'JOIN':

outputJoinMessage(message);

break;

case 'LEAVE':

outputLeaveMessage(message);

break;

case 'CHAT':

outputChatMessage(message);

break;

case 'FILE':

outputFileMessage(message);

break;

}

}

function outputJoinMessage(message) {

var messageElement = document.createElement('li');

messageElement.classList.add('event-message');

message.content = message.sender + ' joined!';

var textElement = document.createElement('p');

var messageText = document.createTextNode(message.content);

textElement.appendChild(messageText);

messageElement.appendChild(textElement);

messageArea.appendChild(messageElement);

messageArea.scrollTop = messageArea.scrollHeight;

}

function outputLeaveMessage(message) {

var messageElement = document.createElement('li');

messageElement.classList.add('event-message');

message.content = message.sender + ' left!';

var textElement = document.createElement('p');

var messageText = document.createTextNode(message.content);

textElement.appendChild(messageText);

messageElement.appendChild(textElement);

messageArea.appendChild(messageElement);

messageArea.scrollTop = messageArea.scrollHeight;

}

function outputChatMessage(message) {

var messageElement = document.createElement('li');

messageElement.classList.add('chat-message');

// вывод аватарки пользователя

var avatarElement = document.createElement('i');

var avatarText = document.createTextNode(message.sender[0]);

avatarElement.appendChild(avatarText);

avatarElement.style['background-color'] = getAvatarColor(message.sender);

messageElement.appendChild(avatarElement);

// вывод имени пользователя

var usernameElement = document.createElement('span');

var usernameText = document.createTextNode(message.sender);

usernameElement.appendChild(usernameText);

messageElement.appendChild(usernameElement);

// вывод текста сообщения

var textElement = document.createElement('p');

var messageText = document.createTextNode(message.content);

textElement.appendChild(messageText);

console.log(message);

messageElement.appendChild(textElement);

// вывод времени сообщения

var messageTimestamp = document.createElement('p');

messageTimestamp.classList.add('time');

var dateText = document.createTextNode(message.timestamp);

messageTimestamp.appendChild(dateText);

messageElement.appendChild(messageTimestamp);

messageArea.appendChild(messageElement);

messageArea.scrollTop = messageArea.scrollHeight;

}

function outputFileMessage(message) {

var messageElement = document.createElement('li');

messageElement.classList.add('chat-message');

// вывод аватарки пользователя

var avatarElement = document.createElement('i');

var avatarText = document.createTextNode(message.sender[0]);

avatarElement.appendChild(avatarText);

avatarElement.style['background-color'] = getAvatarColor(message.sender);

messageElement.appendChild(avatarElement);

// вывод имени пользователя

var usernameElement = document.createElement('span');

var usernameText = document.createTextNode(message.sender);

usernameElement.appendChild(usernameText);

messageElement.appendChild(usernameElement);

// вывод файла

var image = document.createElement('img');

image.classList.add('send-img');

image.src = message.content;

var imageWrapper = document.createElement('div');

imageWrapper.appendChild(image);

messageElement.appendChild(imageWrapper);

// вывод времени сообщения

var messageTimestamp = document.createElement('p');

messageTimestamp.classList.add('time');

var dateText = document.createTextNode(message.timestamp);

messageTimestamp.appendChild(dateText);

messageElement.appendChild(messageTimestamp);

messageArea.appendChild(messageElement);

messageArea.scrollTop = messageArea.scrollHeight;

}

function getAvatarColor(messageSender) {

var hash = 0;

for (var i = 0; i < messageSender.length; i++) {

hash = 31 \* hash + messageSender.charCodeAt(i);

}

var index = Math.abs(hash % avatarColors.length);

return avatarColors[index];

}

function getMessageTime() {

var now = new Date();

return now.toLocaleString("ru");

}

usernameForm.addEventListener('submit', connect, true);

messageForm.addEventListener('submit', sendMessage, true);

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 225 ПЗ | | | | Пояснительная записка | | | | 50 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951002 225 СП | | | | «Чат с поддержкой файлового обмена»,А1, схема программы, чертеж | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 225 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | «Чат с поддержкой  файлового обмена»  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Протасеня.В.Н. |  |  | Т |  | |  | 50 | 50 |
| Пров. | | Мельникова Е.В. |  |  | Кафедра ПОИТ  гр. 951002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |