Chat Application Final Report

Title: Chat Application with WebSocket Support

Author: [Tocan Robert-Alexandru(Group 30432/2); Dragoi Cosmin(Group 30432/1)]

Abstract

This report presents the design and implementation of a chat application built using Spring Boot with WebSocket support. The application facilitates real-time messaging between users and includes features such as user join/leave notifications and typing status updates. The project uses a publish/subscribe messaging model via STOMP (Simple Text Oriented Messaging Protocol) over WebSockets. The Java Collections API was utilized for managing data structures such as lists for user mentions and session data. The report provides an overview of the system's design, implementation, and testing, including UML diagrams, source code, and experimental results.

Team:

- **[Tocan Robert-Alexandru]**
- **[Dragoi Cosmin]**

A. Final Project

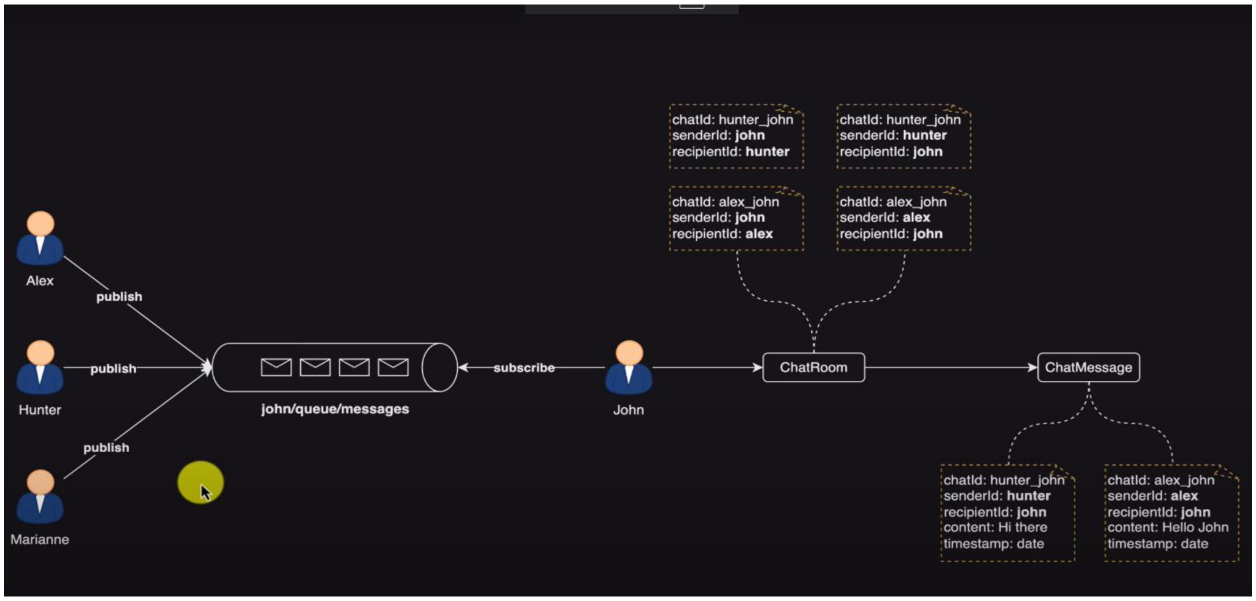
1. Design

The design of the chat application includes the following key components:

a. Use Case Diagram

The Use Case Diagram below outlines the main interactions between users and the chat application:

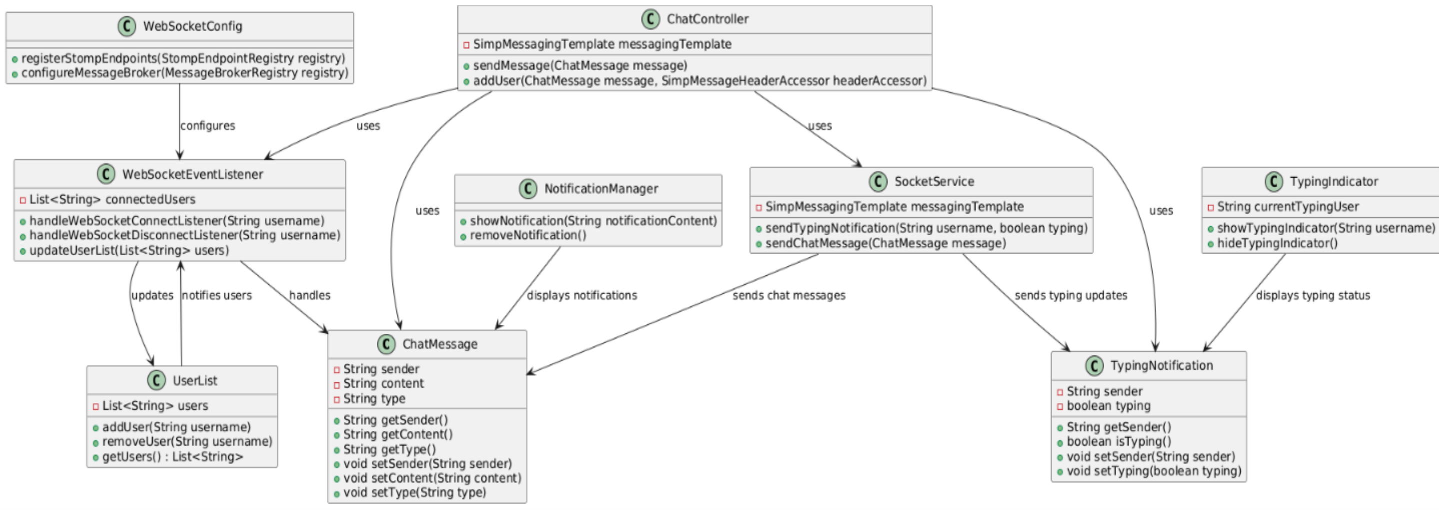
- **Actors: Users**
- **Use Cases: Send messages, Join chat, Leave chat, Receive notifications, View typing indicators.**



b. Class Diagram

The Class Diagram represents the structure of the application's main components, including:

- **ChatMessage: Represents chat messages.**
- **TypingNotification: Represents typing status updates.**
- **ChatController: Handles WebSocket message routing.**
- **WebSocketEventListener: Handles WebSocket lifecycle events.**

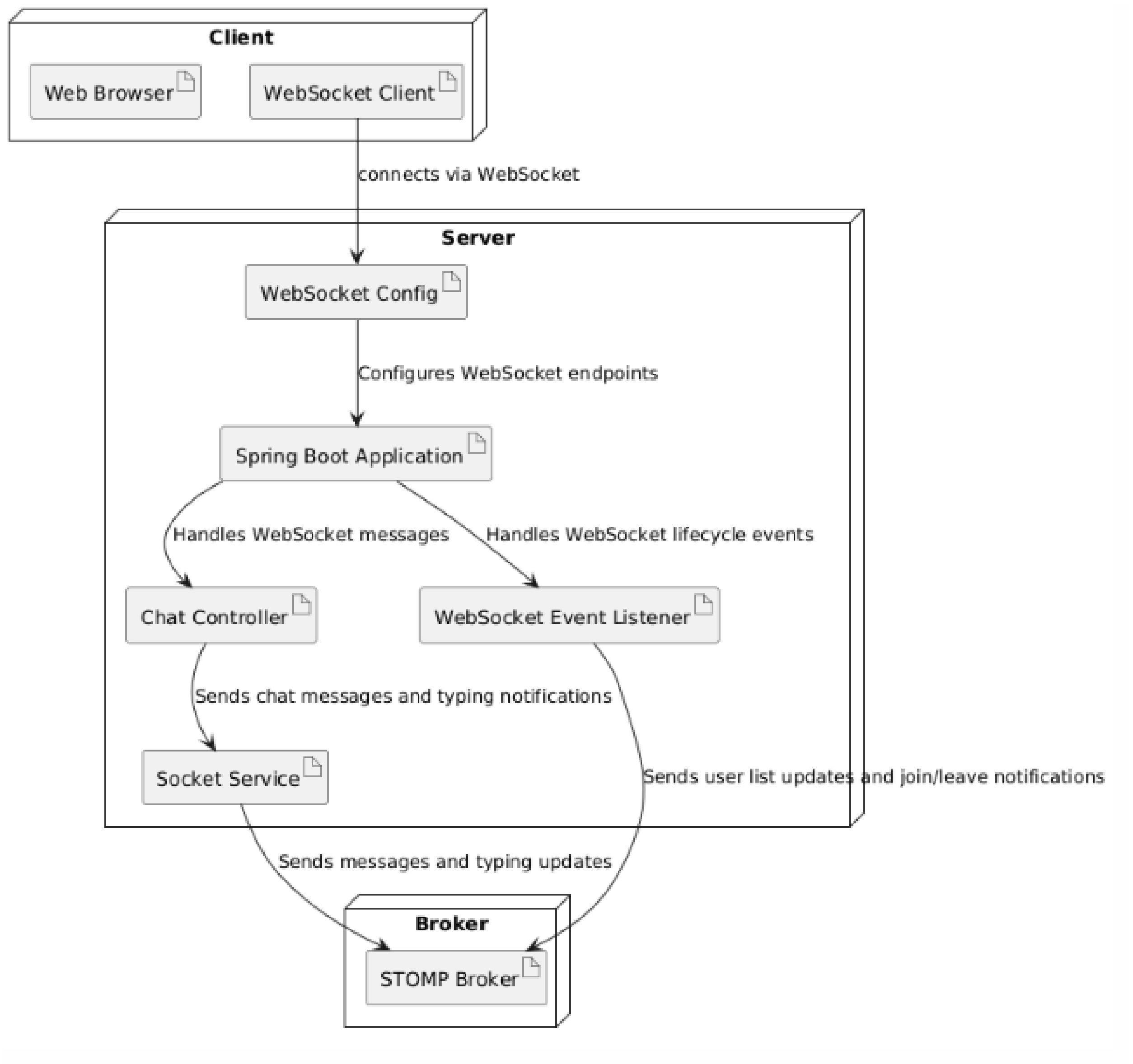


WebSocketConfig
registerStompEndpoints(StompEndpointRegistry registry)
configureMessageBroker(MessageBrokerRegistry registry)
ChatController
SimpMessagingTemplate messagingTemplate
sendMessage(ChatMessage message)
addUser(ChatMessage message, SimpMessageHeaderAccessor headerAccessor)
configures
uses
uses
uses
uses
WebSocketEventListener
List<String> connectedUsers
handleWebSocketConnectListener(String username)
handleWebSocketDisconnectListener(String username)
updateUserList(List<String> users)
NotificationManager
showNotification(String notificationContent)
removeNotification()
SocketService
SimpMessagingTemplate messagingTemplate
sendTypingNotification(String username, boolean typing)
sendChatMessage(ChatMessage message)
TypingIndicator
String currentTypingUser
showTypingIndicator(String username)
hideTypingIndicator()
updates
notifies users
handles
displays notifications
sends chat messages
sends typing updates
displays typing status
UserList
List<String> users
addUser(String username)
removeUser(String username)
getUsers() : List<String>
ChatMessage
String sender
String content
String type
String getSender()
String getContent()
String getType()
void setSender(String sender)
void setContent(String content)
void setType(String type)
TypingNotification
String sender
boolean typing
String getSender()
boolean isTyping()
void setSender(String sender)
void setTyping(boolean typing)

c. Deployment Diagram

The Deployment Diagram illustrates the physical deployment of the application:

- **Client: Connects to the WebSocket endpoint using a web browser or other WebSocket client.**
- **Server: Hosts the Spring Boot application.**
- **Broker: Provides message routing via STOMP.**


Client
Web Browser
WebSocket Client
connects via WebSocket
Server
WebSocket Config
Configures WebSocket endpoints
Spring Boot Application
Handles WebSocket messages
Handles WebSocket lifecycle events
Chat Controller
WebSocket Event Listener
Sends chat messages and typing notifications
Socket Service
Sends user list updates and join/leave notifications
Sends messages and typing updates
Broker
STOMP Broker

2. Experimental Results

The application was tested for the following scenarios:

- **Real-time messaging: Verified that messages sent by one user are received by all other connected users.**
- **User join/leave notifications: Confirmed that join and leave events are broadcast to all users.**
- **Typing notifications: Verified that typing indicators are displayed in real-time.**
- **List of active users**

Performance testing showed low latency and efficient handling of multiple simultaneous connections.

B. Final Project Implementation

1. Source Code

The source code for the project is organized into several key components. This section provides an overview of each part of the code and its functionality.

---

a. Backend Code (Spring Boot)

ChatController.java

This class handles WebSocket messages, including sending and receiving chat messages and handling user join and leave events.

```
package com.tocosm.chat.chat;

import com.tocosm.chat.config.WebSocketEventListener;
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.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Controller;
```

```
@Controller
public class ChatController {

    private final WebSocketEventListener webSocketEventListener;
    private final SimpMessagingTemplate messagingTemplate;

    public ChatController(WebSocketEventListener webSocketEventListener,
SimpMessagingTemplate messagingTemplate) {
        this.webSocketEventListener = webSocketEventListener;
        this.messagingTemplate = messagingTemplate;
    }

    // Handle sending chat messages
    @MessageMapping("/chat.sendMessage")
    @SendTo("/topic/public")
    public ChatMessage sendMessage(@Payload ChatMessage chatMessage) {
        return chatMessage;
    }

    // Handle user joining the chat (this is done only once when they first connect)
    @MessageMapping("/chat.addUser")
    public void addUser(@Payload ChatMessage chatMessage, SimpMessageHeaderAccessor
headerAccessor) {
        // Add username in the WebSocket session
        headerAccessor.getSessionAttributes().put("username", chatMessage.getSender());

        // Notify everyone that the user joined
        webSocketEventListener.handleWebSocketConnectListener(chatMessage.getSender());
    }

    // Handle typing notifications (now using TypingNotification)
    @MessageMapping("/chat.typing")
    @SendTo("/topic/typing")
    public TypingNotification handleTypingNotification(@Payload ChatMessage chatMessage) {
        // Check if the user is typing or has stopped typing
        boolean isTyping = chatMessage.getContent() != null &&
!chatMessage.getContent().isEmpty();
```

```
        // Send the typing status to the client(s)
        TypingNotification typingNotification = new TypingNotification();
        typingNotification.setSender(chatMessage.getSender());
        typingNotification.setTyping(isTyping);

        messagingTemplate.convertAndSend("/topic/typing", typingNotification);

        return typingNotification;
    }
}
```

WebSocketConfig.java

This configuration class sets up WebSocket endpoints and configures the message broker for handling communication via STOMP.

```
package com.tocosm.chat.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import
org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws").withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.setApplicationDestinationPrefixes("/app");
        registry.enableSimpleBroker("/topic");
    }
}
```

ChatMessage.java

This class represents the structure of chat messages, containing properties like sender, content, and type (e.g., CHAT, JOIN, LEAVE).

```
package com.tocosm.chat.chat;

import java.util.List;

public class ChatMessage {

    private MessageType type;
    private String content;
    private String sender;
    private List<String> mentionedUsers; // List of users mentioned in the message
private boolean typing;
    // Default constructor
    public ChatMessage() {
    }

    public ChatMessage(MessageType type, String content, String sender, List<String>
mentionedUsers, boolean typing) {
    }

    // All-args constructor
    public ChatMessage(MessageType type, String content, String sender, List<String>
mentionedUsers) {
        this.type = type;
        this.content = content;
        this.sender = sender;
        this.mentionedUsers = mentionedUsers;
        this.typing=typing;
    }

    // Getters and setters
    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 boolean isTyping() {
    return typing;
}
public List<String> getMentionedUsers() {
    return mentionedUsers;
}

public void setMentionedUsers(List<String> mentionedUsers) {
    this.mentionedUsers = mentionedUsers;
}

// Builder pattern (if you want to keep this feature)
public static class Builder {
    private MessageType type;
    private String content;
    private String sender;
```

```
        private List<String> mentionedUsers;
        private boolean typing;
        public Builder type(MessageType type) {
            this.type = type;
            return this;
        }

        public Builder content(String content) {
            this.content = content;
            return this;
        }

        public Builder sender(String sender) {
            this.sender = sender;
            return this;
        }
        public Builder typing(boolean typing) {
            this.typing = typing;
            return this;
        }

        public Builder mentionedUsers(List<String> mentionedUsers) {
            this.mentionedUsers = mentionedUsers;
            return this;
        }

        public ChatMessage build() {
            return new ChatMessage(type, content, sender, mentionedUsers, typing);
        }
    }
}
```

WebSocketEventListener.java

This class listens for WebSocket lifecycle events such as user connections and disconnections and updates the list of connected users accordingly.

```
package com.tocosm.chat.config;

import com.tocosm.chat.chat.ChatMessage;
import com.tocosm.chat.chat.MessageType;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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.SessionDisconnectEvent;

import java.util.ArrayList;
import java.util.List;

@Component
public class WebSocketEventListener {

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

    private final SimpMessageSendingOperations messagingTemplate;

    // Constructor for dependency injection
    public WebSocketEventListener(SimpMessageSendingOperations messagingTemplate) {
        this.messagingTemplate = messagingTemplate;
    }

    // Handling disconnect events (user leaving the chat)
    @EventListener
    public void handleWebSocketDisconnectListener(SessionDisconnectEvent event) {
        StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());
        String username = (String) headerAccessor.getSessionAttributes().get("username");

        if (username != null) {
            log.info("User disconnected: {}", username);
            // Creating a message for the user who left
            List<String> mentionedUsers = new ArrayList<>(); // Empty list, since this is a "leave"
message
            ChatMessage chatMessage = new ChatMessage(MessageType.LEAVE, username + " left
the chat", username, mentionedUsers);

            // Send the "leave" message to the public topic
            messagingTemplate.convertAndSend("/topic/public", chatMessage);
        }
    }

    // Function to handle when a new user connects
    public void handleWebSocketConnectListener(String username) {
        if (username != null) {
            log.info("User connected: {}", username);

            // Creating a message for the user who joined
            List<String> mentionedUsers = new ArrayList<>(); // Empty list, since this is a "join"
message
            ChatMessage chatMessage = new ChatMessage(MessageType.JOIN, username + " joined
the chat", username, mentionedUsers);

            // Send the "join" message to the public topic
            messagingTemplate.convertAndSend("/topic/public", chatMessage);
        }
    }
}
```

MessageType.java

Defines different message types, such as CHAT, JOIN, LEAVE, etc., that help differentiate the kinds of messages sent over the WebSocket.

```
package com.tocosm.chat.chat;

public enum MessageType {
   CHAT,
   JOIN,
   NOTIFY, NOTIFICATION, LEAVE
}
```

TypingNotification.java

This class handles the typing status notification, indicating when a user is typing or has stopped typing.

```
package com.tocosm.chat.chat;

public class TypingNotification {
   private String sender;
   private boolean typing;

   // Getters and Setters
   public String getSender() {
      return sender;
   }

   public void setSender(String sender) {
      this.sender = sender;
   }

   public boolean isTyping() {
      return typing;
   }

   public void setTyping(boolean typing) {
      this.typing = typing;
   }
}
```

<u>ChatApplication.java</u>

This is the main class for the Spring Boot application that starts the server and manages the application's lifecycle.

Place your ChatApplication code here.

```
package com.tocosm.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);
   }

}
```

b. Frontend Code (HTML, JavaScript, CSS)

index.html

This is the main HTML file for the chat application, including the chat interface, input fields, and the user list sidebar.

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-
scale=1.0">
  <title>Spring Boot WebSocket Chat Application</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 to enter the Chatroom</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>Spring WebSocket ChatApp</h2>
        </div>
        <div class="connecting">
            Connecting...
        </div>
        <ul id="messageArea">
        </ul>
        <form id="messageForm" name="messageForm">
            <div class="form-group">
                <div class="input-group clearfix">
                    <input type="text" id="message" placeholder="Type a message..."
autocomplete="off" class="form-control"/>
                    <button type="submit" class="primary">Send</button>
                </div>
            </div>
        </form>
    </div>
</div>

<!-- Popup Notification -->
<div id="notification-popup" class="notification-popup hidden">
    <p id="notification-text"></p>
</div>

<!-- Sidebar for connected users -->
<div id="user-list" class="user-list">
    <h3>Connected Users</h3>
    <ul id="users"></ul>
</div>
<div id="typing-indicator">Waiting for typing...</div>
  <!-- Typing indicator -->

<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.js

This JavaScript file is responsible for managing the WebSocket connection, handling user input, updating the message area, and managing the user list.

```
'use strict';

var usernamePage = document.querySelector('#username-page');
var chatPage = document.querySelector('#chat-page');
var usernameForm = document.querySelector('#usernameForm');
var messageForm = document.querySelector('#messageForm');
var messageInput = document.querySelector('#message');
var messageArea = document.querySelector('#messageArea');
var connectingElement = document.querySelector('.connecting');

// New elements for the user list
var userList = document.querySelector('#users');  // The list of connected users
var userSidebar = document.querySelector('#user-list');  // Sidebar for users

var stompClient = null;
var username = null;

var colors = [
    '#2196F3', '#32c787', '#00BCD4', '#ff5652',
    '#ffc107', '#ff85af', '#FF9800', '#39bbb0'
];

var connectedUsers = [];  // Array to hold the connected users

function connect(event) {
    username = document.querySelector('#name').value.trim();
    if (username) {
        usernamePage.classList.add('hidden');
        chatPage.classList.remove('hidden');
        userSidebar.classList.add('show');  // Show the user list sidebar

        var socket = new SockJS('/ws');
        stompClient = Stomp.over(socket);

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

function onConnected() {
    // Subscribe to the Public Topic, User List Update, and Typing Indicator
    stompClient.subscribe('/topic/public', onMessageReceived);
    stompClient.subscribe('/topic/userList', onUserListUpdate);
    stompClient.subscribe('/topic/typing', onTypingMessageReceived); // Subscribe to typing
indicator

    // Tell your username to the server
    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,
            type: 'CHAT'
        };
        stompClient.send("/app/chat.sendMessage", {}, JSON.stringify(chatMessage));
        messageInput.value = '';
    }
    event.preventDefault();
}

function onMessageReceived(payload) {
    var message = JSON.parse(payload.body);
    var messageElement = document.createElement('li');

    // Check if the message is a 'JOIN' or 'LEAVE' event
    if (message.type === 'JOIN') {
        messageElement.classList.add('event-message');
        message.content = message.sender + ' joined!';

        // Add user to the connected users list
        if (!connectedUsers.includes(message.sender)) {
            connectedUsers.push(message.sender);
            updateUserList();
        }
    } else if (message.type === 'LEAVE') {
        messageElement.classList.add('event-message');
        message.content = message.sender + ' left!';

        // Remove user from the connected users list
        connectedUsers = connectedUsers.filter(user => user !== message.sender);
        updateUserList();
    } else {
        // Handle regular chat messages
        messageElement.classList.add('chat-message');

        // Display the avatar (first letter of the sender's name)
        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);

        var usernameElement = document.createElement('span');
        var usernameText = document.createTextNode(message.sender);
        usernameElement.appendChild(usernameText);
        messageElement.appendChild(usernameElement);

        // Check for mentions (for this user)
        if (message.content.includes('@' + username)) {
            showNotification("You were mentioned in a message by " + message.sender);
        }
    }

    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 onUserListUpdate(payload) {
    // Update the user list when the server sends the updated list
    connectedUsers = JSON.parse(payload.body);
    updateUserList();
}

function updateUserList() {
    userList.innerHTML = '';  // Clear the list

    connectedUsers.forEach(function(user) {
        var userElement = document.createElement('li');
        userElement.textContent = user;
        userList.appendChild(userElement);
    });
}

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 % colors.length);
    return colors[index];
}

// Function to show a notification (customize this as needed)
function showNotification(notificationContent) {
    // Create the notification element
    var notificationElement = document.createElement('div');
    notificationElement.classList.add('notification-popup');
    notificationElement.innerHTML = '<p>' + notificationContent + '</p>';

    // Add it to the body
    document.body.appendChild(notificationElement);

    // Make the notification visible
    notificationElement.style.display = 'block';

    // Remove the notification after the animation
    setTimeout(function() {
        notificationElement.style.display = 'none';
        notificationElement.remove();
    }, 5000); // Hide after 5 seconds
}

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

var typingTimeout = null; // Timeout variable to detect when the user stops typing
var typingElement = document.querySelector('#typing-indicator'); // The div that will show the
typing indicator

// Function to detect when a user starts typing
var typingTimeout = null; // Timeout variable to detect when the user stops typing

function handleTyping() {
  // Send a typing notification to the server
  stompClient.send("/app/chat.typing", {}, JSON.stringify({sender: username, typing: true}));

  // Clear any previous timeout for typing stop detection
  clearTimeout(typingTimeout);

  // Set a timeout to indicate when the user has stopped typing (2 seconds delay)
  typingTimeout = setTimeout(function() {
    stompClient.send("/app/chat.typing", {}, JSON.stringify({sender: username, typing: false}));
  }, 2000); // After 2 seconds of inactivity, send the stop typing notification
}

// Listen for keypress in the message input field
messageInput.addEventListener('input', handleTyping);

// Listen for typing indicator messages from the server
function onTypingMessageReceived(payload) {
  var message = JSON.parse(payload.body);

  if (message.typing) {
    // Show the typing indicator with the user's name
    typingElement.innerHTML = message.sender + ' is typing...';
  } else {
    // Hide the typing indicator when the user stops typing
    typingElement.innerHTML = '';
  }
}
```

main.css

This CSS file is used to style the application, including the chat window, user list, and notifications.

```css
/* General box-sizing reset */
* {
   -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-color: #f4f4f4;
   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;
}

h1, h2, h3, h4, h5, h6 {
   margin-top: 20px;
   margin-bottom: 20px;
}

h1 {
   font-size: 1.7em;
}

a {
   color: #6db33f;
}

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: 2px;
    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: #6db33f;
    box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.12);
    color: #fff;
}

button.accent {
    background-color: #6db33f;
    box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.12);
    color: #fff;
}

#username-page {
    text-align: center;
}

.username-page-container {
    background: #fff;
    box-shadow: 0 1px 11px rgba(0, 0, 0, 0.27);
    border-radius: 2px;
    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 {
    position: relative;
    height: 100%;
}

.chat-container {
    max-width: 700px;
    margin-left: auto;
    margin-right: auto;
    background-color: #fff;
    box-shadow: 0 1px 11px rgba(0, 0, 0, 0.27);
    margin-top: 30px;
    height: calc(100% - 60px);
    max-height: 600px;
    position: relative;
}

#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;
}

#messageForm .input-group input {
   float: left;
   width: calc(100% - 85px);
}

#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%;
}

@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;
    }

}

/* Style for Notification Popup */
.notification-popup {
    position: fixed;
    top: 20px;  /* Distance from the top of the page */
    right: 20px;  /* Distance from the right of the page */
    background-color: rgba(0, 123, 255, 0.9);
    color: white;
    padding: 15px;
    border-radius: 10px;
    font-size: 16px;
```

```css
    z-index: 9999;
    display: none; /* Initially hidden */
    animation: fadeIn 0.5s ease-in-out, fadeOut 2s 3s ease-in-out;
}


.notification-popup p {
    margin: 0;
}

@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}

@keyframes fadeOut {
    from {
        opacity: 1;
    }
    to {
        opacity: 0;
    }
}
/* Sidebar for connected users */
.user-list {
    position: fixed;
    top: 20px;
    left: 20px;
    background-color: rgba(0, 0, 0, 0.7);
    color: white;
    padding: 15px;
    border-radius: 10px;
    font-size: 16px;
    max-height: 90%;
    overflow-y: auto;
    z-index: 999;
    width: 200px;
    display: none; /* Initially hidden */
}

.user-list h3 {
    margin-top: 0;
    font-size: 18px;
    font-weight: bold;
}

.user-list ul {
    padding-left: 10px;
    list-style-type: none;
}

.user-list ul li {
    margin: 10px 0;
}

.user-list.show {
    display: block; /* Show the sidebar when needed */
}
```

c. Java Collections API

The application leverages the Java Collections API to manage dynamic data structures effectively:

- List: Used for storing user mentions in ChatMessage.
- Map: Used for session attributes in WebSocketEventListener.
- ArrayList: Utilized for maintaining typing notifications and user-related data.

2. Testing and Verification

The application was tested using manual and automated tests to ensure functionality and robustness. Example test cases:

- Test Case 1: User sends a message.
- Test Case 2: User joins or leaves the chat.
- Test Case 3: Typing indicator updates.

Appendix: Mini Project

Source Code (Mini Project Only)

This mini project was an isolated test of WebSocket communication, created to test sending and receiving messages between two clients before integrating it into the full chat application. Below is the source code for the WebSocket server:

```
public class WebSocketServer {
  public static void main(String[] args) {
    Server server = new Server();
    server.start();

    server.onMessageReceived((message) -> {
      System.out.println("Received message: " + message);
      // Process message
      server.sendMessage("Reply to: " + message);
    });
  }
}
```

C. References

1. **Spring Boot Documentation: https://spring.io/projects/spring-boot**
2. **STOMP Protocol Documentation: https://stomp.github.io**
3. **WebSocket Standards: https://tools.ietf.org/html/rfc6455**
4. **Java Collections Framework: https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html**

End of Document