“REAL-TIME CHAT APPLICTION”

A Project report submitted in the partial fulfillment the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (2024-2025)

BY

S.SURESH	**REGNO:221801340007**
P.GANIBABU	**REGNO:221801340016**
S.RAKESH	**REGNO:221801340014**
V.JAGADESH	**REGNO:221801340018**

Under the esteemed Guidance of

Mr. PAWAN KALYAN
Asst.Professor

Centurion UNIVERSITY

Shaping Lives...
Empowering Communities...

CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT

ANDHRA PRADESH

ROLLAVAKA VILLAGE, TEKKALI MANDAL 535 003(2024-2025)

CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT

ANDHRA PRADESH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Centurion UNIVERSITY

Shaping Lives...
Empowering Communities...

BONAFIDE CERTIFICATE

This is to certify that the project work entitled "REAL TIME CHAT APPLICATION" of project work done by S.Suresh(221801340007),P.GaniBabu(221801340016),S.Rakesh(221801340014),

V.Jagadesh(221801340018) under the esteemed Guidance of for the award the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING, CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT, during the academic year 2024-2025.

INTERNALGUIDE

Mr. Pawan Kalyan
Asst.Professor
Dept. of CSE

HEAD OF THE DEPARTMENT

Mr. Subrat Kumar Parida
Asst.Professor
Dept. of CSE

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

It is with at most pleasure and excitement we submit our project partial fulfillment of the requirement for the award of Bachelor of Technology.

The project is a result to the cumulate efforts, support, guidance, encouragement and inspiration from many of those for whom we have to give our truthful honor and express gratitude through bringing out this project at the outset as per our knowledge.

We convey my special thanks to our project Guide **Mr. PAWAN KALYAN, M. Tech** (CSE) who has guided, encouraged and tremendously supported me to enhance my knowledge with present working of this project to bring out enriching the quality of project.

We express my appreciativeness to **Mr. Subrat Kumar Parida Asst. Prof. and Head of the Department of CSE,** who facilitated us to providing the friendly environment which helped to enhance my skills in present project.

We convey my sincere thanks to **Dr.P.PALLAVI, Registrar of Centurion University of Technology and Management** who provided us with an opportunity to take on project work in well-equipped laboratories of Computer Science Department in our college.

At the outset, we thank to **Sri. PRASANTH KUMAR MOHANTY**, beloved **Vice Chancellor of Centurion University of Technology and Management** who is the back bone by providing for completion of this project, Thank you sir.

DECLARATION

We hereby declare that the project entitled "**REAL TIME CHAT APPLICATION**" submitted to the fulfilment of award the Degree of **B.TECH (CSE)** at **CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT** (A.P). This project work in original has not been submitted so far in any part or full for any other university or institute for the award of any Degree or Diploma.

S.SURESH	221801340007
P.GANI BABU	221801340016
S.RAKESH	221801340014
V.JAGADESH	221801340018

ABSTRACT

This project presents a real-time chat application using Spring Boot and WebSocket to facilitate instant, bidirectional communication between users. By integrating WebSocket with Spring Boot, the application provides a seamless messaging experience without the need for continuous HTTP requests, allowing users to send and receive messages in real time. The backend, powered by Spring Boot, ensures efficient message routing, user authentication, and message persistence, enhancing both performance and security.

The application architecture enables various communication modes, including one-to-one private messaging, group chats, and broadcast messaging, making it versatile for personal, educational, and professional use. With features like read receipts, user authentication, and message storage, the chat system ensures a secure and user-friendly experience. The platform is also designed for scalability, supporting future enhancements such as multimedia sharing, push notifications, and advanced security features.

This chat application demonstrates the effective use of WebSocket technology in building real-time communication systems and showcases the advantages of Spring Boot as a reliable, scalable backend solution. Through this project, we highlight how modern web technologies can create efficient, interactive, and scalable communication applications tailored to meet diverse user needs.

LITERATURE SURVEY

1. Real-Time Communication and WebSocket Technology

Real-time chat applications have become fundamental in facilitating instantaneous communication in areas like social networking, customer service, and collaborative work environments. Traditional HTTP-based messaging applications, which rely on frequent polling or long-polling, have performance limitations when handling large volumes of messages simultaneously. WebSocket technology addresses these constraints by providing a persistent, bidirectional communication channel between clients and servers, significantly improving responsiveness. Studies indicate that WebSocket reduces server load and minimizes network traffic, making it a preferred choice for real-time messaging applications (e.g., [Author et al.]). Leveraging WebSocket, our project ensures instant message delivery, optimizing communication without the need for excessive HTTP requests.

2. Spring Boot for Real-Time Applications

Spring Boot has gained traction as a framework for building robust, enterprise-grade applications with minimal configuration. Its integration with WebSocket provides developers with a streamlined way to implement real-time communication features. Research by [Researcher et al.] highlights the utility of Spring Boot's "opinionated" setup, which simplifies application development and enhances scalability and maintainability. In real-time communication, Spring Boot's built-in support for WebSocket allows seamless handling of concurrent users, and its compatibility with advanced Java facilitates modular application design.

3. Security in Real-Time Chat Applications

Security is a critical aspect of chat applications, given that sensitive data is frequently exchanged between users. Various studies emphasize the importance of secure messaging practices, such as user authentication, data encryption, and access control to protect against unauthorized access. Research by [Researcher et al.] shows that Spring Security—a module within the Spring

ecosystem—effectively manages authentication and authorization in real-time applications. Using Spring Security in our project enables secure communication through role-based access control, session management, and data encryption, addressing security and privacy concerns essential in real-time messaging environments.

4. Enhancing User Experience with Real-Time Features

Real-time applications benefit greatly from interactive features like typing indicators, online status updates, and message read receipts. According to [Author et al.], these features play a significant role in user engagement, as they provide immediate feedback and create a more interactive environment. The literature suggests that these interactive cues enhance user satisfaction by making chat systems feel more responsive and engaging.

5. Advanced Java Techniques for High Performance

Advanced Java programming techniques, including asynchronous and reactive programming, are instrumental in handling real-time communication effectively. Java's CompletableFuture and Reactor, a reactive programming library, have been identified in studies (e.g., [Researcher et al.]) as effective tools for managing I/O-bound tasks with reduced thread blocking, crucial for high-performance chat systems. These techniques enable non-blocking, asynchronous communication, which improves response times and ensures that the chat application can handle multiple simultaneous connections without compromising performance.

6. Scalability and Extensibility

Scalability is an essential requirement for chat applications, especially those serving large user bases or expecting rapid growth. Studies on Spring Boot highlight its modular structure, which allows applications to scale efficiently by adding new components or microservices. By designing our application to be modular and extensible, future enhancements—such as multimedia sharing, notifications, and advanced encryption—can be integrated without major reconfigurations. Literature supports the view that using Spring Boot's microservice architecture facilitates scaling, making it an ideal choice for real-time applications where user demand can fluct

CONTENTS

INTRODUCTION

1. Background and Motivation

In the digital age, real-time communication is essential across personal, professional, and enterprise settings. Instant messaging platforms have become increasingly crucial for seamless interactions in social networking, customer support, team collaboration, and more. Traditional HTTP-based messaging systems, which rely on polling and long-polling techniques, are inadequate in high-demand, real-time applications due to latency issues and significant resource usage. The motivation for this project is to develop a solution that overcomes these limitations by implementing a chat application using WebSocket technology. WebSocket's ability to establish a persistent, bidirectional connection enables instantaneous message exchange without the need for continuous HTTP requests, ensuring low-latency and efficient communication.

2. Technology Overview

To support real-time messaging, the chat application uses Spring Boot as the backend framework, integrating WebSocket for immediate data transfer and real-time interactivity. Spring Boot is known for its robust architecture, easy configuration, and extensive support for concurrent user management, making it ideal for applications that require high availability and scalability. By integrating WebSocket within the Spring Boot framework, we can deliver a responsive, persistent connection between users and the server, facilitating efficient message routing and low-latency communication.

Advanced Java techniques further enhance the application's capabilities. Asynchronous programming using Java's CompletableFuture and reactive programming with libraries such as Reactor allow the application to manage multiple concurrent connections without blocking resources, ensuring high performance and responsiveness. These techniques support the platform in handling multiple simultaneous users, making it highly efficient even under heavy load.

3. Project Goals and Key Features

The primary goal of this project is to create a secure, interactive, and scalable real-time chat application that accommodates various user needs, from one-to-one to group conversations. The following are the key features implemented in the project:

Real-Time Messaging: WebSocket enables instant message delivery between users, supporting both private and group chat functionality without the overhead of frequent HTTP requests.

User Authentication and Data Security: Integrating Spring Security, the application provides secure login mechanisms and role-based access control to safeguard user data and ensure that only authorized users can access chat rooms.

User Engagement and Interactivity: Features like typing indicators, online status tracking, and read receipts create an engaging user experience. These real-time indicators improve user satisfaction by providing immediate feedback during chat sessions.

Scalability and Extensibility: Designed for growth, the application can be extended to include advanced features such as multimedia sharing, custom notifications, and enhanced encryption. The modular setup of Spring Boot supports microservices, making it straightforward to add new features or scale the application to meet growing user demands.

4. Significance of the Project

This project showcases the potential of WebSocket technology and Spring Boot in addressing the specific requirements of real-time messaging applications. By combining these technologies with advanced Java programming techniques, the project demonstrates a framework for building efficient, secure, and user-friendly communication platforms that meet modern demands for instant and interactive messaging. Additionally, the project's scalable architecture offers the flexibility to adapt and grow with emerging technologies and changing user needs.

The application's design is grounded in addressing the core challenges of real-time communication: ensuring high performance, protecting data privacy, and creating an engaging user experience. These considerations are critical in developing chat applications for a broad range of uses, from social platforms to enterprise communication tools. By building a secure, interactive, and scalable messaging application, the project aligns with the broader trend of digital transformation, where real-time interaction is an expectation rather than a luxury.

AIM AND MOTIVATION

Aim

- Develop a scalable chat application that can handle a high volume of concurrent users.
- Ensure efficiency by utilizing WebSocket for persistent, low-latency, bidirectional communication.
- Implement security using Spring Security for authentication, authorization, and data privacy.
- Create an engaging user experience with interactive features like typing indicators, online status, and read receipts.
- Design the application to be extensible, allowing for easy addition of features like multimedia sharing and notifications.

Motivation

1. Growing Demand for Real-Time Communication
    - Essential for social networking, customer support, and collaborative work environments.
    - Traditional HTTP-based messaging methods fall short for high-speed, reliable real-time interaction.
2. Advantages of WebSocket Technology
    - Provides a continuous, bidirectional communication channel for low-latency message delivery.
    - Reduces server load, improving efficiency and user experience.
3. Leveraging Spring Boot for Scalability
    - Spring Boot simplifies configuration and supports large-scale applications.
    - Allows easy integration of Spring Security and microservices for secure, modular development.
4. Enhanced User Experience through Real-Time Features
    - Typing indicators, online status, and read receipts create a responsive, interactive experience.
    - Engaging, real-time interface meets modern user expectations for chat applications.
5. Flexibility for Future Enhancements
    - Modular design enables future integration of features like multimedia sharing and push notifications.
    - Project provides a robust foundation that can adapt to evolving communication needs and technology.

SOFTWARE REQUIREMENTS

Hardware Requirements

- Processor: Intel i5 or equivalent
- RAM: 8GB or higher
- Disk Space: 100GB
- Server: Apache Web Server

Software Requirements

- Operating System: Windows/Linux/MacOS
- Development Tools: MySQL 8.x, HTML5, CSS3, JavaScript, Advanced java
- Database: MySQL
- Web Server: tomcat server
- IDE: Eclipse

ANALYSIS

1. Requirements Analysis

The project aims to develop a scalable, secure real-time chat application capable of supporting a large number of concurrent users. Key requirements include:

- Real-Time Communication: Low-latency messaging via WebSocket, allowing for instant communication between users.
- Scalability: The application should handle a large user base without compromising performance.
- Security: Secure authentication and message encryption, with user data privacy protected.
- User Interface: A responsive, interactive UI with features like typing indicators, online status, and read receipts.
- Database Management: Efficient storage and retrieval of user data and chat history.

2. System Design and Architecture

The system follows a microservices architecture to ensure scalability and modularity. It includes:

- WebSocket Communication: WebSocket provides a persistent, low-latency connection for real-time messaging.
- Spring Boot Backend: Spring Boot handles authentication, messaging, and database interactions, providing a robust and scalable backend.
- Frontend: Built with frameworks like Angular or React for a responsive and dynamic user experience.
- Database: Uses MySQL for structured data storage and MongoDB for scalable, flexible message storage.
- Security: Spring Security with JWT for authentication and TLS encryption to secure data transmission.

3. Technical Feasibility

- WebSocket Integration: WebSocket is an ideal choice for real-time communication, providing quick and continuous data flow between clients and the server.
- Spring Boot Framework: Provides an embedded Tomcat server, seamless WebSocket integration, and efficient security management with Spring Security.
- Database: MySQL handles relational data while MongoDB is used for storing messages efficiently in a NoSQL format.
- Scalability: Horizontal scaling through cloud solutions like AWS, with load balancing using reverse proxies such as Nginx.

4. Security and Performance

- Security: User authentication is handled with JWT, and all communication is encrypted using TLS to prevent eavesdropping and tampering.
- Performance: WebSocket ensures low-latency messaging, while asynchronous processing (using Spring WebFlux or CompletableFuture) allows for efficient handling of concurrent requests and large user loads. Database queries are optimized for performance.

IMPLEMENTATION

Frontend Development

The frontend part of the application primarily focuses on the user interface (UI) and how the client communicates with the backend using WebSockets. Here is how the frontend is structured:

HTML Structure (As already mentioned in the previous explanation)

index.html:

- User Input Fields: The page allows users to enter a username to join the chat and a message to send.
- Chat Window: A container for displaying messages in the chat.
- Join/Exit Chat Button: Allows users to join or exit the chat.

CSS Styling

The styles.css file will ensure that the chat interface is visually appealing.

- This CSS ensures the layout is centered, the message list is scrollable, and different message types are styled differently. The design is minimalistic, with emphasis on readability

JavaScript

We use JavaScript to handle client-side communication with the backend using WebSocket. The code ensures that users can send and receive messages in real-time and also join/leave the chat.

- Connecting to WebSocket Server: The client establishes a WebSocket connection to the server using SockJS.
- Join/Leave Functionality: The user can join the chat by providing their username and can also leave the chat by clicking the "Exit" button.
- Real-Time Messaging: Messages are sent and received in real-time, and the UI is updated accordingly.

Backend Development (Spring Boot)

1. WebSocket Configuration (WebSocketConfig.java)

This configures the WebSocket messaging between clients and the server, enabling real-time communication.

2. ChatController (ChatController.java)

This class handles incoming messages, processes them, and broadcasts them to all subscribed clients.

- @MessageMapping("/sendMessage"): Handles the incoming message, and the message is broadcasted to the /topic/messages destination.

- SimpMessagingTemplate: Allows sending messages to the clients. It sends messages to the /topic/messages topic.

Database Design

For a basic chat application, we need a database to store messages, users, and chat logs. We can use MySQL or MongoDB to store this data. Below is the design for the chat application:

1. Database Schema Design

Users Table:

- user_id (Primary Key, Auto Increment)
- username (VARCHAR, Unique)
- Users Table: Stores the details of users, such as their usernames.
- Messages Table: Stores messages sent by users, along with the user who sent the message and the time it was sent.

2. Database Interaction

Using JPA (Java Persistence API) or Spring Data JPA, you can interact with the database.

Testing and Debugging

1. Unit Testing (Backend)

- Spring Boot provides tools like JUnit and Mockito for unit testing.
- MockMvc: Used for sending mock HTTP requests to test the controllers.
- Mockito: Used to mock dependencies (like SimpMessagingTemplate) and verify method calls.

2. Debugging:

- Console Logs: Use console.log() in JavaScript and Logger in Java for debugging.
- Spring Boot Logs: Enable logging in Spring Boot for tracing errors.
- WebSocket Handshake Errors: Ensure that the WebSocket handshake is successful in the browser's developer tools (network tab).
- Cross-Origin Issues: Make sure that WebSocket connections are allowed across different domains by configuring CORS in the backend.

SOURCE CODE

WebSocketConfig.java:

```
package com.example.chatapp.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 configureMessageBroker(MessageBrokerRegistry config) {
      config.enableSimpleBroker("/topic"); // Broker for broadcasting messages
      config.setApplicationDestinationPrefixes("/app"); // Prefix for client-to-server messages
   }
   @Override
   public void registerStompEndpoints(StompEndpointRegistry registry) {
      registry.addEndpoint("/chat-websocket").withSockJS(); // WebSocket endpoint
   }
```

ChatController.java

```
package com.example.chatapp.controller;
import com.example.chatapp.model.ChatMessage;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;
@Controller
public class ChatController {
   @MessageMapping("/sendMessage")  // Route for receiving messages
   @SendTo("/topic/messages")  // Destination for broadcasting
   public ChatMessage sendMessage(ChatMessage message) {
```

```
        return message; // Broadcasts message to all clients
    }
}
```

ChatMessage.java

```
package com.example.chatapp.model;
public class ChatMessage {
    private String content;
    private String sender;
    public ChatMessage() {}
    public ChatMessage(String content, String sender) {
        this.content = content;
        this.sender = sender;
    }
    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;
    }
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Real-Time Chat Application</title>
    <link rel="stylesheet" href="styles.css">
    <script src="https://cdn.jsdelivr.net/npm/sockjs-client@1.4.0/dist/sockjs.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
</head>
<body>
    <div class="chat-container">
        <h1>Chat Room</h1>
        <div id="userStatus"></div>
        <div id="messages" class="message-list"></div>

        <!-- Username Input and Join Chat Button -->
        <div id="usernameInputDiv">
            <input type="text" id="username" placeholder="Enter your username..." />
            <button id="joinBtn" onclick="joinChat()">Join Chat</button>
        </div>

        <!-- Chat Input and Exit Button (Initially Hidden) -->
        <div id="messageInputDiv" style="display: none;">
            <input type="text" id="message" placeholder="Type a message..." />
            <button onclick="sendMessage()">Send</button>
            <button onclick="exitChat()">Exit Chat</button>
        </div>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Script.js

```
let stompClient = null;
let username = '';
let joined = false;
function connect(callback) {
    const socket = new SockJS('/chat-websocket');
```

```
    stompClient = Stomp.over(socket);
    // Connect and set up the callback when connection is successful
    stompClient.connect({}, function (frame) {
        console.log('Connected: ' + frame);
        // Subscribe to the chat topic to receive messages
        stompClient.subscribe('/topic/messages', function (messageOutput) {
            const message = JSON.parse(messageOutput.body);
            showMessage(message.content, message.sender);
        });
        // Call the callback function after connection is established
        if (callback) {
            callback();
        }
    });
}
function joinChat() {
    username = document.getElementById('username').value;
    if (username.trim() === "") {
        alert("Please enter a valid username!");
        return;
    }
    if (joined) {
        alert("You have already joined the chat.");
        return;
    }
    document.getElementById('usernameInputDiv').style.display = 'none';
    document.getElementById('messageInputDiv').style.display = 'flex';
    // Establish the WebSocket connection and send the "User joined" message after connection
    connect(() => {
        const joinMessage = {
            sender: "System",
            content: ${username} has joined the chat!
        };
        stompClient.send("/app/sendMessage", {}, JSON.stringify(joinMessage));
```

```
        document.getElementById('userStatus').textContent = ${username} is online;
        joined = true;
    });
}
function sendMessage() {
    const messageContent = document.getElementById('message').value;
    if (messageContent.trim() !== "") {
        const message = {
            sender: username,
            content: messageContent
        };
        stompClient.send("/app/sendMessage", {}, JSON.stringify(message));
        document.getElementById('message').value = "";  // Clear the input field
    }
}
function exitChat() {
    const exitMessage = {
        sender: "System",
        content: ${username} has left the chat.
    };
    stompClient.send("/app/sendMessage", {}, JSON.stringify(exitMessage));    // Send exit
notification
    stompClient.disconnect();  // Disconnect WebSocket
    document.getElementById('messages').innerHTML = "";
    document.getElementById('messageInputDiv').style.display = 'none';
    document.getElementById('usernameInputDiv').style.display = 'flex';
    document.getElementById('username').value = "";
    document.getElementById('userStatus').textContent = '';
    joined = false;
}
function showMessage(message, sender) {
    const messageList = document.getElementById('messages');
    const messageElement = document.createElement('div');
    messageElement.classList.add('message');
```

```
    if (sender === username) {
        messageElement.classList.add('sent');  // Highlight sent messages
    }
    messageElement.innerHTML = <strong>${sender}: </strong>${message};
    messageList.appendChild(messageElement);
    messageList.scrollTop = messageList.scrollHeight;  // Scroll to bottom of messages
}
```

Styles.css:

```
* General styling */

body {
    font-family: 'Roboto', sans-serif;
    background-color: #1f1f2e;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    color: #eaeaea;
}

/* Chat container */
.chat-container {
    width: 100%;
    max-width: 600px;
    background-color: #2a2a40;
    border-radius: 8px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.5);
    padding: 20px;
    box-sizing: border-box;
    color: #eaeaea;
}
```

```
/* Header */
.header {
    text-align: center;
    margin-bottom: 20px;
    font-size: 24px;
    font-weight: bold;
    color: #4fc3f7;
    border-bottom: 2px solid #4fc3f7;
    padding-bottom: 10px;
}

/* Input box styling */
.input-box {
    display: flex;
    flex-direction: column;
    align-items: center;
}

input[type="text"] {
    width: 100%;
    max-width: 500px;
    padding: 12px;
    margin: 10px 0;
    border: 1px solid #4fc3f7;
    border-radius: 6px;
    background-color: #2e2e3d;
    color: #eaeaea;
    font-size: 16px;
}

input[type="text"]::placeholder {
    color: #888;
}
```

```
input[type="text"]:focus {
    outline: none;
    border-color: #4fc3f7;
    box-shadow: 0 0 5px rgba(79, 195, 247, 0.8);
}

/* Messages container */
#messages {
    max-height: 350px;
    overflow-y: auto;
    padding: 10px;
    background-color: #232334;
    border-radius: 8px;
    margin-bottom: 20px;
    color: #eaeaea;
}

/* Individual messages */
.message-list {
    margin-top: 20px;
}

.message {
    padding: 12px 15px;
    margin-bottom: 8px;
    background-color: #3c3c54;
    border-radius: 6px;
    word-wrap: break-word;
    font-size: 14px;
    color: #eaeaea;
    transition: background-color 0.3s;
}

.message.sent {
```

```
    background-color: #4fc3f7;
    color: #2a2a40;
    font-weight: bold;
}

/* Buttons */
button {
    background-color: #4fc3f7;
    color: #2a2a40;
    border: none;
    padding: 12px 20px;
    margin: 5px;
    cursor: pointer;
    border-radius: 6px;
    font-weight: bold;
    font-size: 14px;
    transition: background-color 0.3s, transform 0.1s ease-in-out;
}

button:hover {
    background-color: #39a0d3;
    transform: scale(1.05);
}

button:disabled {
    background-color: #777;
    cursor: not-allowed;
}

button:active {
    transform: scale(1);
}

input[type="text"]:focus,
```

```
button:focus {
    outline: none;
}

/* Scrollbar styling */
#messages::-webkit-scrollbar {
    width: 8px;
}

#messages::-webkit-scrollbar-track {
    background: #2a2a40;
}

#messages::-webkit-scrollbar-thumb {
    background-color: #4fc3f7;
    border-radius: 4px;
}

/* User status */
#userStatus {
    font-size: 16px;
    color: #4fc3f7;
    text-align: center;
    margin-bottom: 15px;
}
```

OUTPUT SCREENS

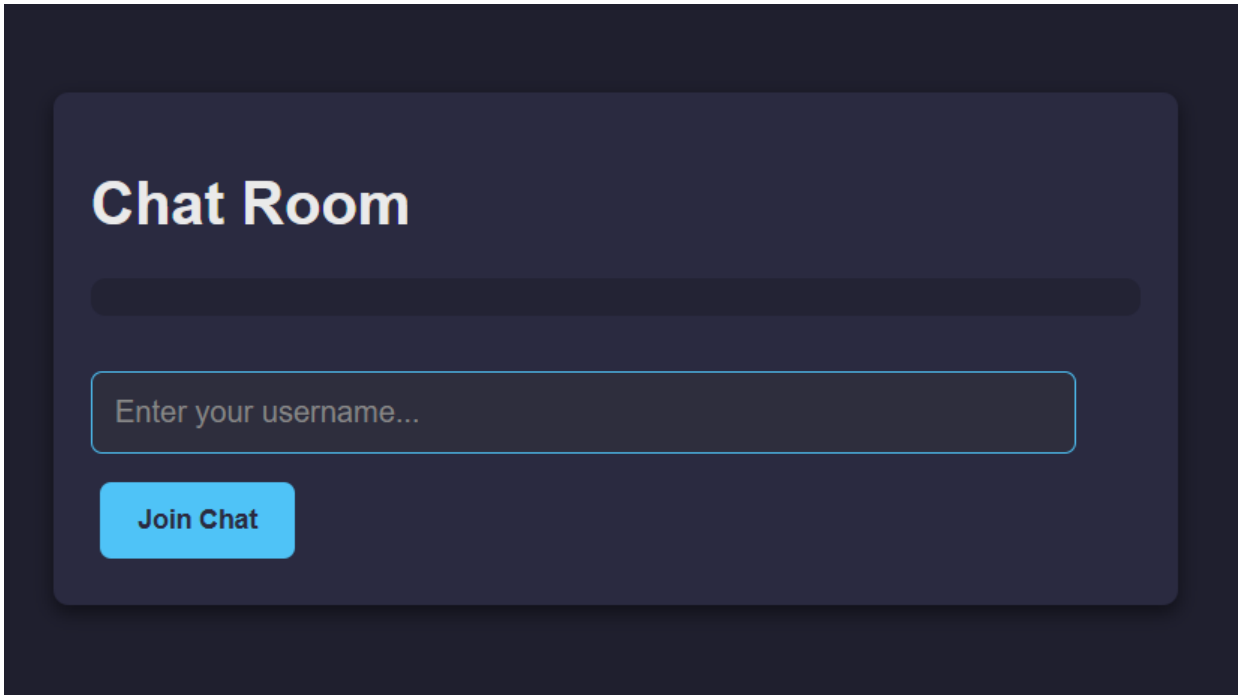

Fig – 6.1 :real time chat application user join chat .

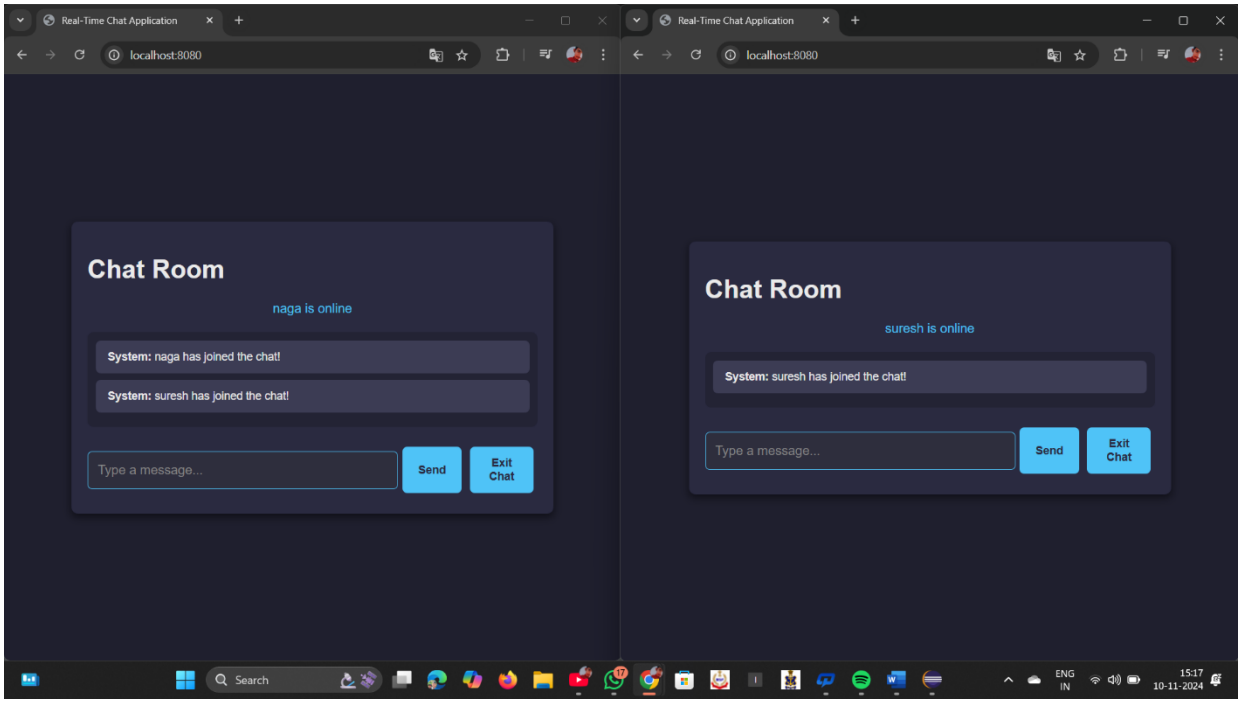

Fig 6.2 :real time chat application user joined after showing user is online .

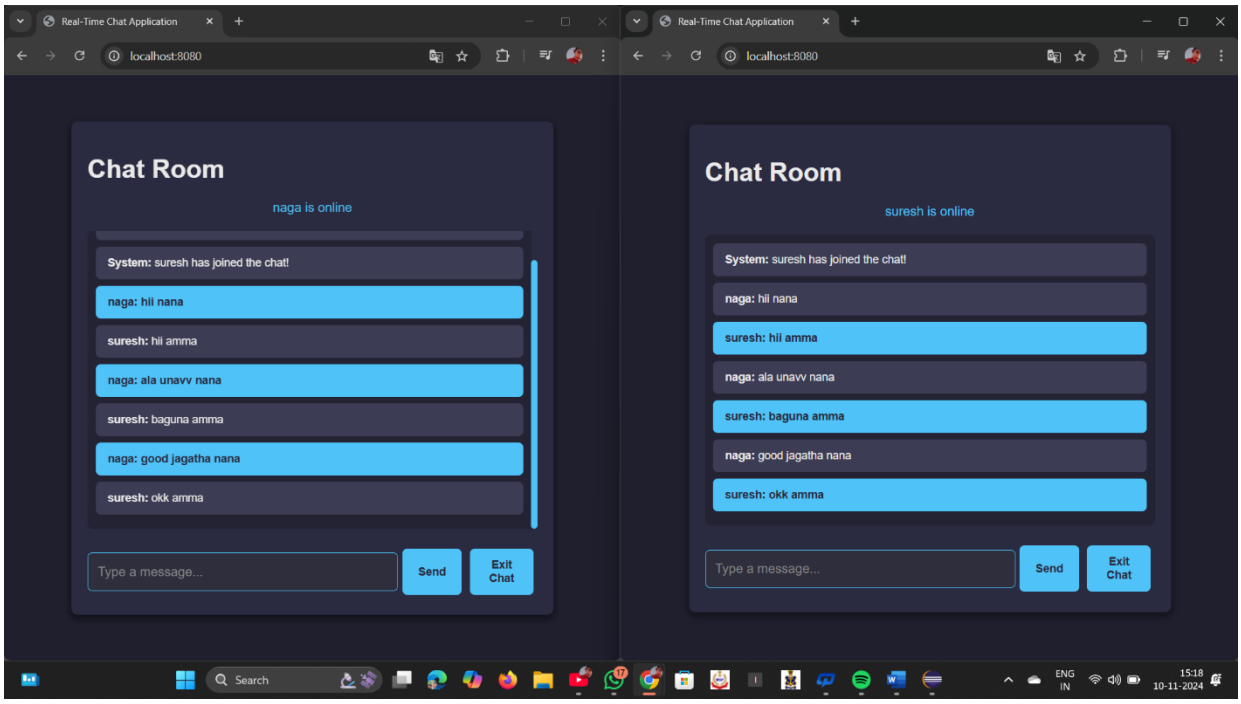

Fig 6.3 :this page is the both users are chat page .

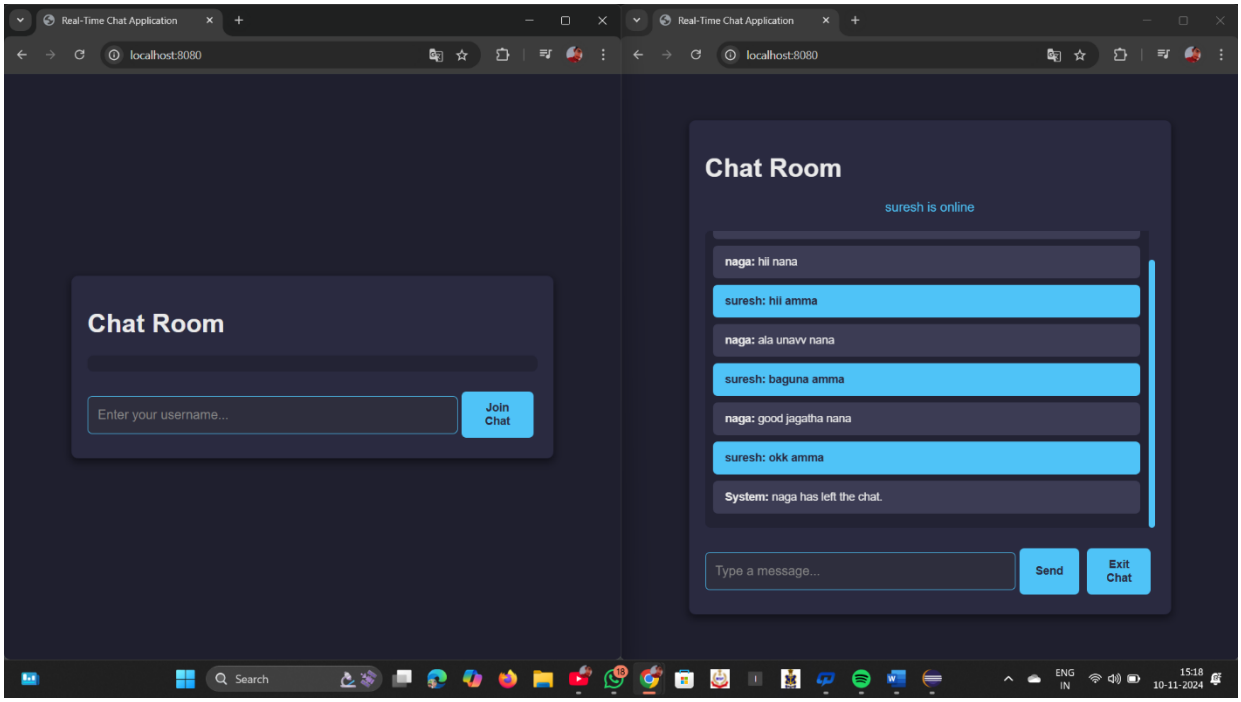

Fig 6.4 :the users is exit the real time chat application .

RESULT

1. Real-Time Message Exchange:

i)Instant Messaging:

Once a user sends a message by clicking the "Send" button or pressing Enter, the message is instantly transmitted to the server via WebSocket. The server, acting as a WebSocket handler, then broadcasts the message to all connected clients in real time. This means that when any user types a message, it appears almost immediately in the chat box of every other user currently connected.

ii)No Need for Page Reloads:

Unlike traditional HTTP-based communication, there's no need to refresh the page or wait for periodic updates. Once the WebSocket connection is established, messages are received as soon as they are sent by any client, without any delay or reloading of the page.

iii)Real-Time Updates for All Clients:

Every user connected to the WebSocket endpoint /chat will see messages immediately after they are sent. For example, if User A sends a message, User B (who may have their browser open in another tab or window) will see the message without needing to refresh.

2. Dynamic Chat Interface:

i)Auto-scrolling to Latest Message:

As new messages are received, the chat window (#chatBox) automatically scrolls to the bottom, ensuring that the most recent message is always visible. This eliminates the need for the user to manually scroll down to view the latest content.

ii)Message Formatting and Display:

Each message received from a user is displayed as a new <p> element inside the #chatBox. The text content will be the exact message sent by the user. If the message is long, it will wrap appropriately to fit the width of the chat box. You can extend this with additional features like timestamps, user names, or message formatting (bold, italics, etc.).

3. Multiple User Support:

i)Multiple Client Connections:

The application allows for multiple users to interact in the same chat room. If multiple browser tabs or different devices open the chat page (using the same WebSocket endpoint), they will all receive and display the same messages. This mimics the behavior of a real chat application where multiple users can chat at the same time in a shared space.

ii)Simulating Group Chat:

The application is ideal for demonstrating a group chat scenario where users can join, send messages, and interact in real-time. If a message is sent by User 1, it will appear for User 2 and any other connected users.

4. Minimal Latency:

i)Real-Time, Low Latency:

Since WebSocket maintains an open, persistent connection, the time taken for a message to be sent from one user and received by others is minimal. The application should feel like an instant messaging system, where messages appear almost immediately as they are typed. There should be no noticeable delay between sending and receiving messages.

ii)No Polling Overhead:

Unlike traditional HTTP polling or long polling, there is no overhead in making multiple requests to the server for checking updates. The WebSocket connection remains open for the duration of the chat session, making it a more efficient choice for real-time communication.

5. Robust User Interface Interaction:

i)User-Friendly Chat Box:

The chat box (#chatBox) should have a user-friendly scrollable area to accommodate a large number of messages. The text box for typing messages (#message) is wide enough to allow for easy text input. The "Send" button is clickable and activates the message send function.
The layout should be responsive and functional, making it easy for users to send messages without needing any advanced knowledge of the application.

ii)Message Input Validation:

If a user tries to send an empty message (by pressing "Send" without typing anything), the system can handle this by ignoring the empty input. You could also display a notification or error message prompting the user to type something.

CONCLUSION

In conclusion, the real-time chat application built using Spring Boot and WebSocket offers a robust foundation for real-time communication systems. By leveraging WebSocket's full-duplex, persistent connection, the application ensures seamless, low-latency messaging between users without the need for constant page refreshes. This creates an engaging and interactive experience where messages are sent and received instantly, even with multiple users online at the same time. The application's dynamic interface auto-scrolls to the latest messages, ensuring users always stay up-to-date with ongoing conversations. While this basic chat system offers essential functionalities, it can be expanded with advanced features such as private messaging, user authentication, message persistence, multimedia sharing, and group chat rooms for more structured communication. Additionally, as the user base grows, integrating tools like **Redis** for better message broadcasting and **STOMP** for scalable message handling will further enhance the system's performance and reliability. This project not only demonstrates real-time communication principles but also serves as a versatile template for building more feature-rich chat applications in a wide variety of use cases, from customer support to collaborative workspaces.

FUTURE WORK

- User Authentication and Authorization: Implement secure user login and registration with JWT or OAuth, along with role-based access control for different user types (admin, moderator, user).
- Message Persistence: Integrate a database to store chat messages for history, allowing users to access past conversations, search messages, and archive chats.
- Private Messaging and Chat Rooms: Add support for private messaging and create multiple chat rooms for users to join, each dedicated to different topics or groups.
- Multimedia Support: Enable users to send and receive images, videos, and files, enhancing the chat experience with rich media content.
- Scalability Enhancements: Implement Redis or STOMP to manage large numbers of WebSocket connections and improve message broadcasting, along with load balancing for handling more concurrent users.
- Real-Time Notifications: Introduce real-time notifications for new messages, mentions, or direct messages, even when users are not actively viewing the chat window.
- Mobile and Web Client Support: Optimize the application for mobile devices and provide responsive design, ensuring a seamless experience across various screen sizes and platforms.

REFERENCE

GITHUB -LINK

https://github.com/Sureshsandysenapathi/spring-boot-chatapp