**Real-Time Chat Application**

In today's fast-paced digital landscape, effective communication is crucial for personal and professional relationships. The Real-Time Chat Application is designed to facilitate instant messaging, media sharing, and notifications, ensuring a seamless communication experience.

**Backend Code (Spring Boot Application)**

java

CopyEdit

package com.example.chatapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Configuration;

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

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

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;

import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

import org.springframework.web.socket.config.annotation.MessageBrokerRegistry;

import javax.persistence.\*;

import java.time.LocalDateTime;

import java.util.Date;

import io.jsonwebtoken.\*;

import java.util.\*;

@SpringBootApplication

public class ChatApplication {

public static void main(String[] args) {

SpringApplication.run(ChatApplication.class, args);

}

}

@Entity

class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(unique = true, nullable = false)

private String username;

private String password;

private String email;

// Getters and setters omitted for brevity

}

@Entity

class Message {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String sender;

private String recipient;

private String content;

private LocalDateTime timestamp;

// Getters and setters omitted for brevity

}

@Configuration

@EnableWebSocketMessageBroker

class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

@Override

public void configureMessageBroker(MessageBrokerRegistry config) {

config.enableSimpleBroker("/topic");

config.setApplicationDestinationPrefixes("/app");

}

@Override

public void registerStompEndpoints(StompEndpointRegistry registry) {

registry.addEndpoint("/ws").setAllowedOrigins("\*").withSockJS();

}

}

@Controller

class ChatController {

@MessageMapping("/chat")

@SendTo("/topic/messages")

public Message sendMessage(Message message) {

message.setTimestamp(LocalDateTime.now());

return message;

}

}

@RestController

@RequestMapping("/auth")

class AuthController {

private final String jwtSecret = "YourSecretKey";

@PostMapping("/login")

public Map<String, String> login(@RequestBody Map<String, String> loginRequest) {

// Assume username/password validation passed

String username = loginRequest.get("username");

String token = Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 3600000))

.signWith(SignatureAlgorithm.HS512, jwtSecret)

.compact();

return Collections.singletonMap("token", token);

}

}

@Configuration

@EnableWebSecurity

class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/auth/\*\*", "/ws/\*\*").permitAll()

.anyRequest().authenticated();

}

}

**Frontend Code (React.js)**

**App.js**

javascript

CopyEdit

import React, { useState, useEffect } from 'react';

import SockJS from 'sockjs-client';

import { Stomp } from '@stomp/stompjs';

const App = () => {

const [message, setMessage] = useState('');

const [messages, setMessages] = useState([]);

let stompClient = null;

useEffect(() => {

const socket = new SockJS('http://localhost:8080/ws');

stompClient = Stomp.over(socket);

stompClient.connect({}, () => {

stompClient.subscribe('/topic/messages', (msg) => {

const receivedMessage = JSON.parse(msg.body);

setMessages((prev) => [...prev, receivedMessage]);

});

});

return () => {

if (stompClient) stompClient.disconnect();

};

}, []);

const sendMessage = () => {

if (stompClient && message.trim() !== '') {

stompClient.send('/app/chat', {}, JSON.stringify({ sender: 'User1', content: message }));

setMessage('');

}

};

return (

<div>

<h1>Real-Time Chat</h1>

<div>

{messages.map((msg, index) => (

<div key={index}>

<strong>{msg.sender}: </strong>

{msg.content}

</div>

))}

</div>

<input

type="text"

value={message}

onChange={(e) => setMessage(e.target.value)}

placeholder="Type a message..."

/>

<button onClick={sendMessage}>Send</button>

</div>

);

};

export default App;