COMANDOS DE ACTIVACION

cd C:\Users\Usuario\Desktop\chat\_empresa

venv\Scripts\activate

python backend/servidor.py

import mysql.connector  
from flask import Flask, render\_template, request  
from flask\_socketio import SocketIO, join\_room, leave\_room, emit  
from flask\_cors import CORS  
import logging  
from datetime import datetime  
  
app = Flask(\_\_name\_\_)  
CORS(app)  
socketio = SocketIO(app, cors\_allowed\_origins="\*")  
  
logging.basicConfig(level=logging.INFO)  
  
# Configuración de la base de datos  
DB\_HOST = 'localhost'  
DB\_USER = 'root'  
DB\_PASSWORD = 'Abril2025'  
DB\_NAME = 'chat\_empresa'  
  
salas = {}  
usuarios\_en\_sala = {}  
usuarios\_info = {}  
usuarios\_conectados\_global = {}  
nombres\_de\_salas = set()  
  
def get\_db\_connection():  
 try:  
 mydb = mysql.connector.connect(  
 host=DB\_HOST,  
 user=DB\_USER,  
 password=DB\_PASSWORD,  
 database=DB\_NAME  
 )  
 return mydb  
 except mysql.connector.Error as err:  
 print(f"Error al conectar a la base de datos: {err}")  
 return None  
  
def get\_usuarios\_en\_sala(sala):  
 sids\_en\_sala = salas.get(sala, [])  
 return [usuarios\_info.get(sid) for sid in sids\_en\_sala if usuarios\_info.get(sid)]  
  
@socketio.on('connect')  
def handle\_connect():  
 sid = request.sid  
 print(f'Cliente conectado: {sid}')  
 cargar\_salas\_desde\_db() # Cargar las salas al conectar el primer cliente  
 emit('usuarios\_conectados', [{'id': id, 'nombre': info['usuario']} for id, info in usuarios\_conectados\_global.items()])  
 emit('actualizar\_lista\_salas', {'salas': list(nombres\_de\_salas)})  
  
def cargar\_salas\_desde\_db():  
 mydb = get\_db\_connection()  
 if mydb:  
 mycursor = mydb.cursor()  
 try:  
 mycursor.execute("SELECT nombre FROM salas")  
 resultados = mycursor.fetchall()  
 for resultado in resultados:  
 nombres\_de\_salas.add(resultado[0])  
 except mysql.connector.Error as err:  
 print(f"Error al cargar las salas desde la base de datos: {err}")  
 finally:  
 mycursor.close()  
 mydb.close()  
  
@socketio.on('disconnect')  
def handle\_disconnect():  
 sid = request.sid  
 nombre\_usuario = usuarios\_info.get(sid)  
 sala\_actual = usuarios\_en\_sala.get(sid)  
  
 print(f'Cliente desconectado: {sid} ({nombre\_usuario if nombre\_usuario else "Desconocido"}) de la sala: {sala\_actual}')  
  
 if sid in usuarios\_conectados\_global:  
 del usuarios\_conectados\_global[sid]  
 emit('usuarios\_conectados', [{'id': id, 'nombre': info['usuario']} for id, info in usuarios\_conectados\_global.items()], broadcast=True)  
  
 if sid in usuarios\_en\_sala:  
 sala = usuarios\_en\_sala[sid]  
 leave\_room(sala)  
 if sid in salas.get(sala, []):  
 salas[sala].remove(sid)  
 del usuarios\_en\_sala[sid]  
 emit('actualizar\_usuarios', {'sala': sala, 'usuarios': get\_usuarios\_en\_sala(sala)}, to=sala)  
 emit('mensaje\_sistema', {'texto': f'{nombre\_usuario if nombre\_usuario else "Un usuario"} ha abandonado la sala {sala}.'}, to=sala)  
  
 if sid in usuarios\_info:  
 del usuarios\_info[sid]  
  
@socketio.on('establecer\_nombre\_usuario')  
def handle\_establecer\_nombre\_usuario(data):  
 sid = request.sid  
 usuario = data.get('usuario')  
 if usuario:  
 usuarios\_info[sid] = usuario  
 usuarios\_conectados\_global[sid] = {'usuario': usuario}  
 print(f'Usuario {usuario} establecido para la sesión {sid}')  
 emit('usuarios\_conectados', [{'id': id, 'nombre': info['usuario']} for id, info in usuarios\_conectados\_global.items()], broadcast=True)  
  
@socketio.on('unirse\_sala')  
def handle\_unirse\_sala(data):  
 sid = request.sid  
 sala = data.get('sala')  
 usuario = usuarios\_info.get(sid, 'Desconocido')  
 sala\_anterior = usuarios\_en\_sala.get(sid)  
  
 if sala\_anterior:  
 leave\_room(sala\_anterior)  
 if sid in salas.get(sala\_anterior, []):  
 salas[sala\_anterior].remove(sid)  
 emit('actualizar\_usuarios', {'sala': sala\_anterior, 'usuarios': get\_usuarios\_en\_sala(sala\_anterior)}, to=sala\_anterior)  
 emit('mensaje\_sistema', {'texto': f'{usuario} ha abandonado la sala {sala\_anterior}.'}, to=sala\_anterior)  
 usuarios\_en\_sala.pop(sid, None)  
  
 join\_room(sala)  
 usuarios\_en\_sala[sid] = sala  
 if sala not in salas:  
 salas[sala] = []  
 if sid not in salas[sala]:  
 salas[sala].append(sid)  
  
 emit('actualizar\_usuarios', {'sala': sala, 'usuarios': get\_usuarios\_en\_sala(sala)}, to=sala)  
 emit('mensaje\_sistema', {'texto': f'{usuario} se ha unido a la sala {sala}.'}, to=sala)  
 emit('usuario\_unido', {'usuario': usuario, 'sala': sala}, broadcast=True)  
  
@socketio.on('mensaje')  
def handle\_mensaje(data):  
 sid = request.sid  
 texto = data.get('texto')  
 destinatario = data.get('destinatario')  
 usuario = usuarios\_info.get(sid, 'Desconocido')  
 sala = usuarios\_en\_sala.get(sid)  
 timestamp = data.get('timestamp') if data.get('timestamp') else int(round(datetime.now().timestamp() \* 1000))  
  
 if texto:  
 if destinatario:  
 destinatario\_sid = None  
 for s, u in usuarios\_info.items():  
 if u == destinatario:  
 destinatario\_sid = s  
 break  
 if destinatario\_sid:  
 emit('mensaje\_privado', {'texto': texto, 'remitente': usuario, 'timestamp': timestamp}, to=destinatario\_sid)  
 emit('mensaje\_privado', {'texto': f'Mensaje privado a {destinatario}: {texto}', 'remitente': usuario, 'timestamp': timestamp}, to=sid)  
 else:  
 emit('mensaje\_sistema', {'texto': f'El usuario {destinatario} no está en línea o no existe.'}, to=sid)  
 elif sala:  
 emit('nuevo\_mensaje', {'usuario': usuario, 'texto': texto, 'timestamp': timestamp}, to=sala)  
  
@socketio.on('crear\_sala')  
def handle\_crear\_sala(data):  
 sid = request.sid  
 nombre\_sala = data.get('nombre\_sala')  
 nombre\_usuario = usuarios\_info.get(sid, 'Desconocido')  
  
 if not nombre\_sala:  
 emit('sala\_creada', {'error': 'El nombre de la sala no puede estar vacío.'}, to=sid)  
 return  
  
 nombre\_sala = nombre\_sala.strip()  
 if not nombre\_sala:  
 emit('sala\_creada', {'error': 'El nombre de la sala no puede estar solo con espacios.'}, to=sid)  
 return  
  
 if len(nombre\_sala) > 50:  
 emit('sala\_creada', {'error': 'El nombre de la sala es demasiado largo (máximo 50 caracteres).'}, to=sid)  
 return  
  
 mydb = get\_db\_connection()  
 if mydb is None:  
 emit('sala\_creada', {'error': 'No se pudo conectar a la base de datos.'}, to=sid)  
 return  
  
 mycursor = mydb.cursor()  
 try:  
 # Verificar si la sala ya existe en la base de datos  
 mycursor.execute("SELECT nombre FROM salas WHERE nombre = %s", (nombre\_sala,))  
 resultado = mycursor.fetchone()  
 if resultado:  
 emit('sala\_creada', {'error': f'La sala "{nombre\_sala}" ya existe.'}, to=sid)  
 return  
  
 sql = "INSERT INTO salas (nombre, creador\_usuario) VALUES (%s, %s)"  
 val = (nombre\_sala, nombre\_usuario)  
 mycursor.execute(sql, val)  
 mydb.commit()  
 print(f'Sala "{nombre\_sala}" creada por {nombre\_usuario} y guardada en la base de datos.')  
 nombres\_de\_salas.add(nombre\_sala)  
 emit('sala\_creada', {'nombre\_sala': nombre\_sala}, broadcast=True)  
 emit('actualizar\_lista\_salas', {'salas': list(nombres\_de\_salas)}, broadcast=True) # Informar a todos de la nueva sala  
  
 join\_room(nombre\_sala)  
 usuarios\_en\_sala[sid] = nombre\_sala  
 salas[nombre\_sala] = salas.get(nombre\_sala, []) + [sid]  
  
 emit('mensaje\_sistema', {  
 'texto': f'Has creado y te has unido a la sala {nombre\_sala}.'  
 }, to=sid)  
  
 emit('actualizar\_usuarios', {  
 'sala': nombre\_sala,  
 'usuarios': get\_usuarios\_en\_sala(nombre\_sala)  
 }, to=nombre\_sala)  
  
 except mysql.connector.Error as err:  
 print(f"Error al crear la sala en la base de datos: {err}")  
 mydb.rollback()  
 emit('sala\_creada', {'error': f'Error al crear la sala: {err}'}, to=sid)  
 finally:  
 if mydb and mydb.is\_connected():  
 mycursor.close()  
 mydb.close()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 socketio.run(app, debug=True, host='0.0.0.0', port=5000)

<!DOCTYPE html>  
<html>  
<head>  
 <title>Chat Empresarial</title>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="estilos.css">  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io.js"></script>  
 <style>  
 body {  
 font-family: sans-serif;  
 margin: 20px;  
 background-color: #f4f4f4;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 div > label {  
 font-weight: bold;  
 margin-right: 5px;  
 }  
  
 select, input[type="text"], button {  
 padding: 8px;  
 margin-bottom: 10px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 box-sizing: border-box;  
 }  
  
 input[type="text"] {  
 width: calc(70% - 10px);  
 margin-right: 5px;  
 }  
  
 #destinatario\_input {  
 width: calc(30% - 10px);  
 }  
  
 button {  
 background-color: #5cb85c;  
 color: white;  
 cursor: pointer;  
 }  
  
 button:hover {  
 background-color: #4cae4c;  
 }  
  
 #mensajes {  
 border: 1px solid #ccc;  
 background-color: white;  
 height: 300px;  
 overflow-y: scroll;  
 padding: 10px;  
 margin-bottom: 10px;  
 border-radius: 4px;  
 display: flex;  
 flex-direction: column;  
 }  
  
 #mensajes p {  
 margin: 5px 10px;  
 padding: 8px 12px;  
 border-radius: 8px;  
 background-color: #e0f7fa; /\* Un color claro para las burbujas \*/  
 border: 1px solid #b2ebf2;  
 align-self: flex-start; /\* Alinear mensajes de otros a la izquierda \*/  
 max-width: 80%;  
 word-break: break-word;  
 position: relative; /\* Para posicionar la marca de tiempo \*/  
 }  
  
 #mensajes p.local-user {  
 background-color: #dcedc8; /\* Otro color para tus mensajes \*/  
 border: 1px solid #c5e1a5;  
 align-self: flex-end; /\* Alinear tus mensajes a la derecha \*/  
 }  
  
 .mensaje-sistema {  
 font-style: italic;  
 color: gray;  
 background-color: #f5f5f5;  
 border: 1px solid #eee;  
 align-self: center; /\* Centrar mensajes del sistema \*/  
 width: fit-content;  
 max-width: 80%;  
 font-size: 0.9em; /\* Reducir un poco el tamaño del texto del sistema \*/  
 color: #777;  
 }  
  
 .mensaje-privado {  
 font-style: italic;  
 color: purple;  
 background-color: #f3e5f5; /\* Otro color para mensajes privados \*/  
 border: 1px solid #e1bee7;  
 align-self: flex-start; /\* Alinear mensajes privados recibidos a la izquierda \*/  
 max-width: 80%;  
 }  
  
 .mensaje-privado.local-user {  
 align-self: flex-end; /\* Alinear tus mensajes privados enviados a la derecha \*/  
 }  
  
 .timestamp {  
 position: absolute;  
 bottom: -1.5em; /\* Ajustar según sea necesario \*/  
 left: 0.5em;  
 font-size: 0.7em;  
 color: #777;  
 }  
  
 .local-user .timestamp {  
 left: auto;  
 right: 0.5em;  
 }  
  
 #usuarios\_en\_sala {  
 list-style-type: none;  
 padding: 10px;  
 margin-bottom: 20px;  
 border: 1px solid #ccc;  
 background-color: #f9f9f9;  
 border-radius: 4px;  
 }  
  
 #usuarios\_en\_sala li {  
 padding: 5px 0;  
 border-bottom: 1px solid #eee;  
 display: flex; /\* Para alinear el nombre y el indicador \*/  
 align-items: center; /\* Centrar verticalmente \*/  
 }  
  
 #usuarios\_en\_sala li:last-child {  
 border-bottom: none;  
 }  
  
 #usuarios\_en\_sala h2 {  
 margin-top: 0;  
 color: #555;  
 font-size: 1.2em;  
 margin-bottom: 10px;  
 text-align: center;  
 }  
  
 #usuarios\_en\_linea {  
 list-style-type: none;  
 padding: 10px;  
 margin-bottom: 20px;  
 border: 1px solid #ccc;  
 background-color: #f9f9f9;  
 border-radius: 4px;  
 }  
  
 #usuarios\_en\_linea li {  
 padding: 5px 0;  
 border-bottom: 1px solid #eee;  
 display: flex; /\* Para alinear el nombre y el indicador \*/  
 align-items: center; /\* Centrar verticalmente \*/  
 }  
  
 #usuarios\_en\_linea li:last-child {  
 border-bottom: none;  
 }  
  
 #usuarios\_en\_linea h2 {  
 margin-top: 0;  
 color: #555;  
 font-size: 1.2em;  
 margin-bottom: 10px;  
 text-align: center;  
 }  
  
 .online-indicator {  
 width: 10px;  
 height: 10px;  
 border-radius: 50%;  
 background-color: #4CAF50; /\* Verde para en línea \*/  
 margin-left: 8px;  
 display: inline-block;  
 }  
 </style>  
</head>  
<body>  
 <h1>Chat Empresarial</h1>  
  
 <div>  
 <h2>Usuarios en Línea:</h2>  
 <ul id="usuarios\_en\_linea">  
 </ul>  
 </div>  
  
 <div>  
 <label for="sala">Seleccionar Sala:</label>  
 <select id="sala">  
 <option value="general">General</option>  
 <option value="desarrollo">Desarrollo</option>  
 <option value="ventas">Ventas</option>  
 </select>  
 <button id="unirse\_sala\_boton">Unirse</button>  
 </div>  
  
 <div>  
 <h2>Usuarios en la Sala:</h2>  
 <ul id="usuarios\_en\_sala">  
 </ul>  
 </div>  
  
 <div>  
 <h2>Crear Nueva Sala</h2>  
 <input type="text" id="nueva\_sala\_nombre" placeholder="Nombre de la nueva sala">  
 <button id="crear\_sala\_boton">Crear Sala</button>  
 <p id="mensaje\_creacion\_sala" style="color: red;"></p>  
 </div>  
  
 <div id="mensajes"></div>  
 <div>  
 <input type="text" id="mensaje\_input" placeholder="Escribe tu mensaje...">  
 <input type="text" id="destinatario\_input" placeholder="(Opcional) Nombre del destinatario (privado)">  
 <button id="enviar\_boton">Enviar</button>  
 </div>  
  
 <audio id="notificacion\_audio" style="display:none;">  
 <source src="notification-2-269292.mp3" type="audio/mpeg">  
 <source src="notificacion.wav" type="audio/wav">  
 Tu navegador no soporta la reproducción de audio.  
 </audio>  
  
 <script>  
 const socket = io('http://localhost:5000');  
 const mensajesDiv = document.getElementById('mensajes');  
 const mensajeInput = document.getElementById('mensaje\_input');  
 const destinatarioInput = document.getElementById('destinatario\_input');  
 const enviarBoton = document.getElementById('enviar\_boton');  
 const salaSelect = document.getElementById('sala');  
 const unirseSalaBoton = document.getElementById('unirse\_sala\_boton');  
 const usuariosEnSalaLista = document.getElementById('usuarios\_en\_sala');  
 const usuariosEnLineaLista = document.getElementById('usuarios\_en\_linea');  
 const nuevaSalaNombreInput = document.getElementById('nueva\_sala\_nombre');  
 const crearSalaBoton = document.getElementById('crear\_sala\_boton');  
 const mensajeCreacionSala = document.getElementById('mensaje\_creacion\_sala');  
 const usuario = prompt('Escriba su nombre:', 'Usuario Anónimo');  
  
 let originalTitle = document.title;  
 let newMessage = false;  
 const notificacionAudio = document.getElementById('notificacion\_audio');  
  
 window.addEventListener('focus', () => {  
 newMessage = false;  
 document.title = originalTitle;  
 });  
  
 socket.on('connect', () => {  
 console.log('Conectado al servidor como:', usuario);  
 socket.emit('establecer\_nombre\_usuario', { usuario: usuario });  
 });  
  
 socket.on('nuevo\_mensaje', (data) => {  
 const nuevoMensaje = document.createElement('p');  
 nuevoMensaje.textContent = `${data.usuario}: ${data.texto}`;  
 if (data.usuario === usuario) {  
 nuevoMensaje.classList.add('local-user');  
 }  
 const timestampDiv = document.createElement('div');  
 timestampDiv.classList.add('timestamp');  
 const date = new Date(data.timestamp);  
 timestampDiv.textContent = date.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' }) + ' - ' + date.toLocaleDateString();  
 nuevoMensaje.appendChild(timestampDiv);  
 mensajesDiv.appendChild(nuevoMensaje);  
 mensajesDiv.scrollTop = mensajesDiv.scrollHeight;  
  
 if (!document.hasFocus() && notificacionAudio) {  
 newMessage = true;  
 document.title = "(¡Nuevo mensaje!) " + originalTitle;  
 notificacionAudio.play();  
 }  
 });  
  
 socket.on('mensaje\_privado', (data) => {  
 console.log('Mensaje privado recibido en el frontend:', data);  
 const nuevoMensaje = document.createElement('p');  
 nuevoMensaje.textContent = `(Privado de ${data.remitente}): ${data.texto}`;  
 nuevoMensaje.classList.add('mensaje-privado');  
 if (data.remitente === usuario) {  
 nuevoMensaje.classList.add('local-user');  
 }  
 const timestampDiv = document.createElement('div');  
 timestampDiv.classList.add('timestamp');  
 const date = new Date(data.timestamp);  
 timestampDiv.textContent = date.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' }) + ' - ' + date.toLocaleDateString();  
 nuevoMensaje.appendChild(timestampDiv);  
 mensajesDiv.appendChild(nuevoMensaje);  
 mensajesDiv.scrollTop = mensajesDiv.scrollHeight;  
  
 if (!document.hasFocus() && notificacionAudio) {  
 newMessage = true;  
 document.title = "(¡Nuevo mensaje privado!) " + originalTitle;  
 notificacionAudio.play();  
 }  
 });  
  
 socket.on('mensaje\_sistema', (data) => {  
 const nuevoMensaje = document.createElement('p');  
 nuevoMensaje.textContent = `[Sistema]: ${data.texto}`;  
 nuevoMensaje.classList.add('mensaje-sistema');  
 mensajesDiv.appendChild(nuevoMensaje);  
 mensajesDiv.scrollTop = mensajesDiv.scrollHeight;  
 });  
  
 socket.on('usuario\_unido', (data) => {  
 const nuevoMensaje = document.createElement('p');  
 nuevoMensaje.textContent = `[Sistema]: ${data.usuario} se ha unido a la sala ${data.sala}`;  
 nuevoMensaje.classList.add('mensaje-sistema');  
 mensajesDiv.appendChild(nuevoMensaje);  
 mensajesDiv.scrollTop = mensajesDiv.scrollHeight;  
 });  
  
 socket.on('actualizar\_usuarios', (data) => {  
 if (data.sala === salaSelect.value) {  
 usuariosEnSalaLista.innerHTML = '';  
 data.usuarios.forEach(nombre => {  
 const li = document.createElement('li');  
 li.textContent = nombre;  
 usuariosEnSalaLista.appendChild(li);  
 });  
 }  
 });  
  
 socket.on('usuarios\_conectados', (usuariosConectados) => {  
 usuariosEnLineaLista.innerHTML = '';  
 usuariosConectados.forEach(user => {  
 const li = document.createElement('li');  
 li.textContent = user.nombre;  
 const onlineIndicator = document.createElement('span');  
 onlineIndicator.classList.add('online-indicator');  
 li.appendChild(onlineIndicator);  
 usuariosEnLineaLista.appendChild(li);  
 });  
 });  
  
 socket.on('sala\_creada', (data) => {  
 if (data.error) {  
 mensajeCreacionSala.textContent = `Error al crear la sala: ${data.error}`;  
 setTimeout(() => {  
 mensajeCreacionSala.textContent = '';  
 }, 3000);  
 } else if (data.nombre\_sala) {  
 const nuevaOpcion = document.createElement('option');  
 nuevaOpcion.value = data.nombre\_sala;  
 nuevaOpcion.textContent = data.nombre\_sala;  
 salaSelect.appendChild(nuevaOpcion);  
 salaSelect.value = data.nombre\_sala; // Seleccionar automáticamente la nueva sala  
 socket.emit('unirse\_sala', { sala: data.nombre\_sala, usuario: usuario }); // Unirse a la nueva sala  
 alert(`Sala "${data.nombre\_sala}" creada y te has unido.`);  
 }  
 });  
  
 socket.on('actualizar\_lista\_salas', (data) => {  
 const salas = data.salas;  
 salaSelect.innerHTML = ''; // Limpiar las opciones existentes  
 salas.forEach(sala => {  
 const option = document.createElement('option');  
 option.value = sala;  
 option.textContent = sala;  
 salaSelect.appendChild(option);  
 });  
 });  
  
 crearSalaBoton.addEventListener('click', () => {  
 console.log('Botón "Crear Sala" clickeado.');  
 const nombreNuevaSala = nuevaSalaNombreInput.value.trim();  
 if (nombreNuevaSala) {  
 console.log('Emitiendo evento "crear\_sala" con nombre:', nombreNuevaSala);  
 socket.emit('crear\_sala', { nombre\_sala: nombreNuevaSala });  
 nuevaSalaNombreInput.value = ''; // Limpiar el input después de emitir  
 } else {  
 mensajeCreacionSala.textContent = 'Por favor, ingrese un nombre para la sala.';  
 setTimeout(() => {  
 mensajeCreacionSala.textContent = '';  
 }, 3000);  
 }  
 });  
  
 enviarBoton.addEventListener('click', () => {  
 const texto = mensajeInput.value.trim();  
 const destinatario = destinatarioInput.value.trim();  
 if (texto) {  
 const payload = { usuario: usuario, texto: texto };  
 if (destinatario) {  
 payload['destinatario'] = destinatario;  
 }  
 socket.emit('mensaje', payload);  
 mensajeInput.value = '';  
 destinatarioInput.value = '';  
 }  
 });  
  
 unirseSalaBoton.addEventListener('click', () => {  
 const salaSeleccionada = salaSelect.value;  
 socket.emit('unirse\_sala', { sala: salaSeleccionada, usuario: usuario });  
 mensajesDiv.innerHTML = '';  
 usuariosEnSalaLista.innerHTML = '';  
 });  
 </script>  
</body>  
</html>