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

import axios from 'axios';

import { io } from 'socket.io-client';

import './Chat.css';

import { useParams } from 'react-router-dom';

import { useNavigate } from 'react-router-dom';

function Chat() {

  const [user, setUser] = useState(null);

  const [students, setStudents] = useState([]);

  const [chattedStudents, setChattedStudents] = useState([]);

  const [activeStudent, setActiveStudent] = useState(null);

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

  const [newMessage, setNewMessage] = useState('');

  const [unreadMessages, setUnreadMessages] = useState(() => {

    // Load from localStorage on first render

    const stored = localStorage.getItem("unreadMessages");

    return stored ? JSON.parse(stored) : {};

  });

  useEffect(() => {

    // Save unread messages to localStorage whenever they change

    localStorage.setItem("unreadMessages", JSON.stringify(unreadMessages));

  }, [unreadMessages]);

  const [favoriteStudents, setFavoriteStudents] = useState([]);

  const socketRef = useRef(null);

  const chatContainerRef = useRef(null);

  const { studentId } = useParams();

  const [student, setStudent] = useState(null);

useEffect(() => {

  const fetchStudent = async () => {

    if (!studentId) {

      console.error('No studentId provided.');

      return;

    }

    const token = sessionStorage.getItem('token');

    if (!token) {

      console.error('No token found. User not authenticated.');

      return;

    }

    try {

      const response = await axios.get(`http://localhost:3001/api/verifiedStudents/${studentId}`, {

        headers: { Authorization: `Bearer ${token}` },

      });

      setStudent(response.data);

    } catch (err) {

      console.error('Error fetching student details:', err.response?.data || err.message);

    }

  };

  // Only fetch if studentId exists

  if (studentId) fetchStudent();

}, [studentId]);

  // Fetch the logged-in user details

  useEffect(() => {

    const fetchUser = async () => {

      const token = sessionStorage.getItem('token');

      if (!token) {

        console.error('No token found. User not authenticated.');

        return;

      }

      try {

        const response = await axios.get('http://localhost:3001/api/getUserDetails', {

          headers: {

            Authorization: `Bearer ${token}`,

          },

        });

        if (response.data && response.data.\_id) {

          setUser(response.data);

        } else {

          console.error('User data is missing \_id:', response.data);

        }

      } catch (err) {

        console.error('Error fetching user:', err.response?.data || err.message);

      }

    };

    fetchUser();

  }, []);

  // Fetch chatted students

  useEffect(() => {

    const fetchChattedStudents = async () => {

      try {

        const token = sessionStorage.getItem('token');

        if (!token) {

          console.error('Token is missing');

          return;

        }

        const response = await axios.get('http://localhost:3001/api/chattedStudents', {

          headers: {

            Authorization: `Bearer ${token}`,

          },

          withCredentials: true,

        });

        console.log('Chatted students:', response.data); // Log the response

        setChattedStudents(response.data); // Update your state with the data

      } catch (err) {

        console.error('Error fetching chatted students:', err.response?.data || err.message);

      }

    };

    fetchChattedStudents();

  }, [user]);

  // Fetch all students except the logged-in user

  useEffect(() => {

    const fetchStudents = async () => {

      if (user) {

        try {

          const { data } = await axios.get('http://localhost:3001/api/verifiedStudents', { withCredentials: true });

          setStudents(data.filter(student => student.\_id !== user.\_id));  // Remove logged-in user

        } catch (err) {

          console.error('Error fetching students:', err);

        }

      }

    };

    fetchStudents();

  }, [user]);

  useEffect(() => {

    if (!socketRef.current) {

      socketRef.current = io('http://localhost:3001');

      socketRef.current.on('connect', () => {

        console.log('Connected to socket server. Socket ID:', socketRef.current.id);

      });

      socketRef.current.on('disconnect', () => {

        console.log('Disconnected from socket server');

      });

    }

    const handleIncomingMessage = (message) => {

      const { senderId, receiverId } = message;

      console.log('Incoming message:', message);

      console.log('Current user:', user?.\_id);

      console.log('Active student:', activeStudent?.\_id);

      if (!activeStudent) {

        const peerId = senderId !== user.\_id ? senderId : receiverId;

        console.log('No student selected, setting unread for:', peerId);

        setUnreadMessages((prev) => {

          const updated = { ...prev, [peerId]: (prev[peerId] || 0) + 1 };

          console.log('Unread updated (no student selected):', updated);

          return updated;

        });

        setChattedStudents((prev) => {

          const exists = prev.some(s => s.\_id === peerId);

          if (exists) return prev;

          const found = students.find(s => s.\_id === peerId);

          return found ? [...prev, found] : prev;

        });

        return;

      }

      const isActiveChat =

        senderId === activeStudent.\_id || receiverId === activeStudent.\_id;

      console.log('Is active chat:', isActiveChat);

      const isSentByCurrentUser = senderId === user.\_id;

      if (isActiveChat) {

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

        if (!isSentByCurrentUser) {

          setUnreadMessages((prev) => {

            const updated = { ...prev };

            delete updated[senderId];

            console.log('Cleared unread for active chat:', updated);

            return updated;

          });

        }

      } else {

        if (!isSentByCurrentUser) {

          setUnreadMessages((prev) => {

            const updated = { ...prev, [senderId]: (prev[senderId] || 0) + 1 };

            console.log('Incremented unread for:', senderId, updated);

            return updated;

          });

        }

      }

    };

    socketRef.current.on('newMessage', handleIncomingMessage);

    return () => {

      socketRef.current.off('newMessage', handleIncomingMessage);

    };

  }, [activeStudent, user, students]);

  const fetchChatHistory = async (student) => {

    try {

      const token = sessionStorage.getItem('token'); // Or wherever you store the auth token

      const { data } = await axios.get(`http://localhost:3001/api/chat/${student.\_id}`, {

        params: { userId: user.\_id },

        headers: {

          Authorization: `Bearer ${token}`,

        },

      });

      setMessages(data);

    } catch (err) {

      console.error('Error fetching chat history:', err);

    }

  };

  const handleStartChat = async (student) => {

    if (!user || !student) {

      console.log("User or student not provided. Exiting function.");

      return;

    }

    console.log("Starting chat with student:", student);

    setActiveStudent(student);

    setMessages([]); // Clear previous chat history

    console.log("Cleared previous chat history.");

    const senderId = user.\_id;

    const receiverId = student.\_id;

    const roomName = [senderId, receiverId].sort().join('-');

    console.log("Generated room name:", roomName);

    setChattedStudents((prev) => {

      const exists = prev.some((s) => s.\_id === student.\_id);

      if (exists) {

        console.log("Student already in chatted list.");

        return prev;

      } else {

        console.log("Adding student to chatted list.");

        return [...prev, student];

      }

    });

    if (socketRef.current) {

      console.log("Emitting 'joinRoom' with roomName:", roomName);

      socketRef.current.emit('joinRoom', roomName, (err) => {

        if (err) {

          console.error('Error joining room:', err);

        } else {

          console.log(`Joined room: ${roomName}`);

        }

      });

    }

    // Clear only the unread for the student you're now chatting with

    setUnreadMessages((prev) => {

      const updated = { ...prev };

      delete updated[student.\_id];

      console.log("Updated unread messages:", updated);

      return updated;

    });

    fetchChatHistory(student);

    console.log("Fetching chat history for student:", student.\_id);

    // Mark messages as read when starting the chat

    try {

      console.log("Sending request to mark messages as read for:", { senderId, receiverId });

      const response = await axios.put('http://localhost:3001/api/messages/markAsRead', {

        senderId: user.\_id,

        receiverId: student.\_id,

      });

      console.log('Messages marked as read:', response.data);

    } catch (err) {

      console.error('Error marking messages as read:', err.response?.data || err.message);

    }

  };

  const [isSending, setIsSending] = useState(false);

  const handleSendMessage = () => {

    if (!newMessage.trim() || !activeStudent || !user) {

      console.error('Missing information');

      return;

    }

    const senderId = user.\_id;

    const receiverId = activeStudent.\_id;

    const roomName = [senderId, receiverId].sort().join('-');

    const msg = { senderId, receiverId, text: newMessage };

    socketRef.current?.emit('newMessage', { room: roomName, message: msg });

    setNewMessage(''); // Clear the message input field

  };

  // Ensure that the chat container scrolls to the bottom when a new message is added

  useEffect(() => {

    if (chatContainerRef.current) {

      chatContainerRef.current.scrollTop = chatContainerRef.current.scrollHeight;

    }

  }, [messages]);

  const [searchQuery, setSearchQuery] = useState('');

  const handleSearchChange = (e) => {

    setSearchQuery(e.target.value);

  };

  // Filter students based on search query and exclude chatted students

  const filteredStudents = students.filter((student) =>

    !chattedStudents.some(chattedStudent => chattedStudent.\_id === student.\_id) &&

    (student.name.toLowerCase().includes(searchQuery.toLowerCase()) ||

     student.specification.toLowerCase().includes(searchQuery.toLowerCase()))

  );

  const toggleFavorite = async (studentId) => {

    try {

      // Check if the student is already in favorites

      if (favoriteStudents.includes(studentId)) {

        // Remove from favorites

        setFavoriteStudents((prev) => prev.filter((id) => id !== studentId));

      } else {

        // Add to favorites

        setFavoriteStudents((prev) => [...prev, studentId]);

        // Send POST request to backend to save the favorite

        const token = sessionStorage.getItem('token'); // Get the token

        if (!token) {

          console.error('User is not authenticated');

          return;

        }

        const response = await axios.post(

          `http://localhost:3001/api/favoriteStudent/${studentId}`,

          {},

          {

            headers: {

              Authorization: `Bearer ${token}`,

            },

          }

        );

        console.log(response.data.message); // Show success message from backend

      }

    } catch (error) {

      console.error('Error favoriting student:', error.response?.data || error.message);

    }

  };

  const unfavoriteStudent = async (favoriteStudentId) => {

    try {

      const token = sessionStorage.getItem('token'); // Get the token

      if (!token) {

        console.error('User is not authenticated');

        return;

      }

      // Send DELETE request to remove the student from favorites

      const response = await axios.delete(

        `http://localhost:3001/api/unfavoriteStudent/${favoriteStudentId}`,

        {

          headers: {

            Authorization: `Bearer ${token}`,

          },

        }

      );

      console.log(response.data.message); // Show success message from backend

      // Remove from the favoriteStudents list in state

      setFavoriteStudents((prev) => prev.filter((id) => id !== favoriteStudentId));

    } catch (error) {

      console.error('Error unfavoriting student:', error.response?.data || error.message);

    }

  };

  const token = sessionStorage.getItem('token');

  const fetchFavorites = async () => {

    if (!token) {

      console.log('User is not logged in');

      return;

    }

    try {

      const response = await axios.get('http://localhost:3001/api/favoriteStudents', {

        headers: { Authorization: `Bearer ${token}` },

      });

      const favoriteStudentIds = response.data.map(fav => fav.favoriteStudentId.\_id);

      setFavoriteStudents(favoriteStudentIds);

    } catch (error) {

      console.error('Error fetching favorite students:', error);

    }

  };

  useEffect(() => {

    if (token) {

      fetchFavorites();

    }

  }, [token]);

  // Filter favorite students

  const favoriteStudentsList = chattedStudents.filter(student => favoriteStudents.includes(student.\_id));

  // Filter non-favorite students

  const nonFavoriteStudentsList = chattedStudents.filter(student => !favoriteStudents.includes(student.\_id));

  const sortedChattedStudents = [...favoriteStudentsList, ...nonFavoriteStudentsList];

  useEffect(() => {

    const fetchStudentAndStartChat = async () => {

      if (!studentId || !user) return; // <- wait for both studentId AND user

      const token = sessionStorage.getItem('token');

      try {

        const res = await axios.get(`http://localhost:3001/api/verifiedStudents/${studentId}`, {

          headers: { Authorization: `Bearer ${token}` },

        });

        const student = res.data;

        setActiveStudent(student);

        const senderId = user.\_id;

        const receiverId = student.\_id;

        const roomName = [senderId, receiverId].sort().join('-');

        socketRef.current?.emit('joinRoom', roomName, (err) => {

          if (err) console.error("Socket join error:", err);

          else console.log("Joined room:", roomName);

        });

        const msgRes = await axios.get(`http://localhost:3001/api/chat/${student.\_id}`, {

          params: { userId: user.\_id, peerId: student.\_id },

          headers: { Authorization: `Bearer ${token}` },

        });

        setMessages(msgRes.data);

      } catch (error) {

        console.error("Error loading student/chat:", error);

      }

    };

    fetchStudentAndStartChat();

  }, [studentId, user]); // <- add user to dependencies

  useEffect(() => {

  if (!activeStudent) {

    if (Object.keys(unreadMessages).length > 0) {

      console.log("🔔 You have unread messages:", unreadMessages);

    }

  }

}, [unreadMessages, activeStudent]);

  return (

    <div className="Chat-container">

    <div className="sidebar">

  <input

    type="text"

    value={searchQuery}

    onChange={handleSearchChange}

    placeholder="Search peers by name or specification..."

  />

  {searchQuery.trim() && (

    filteredStudents.length > 0 ? (

      filteredStudents.map(student => (

        <div key={student.\_id} onClick={() => handleStartChat(student)}>

          {student.name} - {student.specification}

        </div>

      ))

    ) : (

      <div>No matching students found.</div>

    )

  )}

  <h3 className="catted">Chats</h3>

  {chattedStudents.length === 0 ? (

    <div>No chatted students found.</div>

  ) : (

    // Show chats

    chattedStudents.map(student => (

      <div key={student.\_id} onClick={() => handleStartChat(student)}>

        {/\* Show unread indicator \*/}

        {unreadMessages[student.\_id] && (

          <span className="unread-indicator">●</span>

        )}

        <span>{student.name}</span>

      </div>

    ))

  )}

  {/\* If no student selected, still show unread messages indicator \*/}

  {!activeStudent && (

    Object.keys(unreadMessages).length > 0 ? (

      <div className="unread-indicator-global">

        You have unread messages

      </div>

    ) : (

      <div>No unread messages</div>

    )

  )}

</div>

      <div className="chat-box">

        {activeStudent ? (

          <>

            <h2>Chat with {activeStudent.name}</h2>

            <div className="chat-messages" ref={chatContainerRef}>

              {messages.slice().reverse().map((msg, idx) => (

                <div key={idx} className={`message ${msg.senderId === user.\_id ? 'sent' : 'received'}`}>

                  <strong>{msg.senderId === user.\_id ? 'You' : activeStudent.name}</strong>: {msg.text}

                </div>

              ))}

            </div>

            <div className="chat-input">

              <input

                type="text"

                value={newMessage}

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

                placeholder="Type a message..."

                onKeyDown={(e) => e.key === 'Enter' && handleSendMessage()}

              />

              <button onClick={handleSendMessage} disabled={isSending}>Send</button>

            </div>

          </>

        ) : (

          <h2>No student selected</h2>

        )}

      </div>

    </div>

  );

}

export default Chat;

/\* Global styles \*/

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

html, body {

  height: 88vh;

  width: 100%;

  font-family: 'Arial', sans-serif;

  background-color: #f4f4f4;

  display: flex;

  flex-direction: column;

}

/\* Main container \*/

.Chat-container {

  display: flex;

  height: 75vh;

  width: 100vw;

}

/\* Sidebar (left) \*/

.sidebar {

  width: 330px;

  background-color: #ffffff;

  border-color: #485460;

  color: white;

  padding: 20px;

  display: flex;

  flex-direction: column;

  justify-content: flex-start;

  overflow-y: auto; /\* Enable scrolling for the sidebar if needed \*/

}

.sidebar input {

  padding: 10px;

  margin-bottom: 15px;

  border: 1px solid #ccc;

  border-radius: 4px;

  font-size: 14px;

}

.catted{

  text-align: left;

}

.sidebar div {

  padding: 10px;

  margin-bottom: 10px;

  cursor: pointer;

  background-color: #48742F;

  border-radius: 5px;

  transition: background-color 0.3s ease;

}

.sidebar div:hover {

  background-color:#48742F;

}

.sidebar div.active {

  background-color: #1e2b35;

}

.sidebar div.active:hover {

  background-color: #1a232e;

}

/\* Chat box (right) \*/

.chat-box {

  flex: 1;

  display: flex;

  flex-direction: column;

  padding: 20px;

}

.chat-box h2 {

  color: #333;

  margin-bottom: 15px;

}

/\* Chat messages \*/

.chat-messages {

  flex: 1;

  overflow-y: auto;

  margin-bottom: 20px;

}

.chat-input {

  display: flex;

  padding: 5px;

  background-color: #fff;

  position: sticky; /\* Sticks to the bottom of the chat container \*/

}

.chat-input input {

  flex-grow: 1; /\* Make the input box flexible \*/

  padding: 10px;

  font-size: 14px;

  border: 1px solid #ccc;

  border-radius: 4px;

  margin-right: 10px;

  resize: none; /\* Optional: Remove resizing behavior \*/

  min-width: 150px; /\* Optional: Set a minimum width \*/

}

.chat-input button {

  padding: 10px 20px;

  font-size: 14px;

  background-color: #48742F;

  color: white;

  border-radius: 4px;

  cursor: pointer;

}

.chat-input button:disabled {

  background-color: #ccc;

}

.chat-messages {

  display: flex;

  flex-direction: column;

  padding: 10px;

  overflow-y: auto;

  scroll-behavior: smooth; /\* Enables smooth scrolling \*/

}

.message {

  padding: 10px;

  margin: 5px;

  border-radius: 8px;

  max-width: 40%; /\* Maximum width for the message \*/

  word-wrap: break-word; /\* Allow long words to break and wrap onto the next line \*/

  white-space: pre-wrap; /\* Ensures the line breaks are preserved \*/

}

.sent {

  background-color: #4CAF50;

  align-self: flex-end;

  color: #f4f4f4;

  border-radius: 8px;

  padding: 10px;

  margin: 5px;

}

.received {

  background-color: #9e7608;

  align-self: flex-start;

  color: #f4f4f4;

}

/\* Scroll behavior \*/

.chat-messages::-webkit-scrollbar {

  width: 6px;

}

.chat-messages::-webkit-scrollbar-thumb {

  background-color:#48742F;

  border-radius: 10px;

}

.green-dot {

  width: 10px;

  height: 10px;

  background-color: rgb(183, 245, 12);

  border-radius: 50%;

  display: inline-block;

  margin-left: 5px;

}

.heart-icon {

  cursor: pointer;

  margin-left: 10px;

  font-size: 18px;

}

.heart-icon.active {

  color: red;

}

.chatted-student {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 6px 0;

}

.chat-messages {

  max-height: 400px;

  overflow-y: auto;

  display: flex;

  flex-direction: column-reverse;

}

.chat-input {

  display: flex;

  justify-content: space-between;

}

.chat-input input {

  width: 85%;

}

.unread-badge {

  background-color: rgb(45, 238, 45);

  color: white;

  border-radius: 50%;

  padding: 4px 8px;

  font-size: 12px;

  margin-left: 8px;

  display: inline-block;

  min-width: 20px;

  text-align: center;

  animation: pulse 1.5s infinite;

}

@keyframes pulse {

  0% {

    transform: scale(1);

    opacity: 0.7;

  }

  50% {

    transform: scale(1.2);

    opacity: 1;

  }

  100% {

    transform: scale(1);

    opacity: 0.7;

  }

}

app.put('/api/messages/markAsRead', async (req, res) => {

  const { senderId, receiverId } = req.body;

  if (!senderId || !receiverId) {

    return res.status(400).json({ error: 'Missing senderId or receiverId' });

  }

  try {

    await Message.updateMany(

      { senderId, receiverId, isRead: false },

      { $set: { isRead: true } }

    );

    res.sendStatus(200);

  } catch (err) {

    res.status(500).json({ error: 'Failed to mark messages as read' });

  }

});