**AIM: Database: Using MongoDB, build the CRUD operations with NodeJS.**

**CODE:**

// <script src="/socket.io/socket.io.js"></script>

  <script>

    // Current username

    let currentUsername = '';

    // Format timestamp

    function formatTimestamp(timestamp) {

      const date = new Date(timestamp);

      return date.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' });}

    // Handle username submission

    usernameForm.addEventListener('submit', (e) => {

      e.preventDefault();

      currentUsername = usernameInput.value.trim();

      if (currentUsername) {

        // Create user via API

        fetch('/api/users', {

          method: 'POST',

          headers: {

            'Content-Type': 'application/json'},

          body: JSON.stringify({ username: currentUsername })})

        .then(response => {

          if (response.status === 409) {

            // Username exists, just join

            return { success: true };}

          return response.json();})

        .then(data => {

          if (data.error) {

            alert(`Error: ${data.error}`);

            return;}          // Join the chat

          socket.emit('join', currentUsername);

          usernameScreen.style.display = 'none';

          chatScreen.style.display = 'flex';

          // Fetch online users

          fetchUsers();})

        .catch(error => {

          console.error('Error:', error);

          alert('An error occurred. Please try again.');  });}});

    // Handle sending messages

    chatForm.addEventListener('submit', (e) => {

      e.preventDefault();

      const message = messageInput.value.trim();

      if (message && currentUsername) {

        // Send message via API

        fetch('/api/messages', {

          method: 'POST',

          headers: {

            'Content-Type': 'application/json'  },

          body: JSON.stringify({ user: currentUsername, text: message })  })

        .then(response => response.json())

        .then(data => {

          if (data.error) {

            alert(`Error: ${data.error}`);

            return;}

          messageInput.value = '';

          messageInput.focus();})

        .catch(error => {

          console.error('Error:', error);

          alert('Failed to send message. Please try again.');  });}  });

    // Fetch all messages

    function fetchMessages() {

      fetch('/api/messages')

        .then(response => response.json())

        .then(messages => {

          chatMessages.innerHTML = '';

          messages.forEach(message => {

            displayMessage(message);});

          // Auto-scroll to the bottom

          chatMessages.scrollTop = chatMessages.scrollHeight;})

        .catch(error => {

          console.error('Error fetching messages:', error);});  }

    // Fetch all users

    function fetchUsers() {

      fetch('/api/users')

        .then(response => response.json())

        .then(users => {

          usersList.innerHTML = '';

          users.forEach(user => {

            const li = document.createElement('li');

            li.textContent = user.username;

            if (user.username === currentUsername) {

              li.textContent += ' (You)';

              li.style.fontWeight = 'bold';  }

            usersList.appendChild(li);});})

        .catch(error => {

          console.error('Error fetching users:', error);});}

    // Display a message

    function displayMessage(message) {

      const messageElement = document.createElement('div');

      if (message.user === 'System') {

        messageElement.classList.add('message', 'system-message');

        messageElement.innerHTML = `

          <div>${message.text}</div>

          <div class="timestamp">${formatTimestamp(message.timestamp)}</div>`;} else if (message.user === currentUsername) {

        messageElement.classList.add('message', 'user-message');

        messageElement.innerHTML = `

          <div class="actions">

            <button class="edit-btn" data-id="${message.\_id}">Edit</button>

            <button class="delete-btn" data-id="${message.\_id}">Delete</button>

          </div>

          <div>${message.text}</div>

          <div class="timestamp">${formatTimestamp(message.timestamp)}</div>`;

        // Add event listeners for edit and delete buttons

        setTimeout(() => {

          const editBtn = messageElement.querySelector('.edit-btn');

          const deleteBtn = messageElement.querySelector('.delete-btn');

          if (editBtn) {

            editBtn.addEventListener('click', function() {

              const messageId = this.getAttribute('data-id');

              openEditModal(messageId, message.text);});}

          if (deleteBtn) {

            deleteBtn.addEventListener('click', function() {

              const messageId = this.getAttribute('data-id');

              openDeleteModal(messageId);});}

        }, 0);

      } else {

        messageElement.classList.add('message', 'other-message');

        messageElement.innerHTML = `

          <div><strong>${message.user}</strong>: ${message.text}</div>

          <div class="timestamp">${formatTimestamp(message.timestamp)}</div>`;}

      chatMessages.appendChild(messageElement);}

    // Open edit modal

    function openEditModal(messageId, text) {

      editMessageId.value = messageId;

      editMessage.value = text;

      editModal.style.display = 'flex';}

    // Open delete modal

    function openDeleteModal(messageId) {

      deleteMessageId.value = messageId;

      deleteModal.style.display = 'flex';}

    // Edit form submit handler

    editForm.addEventListener('submit', (e) => {

      e.preventDefault();

      const messageId = editMessageId.value;

      const newText = editMessage.value.trim();

      if (newText) {

        // Update message via API

        fetch(`/api/messages/${messageId}`, {

          method: 'PUT',

          headers: {

            'Content-Type': 'application/json'},

          body: JSON.stringify({ text: newText })})

        .then(response => response.json())

        .then(data => {

          if (data.error) {

            alert(`Error: ${data.error}`);

            return;}

          // Close modal

          editModal.style.display = 'none';})

        .catch(error => {

          console.error('Error:', error);

          alert('Failed to update message. Please try again.');});}  });

       // Cancel edit button handler

    cancelEdit.addEventListener('click', () => {

      editModal.style.display = 'none';  });

    // Delete confirmation handler

    confirmDelete.addEventListener('click', () => {

      const messageId = deleteMessageId.value;

      // Delete message via API

      fetch(`/api/messages/${messageId}`, {

        method: 'DELETE'})

      .then(response => response.json())

      .then(data => {

        if (data.error) {

          alert(`Error: ${data.error}`);

          return;}

        // Close modal

        deleteModal.style.display = 'none';

      .catch(error => {

        console.error('Error:', error);

        alert('Failed to delete message. Please try again.');});  });

    // Cancel delete button handler

    cancelDelete.addEventListener('click', () => {

      deleteModal.style.display = 'none';});

       // Refresh button handler

    refreshButton.addEventListener('click', () => {

      fetchMessages();

      fetchUsers();});

       // Logout button handler

    logoutButton.addEventListener('click', () => {

      currentUsername = '';

      chatScreen.style.display = 'none';

      usernameScreen.style.display = 'flex';

      usernameInput.value = '';});    // Socket.io events

    // Receive message history

    socket.on('messageHistory', (messages) => {

      chatMessages.innerHTML = '';

      messages.forEach(message => {

        displayMessage(message);  });

      // Auto-scroll to the bottom

      chatMessages.scrollTop = chatMessages.scrollHeight;});

      // Receive new message

    socket.on('message', (message) => {

      displayMessage(message);

      // Auto-scroll to the bottom

      chatMessages.scrollTop = chatMessages.scrollHeight;});

    // Message updated

    socket.on('messageUpdated', (message) => {

      // Refresh messages to show updated content

      fetchMessages();  });

    // Message deleted

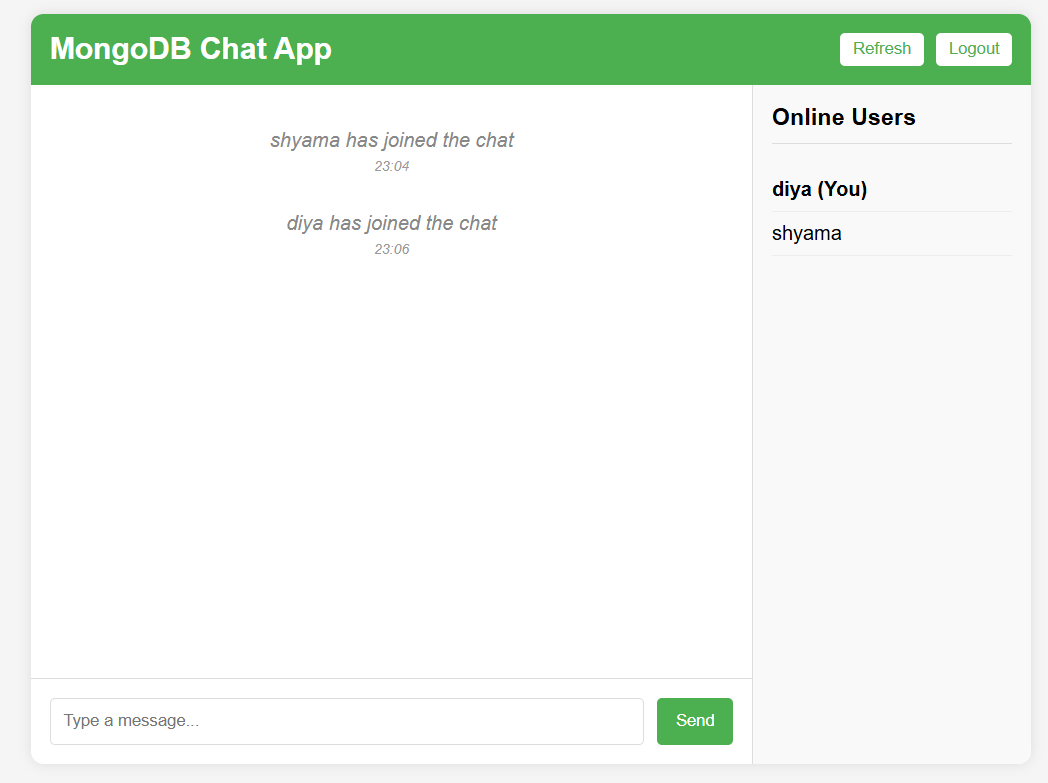
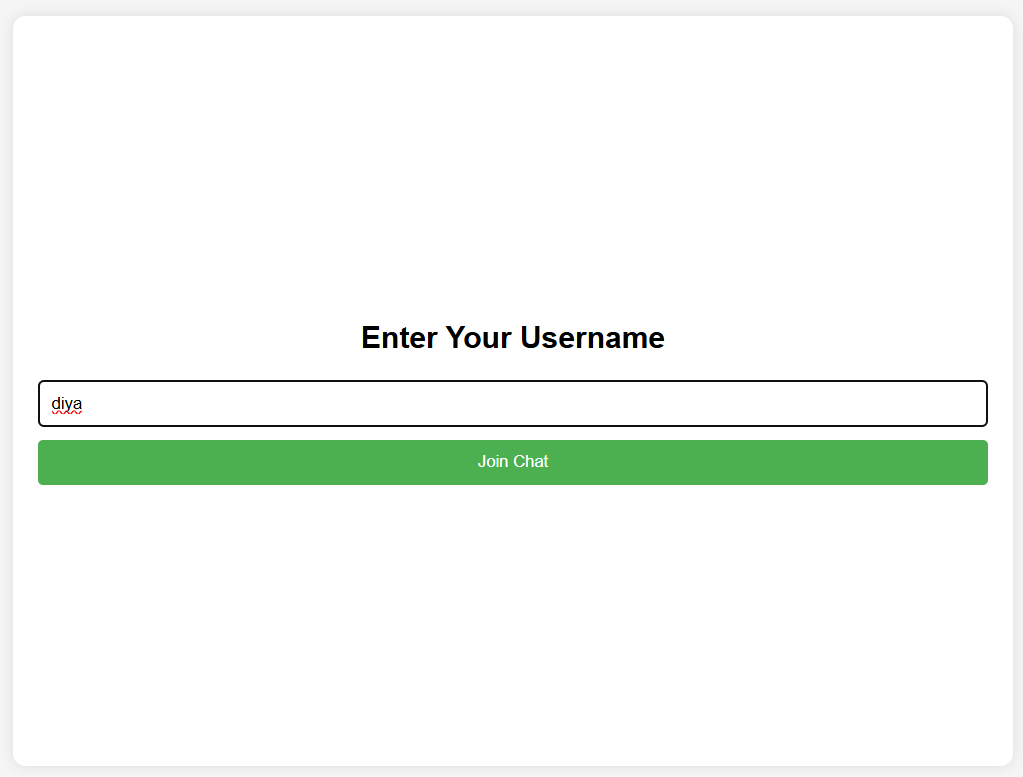
    socket.on('messageDeleted', (data) => {

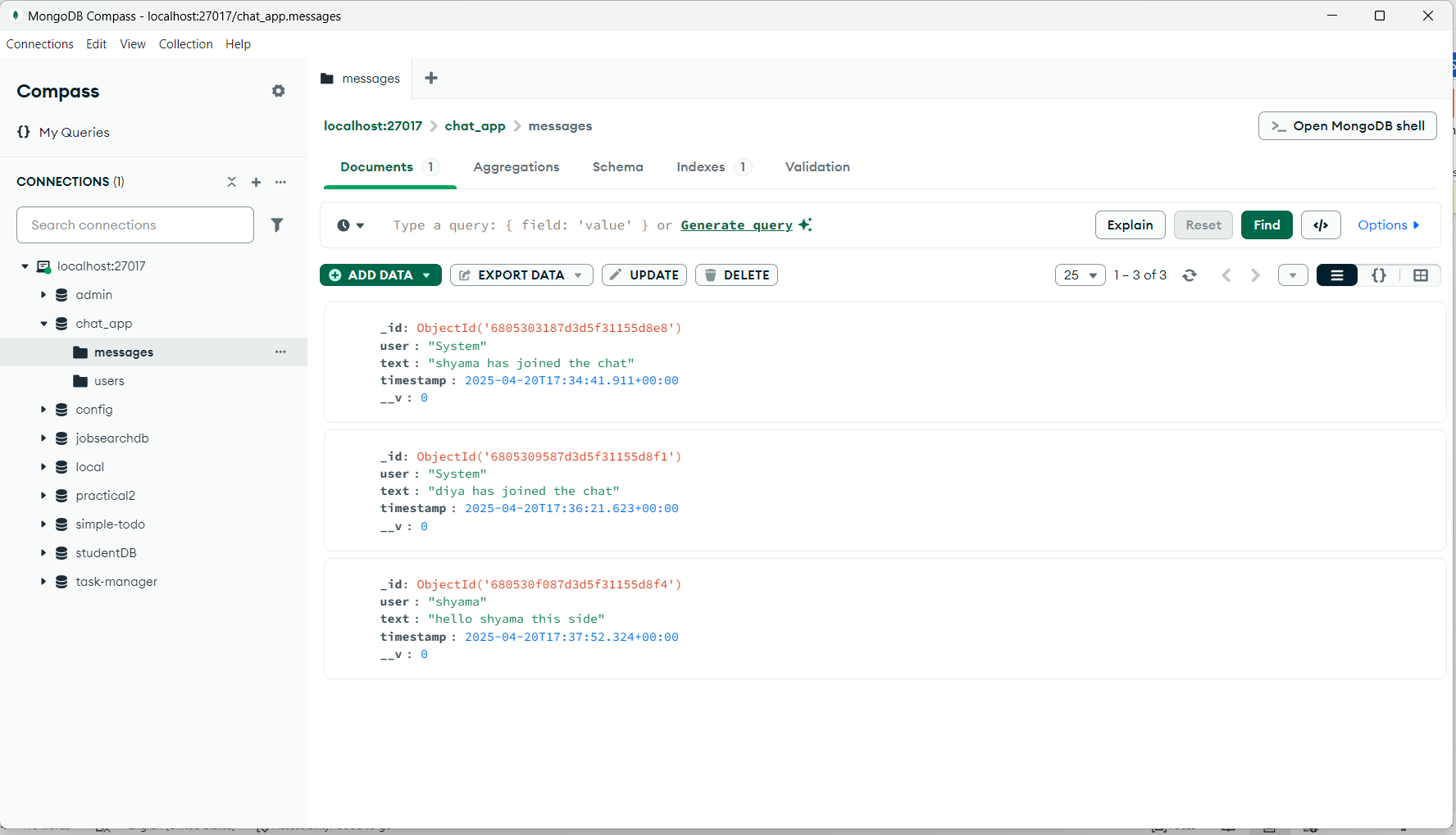
      // Refresh messages to remove deleted message

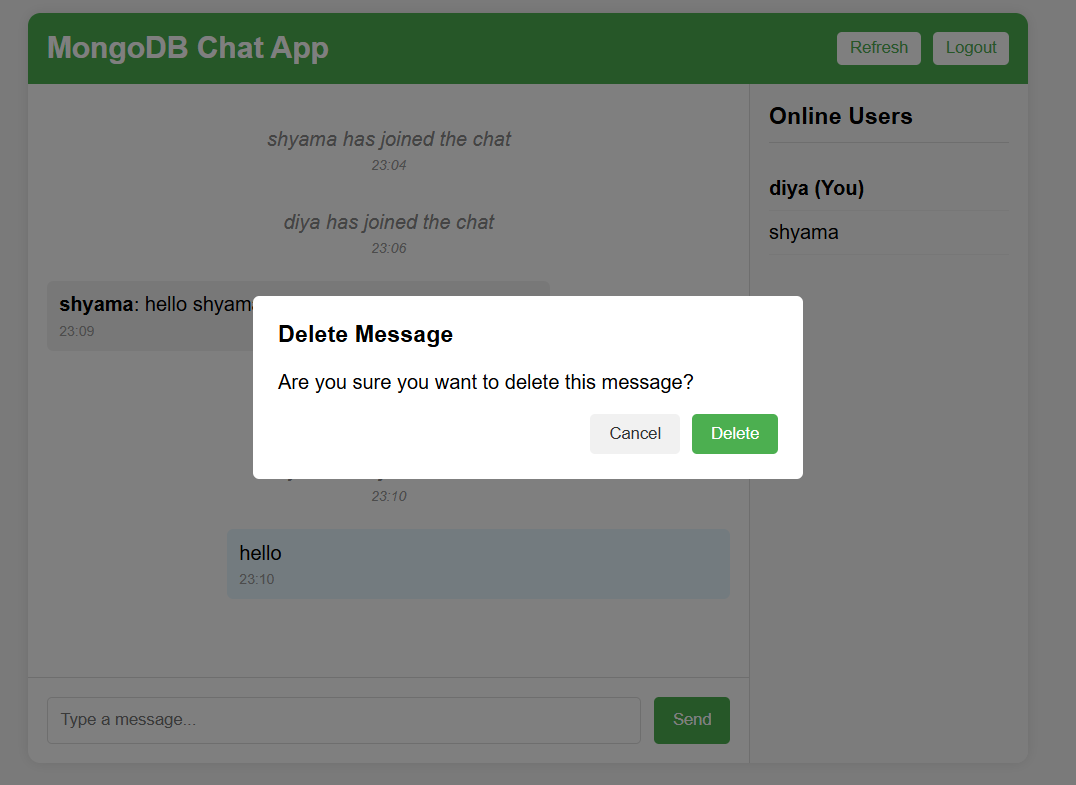
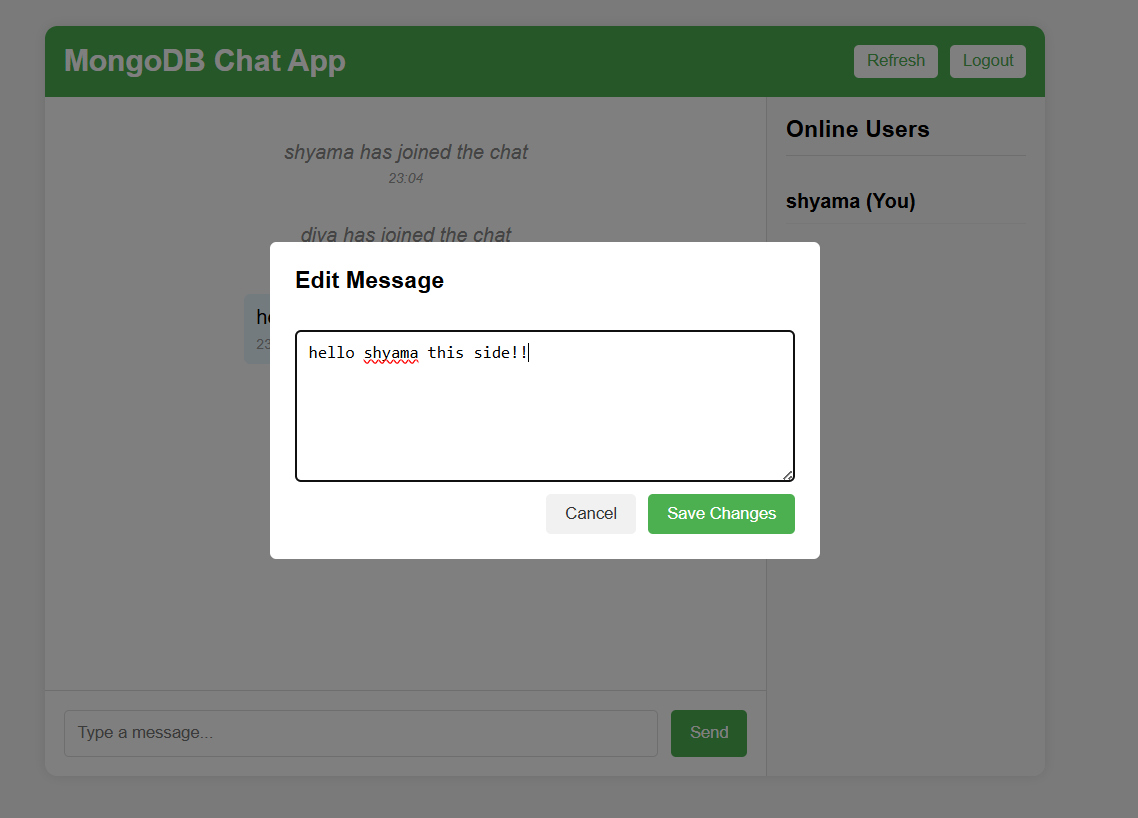
      fetchMessages();});

  </script>

**OUTPUT:**

****

****

****