A Course Based Project Report on

REAL TIME CHAT WITH FILE SHARING

Submitted to the

Department of CSE-(CyS, DS) and AI&DS

in partial fulfilment of the requirements for the completion of course

COMPUTER NETWORKS AND ETHICAL HACKING LABORATORY (22PC2CY201)

BACHELOR OF TECHNOLOGY

IN

CSE-Data Science

Submitted by

G. SRI NITHYA	23071A6786
G. ARAVIND KUMAR	23071A6787
G. VISHNU PHANI	23071A6788
G. RUTHVIK	23071A6789
G. ANIRUDH	23071A6790

Under the guidance of

Mrs. G. USHA

Assistant Professor

Department of CSE-(CyS, DS) and AI&DS

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade, NBA

VignanaJyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India

November-2025

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous, ISO 21001:2018& QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE, PE, AMS, SWEProgrammes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF (2024) Rank band:151-200in EngineeringCategory
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSEVignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.
Telephone No: 040-2304 2758/59/60, Fax: 040-23042761
E-mail: postbox@vnrvjiet.ac.in, Website: www.vnrvjiet.ac.in

Department of CSE-(CyS, DS) and AI&DS

TAMASOMA JYOTIRGAMAYA
Estd.1995

CERTIFICATE

This is to certify that the project report entitled "**REAL TIME CHAT WITH FILE SHARING**" is a bonafide work done under our supervision and is being submitted by **Miss. G. Sri Nithya (23071A6786), Mr. G. Aravind Kumar (23071A6787), Mr. G. Vishnu Phani (23071A6788), Mr. G. Ruthvik (23071A6789), Mr. G. Anirudh (23071A6790),** in partial fulfillment for the award of the degree of **Bachelor of Technology** in **CSE-Data Science**, of the VNRVJIET, Hyderabad during the academic year 2025-2026.

Mrs. G. Usha
Assistant Professor
Dept of CSE-(CyS, DS) and AI&DS

Dr. T. Sunil Kumar
Professor& HOD
Dept of CSE-(CyS, DS) and AI&DS

Course based Projects Reviewer

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade,

VignanaJyothi Nagar, Pragathi Nagar, Nizampet(SO), Hyderabad-500090, TS, India

Department of CSE-(CyS, DS) and AI&DS

DECLARATION

We declare that the course based project work entitled "**REAL TIME CHAT WITH FILE SHARING**" submitted in the Department of **CSE-(CyS, DS) and AI&DS**, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in CSE-Data Science** is a bonafide record of our own work carried out under the supervision of **Mrs. G. Usha , Assistant Professor, Department of CSE-(CyS, DS) and AI&DS, VNRVJIET.** Also, we declare that the matter embodied in this thesis has not been submitted by us in full or in any part thereof for the award of any degree/diploma of any other institution or university previously.

Place: Hyderabad.

G. Sri Nithya (23071A6786)

G. Aravind Kumar (23071A6787)

G. Vishnu Phani (23071A6788)

G. Ruthvik (23071A6789)

G. Anirudh (23071A6790)

ACKNOWLEDGEMENT

We express our deep sense of gratitude to our beloved President, Sri. D. Suresh Babu, VNR Vignana Jyothi Institute of Engineering & Technology for the valuable guidance and for permitting us to carry out this project.

With immense pleasure, we record our deep sense of gratitude to our beloved Principal, Dr. C.D Naidu, for permitting us to carry out this project.

We express our deep sense of gratitude to our beloved Professor **Dr. T. Sunil Kumar**, Professor and Head, Department of CSE-(CyS, DS) and AI&DS, VNR Vignana Jyothi Institute of Engineering & Technology, Hyderabad-500090 for the valuable guidance and suggestions, keen interest and through encouragement extended throughout the period of project work.

We take immense pleasure to express our deep sense of gratitude to our beloved Guide, Mrs. G.Usha, Assistant Professor in CSE-(CyS, DS) and AI&DS, VNR Vignana Jyothi Institute of Engineering & Technology, Hyderabad, for his/her valuable suggestions and rare insights, for constant source of encouragement and inspiration throughout my project work.

We express our thanks to all those who contributed for the successful completion of our project work.

G. SRI NITHYA	23071A6786
G. ARAVIND KUMAR	23071A6787
G. VISHNU PHANI	23071A6788
G. RUTHVIK	23071A6789
G. ANIRUDH	23071A6790

ABSTRACT

Real Time chat with file sharing is a real-time, LAN-based chat application designed to facilitate seamless communication and file sharing within a local network. This application supports broadcast and private messaging, alongside a dynamic user list and real-time file exchange, making it ideal for team collaboration and instant information sharing. Built with Node.js, Express, and Socket.IO, Real Time chat with file sharing enables users to set unique usernames, send messages to all connected users or privately to specific users, and share files instantly. The file-sharing functionality allows users to select recipients, making it flexible for both one-to-one and one-to-many interactions. The backend architecture handles user management, message broadcasting, private messaging, and secure file transfers. Real Time chat with file sharing dynamically updates the user list as participants connect or disconnect, creating an intuitive, user-friendly experience. This project showcases the application of Web-Socket technology for real-time communication over LAN, serving as an educational example of building interactive networked applications. Real Time chat with file sharing demonstrates how local network-based communication and collaboration tools can be effectively designed, enhancing understanding of networking principles and Web-Socket-driven data exchange within confined network.

TABLE OF CONTENTS

INTRODUCTION

In contemporary collaborative environments, whether they are offices, educational institutions, or small project teams, the need for instant, reliable communication and efficient file exchange is paramount. Traditional methods often rely on email or external cloud services, which can introduce latency, depend heavily on stable internet access, and may not be suitable for closed or local network setups. This reliance on external infrastructure presents a significant challenge, particularly in scenarios where internet connectivity is either restricted, unreliable, or simply unnecessary for internal team coordination. The gap identified is the absence of a dedicated, self-contained system that facilitates both real-time messaging and secure file sharing *entirely* within a Local Area Network (LAN).

The Real-Time Chat with File Sharing application directly addresses this deficiency. It is conceived and designed as a robust, LAN-based solution for seamless communication and collaboration. The system is engineered to operate exclusively within a local network, eliminating the dependency on external servers or cloud-based platforms. This local-only architecture is crucial, as it guarantees low-latency performance and ensures secure, private data exchange among connected users, making it an ideal tool for environments that prioritize internal network traffic and security.

The application's core functionality revolves around two main features: real-time communication and dynamic file sharing. Utilizing modern web technologies—specifically Node.js, Express, and Socket.IO—the platform establishes bidirectional, Web-Socket-driven connections that underpin its real-time capabilities. Users can instantly set unique usernames, send messages tonum all participants via broadcast messaging, or engage in focused, private chats with specific individuals. Furthermore, the system includes a dynamic user list that updates in real-time, providing an intuitive overview of connected peers.

1.1 PROBLEM DEFINITION

The problem addressed by Real Time chat with file sharing is the lack of a dedicated system for real-time communication and file sharing within local area networks (LANs), particularly in environments like offices, schools, or small network setups where internet connectivity may be limited or unnecessary. Existing solutions often depend on external servers or cloud-based services, making them unsaitable for closed networks. Real Time chat with file sharing aims to overcome this challenge by providing a LAN-based interactive chat application that supports low-latency communication, dynamic user management, and real-time file sharing. The solution is designed to operate entirely within a local network, ensuring seamless and secure collaboration without reliance on the internet, while offering an intuitive user experience for broadcast and private messaging.

This dependence on external infrastructure introduces vulnerabilities, including potential data privacy concerns, increased operational latency due to data traveling outside the local environment, and an outright failure point if external internet connectivity is compromised or unavailable. Therefore, the Real Time Chat with File Sharing project is not merely an alternative, but a foundational requirement for environments needing guaranteed, high-speed, and secure internal communication. By strictly limiting its operation to the LAN, the application effectively mitigates these risks, establishing a resilient communication channel tailored specifically for the needs of closed, controlled, and performance-critical local networks.

1.2 OBJECTIVE

The objective of the Real-Time Chat with File Sharing project is to develop a LAN-based communication system that enables users to engage in real-time messaging and file sharing without requiring internet connectivity. The application leverages modern web technologies, such as Node.js, Express, and Socket.IO, to facilitate low-latency, bidirectional communication. It aims to provide functionalities like broadcast messaging, private chats, dynamic user management, and interactive file sharing, creating a seamless and user-friendly platform for collaboration within closed network environments. This project also serves as an educational tool to demonstrate the practical application of Web-Socket technol.

In serving as an educational tool, the project offers a tangible demonstration of fundamental networking principles and the power of Web-Socket-driven data exchange. By implementing features like dynamic user list updates and secure file transfer protocols entirely over a LAN, the application practically illustrates concepts such as client-server architecture, broadcast mechanisms, and the persistence of real-time connections. The use of the Node.js, Express, and Socket.IO stack showcases a modern, scalable approach to building interactive networked applications, providing a valuable blueprint for students and developers seeking to understand and deploy low-latency communication systems within self-contained network boundaries.

SOURCE CODE

Server.js:

```
// server.js (edited, robust + backward compatible)
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = new Server(server);

// serve static frontend from ./public
app.use(express.static('public'));

// map socket.id -> username
let users = {};

io.on('connection', (socket) => {
  console.log('socket connected:', socket.id);

  // set username for this socket
  socket.on('set username', (rawUsername) => {
    if (!rawUsername) return;
    const username = String(rawUsername).trim();
    if (!username) return;

    users[socket.id] = username;
    io.emit('update users', Object.values(users));
    console.log(`username set for ${socket.id} -> ${username}`);
  });

  // helper to normalize incoming text payloads that might be string or { message }
  function extractText(payload) {
    if (typeof payload === 'string') return payload;
    if (payload && typeof payload === 'object' && 'message' in payload) {
      return String(payload.message || '');
    }
    return '';
  }

  // public chat message (accepts string or { message })
  socket.on('chat message', (payload) => {
    const msg = extractText(payload);
    const username = users[socket.id];
    if (!username || !msg) return;
```

```
  io.emit('chat message', { from: username, message: msg });
});

// private message to a username
socket.on('private message', (data) => {
  const { to, message } = data || {};
  if (!to || !message) {
    // optionally notify sender of malformed request
    socket.emit('error message', { reason: 'private message requires { to, message }' });
    return;
  }

  const recipientSocketId = Object.keys(users).find((id) => users[id] === to);
  if (recipientSocketId) {
    io.to(recipientSocketId).emit('private message', {
      from: users[socket.id] || 'Unknown',
      message,
    });
  } else {
    // notify sender that recipient not found
    socket.emit('error message', { reason: `User "${to}" not found or not connected.` });
  }
});

// broadcast message (accepts string or { message })
socket.on('broadcast message', (payload) => {
  const msg = extractText(payload);
  const username = users[socket.id];
  if (!username || !msg) return;
  io.emit('broadcast message', { from: username, message: msg });
});

// file upload (base64 content)
socket.on('file upload', (fileData) => {
  const { filename, fileType, content, to } = fileData || {};
  const username = users[socket.id];
  if (!username || !filename || !content) return;

  if (to) {
    // private file: send to specific user if found
    const recipientSocketId = Object.keys(users).find((id) => users[id] === to);
    if (recipientSocketId) {
      io.to(recipientSocketId).emit('file upload', {
        from: username,
        filename,
        fileType,
        content,
        isPrivate: true,
      });
```

```
      } else {
        socket.emit('error message', { reason: `User "${to}" not found for file transfer.` });
      }
    } else {
      // broadcast file to all users
      io.emit('file upload', {
        from: username,
        filename,
        fileType,
        content,
        isPrivate: false,
      });
    }
  });

  // handle disconnect
  socket.on('disconnect', () => {
    console.log('socket disconnected:', socket.id);
    delete users[socket.id];
    io.emit('update users', Object.values(users));
  });
});

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => {
  console.log(`listening on *:${PORT}`);
});
```

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 with File Sharing</title>

  <!-- <<< CSS left exactly as provided by you (no changes) >>> -->
  <style>
    body {
      font-family: Arial, sans-serif;
      background-image: url(Screenshot\ 2025-11-07\ 202858.png);
      background-repeat: no-repeat;
      background-size: cover; margin:0 ;
      padding: 0;
    }

    .chat-container {
      width: 80%;
      margin: 50px auto;
      /* background: white; */
      background-color: rgba(255, 255, 255, 0.5);
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    }
    #messages {
      list-style-type: none;
      padding: 0;
      max-height: 300px; overflow-y: auto;
      margin-bottom: 20px;
    }
    #messages li {
      padding: 8px;
      background: #f9f9f9;
      margin-bottom: 8px;
      border-radius: 4px;
    }
    #users{
      list-style-type: none;
      padding: 0; margin-bottom: 20px;
    }
    #users li{
      padding: 8px;
      background: #f9f9f9;
```

```
    margin-bottom: 8px;
    border-radius: 4px;
    cursor: pointer;
   }
   #input, #fileInput{
    width: 90%;
    padding: 20px;
    margin-top: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
    background-color: rgba(255, 255, 255, 0.5);
   }
   #broadcastInput, #privateInput,#usernameInput ,#privateUser,#fileRecipient{
    padding: 15px;
    border-radius: 0.5rem;
    border: none;
    background-color: rgba(255, 255, 255, 0.5);
   }
   #sendBroadcast, #sendPrivate, #sendFile, #setUserameButton{
    margin-top: 20px;
    cursor: pointer;
    padding: 0.5rem;
    background-color: blue;
    color: white;
    border: none;
    padding: 10px;
    border-radius: 0.5rem;
   }
  </style>
  <!-- <<< end CSS >>> -->
</head>
<body>
  <div class="chat-container">
   <h2>Real-time Chat with File Sharing</h2>

   <div id="setUsername">
    <input id="usernameInput" placeholder="Enter your username" />
    <button id="setUsernameButton">Set Username</button>
   </div>

   <h3>Connected Users:</h3>
   <ul id="users"></ul>

   <h3>Messages:</h3>
   <ul id="messages"></ul>

   <div id="privateMessage">
    <input id="privateUser" placeholder="Recipient Username" />
    <input id="privateInput" placeholder="Private Message" />
```

```
    <button id="sendPrivate">Send Private Message</button>
  </div>

  <div id="broadcastMessage">
    <input id="broadcastInput" placeholder="Enter a message to broadcast" />
    <button id="sendBroadcast">Send to All Users</button>
  </div>

  <input id="input" autocomplete="off" placeholder="Type a message..." />

  <div id="fileSharing">
    <input type="file" id="fileInput" />
    <input id="fileRecipient" placeholder="Recipient Username (leave blank for broadcast)" />
    <button id="sendFile">Send File</button>
  </div>
</div>

<!-- socket.io client -->
<script src="/socket.io/socket.io.js"></script>
<script>
  // Keep JS robust: wait for DOM load and guard element access
  window.addEventListener('load', () => {
    const socket = io();

    // track username
    let username = '';

    // helper to safely get element
    const $ = id => document.getElementById(id);

    // elements (may be null if not present)
    const setUsernameBtn = $('setUsernameButton');
    const usernameInput = $('usernameInput');
    const setUsernameDiv = $('setUsername');
    const usersListEl = $('users');
    const messagesEl = $('messages');
    const mainInput = $('input'); // matches your id
    const sendPrivateBtn = $('sendPrivate');
    const privateUserInput = $('privateUser');
    const privateInput = $('privateInput');
    const broadcastInput = $('broadcastInput');
    const sendBroadcastBtn = $('sendBroadcast');
    const fileInput = $('fileInput');
    const fileRecipient = $('fileRecipient');
    const sendFileBtn = $('sendFile');

    // Utility: escape text for HTML
    function escapeHtml(str) {
      if (!str) return '';
```

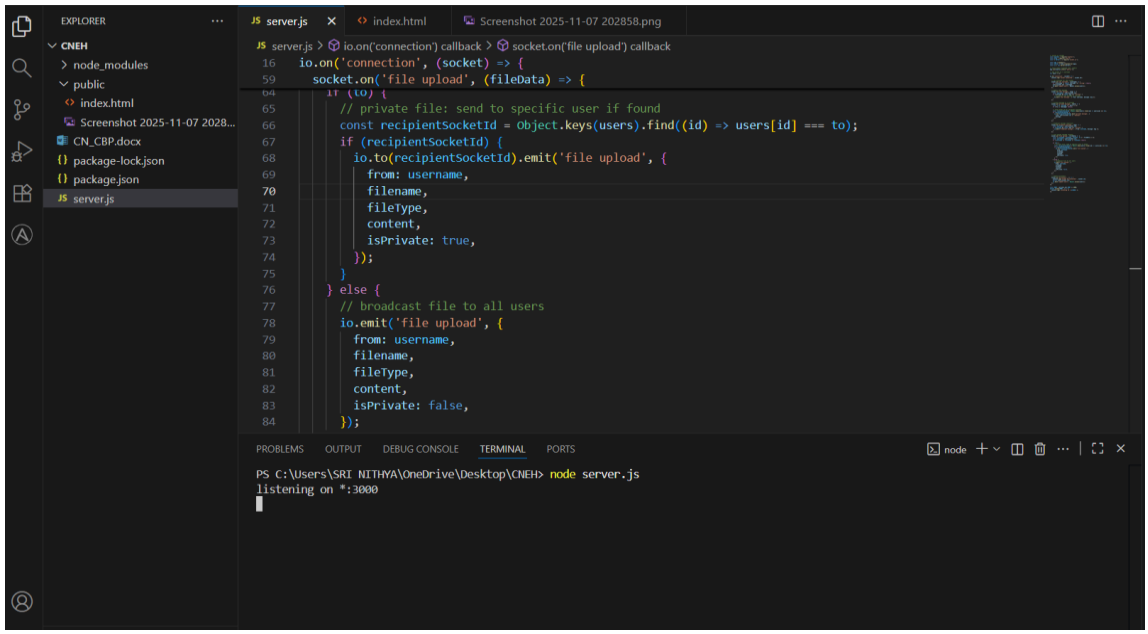
```
  return String(str)
    .replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&#039;');
}

function appendListMessage(text) {
  if (!messagesEl) return;
  const li = document.createElement('li');
  li.innerHTML = text;
  messagesEl.appendChild(li);
  messagesEl.scrollTop = messagesEl.scrollHeight;
}

// Set username
if (setUsernameBtn && usernameInput) {
  setUsernameBtn.addEventListener('click', () => {
    const input = usernameInput.value.trim();
    if (input) {
      username = input;
      socket.emit('set username', username);
      if (setUsernameDiv) setUsernameDiv.style.display = 'none';
    } else {
      alert('Please set a username before sending messages.');
    }
  });
}

// Update user list from server
socket.on('update users', (userList) => {
  if (!usersListEl) return;
  usersListEl.innerHTML = '';
  userList.forEach((user) => {
    const li = document.createElement('li');
    li.textContent = user;
    li.addEventListener('click', () => {
      if (privateUserInput) privateUserInput.value = user;
      if (fileRecipient) fileRecipient.value = user;
    });
    usersListEl.appendChild(li);
  });
});

// Display chat message
socket.on('chat message', (data) => {
  // server may send { from, message } or string; handle both
  let from = '', message = '';
```

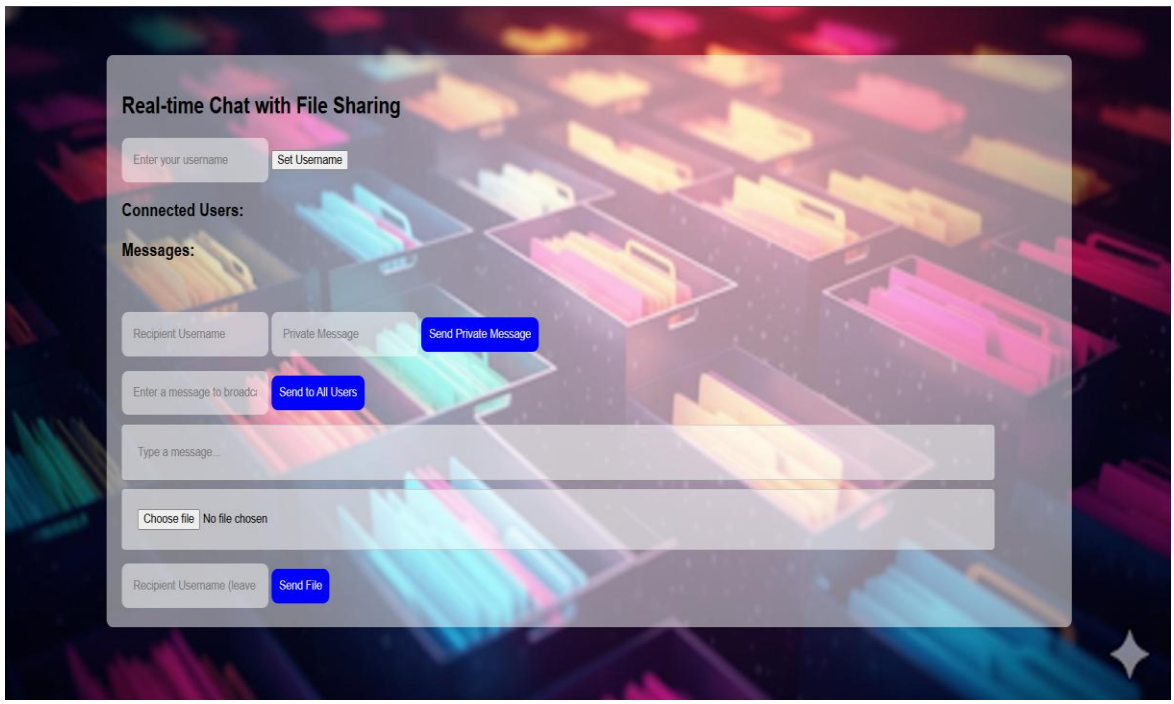
```
      if (typeof data === 'string') {
        message = data;
      } else if (data && typeof data === 'object') {
        from = data.from || '';
        message = data.message || '';
      }
      appendListMessage(`<strong>${escapeHtml(from)}</strong>: ${escapeHtml(message)}`);
    });

    // Display private message
    socket.on('private message', (data) => {
      const from = data?.from || '';
      const message = data?.message || '';
      appendListMessage(`<em>Private message from ${escapeHtml(from)}:</em> ${escapeHtml(message)}`);
    });

    // Display broadcast message
    socket.on('broadcast message', (data) => {
      const from = data?.from || '';
      const message = data?.message || (typeof data === 'string' ? data : '');
      appendListMessage(`<strong>Broadcast message from ${escapeHtml(from)}:</strong>
${escapeHtml(message)}`);
    });

    // Handle file upload (incoming)
    socket.on('file upload', (data) => {
      const { from = '', filename = '', fileType = 'application/octet-stream', content = '', isPrivate = false } = data || {};
      const li = document.createElement('li');
      const link = document.createElement('a');
      link.href = `data:${fileType};base64,${content}`;
      link.download = filename || 'file';
      link.textContent = `${isPrivate ? 'Private' : 'Broadcast'} file from ${from}: ${filename}`;
      li.appendChild(link);
      if (messagesEl) {
        messagesEl.appendChild(li);
        messagesEl.scrollTop = messagesEl.scrollHeight;
      }
    });

    // Send chat message by pressing Enter in main input
    if (mainInput) {
      mainInput.addEventListener('keypress', (e) => {
        if (e.key === 'Enter') {
          if (!username) return alert('Set username first');
          const message = e.target.value;
          if (message && message.trim()) {
            // emit string message (server may accept string)
            socket.emit('chat message', message.trim());
            e.target.value = '';
```

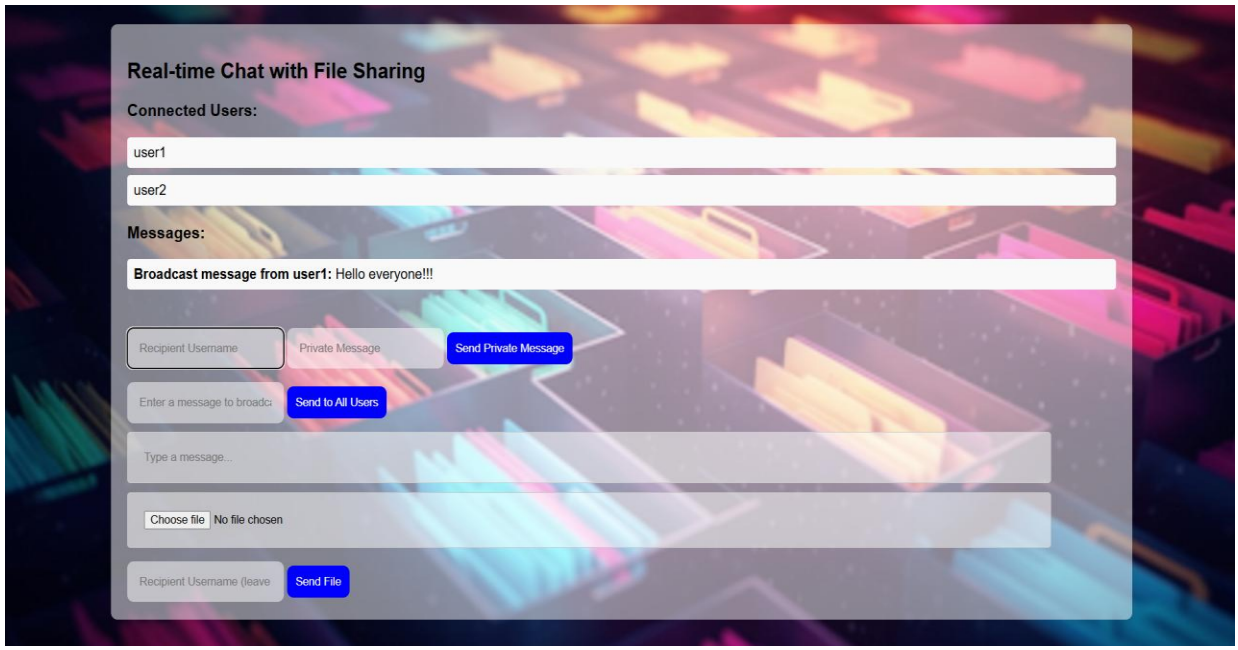
```
      }
    }
  });
}

// Send private message
if (sendPrivateBtn) {
  sendPrivateBtn.addEventListener('click', () => {
    if (!username) return alert('Set username first');
    const to = privateUserInput ? privateUserInput.value.trim() : '';
    const message = privateInput ? privateInput.value.trim() : '';
    if (!to || !message) {
      alert('Provide recipient and message.');
      return;
    }
    socket.emit('private message', { to, message });
    if (privateInput) privateInput.value = '';
  });
}

// Send broadcast message
if (sendBroadcastBtn) {
  sendBroadcastBtn.addEventListener('click', () => {
    if (!username) return alert('Set username first');
    const message = broadcastInput ? broadcastInput.value.trim() : '';
    if (!message) return;
    socket.emit('broadcast message', message );
    if (broadcastInput) broadcastInput.value = '';
  });
}

// Send file
if (sendFileBtn) {
  sendFileBtn.addEventListener('click', () => {
    if (!username) return alert('Set username first');
    if (!fileInput) return alert('Choose a file first');
    if (!fileInput.files || !fileInput.files.length) return alert('Choose a file first');
    const recipient = fileRecipient ? fileRecipient.value.trim() : '';
    const file = fileInput.files[0];
    const reader = new FileReader();
    reader.onload = () => {
      const content = reader.result.split(',')[1] || '';
      socket.emit('file upload', {
        filename: file.name,
        fileType: file.type || 'application/octet-stream',
        content,
        to: recipient || null
      });
      // clear inputs
```

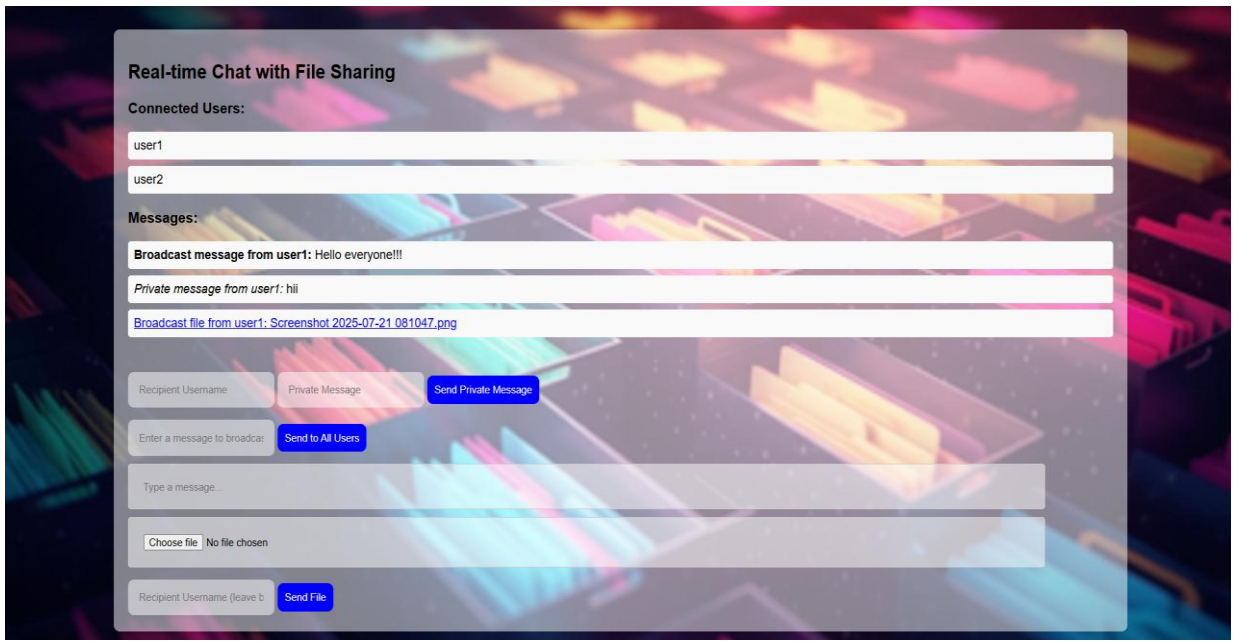
```
          fileInput.value = '';
          if (fileRecipient) fileRecipient.value = '';
        };
        reader.readAsDataURL(file);
      });
    }

  }); // end load listener
 </script>
</body>
</html>
```

OUTPUTS


EXPLORER
CNEH
node_modules
public
index.html
Screenshot 2025-11-07 2028...
CN_CBP.docx
package-lock.json
package.json
server.js
server.js
index.html
Screenshot 2025-11-07 202858.png
server.js > io.on('connection') callback > socket.on('file upload') callback
io.on('connection', (socket) => {
socket.on('file upload', (fileData) => {
if (to) {
// private file: send to specific user if found
const recipientSocketId = Object.keys(users).find((id) => users[id] === to);
if (recipientSocketId) {
io.to(recipientSocketId).emit('file upload', {
from: username,
filename,
fileType,
content,
isPrivate: true,
});
}
} else {
// broadcast file to all users
io.emit('file upload', {
from: username,
filename,
fileType,
content,
isPrivate: false,
});
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
node
PS C:\Users\SRI NITHYA\OneDrive\Desktop\CNEH> node server.js
listening on *:3000


Real-time Chat with File Sharing
Enter your username
Set Username
Connected Users:
Messages:
Recipient Username
Private Message
Send Private Message
Enter a message to broadc
Send to All Users
Type a message...
Choose file
No file chosen
Recipient Username (leave
Send File


Real-time Chat with File Sharing
Connected Users:
user1
user2
Messages:
Broadcast message from user1: Hello everyone!!!
Recipient Username
Private Message
Send Private Message
Enter a message to broadc
Send to All Users
Type a message...
Choose file
No file chosen
Recipient Username (leave
Send File


Real-time Chat with File Sharing
Connected Users:
user1
user2
Messages:
Broadcast message from user1: Hello everyone!!!
Private message from user1: hii
Broadcast file from user1: Screenshot 2025-07-21 081047.png
Recipient Username
Private Message
Send Private Message
Enter a message to broadca
Send to All Users
Type a message...
Choose file
No file chosen
Recipient Username (leave b
Send File

CONCLUSION

The Real-Time Chat with File Sharing project demonstrates the effectiveness of leveraging modern web technologies like Node.js, Express, and Socket.IO to create a seamless, secure, and efficient LAN-based communication platform. By eliminating the dependency on internet connectivity, the application proves invaluable in environments such as offices, schools, or localized setups. It successfully integrates key features like broadcast messaging, private chats, dynamic user management, and real-time file sharing, fostering collaboration and teamwork within a closed network. This project not only provides a practical solution for real-time communication but also highlights the potential of Web-Socket-driven technology in developing responsive and low-latency applications, bridging the oretical concepts with real-world implementation This project not only fulfills practical communication requirements but also serves as an educational demonstration of how modern web frameworks can be used to build responsive and reliable local network applications.

REFERENCES

- https://www.tutorialspoint.com/socket.io/
- https://socket.io/docs/v4/
- https://developer.mozilla.org/en-US/docs/Web/API/WebSockets%20API
- https://expressjs.com/