Chat Application using Socket.IO and Node.js with the help of TCP and UDP port

Presented by-

DreamWeavers Squad

Our Team

Nandan - RA2211031010046

Anuj Tiwari - RA2211031010052

Aditya Dubey - RA2211031010072

Ayan Saxena - RA2211031010063

Topic

- Problem Statement
- Literature Review
- Plan of Action/Member Tasks
- References

PROBLEM STATEMENT

Overview:

• Need for real-time chat communication between users over the internet.

Challenges:

Handling multiple clients, minimizing message latency.

Objectives:

•Develop a chat application with user registration, messaging, and email notifications.

Literature Survey

WebSocket Technology:

Real-time communication, bidirectional data transfer.

•Comparison with HTTP-based communication.

• Review of similar applications: WhatsApp, Slack, etc.

•Findings: Socket.IO for real-time messaging, MongoDB for database, and Nodemailer for email services.

Proposed Sys. Architecture

- User registration and authentication (MongoDB).
- Real-time messaging (Socket.IO).
- Email notification (Nodemailer).
- TCP server setup for direct communication.

Flowchart

- 1. User registers.
- 2. Data stored in MongoDB.
- 3. Real-time messaging through Socket.IO.
- 4. Email sent upon registration.
- 5. TCP connection established for direct communication.

Results

- Screenshots of MongoDB database showing users and messages.
- Real-time chat interface.
- Email notifications sent to users.
- Analysis: Application performance, latency, and reliability.

Conclusion

Summary:

•Successfully developed a real-time chat application.

•Future Work: Enhance security, add file sharing, improve authentication.

•Lessons Learned: Challenges and solutions during development.

Code Implementation

Key Features:

- MongoDB connection for user and message storage.
- Real-time communication using Socket.IO.
- Nodemailer for email notifications.
- TCP server for direct communication.

Code Highlights:

- MongoDB connection setup.
- WebSocket event handling.
- Nodemailer configuration for sending emails.

TCP vs. UDP in Computer Networks

TCP (Transmission Control Protocol):

•Reliable, connection-oriented protocol.

•Ensures data is delivered in the correct order.

•Example Use Case: Web applications, file transfers.

UDP (User Datagram Protocol):

•Connectionless, faster but less reliable.

•Suitable for applications where speed is critical and occasional data loss is acceptable.

•Example Use Case: Video streaming, online gaming.

TCP Server Code Example

TCP Server using Node.js:

```
const net = require('net');
const server = net.createServer((socket) => {
console.log('New TCP connection:', socket.remoteAddress);
socket.on('data', (data) => {
console.log('Received:', data.toString());
});
socket.on('close', () => {
console.log('Connection closed');
});
});
server.listen(6000, () => console.log('TCP server running on port 6000'));`
```

UDP Server Code Example

UDP Server using Node.js:

```
const dgram = require('dgram');
const server = dgram.createSocket('udp4');
server.on('message', (msg, rinfo) => {
console.log(`Received: ${msg} from ${rinfo.address}:${rinfo.port}`);
});
server.bind(6000, () => console.log('UDP server listening on port 6000'));
```

Thank You