컴퓨터 네트워크

Design Project

복수 사용자 메신저 프로그램 보고서

팀명: 김영광_구자빈_윤성욱

1. 개요

이 프로젝트는 복수 사용자를 지원하는 메신저 프로그램을 React와 Node.js 기반으로 개발하는 것을 목표로 한다. 이 프로그램은 개인 채팅 및 단체 채팅 기능을 제공하며, 실시간 통신을 위해 Socket.IO를 사용한다.

2. 프로그램 구조

클라이언트 (React)

- 상태 관리: `useState`를 사용하여 사용자 ID, 온라인 사용자 목록, 개인 및 단체 채팅 데이터 등을 관리한다.

- 효과 관리: `useEffect`를 사용하여 초기 로그인 상태를 설정하고, Socket.IO 이벤트 리스너를 설정하며, 필요한 API 호출을 수행한다.

- UI 구성:

  - 로그인 및 로그아웃 기능

  - 온라인 사용자 목록 표시 및 선택

  - 개인 채팅 및 단체 채팅 인터페이스

  - 채팅 메시지 보내기 및 받기

  - 단체 채팅방 생성 및 초대 기능

서버 (Node.js + Express + Socket.IO)

- 데이터 저장: 사용자 정보, 개인 채팅 및 단체 채팅 데이터는 JSON 파일로 관리된다.

- API 제공:

- 사용자 로그인 및 로그아웃 처리

- 온라인 사용자 및 채팅 데이터 제공

- 단체 채팅방 관리 (생성, 초대, 삭제)

- 소켓 통신: 클라이언트와의 실시간 통신을 관리하며, 메시지 송수신 및 사용자 상태 업데이트를 처리한다.

3. 주요 기능

- 로그인 및 세션 관리: 사용자는 ID를 입력하여 로그인하며, 세션 정보는 서버와 클라이언트 모두에서 관리된다.

- 실시간 사용자 상태 업데이트: 사용자의 접속 상태는 실시간으로 업데이트되며, 온라인 상태의 사용자만 메시지를 주고받을 수 있다.

- 개인 및 단체 채팅: 사용자는 다른 온라인 사용자와 1:1로 메시지를 주고받거나, 여러 사용자가 참여하는 단체 채팅방에서 소통할 수 있다.

- 단체 채팅방 관리: 사용자는 단체 채팅방을 생성하고, 다른 사용자를 초대하거나 채팅방을 삭제할 수 있다.

4. 사용 편의성 및 확장성

- 인터페이스: 클라이언트 인터페이스는 최대한 간단하게 유지되며, 모든 주요 기능은 명확하게 접근 가능하다.

- 확장성: 시스템은 향후 기능 추가를 쉽게 할 수 있도록 설계되었다. 예를 들어, 메시지 포맷은 HTTP 요청과 유사하게 헤더와 바디로 구성되어 향후 기능 확장을 용이하게 한다.

5. 코드 주요 함수 기능 설명

로그인 및 세션 관리

- **handleLogin**: 사용자가 ID 를 입력하고 '로그인' 버튼을 클릭하면 이 함수가 호출된다. 입력된 ID 와 소켓 ID 를 서버로 전송하며, 로그인 성공 시 사용자의 온라인 상태 및 세션 정보가 업데이트된다.

- **handleLogout**: 현재 로그인한 사용자가 '로그아웃' 버튼을 클릭할 경우 이 함수가 호출된다. 사용자의 온라인 상태를 오프라인으로 변경하고 세션 정보를 초기화한다.

메시지 전송 및 관리

- **handlePrivateMessageSend**: 개인 채팅에서 메시지를 전송하는 함수. 사용자가 메시지 입력 후 전송 버튼을 클릭하면 해당 메시지를 소켓을 통해 다른 사용자에게 직접 전송한다.
- **handleGroupMessageSend**: 단체 채팅에서 메시지를 전송하는 함수. 해당 채팅방의 멤버인지 확인 후 메시지를 채팅방에 속한 모든 사용자에게 전송한다.

채팅방 관리

- **createGroupChat**: 새로운 단체 채팅방을 생성하는 함수. 사용자가 채팅방 이름을 입력하고 생성 버튼을 클릭하면 해당 이름으로 새로운 채팅방을 생성한다.
- **inviteToGroupChat**: 특정 사용자를 단체 채팅방에 초대하는 함수. 초대하려는 사용자의 ID 를 서버에 전송하여 초대 처리를 수행한다.
- **deleteGroupChat**: 단체 채팅방을 삭제하는 함수. 채팅방 관리자만이 해당 기능을 수행할 수 있으며, 삭제 시 모든 채팅 데이터가 제거된다.

2. 서버 사이드 주요 함수 및 기능 (Node.js + Express + Socket.IO)

사용자 및 세션 관리

- **/login, /logout 엔드포인트**: 사용자 로그인 및 로그아웃을 처리하는 API. 로그인 시 사용자 ID 및 소켓 ID 를 받아 사용자의 온라인 상태를 업데이트하고, 로그아웃 시 사용자의 온라인 상태를 오프라인으로 설정한다.

메시지 관리

- **socket.on('message',...)**: 개인 메시지를 처리하는 소켓 이벤트. 메시지를 받은 후 저장하고 해당 사용자에게 메시지를 전달한다.
- **socket.on('groupMessage',...)**: 단체 메시지를 처리하는 소켓 이벤트. 메시지를 해당 단체 채팅방의 모든 멤버에게 전달한다.

채팅방 관리

- **/create-group-chat, /invite-to-group, /delete-group-chat 엔드포인트**: 단체 채팅방의 생성, 초대, 삭제를 처리하는 API. 각각 새로운 채팅방을 생성하거나, 사용자를 채팅방에 초대하고, 채팅방을 삭제하는 기능을 제공한다.

5. 주요코드 설명

5.1 server.js

1. 모듈 및 서버 설정

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const fs = require('fs');
const cors = require('cors');
const path = require('path');

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

const PORT = process.env.PORT || 5005;

let onlineUsers = {};
let privateChats = {};
let groupChats = {};
```

- **모듈 임포트**: Express, http, socket.io, fs, cors, path 모듈을 불러옵니다.
- **서버 생성**: Express 앱을 생성하고 HTTP 서버와 Socket.IO 서버를 설정합니다.
- **데이터 초기화**: 온라인 사용자, 개인 채팅, 단체 채팅 정보를 저장할 객체를 초기화합니다.

2. 서버 시작 시 데이터 로드

```
if (fs.existsSync('onlineUsers.json')) {
    const data = fs.readFileSync('onlineUsers.json');
    onlineUsers = JSON.parse(data);
}
if (fs.existsSync('privateChats.json')) {
    const data = fs.readFileSync('privateChats.json');
    privateChats = JSON.parse(data);
}
if (fs.existsSync('groupChats.json')) {
    const data = fs.readFileSync('groupChats.json');
    groupChats = JSON.parse(data);
}
```

- **데이터 복원**: 서버가 시작될 때 기존에 저장된 사용자, 세션, 채팅 데이터를 JSON 파일로부터 읽어옵니다.

3. 미들웨어 및 정적 파일 제공

```
app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname, 'build')));
```

- **CORS 허용**: 다양한 출처의 요청을 허용합니다.
- **JSON 파싱**: 요청 본문을 JSON 형식으로 파싱합니다.
- **정적 파일 제공**: React 애플리케이션의 빌드된 정적 파일을 제공합니다.

4. API 엔드포인트

```
app.post('/login', (req, res) => {
    const { id, ip, port, socketId } = req.body;
    onlineUsers[id] = { ip, port, online: true, socketId };
    fs.writeFileSync('onlineUsers.json', JSON.stringify(onlineUsers));
    res.send(onlineUsers);
});

app.post('/logout', (req, res) => {
    const { id } = req.body;
    if (onlineUsers[id]) {
        onlineUsers[id].online = false;
        fs.writeFileSync('onlineUsers.json', JSON.stringify(onlineUsers));
        res.send({ success: true });
    } else {
        res.send({ success: false });
    }
});

app.get('/online-users', (req, res) => {
    res.send(onlineUsers);
});

app.get('/private-chats', (req, res) => {
    res.send(privateChats);
});

app.get('/group-chats', (req, res) => {
    res.send(groupChats);
});
```

```
// 단체 채팅방 생성 엔드포인트
app.post('/create-group-chat', (req, res) => {
    const { roomName, creatorId } = req.body;
    if (groupChats[roomName]) {
        return res.status(400).send({ success: false, message: '채팅방 이름이 이미 존재합니
    }
    groupChats[roomName] = {
        members: [creatorId],
        messages: []
    };
    fs.writeFileSync('groupChats.json', JSON.stringify(groupChats));
    res.send({ success: true, roomName, members: groupChats[roomName].members });
});

// 단체 채팅방에 사용자 초대 엔드포인트
app.post('/invite-to-group', (req, res) => {
    const { roomName, inviterId, inviteeId } = req.body;
    if (!groupChats[roomName]) {
        return res.status(400).send({ success: false, message: '채팅방이 존재하지 않습니다.
    }
    if (!groupChats[roomName].members.includes(inviterId)) {
        return res.status(400).send({ success: false, message: '초대할 권한이 없습니다.' })
    }
    if (!groupChats[roomName].members.includes(inviteeId)) {
        groupChats[roomName].members.push(inviteeId);
        fs.writeFileSync('groupChats.json', JSON.stringify(groupChats));
        const userSocketId = onlineUsers[inviteeId]?.socketId;
        if (userSocketId) {
            io.to(userSocketId).emit('invitedToGroup', { roomName });
        }
    }
    res.send({ success: true, roomName, members: groupChats[roomName].members });
});
```

```
// 단체 채팅방 삭제 엔드포인트
app.post('/delete-group-chat', (req, res) => {
    const { roomName, requesterId } = req.body;
    if (!groupChats[roomName]) {
        return res.status(400).send({ success: false, message: '채팅방이 존재하지 않습니다.
    }
    if (!groupChats[roomName].members.includes(requesterId)) {
        return res.status(400).send({ success: false, message: '채팅방 삭제 권한이 없습니다
    }
    delete groupChats[roomName];
    fs.writeFileSync('groupChats.json', JSON.stringify(groupChats));
    io.to(roomName).emit('groupDeleted', { roomName });
    res.send({ success: true });
});

app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname, 'build', 'index.html'));
});
```

- **로그인/로그아웃 엔드포인트**: 사용자의 로그인 상태를 관리합니다.
- **데이터 제공 엔드포인트**: 온라인 사용자 목록, 개인 채팅, 단체 채팅 목록을 제공합니다.
- **단체 채팅방 관리 엔드포인트**: 단체 채팅방 생성, 초대, 삭제 기능을 제공합니다.

5. 소켓 이벤트 핸들러

```
io.on('connection', (socket) => {
    console.log('New client connected');

    socket.on('join', (id) => {
        if (onlineUsers[id]) {
            onlineUsers[id].online = true;
            onlineUsers[id].socketId = socket.id;
            fs.writeFileSync('onlineUsers.json', JSON.stringify(onlineUsers));
            socket.join(id);
            console.log(`${id} joined`);
            io.emit('updateUsers', onlineUsers);
        }
    });

    socket.on('leave', (id) => {
        if (onlineUsers[id]) {
            onlineUsers[id].online = false;
            fs.writeFileSync('onlineUsers.json', JSON.stringify(onlineUsers));
            io.emit('updateUsers', onlineUsers);
        }
    });
```

... 사이 여러 가지 소켓 이벤트코드 존재(생략)

```
    socket.on('disconnect', () => {
        console.log('Client disconnected');
        let disconnectedUser;
        for (let [key, value] of Object.entries(onlineUsers)) {
            if (value.socketId === socket.id) {
                onlineUsers[key].online = false;
                disconnectedUser = key;
                break;
            }
        }
        if (disconnectedUser) {
            fs.writeFileSync('onlineUsers.json', JSON.stringify(onlineUsers));
            io.emit('updateUsers', onlineUsers);
        }
    });
});

server.listen(PORT, () => console.log(`Server is running on port ${PORT}`));
```

- **소켓 연결 및 이벤트**: 사용자가 접속, 접속 해제, 메시지 송수신, 단체 채팅방 메시지 송수신

등의 이벤트를 처리합니다.

- **데이터 업데이트**: 각 이벤트에 따라 사용자, 개인 채팅, 단체 채팅 데이터를 업데이트하고, 이를 파일에 저장합니다.

5.2 App.js

1. 클라이언트 설정 및 상태 관리

```
import React, { useState, useEffect } from 'react';
import io from 'socket.io-client';

const socket = io('http://localhost:5005');

function App() {
    const [id, setId] = useState('');
    const [onlineUsers, setOnlineUsers] = useState({});
    const [isLoggedIn, setIsLoggedIn] = useState(false);
    const [privateChats, setPrivateChats] = useState({});
    const [activePrivateChat, setActivePrivateChat] = useState('');
    const [privateMessage, setPrivateMessage] = useState('');
    const [groupChats, setGroupChats] = useState({});
    const [activeGroupChat, setActiveGroupChat] = useState('');
    const [groupMessage, setGroupMessage] = useState('');
    const [roomName, setRoomName] = useState('');
    const [error, setError] = useState('');
```

- **상태 관리**: 여러 useState 훅을 사용하여 ID, 온라인 사용자 목록, 개인 및 단체 채팅 데이터, 에러 메시지 등을 관리합니다.

2. 초기화 및 이벤트 리스너 설정

```
useEffect(() => {
    const savedId = sessionStorage.getItem('id');
    if (savedId) {
        setId(savedId);
        setIsLoggedIn(true);
        socket.emit('join', savedId);
    }

    if (isLoggedIn) {
        fetch('http://localhost:5005/online-users')
            .then(response => response.json())
            .then(data => {
                setOnlineUsers(data);
            })
            .catch(err => console.error('Failed to fetch online users:', err));

        fetch('http://localhost:5005/private-chats')
            .then(response => response.json())
            .then(data => {
                setPrivateChats(data);
            })
            .catch(err => console.error('Failed to fetch private chats:', err));

        fetch('http://localhost:5005/group-chats')
            .then(response => response.json())
            .then(data => {
                setGroupChats(data);
            })
```

... 이외의 코드들 생략

```
        socket.on('groupDeleted', ({ roomName }) => {
            setGroupChats(prevChats => {
                const updatedChats = { ...prevChats };
                delete updatedChats[roomName];
                return updatedChats;
            });
            if (activeGroupChat === roomName) {
                setActiveGroupChat('');
            }
            alert(`단체 채팅방 ${roomName}이 삭제되었습니다.`);
        });

        socket.on('error', ({ message }) => {
            setError(message);
            setTimeout(() => setError(''), 3000);  // 3초 후 에러 메시지 제거
        });

        return () => {
            socket.off('updateUsers');
            socket.off('message');
            socket.off('newGroupMessage');
            socket.off('invitedToGroup');
            socket.off('groupDeleted');
            socket.off('error');
        };
    }
}, [isLoggedIn, activeGroupChat]);
```

- **초기화**: 저장된 ID가 있는 경우 자동 로그인 처리.
- **API 호출**: 세션 및 개인 채팅 데이터 로드.
- **이벤트 리스너**: 소켓 이벤트 리스너를 설정하여 실시간 데이터를 수신.

3. 로그인 및 로그아웃 처리

```
const handleLogin = () => {
    fetch('http://localhost:5005/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ id, socketId: socket.id })
    }).then(res => res.json())
        .then(data => {
            setOnlineUsers(data);
            setIsLoggedIn(true);
            sessionStorage.setItem('id', id);
            alert('로그인 성공');
            socket.emit('join', id);
        });
};

const handleLogout = () => {
    fetch('http://localhost:5005/logout', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ id })
    }).then(() => {
        setIsLoggedIn(false);
        setId('');
        sessionStorage.removeItem('id');
        alert('로그아웃 성공');
        socket.emit('leave', id);
    });
};
```

- **로그인**: 서버에 로그인 요청을 보내고, 성공 시 온라인 사용자 목록을 갱신.
- **로그아웃**: 서버에 로그아웃 요청을 보내고, 클라이언트 상태를 초기화.

4. 메시지 전송 및 세션 관리

- **개인 메시지 전송**: 사용자가 메시지를 입력하여 전송하면, 해당 메시지를 소켓을 통해 상대방에게 전송합니다.
- **단체 메시지 전송**: 단체 채팅방에서 메시지를 전송하면, 세션에 참여한 모든 사용자에게 메시지를 전송합니다.

- **단체 채팅방 생성**: 새로운 채팅방을 생성하고, 사용자가 해당 방에 참여합니다.
- **사용자 초대**: 특정 사용자를 단체 채팅방에 초대합니다.