

# web\_2 김동훈

B. MERN + socket.io를 이용한 채팅 앱  
→ socket.io로 실시간 소통 가능  
→ example: discord 등

<https://joseph0926.tistory.com/87>

<https://soonysoon.tistory.com/79>

참고 사이트

- mern : mongo Db + express.js + react.js + node.js



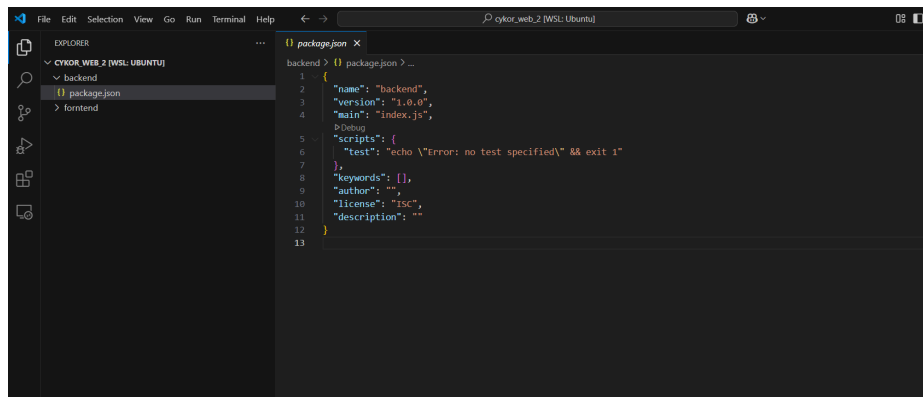
이때 대략적인 순서는

1. 클라이언트 - 서버 소켓 연결
2. 클라이언트에서 서버의 알림을 출력해 연결되었음을 보여줌
3. 클라이언트에서 메시지를 입력 → 서버로 전송
4. 서버에서 메시지 수신 → DB 저장 + 전체 클라이언트에 전송
5. 클라이언트가 메시지 받아서 렌더링

## backend 설정

node.js 프로젝트 초기화

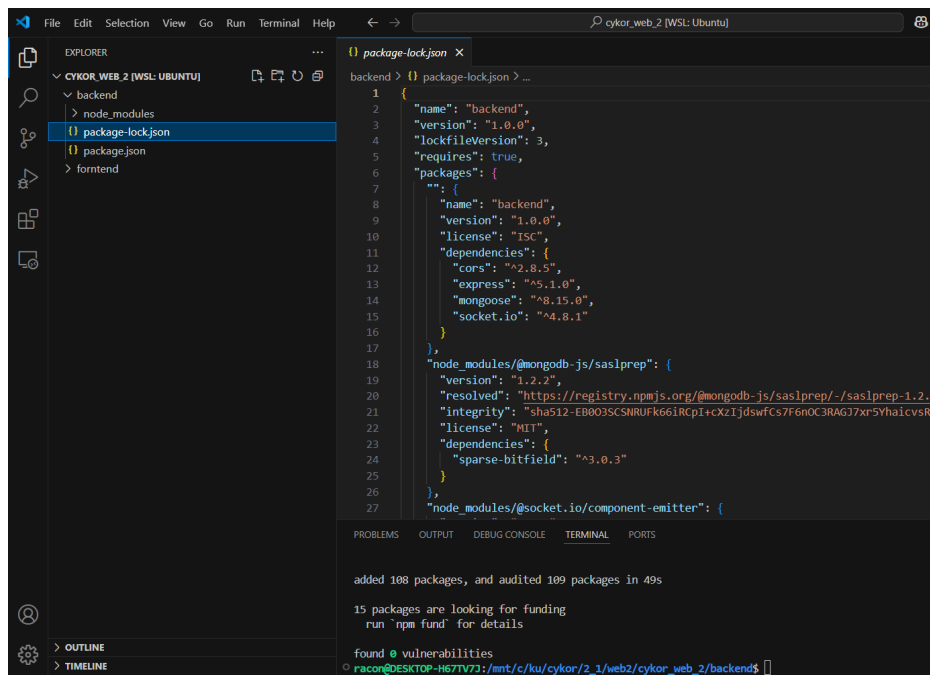
```
npm init -y
```



npm install express socket.io cors mongoose

express socket.io cors mongoose 설치

- express : 웹 서버 프레임워크
- socket.io : 실시간 통신용
- cors : 다른 도메인/포트에서 요청이 오더라도 허용하게 해주는 미들웨어
- mongoose : mongodb 용 라이브러리



//backend index.js 코드 작성

//<https://soonysoon.tistory.com/79> 참고함

// <https://jackcokebb.tistory.com/11>

```

const express = require('express'); //express 프레임 워크 설정
const http = require('http'); //node js http
const { Server } = require('socket.io');
const cors = require('cors'); // cors 상수 저장

const app = express(); // 서버에 필요한 기능인 미들웨어를 어플리케이션에 추가
const server= http.createServer(app);
const socketio = require('socket.io'); //https://smaivnn.tistory.com/2
const io = new socketio.Server(server, {
  cors: { origin: "*",methods: ['GET', 'POST'] } //모든 출처 허용, get,post 메서드 허용
});

app.use(cors()); //해당 cors 미들웨어 적용

io.on('connection', (socket) => { //클라이언트가 socket.io로 연결한 경우
  console.log('사용자 연결됨:', socket.id); //연결된 경우 log

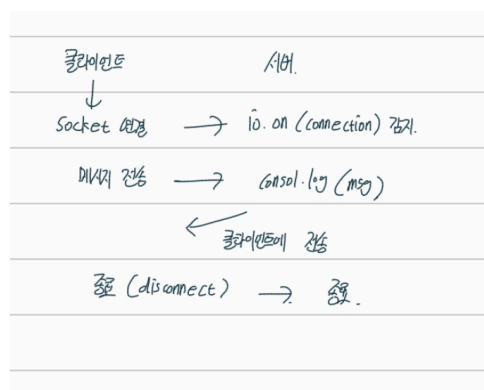
  // 채팅 메시지 수신
  socket.on('chat message', (msg) => { //클라이언트가 메시지를 보내는 경우
    console.log('받은 메시지:', msg); // 서버에 받은 메시지 저장
    io.emit('chat message', msg); // 전체 클라이언트에게 메시지 전송 -> 모든 연결된 사용자
  });

  socket.on('disconnect', () => { //연결 종료
    console.log('사용자 연결 종료:', socket.id);
  });
});

//서버 설정 -> node.js 서버를 3000에서 실행
const PORT = 3001;
server.listen(PORT, () => {
  console.log("서버가 http://localhost:" + PORT + " 에서 실행중");
});

```

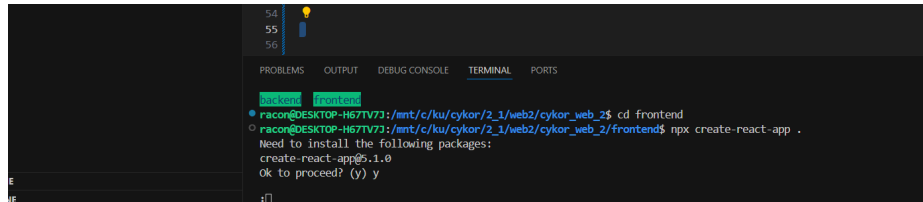
1. 서버는 3001번 포트에서 수행
2. 클라이언트가 소켓으로 연결하면 connection 이벤트 발생
3. 메시지를 보내면 서버가 전체 클라이언트에게 다시 전송
4. 연결 종료 로그 출력



## frontend 설정

### 1. 리엑트 설정

```
npx create-react-app .
```



### 2. socket.io 설치

```
npm install socket.io-client
```

- 참고 : 리엑트 jsx 문법

[https://velog.io/@gyumin\\_2/React-JSX란정의-장점-문법-특징-등#jsx-문법](https://velog.io/@gyumin_2/React-JSX란정의-장점-문법-특징-등#jsx-문법)

```
import React, { useState, useEffect } from 'react'; //리엑트 라이브러리
import { io } from 'socket.io-client'; //socket 서버 연결

const socket = io('http://localhost:3001');

//참고 https://velog.io/@cychann/React-Class-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-vs-Function-%E
function App(){
  const [message, setMessage] = useState(""); //메시지
  const [chat, setChat] = useState([]); //기존 메시지 목록

  // https://xiubindevel.tistory.com/100
  // 메시지 불러오기
  useEffect(() => { //화면이 그려지면 한번 실행
    socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]);});
    //chat message 이벤트 발생시 수행
    //서버에서 전달된 msg를 setchat 호출
    //기존 메시지 배열 복사, 뒤에 msg 연결

    return () => { socket.off('chat message');}; //중복 등록되지 않도록 제거
  }, []);
```

```

// 메시지 전송 처리
const sendMessage = (event) => { //https://programming119.tistory.com/100
  event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
  if(message.trim() === '') return; //빈 메시지 방지
  socket.emit('chat message', message); // 입력받은 메시지 비우기
  setMessage(''); //메시지 비우기
};

return(
  <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>
    <h2> 실시간 채팅 </h2>
    <form onSubmit={sendMessage}>
      <input
        type="text"
        value={message}
        onChange={(event) => setMessage(event.target.value)}
        placeholder="메시지 입력"
        style={{ width: '70%', padding: '8px' }}
      />
      <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
        메시지 전송
      </button>
    </form>

    { /* 채팅 출력 품품 */ }
    <ul style={{ marginTop: '20px' }}>
      {chat.map((msg, idx) => ( <li key={idx} style={{ marginBottom: '5px' }}>{msg}</li>))}
    </ul>

  </div>
);

}

export default App;

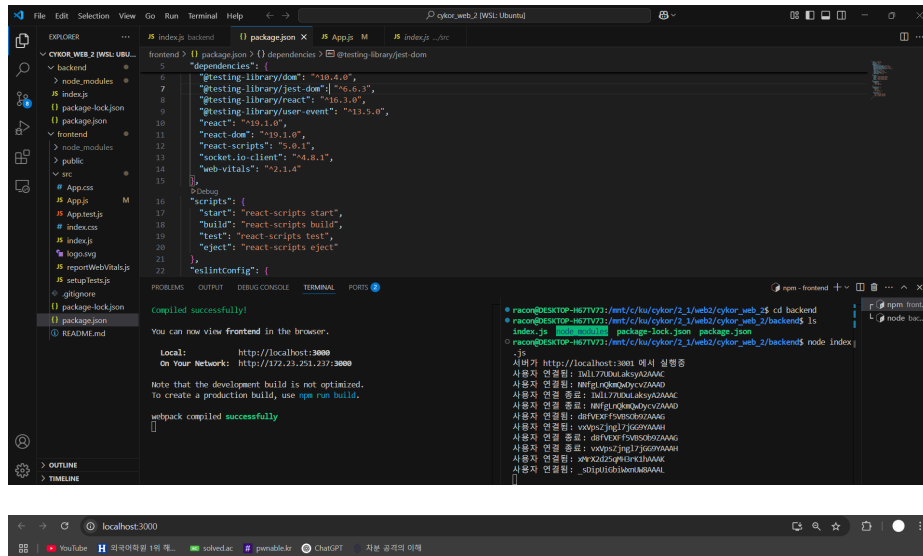
```

---

## 서버 실행

#### 실시간 채팅

backend에서 node index.js 실행  
frontend 에서



정상적으로 서로 다른 페이지에서 채팅 메시지가 보이는 것을 확인할 수 있다

## mongoDB 연결

```

# reportWebVitals.js
# setupTests.js
# ignore
package-lock.json
package.json
README.md

The server generated these startup warnings when booting
2025-05-29T15:19:32.594+09:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://docs.mongodb.org/core/prodnotes-filesystem
2025-05-29T15:19:32.938+09:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2025-05-29T15:19:32.938+09:00: /sys/kernel/mm/transparent_hugopage/enabled is 'always'. We suggest setting it to 'never' in this binary version
2025-05-29T15:19:32.938+09:00: vm.max_map_count is too low

-----
test> exit
* racon@DESKTOP-H67TV73:/mnt/c/ku/cykor/2_1/web2/cykor_web_2/backend$ npm install mongoose
changed 1 package, and audited 189 packages in 22s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
* racon@DESKTOP-H67TV73:/mnt/c/ku/cykor/2_1/web2/cykor_web_2/backend$

```

## mongodb설치

```

run `npm fund` for details

found 0 vulnerabilities
* racon@DESKTOP-H67TV73:/mnt/c/ku/cykor/2_1/web2/cykor_web_2/backend$ mongosh
Current Mongosh Log ID: 6837fee0c732ba471fc59f34
Connecting to:
  mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.1
Using MongoDB:
  6.0.23
Using Mongosh:
  2.5.1

For mongosh info see: https://www.mongodb.com/docs/mongosh-shell/

-----
The server generated these startup warnings when booting
2025-05-29T15:19:32.594+09:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://

```

## 코드 수정

1. mongodb 데이터 저장용 코드

참고 :

1. [https://velog.io/@soshin\\_dev/Node.js-Mongoose-%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%B4%EB%B3%B4%EC%9E%90](https://velog.io/@soshin_dev/Node.js-Mongoose-%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%B4%EB%B3%B4%EC%9E%90)
2. <https://mongoosejs.com/docs/guide.html>

## chatsetting.js

```

const mongoose = require('mongoose');

const chatstruct = new mongoose.Schema({ //
  message: String,
  timestamp: { type: Date, default: Date.now } //현재 시간 저장
});

module.exports = mongoose.model('chat', chatstruct); //mongodb의 chat에 저장

```

위의 코드를 바탕으로 backend,frontend 코드 변환

## backedn - index.js

```

//backend index.js 코드 작성

//https://soonysoon.tistory.com/79 참고함
// https://jackcokebb.tistory.com/11

```

```

const express = require('express'); //express 프레임 워크 설정
const http = require('http'); //node js http
const { Server } = require('socket.io');
const cors = require('cors'); // cors 상수 저장
const mongoose=require('mongoose');
const chat = require('./chatsetting');

//mongodb 연결
// https://koreankoder.tistory.com/15
mongoose.connect('mongodb://localhost:27017/chatdb', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()⇒console.log("connected"))
.catch(err⇒console.error("error : ",err));

const app = express(); // 서버에 필요한 기능인 미들웨어를 어플리케이션에 추가
const server= http.createServer(app);
const socketio = require('socket.io'); //https://smaivnn.tistory.com/2
const { timeStamp } = require('console');
const io = new socketio.Server(server, {
  cors: { origin: "*",methods: ['GET', 'POST'] } //모든 출처 허용, get,post 메서드 허용
});

app.use(cors()); //해당 cors 미들웨어 적용

io.on('connection', async(socket) ⇒ { //클라이언트가 socket.io로 연결한 경우
  console.log('사용자 연결됨:', socket.id); //연결된 경우 log

  //기존 메시지 정보 수신
  const chathistory =await chat.find().sort({ timestamp:1}).limit(50);
  socket.emit('chat history', chathistory);

  // 채팅 메시지 수신
  socket.on('chat message', async(msg) ⇒ { //클라이언트가 메시지를 보내는 경우
    console.log('받은 메시지:', msg); // 서버에 받은 메시지 저장
    const saved= await chat.create({message:msg});
    io.emit('chat message', saved.message); // 전체 클라이언트에게 메시지 전송 -> 모든 연결된 사용자
  });

  socket.on('disconnect', () ⇒ { //연결 종료
    console.log('사용자 연결 종료:', socket.id);
  });
});

//서버 설정 -> node.js 서버를 3001에서 실행
const PORT = 3001;
server.listen(PORT, () ⇒ {
  console.log("서버가 http://localhost:" + PORT + " 에서 실행중");
});

```



```
console.log('받은 메시지:', msg); // 서버에 받은 메시지 저장
const saved= await chat.create({message:msg});
```

부분을 통해서 서버에 데이터를 저장

## frontend-App.js

```
import React, { useState, useEffect } from 'react'; //리액트 라이브러리
import { io } from 'socket.io-client'; //socket 서버 연결

const socket = io('http://localhost:3001');

//참고 https://velog.io/@cychann/React-Class-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-vs-Function-%E
function App(){
  const [message, setMessage] = useState(''); //메시지
  const [chat, setChat] = useState([]); //기존 메시지 목록

  // https://xiubindex.tistory.com/100
  // 메시지 불러오기
  useEffect(() => { //화면이 그려지면 한번 시행

    //과거 메시지 수신
    socket.on('chat history', (messages) => { setChat(messages.map((m) => m.message)); });
    socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]); });
    //chat message 이벤트 발생시 수행
    //서버에서 전달된 msg를 setchat 호출
    //기존 메시지 배열 복사, 뒤에 msg 연결

    return () => { socket.off('chat message');
      socket.off('chat history');
    }; //중복 등록되지 않도록 제거
  }, []);

  // 메시지 전송 처리
  const sendMessage = (event) => { //https://programming119.tistory.com/100
    event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
    if(message.trim() === '') return; //빈 메시지 방지
    socket.emit('chat message', message); // 입력받은 메시지 비우기
    setMessage(''); //메시지 비우기
  };

  return(
    <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>
      <h2> 실시간 채팅 </h2>
      <form onSubmit={sendMessage}>
        <input
          type="text"
          value={message}
          onChange={(event) => setMessage(event.target.value)}
          placeholder="메시지 입력"
          style={{ width: '70%', padding: '8px'}}
        />
    </div>
  );
}
```

```

    <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
      메시지 전송
    </button>
  </form>

  {/* 채팅 출력 폼폼 */}
  <ul style={{ marginTop: '20px' }}>
    {chat.map((msg, idx) => ( <li key={idx} style={{ marginBottom: '5px' }}>{msg}</li>))}
  </ul>

</div>
);

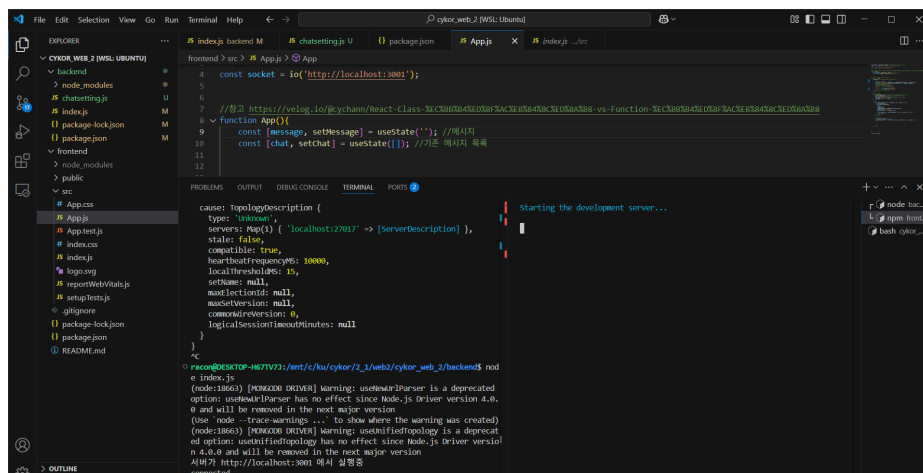
}

export default App;

```

과거 메시지도 출력되도록 수정

## 테스트



#### 실시간 채팅

메시지 입력

메시지 전송

#### 실시간 채팅

메시지 입력

메시지 전송

• test

정상적으로 다른 페이지에서도 해당 내용이 보이고, 새로고침해도 유지되는것을 확인할 수 있다.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2
option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(node:18663) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
서버가 http://localhost:3001 에서 실행중
connected
사용자 연결됨: nvves071XVZqhrveAAAC
사용자 연결됨: asbAQkp9dSkuZwcAAAD
받은 메시지: test
사용자 연결 종료: asbAQkp9dSkuZwcAAAD
사용자 연결됨: siMljqu01IRuIKZAAAF
^C
racon@DESKTOP-H67TV73 /mnt/c/ku/cykor/2_1/web2/cykor_web_2/backend$ node index.js
(node:26104) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(node:26104) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
서버가 http://localhost:3001 에서 실행중
connected
사용자 연결됨: GXg-60h3-D7Cif_yAAAB
||
Compiled successfully!
You can now view frontend in the browser.
Local: http://localhost:3000
On Your Network: http://172.23.251.23/3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
```

#### 실시간 채팅

메시지 입력

메시지 전송

• test

추가적으로 mongodb가 꺼지지 않으면 해당 backend와 frontend를 다시 실행해도 해당 메시지들이 저장되어있는것을 볼 수 있었다

## 로그인 기능 구현

usersetting.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: { type: String, unique: true },
  password: String
});

module.exports = mongoose.model('user', userSchema);
```

이후 chatsetting.js 또한 닉네임을 추가한다

```
const mongoose = require('mongoose');

const chatstruct = new mongoose.Schema({ //
  nickname: String,
  message: String,
  timestamp: { type: Date, default: Date.now } //현재 시간 저장
});

module.exports = mongoose.model('chat', chatstruct); //mongodb의 chat에 저장
```

참고 : <https://velog.io/@jihukimme/React-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EB%A1%9C%EA%B7%B8%EC%9D%B8-%ED%9A%8C%EC%9B%90%EA%B0%80%EC%9E%85-%EB%A1%9C%EA%B7%B8%EC%95%84%EC%9B%83-%EA%B5%AC%ED%98%84%ED%95%98%EA%B8%B0>

로그인 세팅

```
const express = require('express'); //express 웹 프레임워크
const router = express.Router(); //라우터 -> register,login 생성
const User = require('./usersetting');

//https://velog.io/@jihukimme/React-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EB%A1%9C%EA%B7%B8%EC%9D%B8-%ED%9A%8C%EC%9B%90%EA%B0%80%EC%9E%85-%EB%A1%9C%EA%B7%B8%EC%95%84%EC%9B%83-%EA%B5%AC%ED%98%84%ED%95%98%EA%B8%B0
router.post('/register', async (req, res) => { //회원가입
  const { username, password } = req.body; //https://www.google.com/search?q=req.body&rlz=1C1GCEU_koKR1161KR116
  try {
    const exist = await User.findOne({ username }); //이미 같은 id가 있는지 확인
    if (exist) return res.status(400).json({ message: '이미 저장된 id' }); //저장된 값이 있는 경우

    const user = new User({ username, password }); //user 정보 획득 및 저장
    await user.save();
    res.status(201).json({ message: '회원가입 성공' });
  } catch (err) {
    res.status(500).json({ message: '서버 오류', error: err });
  }
});

// 로그인
```

```

router.post('/login', async (req, res) => { //request,response
  const { username, password } = req.body; //request에서 id,password 추출
  const user = await User.findOne({ username, password }); //해당 요소 찾기
  if (!user) return res.status(401).json({ message: '로그인 실패' }); //user 데이터가 없는 경우

  res.status(200).json({ message: '로그인 성공', user: { username } }); //로그인 response 전송
});

module.exports = router;

```

backend : index.js 도 변경

```

//backend index.js 코드 작성

//https://soonysoon.tistory.com/79 참고함
// https://jackcokebb.tistory.com/11

const express = require('express'); //express 프레임 워크 설정
const http = require('http'); //node js http
const { Server } = require('socket.io');
const cors = require('cors'); // cors 상수 저장
const mongoose=require('mongoose');
const chat = require('./chatsetting');

//mongodb 연결
// https://koreankoder.tistory.com/15
mongoose.connect('mongodb://localhost:27017/chatdb', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()=>console.log("connected"))
.catch(err=>console.error("error : ",err));

const app = express(); // 서버에 필요한 기능인 미들웨어를 어플리케이션에 추가
const server= http.createServer(app);
const socketio = require('socket.io'); //https://smaivnn.tistory.com/2
const { timeStamp } = require('console');
const authRoutes = require('./auth');
const io = new socketio.Server(server, {
  cors: { origin: "*",methods: ['GET', 'POST'] } //모든 출처 허용, get,post 메서드 허용
});

app.use(cors()); //해당 cors 미들웨어 적용
app.use(express.json()); //req.body에서 json형태의 데이터 읽기
app.use('/api/auth', authRoutes); //https://velog.io/@rhftnqls/auth-%EB%AF%B8%EB%93%A4%EC%9B%A8%EC%96%
//auto routes 만들기

io.on('connection', async(socket) => { //클라이언트가 socket.io로 연결한 경우
  console.log('사용자 연결됨:', socket.id); //연결된 경우 log

  //기존 메시지 정보 수신
  const chathistory =await chat.find().sort({ timestamp:1}).limit(50);
  socket.emit('chat history', chathistory);

```

```

// 채팅 메시지 수신
socket.on('chat message', async({ nickname, message }) => { //클라이언트가 메시지를 보내는 경우 + nickname도 추가
  console.log('받은 메시지:', { nickname, message }); // 서버에 받은 메시지 저장
  const saved= await chat.create({ nickname, message });
  io.emit('chat message', { nickname: saved.nickname, message: saved.message }); // 전체 클라이언트에게 메시지 전송 -
});

socket.on('disconnect', () => { //연결 종료
  console.log('사용자 연결 종료:', socket.id);
});
});

//서버 설정 -> node.js 서버를 3001에서 실행
const PORT = 3001;
server.listen(PORT, () => {
  console.log("서버가 http://localhost:" + PORT + " 에서 실행중");
});

```

frontend에도 로그인 및 회원가입 기능 추가

```

import React, { useState, useEffect } from 'react'; //리액트 라이브러리
import { io } from 'socket.io-client'; //socket 서버 연결

const socket = io('http://localhost:3001');

//참고 https://velog.io/@cychann/React-Class-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-vs-Function-%E
function App(){
  const [message, setMessage] = useState(""); //메시지
  const [chat, setChat] = useState([]); //기존 메시지 목록

  //로그인
  const [user, setUser] = useState(null);
  const [loginform, setLoginForm] = useState({ username: '', password: '' });
  const [isRegister, setIsRegister] = useState(false);

  // https://xiubindex.tistory.com/100
  // 메시지 불러오기
  useEffect(() => { //화면이 그려지면 한번 실행

    //과거 메시지 수신
    socket.on('chat history', (messages) => { setChat(messages);});
    socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]);});
    //chat message 이벤트 발생시 수행
    //서버에서 전달된 msg를 setchat 호출
    //기존 메시지 배열 복사, 뒤에 msg 연결

    return () => { socket.off('chat message');

```

```

        socket.off('chat history');
    }; //중복 등록되지 않도록 제거
},[]);

//로그인 처리
const handleAuth = async (e) => { //로그인/회원가입 처리 함수
    e.preventDefault();
    const url = `http://localhost:3001/api/auth/${isRegister ? 'register' : 'login'}`; //isRegister가 true인 경우 register 페이지로 0
    const res = await fetch(url, { //요청 전송
        method: 'POST',
        headers: { 'Content-Type': 'application/json' }, //json 형태
        body: JSON.stringify(loginform) //loginform과 같은 형태를 json으로 변환하여 전송
    });

    const data = await res.json(); //응답 처리
    if (res.ok) { //응답 성공한 경우
        setUser({ username: loginform.username }); // 로그인 성공 시 유저 설정
    } else {
        alert(data.message);
    }
};

// 메시지 전송 처리
const sendMessage = (event) => { //https://programming119.tistory.com/100
    event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
    if(message.trim() === '') return; //빈 메시지 방지
    socket.emit('chat message', { nickname: user.username, message }); // 입력받은 메시지 비우기
    setMessage(""); //메시지 비우기
};

//페이지
if (!user) { //로그인 되어있는 경우
    return (
        <div style={{ padding: '30px', textAlign: 'center' }}>
            <h2>{isRegister ? '회원가입' : '로그인'}</h2>
            <form onSubmit={handleAuth}>
                <input
                    type="text"
                    placeholder="아이디"
                    value={loginform.username}
                    onChange={(e) => setLoginForm({ ...loginform, username: e.target.value })} //loginform username과 password에
                /><br /><br />
                <input
                    type="password"
                    placeholder="비밀번호"
                    value={loginform.password}
                    onChange={(e) => setLoginForm({ ...loginform, password: e.target.value })}
                /><br /><br />
                <button type="submit">{isRegister ? '회원가입' : '로그인'}</button> {/*버튼을 통해 폼 전환이 되도록 한다*/}
            <p
                style={{ marginTop: '10px', cursor: 'pointer', color: 'blue' }}
                onClick={() => setIsRegister(!isRegister)}
            </p>

```

```

    >
    {isRegister ? '로그인 전환' : '회원가입 전환'}
  </p>
</form>
</div>
);
}

return(
  <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>
    <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
      <h2>실시간 채팅</h2>
      <button onClick={() => setUser(null)} style={{ padding: '6px 10px', fontSize: '12px' }}>
        로그아웃
      </button>

    </div>
    <form onSubmit={sendMessage}>
      <input
        type="text"
        value={message}
        onChange={(event) => setMessage(event.target.value)}
        placeholder="메시지 입력"
        style={{ width: '70%', padding: '8px' }}
      />
      <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
        메시지 전송
      </button>
    </form>
    {/* 채팅 출력 폼, user추가 */}
    <ul style={{ marginTop: '20px' }}>
      {chat.map((msg, idx) => ( <li key={idx} style={{ marginBottom: '5px' }}> <strong>{msg.nickname}</strong>: {msg.m
    </ul>
  </div>
);
}

export default App;

```

## 테스트

### 1. mongodb 실행

```
sudo systemctl start mongod
```



#### 회원가입

  
  
  
[로그인 전환](#)

#### 로그인

  
  
  
[회원가입 전환](#)

test 계정으로 계정 생성

#### 회원가입

  
  
  
[로그인 전환](#)

test2계정도 추가하고 테스트한다

#### 실시간 채팅

- : test
- **test**: hello world
- **test2**: hello world, test

서로 test, test2 이름과 메시지가 보인다

## 단톡방 기능 추가

chatsetting.js에 roomid를 추가해 해당 room에서만 보일 수 있도록 한다

```
const mongoose = require('mongoose');

const chatstruct = new mongoose.Schema({ //
  nickname: String,
  message: String,
  roomid:String,
  timestamp: { type: Date, default: Date.now } //현재 시간 저장
});

module.exports = mongoose.model('chat', chatstruct); //mongodb의 chat에 저장
```

backend

roomid를 설정해 해당 값에 맞춰서만 전송한다

```
//backend index.js 코드 작성

//https://soonysoon.tistory.com/79 참고함
// https://jackcokebb.tistory.com/11

const express = require('express'); //express 프레임 워크 설정
const http = require('http'); //node js http
const { Server } = require('socket.io');
const cors = require('cors'); // cors 상수 저장
const mongoose=require('mongoose');
const chat = require('./chatsetting');

//mongodb 연결
// https://koreankoder.tistory.com/15
mongoose.connect('mongodb://localhost:27017/chatdb', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log("connected"))
.catch(err=>console.error("error : ",err));

const app = express(); // 서버에 필요한 기능인 미들웨어를 어플리케이션에 추가
const server= http.createServer(app);
const socketio = require('socket.io'); //https://smaivnn.tistory.com/2
const { timeStamp } = require('console');
const authRoutes = require('./auth');
const io = new socketio.Server(server, {
  cors: { origin: "*",methods: ['GET', 'POST'] } //모든 출처 허용, get,post 메서드 허용
});

app.use(cors()); //해당 cors 미들웨어 적용
app.use(express.json());//req.body에서 json형태의 데이터 읽기
app.use('/api/auth', authRoutes); //https://velog.io/@rhftnqls/auth-%EB%AF%B8%EB%93%A4%EC%9B%A8%EC%96%
//auto routes 만들기
```



```

// https://xiubindev.tistory.com/100
// 메시지 불러오기
//방 입장시 서버에 표시
useEffect(() => {
  socket.emit('join room', roomid);
}, [roomid]);

useEffect(() => { //화면이 그려지면 한번 시행

  //과거 메시지 수신
  socket.on('chat history', (messages) => { setChat(messages);});
  socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]);});
  //chat message 이벤트 발생시 수행
  //서버에서 전달된 msg를 setchat 호출
  //기존 메시지 배열 복사, 뒤에 msg 연결

  return () => { socket.off('chat message');
    socket.off('chat history');
  }; //중복 등록되지 않도록 제거
}, []);
useEffect(() => {chatendRef.current?.scrollIntoView({ behavior: 'smooth' });}, [chat]);

//로그인 처리
const handleAuth = async (e) => { //로그인/회원가입 처리 함수
  e.preventDefault();
  const url = `http://localhost:3001/api/auth/${isRegister ? 'register' : 'login'}`; //isRegister가 true인 경우 register 페이지로 0
  const res = await fetch(url, { //요청 전송
    method: 'POST',
    headers: { 'Content-Type': 'application/json' }, //json 형태
    body: JSON.stringify(loginform) //loginform과 같은 형태를 json으로 변환하여 전송
  });

  const data = await res.json(); //응답 처리
  if (res.ok) { //응답 성공한 경우
    setUser({ username: loginform.username }); // 로그인 성공 시 유저 설정
  } else {
    alert(data.message);
  }
};

// 메시지 전송 처리
const sendMessage = (event) => { //https://programming119.tistory.com/100
  event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
  if(message.trim() === '') return; //빈 메시지 방지
  socket.emit('chat message', { nickname: user.username, message, roomid }); // 입력받은 메시지 비우기, roomid 기능 추가
  setMessage(''); //메시지 비우기
};

//페이지

```

```

if (!user) { //로그인 되어있는 경우
  return (
    <div style={{ padding: '30px', textAlign: 'center' }}>
      <h2>{isRegister ? '회원가입' : '로그인'}</h2>
      <form onSubmit={handleAuth}>
        <input
          type="text"
          placeholder="아이디"
          value={loginform.username}
          onChange={(e) => setLoginForm({ ...loginform, username: e.target.value })} //loginform username과 password에
        /><br /><br />
        <input
          type="password"
          placeholder="비밀번호"
          value={loginform.password}
          onChange={(e) => setLoginForm({ ...loginform, password: e.target.value })}
        /><br /><br />
        <button type="submit">{isRegister ? '회원가입' : '로그인'}</button> {/*버튼을 통해 폼 전환이 되도록 한다*/}
      <p
        style={{ marginTop: '10px', cursor: 'pointer', color: 'blue' }}
        onClick={() => setIsRegister(!isRegister)}
      >
        {isRegister ? '로그인 전환' : '회원가입 전환'}
      </p>
    </form>
  </div>
  );
}

return(
  <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>
    <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
      <h2>실시간 채팅- {roomId}</h2> {/*방 이름 추가 */}
      <button onClick={() => setUser(null)} style={{ padding: '6px 10px', fontSize: '12px' }}>
        로그아웃
      </button>
    </div>
    <div style={{ marginBottom: '10px' }}>
      <button onClick={() => setRoomid('room1')} style={{ marginRight: '5px' }}>방 1</button>
      <button onClick={() => setRoomid('room2')} style={{ marginRight: '5px' }}>방 2</button>
      <button onClick={() => setRoomid('room3')}>방 3</button>
    </div>

    <form onSubmit={sendMessage}>
      <input
        type="text"
        value={message}
        onChange={(event) => setMessage(event.target.value)}
        placeholder="메시지 입력"
        style={{ width: '70%', padding: '8px' }}
      />
      <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
        메시지 전송
      </button>
    </form>
    {/* 채팅 출력 폼, user추가 + 자동 스크롤, 메시지 필터링 */}
  <ul style={{ marginTop: '20px' }}>

```

```

    {chat
      .filter((msg) => msg.roomid === roomid)
      .map((msg, idx) => (
        <li key={idx} style={{ marginBottom: '5px', textAlign: msg.nickname === user.username ? 'right' : 'left' }}>
          <strong>{msg.nickname}</strong>: {msg.message}
          <div style={{ fontSize: '10px', color: '#888' }}>{new Date(msg.timestamp).toLocaleTimeString()}</div>
        </li>
      ))
    <div ref={chatEndRef} />
  </ul>
</div>
);
}

export default App;

```

## 테스트

test 계정으로 로그인해서 room1에 메시지 입력

실시간 채팅- room1

로그아웃

방 1 | 방 2 | 방 3

메시지 입력

메시지 전송

---

실시간 채팅- room1

로그아웃

방 1 | 방 2 | 방 3

메시지 입력

메시지 전송

- test: chat test-1  
2021. 12. 15 14:15

실시간 채팅- room2

로그아웃

방 1 | 방 2 | 방 3

메시지 입력

메시지 전송

정상적으로 room1에서만 보이는 것을 확인할 수 있다



test2 계정으로 접속한 경우에도 정상적으로 보이는 것을 확인할 수 있다

## 친구추가 기능

친구 추가, 목록 확인, 친구 제거를 위한 api 추가

```
const express = require('express'); //express 웹 프레임워크
const router = express.Router(); //라우터 -> register,login 생성
const User = require('./usersetting');
```

```
//https://velog.io/@jihukimme/React-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EB%A1%9C%EA%B7%
router.post('/register', async (req, res) => { //회원가입
  const { username, password } = req.body; //https://www.google.com/search?q=req.body&rlz=1C1GCEU_koKR1161KR116'
  try {
    const exist = await User.findOne({ username }); //이미 같은 id가 있는지 확인
    if (exist) return res.status(400).json({ message: '이미 저장된 id' }); //저장된 값이 있는 경우

    const user = new User({ username, password }); //user 정보 획득 및 저장
    await user.save();
    res.status(201).json({ message: '회원가입 성공' });
  } catch (err) {
    res.status(500).json({ message: '서버 오류', error: err });
  }
});
```

```
// 로그인
router.post('/login', async (req, res) => { //request,response
  const { username, password } = req.body; //request에서 id,password 추출
  const user = await User.findOne({ username, password }); //해당 요소 찾기
  if (!user) return res.status(401).json({ message: '로그인 실패' }); //user 데이터가 없는 경우

  res.status(200).json({ message: '로그인 성공', user: { username } }); //로그인 response 전송
});
```

```
router.post('/addfriend', async (req, res) => {
  const { username, friendusername } = req.body;
  try{
    const user = await User.findOne({ username }); //사용자
    const friend = await User.findOne({ username: friendusername }); //추가하려는 이름
    if (!user || !friend) { // 둘중 하나가 없는 유저인 경우
      return res.status(404).json({ message: '없는 유저' });
    }
  }
});
```

```

    }

    const alreadyfriend = user.friends.includes(friend._id); //이미 친구추가되어 있는지 체크
    if (alreadyfriend) {
        return res.status(400).json({ message: '이미 친구추가됨' });
    }

    //친구 추가
    user.friends.push(friend._id);
    friend.friends.push(user._id);
    await user.save();
    await friend.save();
    res.status(200).json({ message: '친구 추가 완료', friend: friend.username }); //친구 추가 완료 체크

} catch (err) { //error 확인
    console.error(err);
    res.status(500).json({ message: '서버 오류' });
}
});

//친구 목록 확인용 api
router.post('/list', async (req, res) => {
    const { username } = req.body;

    try {
        const user = await User.findOne({ username }); //user찾기
        if (!user) return res.status(404).json({ message: '사용자 없음' });

        const friends = await User.find({ _id: { $in: user.friends }, 'username' }); //friend 확인
        res.json({ friends: friends.map(f => f.username) });
    } catch (err) {
        res.status(500).json({ message: '서버 에러' });
    }
});

//친구 제거 api
router.post('/removefriend', async (req, res) => {
    const { username, friendusername } = req.body;
    try {
        const user = await User.findOne({ username }); //user찾기 -> 기존과 동일
        const friend = await User.findOne({ username: friendusername });

        if (!user || !friend) { //둘중에 데이터가 없는 경우
            return res.status(404).json({ message: '없는 유저' });
        }

        //배열에서 제거
        user.friends = user.friends.filter(fld => !fld.equals(friend._id)); //친구 id를 제거해서 새로 확인
        friend.friends = friend.friends.filter(fld => !fld.equals(user._id));
        await user.save();
        await friend.save();

        res.status(200).json({ message: '친구 제거 완료', friend: friend.username });
    }
});

```



```

    } catch (err) {
      console.error(err);
      res.status(500).json({ message: '서버 오류' });
    }
  });
};

```

```

module.exports = router;

```

- frontend에서도 해당 친구 추가 기능 추가

```

import React, { useState, useEffect, useRef } from 'react'; //리액트 라이브러리
//https://hihiha2.tistory.com/19
import { io } from 'socket.io-client'; //socket 서버 연결

const socket = io('http://localhost:3001');

//참고 https://velog.io/@cychann/React-Class-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-vs-Function-%E
function App(){
  const [message, setMessage] = useState(""); //메시지
  const [chat, setChat] = useState([]); //기존 메시지 목록

  //로그인
  const [user, setUser] = useState(null);
  const [loginform, setLoginForm] = useState({ username: '', password: '' });
  const [isRegister, setIsRegister] = useState(false);
  const [roomid, setRoomid] = useState('room1');
  const chatendRef = useRef(null);
  const [friendname, setFriendName] = useState(""); //친구추가 입력값 상태
  const [friends, setFriends] = useState([]); //친구 목록

  //친구 목록 가져오기
  const fetchFriends = async (u = user) => {
    if (!u || !u.username) return;
    const res = await fetch('http://localhost:3001/api/auth/list', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ username: u.username })
    });
    const data = await res.json();
    if (res.ok) {
      setFriends(data.friends);
    } else {
      alert(`${data.message}`);
    }
  };

  //친구 제거 함수
  const handleRemoveFriend = async (friendusername) => {
    const res = await fetch('http://localhost:3001/api/auth/removefriend', {

```

```

method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify({ username: user.username, friendusername }) //json형태로 사용
});

const data = await res.json();
if (res.ok) {
  alert(`${data.friend} 친구 제거 완료`);
  fetchFriends(); // 목록 갱신
} else {
  alert(data.message);
}
};

// https://xiubindex.tistory.com/100
// 메시지 불러오기
//방 입장시 서버에 표시
useEffect(() => {
  socket.emit('join room', roomid);
}, [roomid]);

useEffect(() => { //화면이 그려지면 한번 시행

  //과거 메시지 수신
  socket.on('chat history', (messages) => { setChat(messages);});
  socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]);});
  //chat message 이벤트 발생시 수행
  //서버에서 전달된 msg를 setchat 호출
  //기존 메시지 배열 복사, 뒤에 msg 연결

  return () => { socket.off('chat message');
    socket.off('chat history');
  }; //중복 등록되지 않도록 제거
}, []);
useEffect(() => {chatendRef.current?.scrollIntoView({ behavior: 'smooth' });}, [chat]);

//로그인 처리
const handleAuth = async (e) => { //로그인/회원가입 처리 함수
  e.preventDefault();
  const url = `http://localhost:3001/api/auth/${isRegister ? 'register' : 'login'}`; //isRegister가 true인 경우 register 페이지로 0
  const res = await fetch(url, { //요청 전송
    method: 'POST',
    headers: { 'Content-Type': 'application/json' }, //json 형태
    body: JSON.stringify(loginform) //loginform과 같은 형태를 json으로 변환하여 전송
  });

  const data = await res.json(); //응답 처리
  if (res.ok) { //응답 성공한 경우
    const newUser = { username: loginform.username };
    setUser(newUser); // user 상태 설정
    fetchFriends(newUser); // user 값 넘겨주기
  } else {
    alert(data.message);
  }
};

```

```

// 메시지 전송 처리
const sendMessage = (event) => { //https://programming119.tistory.com/100
  event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
  if(message.trim() === '') return; //빈 메시지 방지
  socket.emit('chat message', { nickname: user.username, message, roomid }); // 입력받은 메시지 비우기, roomid 기능 추가
  setMessage(''); //메시지 비우기
};

const handleaddfriend = async () => {
  if (!friendname.trim()) return; //friendname이 공백인 경우 그냥 return
  const res = await fetch('http://localhost:3001/api/auth/addfriend', { // 친구 추가 api
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ //json형태로 전송
      username: user.username,
      friendusername: friendname
    })
  });

  const data = await res.json(); //서버로 받은 값을 data에 저장
  if (res.ok) {
    alert(`${data.friend} 친구추가`);
    setFriendName('');
    fetchFriends(); //친구추가 이후 목록 갱신
  } else {
    alert(`${data.message}`);
  }
};

//페이지
if (!user) { //로그인 되어있는 경우
  return (
    <div style={{ padding: '30px', textAlign: 'center' }}>
      <h2>{isRegister ? '회원가입' : '로그인'}</h2>
      <form onSubmit={handleAuth}>
        <input
          type="text"
          placeholder="아이디"
          value={loginform.username}
          onChange={e => setLoginForm({ ...loginform, username: e.target.value })} //loginform username과 password에
        /><br /><br />
        <input
          type="password"
          placeholder="비밀번호"
          value={loginform.password}
          onChange={e => setLoginForm({ ...loginform, password: e.target.value })}
        /><br /><br />
        <button type="submit">{isRegister ? '회원가입' : '로그인'}</button> { /*버튼을 통해 폼 전환이 되도록 한다*/ }
      <p
        style={{ marginTop: '10px', cursor: 'pointer', color: 'blue' }}
        onClick={() => setIsRegister(!isRegister)}
      >
        {isRegister ? '로그인 전환' : '회원가입 전환'}
      </p>
    </div>
  );
}

```

```

    </p>
  </form>
</div>
);
}

return(
  <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>
    <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
      <h2>실시간 채팅- {roomId}</h2> { /*방 이름 추가 */}
      <button onClick={() => setUser(null)} style={{ padding: '6px 10px', fontSize: '12px' }}>
        로그아웃
      </button>
    </div>
    <div style={{ marginBottom: '10px' }}>
      <button onClick={() => setRoomid('room1')} style={{ marginRight: '5px' }}>방 1</button>
      <button onClick={() => setRoomid('room2')} style={{ marginRight: '5px' }}>방 2</button>
      <button onClick={() => setRoomid('room3')}>방 3</button>
    </div>

    { /*친구추가 세팅*/}
    <div style={{ marginBottom: '15px' }}>
      <input type="text" placeholder="친구 아이디 입력" value={friendname} onChange={(e) => setFriendName(e.target.value)} />
      <button onClick={handleaddfriend} style={{ marginLeft: '8px', padding: '6px 10px' }}>
        친구추가
      </button>
    </div>

    { /*친구 목록 */}
    <div style={{ marginBottom: '20px' }}>
      <h4>친구 목록</h4>
      <ul style={{ paddingLeft: '20px' }}>
        {friends.map((f, i) => (
          <li key={i}>
            {f}
            <button onClick={() => handleRemoveFriend(f)} style={{ marginLeft: '10px', padding: '2px 6px', fontSize: '12px' }}>
              제거
            </button>
          </li>
        ))}
      </ul>
    </div>

    <form onSubmit={sendMessage}>
      <input
        type="text"
        value={message}
        onChange={(event) => setMessage(event.target.value)}
        placeholder="메시지 입력"
        style={{ width: '70%', padding: '8px' }}
      />
      <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
        메시지 전송
      </button>
    </form>
    { /* 채팅 출력 폼, user추가 + 자동 스크롤, 메시지 필터링 */}
  </div>
);
}

```

```

<ul style={{ marginTop: '20px' }}>
  {chat
    .filter((msg) => msg.roomid === roomid)
    .map((msg, idx) => (
      <li key={idx} style={{ marginBottom: '5px', textAlign: msg.nickname === user.username ? 'right' : 'left' }}>
        <strong>{msg.nickname}</strong>: {msg.message}
        <div style={{ fontSize: '10px', color: '#888' }}>{new Date(msg.timestamp).toLocaleTimeString()}</div>
      </li>
    ))}
  <div ref={chatendRef} />
</ul>
</div>
);
}

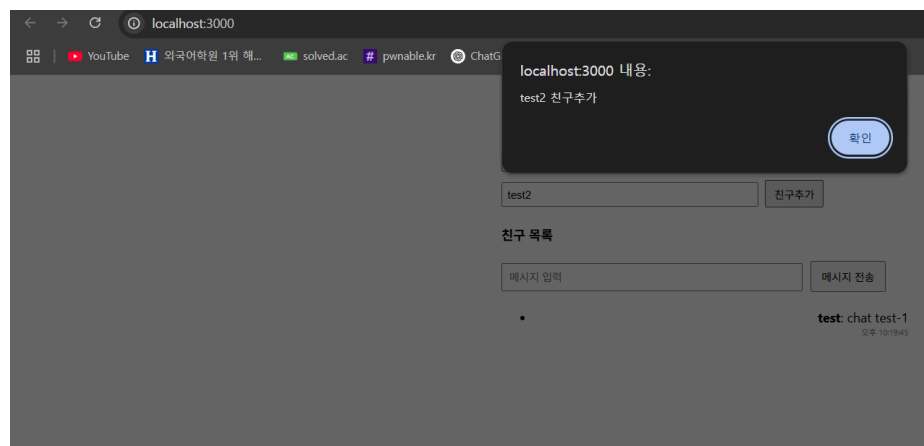
export default App;

```

## test



정상적으로 친구 목록과 친구 추가가 나타나는 것을 확인할 수 있다



실시간 채팅- room1 로그아웃

방 1 | 방 2 | 방 3

친구 아이디 입력 친구추가

친구 목록

- test2 제거

메시지 입력 메시지 전송

• test: chat test-1  
오후 10:19:45

추가적으로 친구추가 기능도 정상적으로 작동하는것을 확인할 수 있다

## 단톡방 구조 변경

단톡방을 고정된 형태가 아니라 방을 만들고 삭제할 수 있도록 변경한다

roomsetting.js

```
const mongoose = require('mongoose');

const roomstruct = new mongoose.Schema({
  name: { type: String, required: true, unique: true },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Room', roomstruct);
```

또한 다음과 같이 auth.js에 방과 관련된 부분을 추가한다

```
router.post('/createroom', async (req, res) => {
  const { name } = req.body;
  if (!name) return res.status(400).json({ message: 'room 이름 입력 필요' }); //방 이름 받아오기
  try {
    const exist = await Room.findOne({ name });
    if (exist) return res.status(400).json({ message: '이미 존재하는 방 이름입니다' });

    const room = new Room({ name }); //새 방
    await room.save();
    res.status(201).json({ message: '방 생성 완료', room }); //신규 room 생성 확인인
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: '서버 오류' });
  }
});

router.get('/rooms', async (req, res) => {
  try {
    const rooms = await Room.find().sort({ createdAt: -1 }); //방 목록 받아오기기
    res.json({ rooms });
  } catch (err) {
    res.status(500).json({ message: '방 목록 조회 오류' });
  }
});
```

```

router.post('/deleteroom', async (req, res) => {
  const { name } = req.body;
  try {
    const result = await Room.deleteOne({ name }); //이름 찾아서 지우기
    if (result.deletedCount === 0) {
      return res.status(404).json({ message: '방 확인 error' });
    }
    res.status(200).json({ message: '방 삭제 완료' });
  } catch (err) {
    res.status(500).json({ message: '서버 오류', error: err });
  }
});

```

이후 dm과 room 설정을 app.js에 추가하였다.

```

const fetchrooms = async () => {
  const res = await fetch('http://localhost:3001/api/auth/rooms');
  const data = await res.json();
  if (res.ok) setRoomList(data.rooms.map(r => r.name));
};

//방 생성
const handlecreateRoom = async () => {
  if (!newroomname.trim()) return;

  const res = await fetch('http://localhost:3001/api/auth/createroom', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: newroomname })
  });

  const data = await res.json();
  if (res.ok) {
    alert('방 생성 완료');
    setnewroomname('');
    fetchrooms();
  } else {
    alert(data.message);
  }
};

//방 제거
const handleDeleteRoom = async (roomname) => {
  if (!window.confirm(`${roomname} ㄹㅇ 삭제할까?`)) return; //확인창

  const res = await fetch('http://localhost:3001/api/auth/deleteroom', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: roomname })
  }); //json 형태로 확인

  const data = await res.json();
  if (res.ok) {

```

```

    alert('삭제 완료');
    fetchrooms();
    if (roomid === roomname) setRoomid('room1'); //삭제된 방의 경우 기본방 room1으로 이동
  } else {
    alert(data.message);
  }
};

```

```
//dm
const handleEnterDM = (friendname) => {
  // 친구 이름과 내 이름을 정렬해서 방 이름을 항상 고정된 순서로 설정
  const sorted = [user.username, friendname].sort();
  const dmRoom = `DM_${sorted[0]}_${sorted[1]}`;
  setRoomid(dmRoom); // 해당 DM 방으로 입장 -> dm으로 입장
};
```

```
import React, { useState, useEffect, useRef } from 'react'; //리엑트 라이브러리
//https://h1hiha2.tistory.com/19
import { io } from 'socket.io-client'; //socket 서버 연결
```

```
const socket = io('http://localhost:3001');
```

```
//참고 https://velog.io/@cychann/React-Class-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-vs-Function-%E
function App(){
```

```
const [message, setMessage] = useState(""); //메시지
const [chat, setChat] = useState([]); //기존 메시지 목록
```

```
//로그인
```

```
const [user, setUser] = useState(null);
const [loginform, setLoginForm] = useState({ username: '', password: '' });
const [isRegister, setIsRegister] = useState(false);
const [roomid, setRoomid] = useState('room1');
const chatendRef = useRef(null);
const [friendname, setFriendName] = useState(''); //친구추가 입력값 상태
const [friends, setFriends] = useState([]); //친구 목록
```

//방 관련 설정

```
const [roomlist, setRoomList] = useState([]);
const [newroomname, setnewroomname] = useState('');
```

## //방 목록

```
const fetchrooms = async () => {
  const res = await fetch('http://localhost:3001/api/auth/rooms');
  const data = await res.json();
  if (res.ok) setRoomList(data.rooms.map(r => r.name));
};
```

//방 생성

```
const handlecreateRoom = async () => {  
  if (!newroomname.trim()) return;
```



```

const res = await fetch('http://localhost:3001/api/auth/createroom', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ name: newroomname })
});

const data = await res.json();
if (res.ok) {
  alert('방 생성 완료');
  setnewroomname('');
  fetchrooms();
} else {
  alert(data.message);
}
};

//방 제거
const handleDeleteRoom = async (roomname) => {
  if (!window.confirm(`${roomname} ㄹㅇ 삭제할까`)) return; //확인창

  const res = await fetch('http://localhost:3001/api/auth/deleteroom', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: roomname })
  }); //json 형태로 확인

  const data = await res.json();
  if (res.ok) {
    alert('삭제 완료');
    fetchrooms();
    if (roomid === roomname) setRoomid('room1'); //삭제된 방의 경우 기본방 room1으로 이동
  } else {
    alert(data.message);
  }
};

//dm
const handleEnterDM = (friendname) => {
  // 친구 이름과 내 이름을 정렬해서 방 이름을 항상 고정된 순서로 설정
  const sorted = [user.username, friendname].sort();
  const dmRoom = `DM_${sorted[0]}_${sorted[1]}`;
  setRoomid(dmRoom); // 해당 DM 방으로 입장 -> dm으로 입장
};

//친구 목록 가져오기
const fetchFriends = async (u = user) => {
  if (!u || !u.username) return;
  const res = await fetch('http://localhost:3001/api/auth/list', {
    method: 'POST',

```

```

    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username: u.username })
  });

  const data = await res.json();
  if (res.ok) {
    setFriends(data.friends);
  } else {
    alert(`${data.message}`);
  }
};

//친구 제거 함수
const handleRemoveFriend = async (friendusername) => {
  const res = await fetch('http://localhost:3001/api/auth/removefriend', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username: user.username, friendusername }) //json형태로 사용
  });

  const data = await res.json();
  if (res.ok) {
    alert(`${data.friend} 친구 제거 완료`);
    fetchFriends(); // 목록 갱신
  } else {
    alert(data.message);
  }
};

// https://xiubindev.tistory.com/100
// 메시지 불러오기
//방 입장시 서버에 표시
useEffect(() => {
  socket.emit('join room', roomid);
}, [roomid]);

useEffect(() => { //화면이 그려지면 한번 시행

  //과거 메시지 수신
  socket.on('chat history', (messages) => { setChat(messages);});
  socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]);});
  //chat message 이벤트 발생시 수행
  //서버에서 전달된 msg를 setchat 호출
  //기존 메시지 배열 복사, 뒤에 msg 연결

  return () => { socket.off('chat message');
    socket.off('chat history');
  }; //중복 등록되지 않도록 제거
}, [roomid]);
useEffect(() => {chatendRef.current?.scrollIntoView({ behavior: 'smooth' });}, [chat]);

useEffect(() => {if (user) {fetchFriends();fetchrooms();}}, [user]);

```

```

//로그인 처리
const handleAuth = async (e) => { //로그인/회원가입 처리 함수
  e.preventDefault();
  const url = `http://localhost:3001/api/auth/${isRegister ? 'register' : 'login'}`; //isRegister가 true인 경우 register 페이지로 0
  const res = await fetch(url, { //요청 전송
    method: 'POST',
    headers: { 'Content-Type': 'application/json' }, //json 형태
    body: JSON.stringify(loginform) //loginform과 같은 형태를 json으로 변환하여 전송
  });

  const data = await res.json(); //응답 처리
  if (res.ok) { //응답 성공한 경우
    const newUser = { username: loginform.username };
    setUser(newUser); // user 상태 설정
    fetchFriends(newUser); // user 값 넘겨주기
    fetchrooms();
  } else {
    alert(data.message);
  }
};

// 메시지 전송 처리
const sendMessage = (event) => { //https://programming119.tistory.com/100
  event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
  if(message.trim() === '') return; //빈 메시지 방지
  socket.emit('chat message', { nickname: user.username, message, roomid }); // 입력받은 메시지 비우기, roomid 기능 추가
  setMessage(''); //메시지 비우기
};

const handleaddfriend = async () => {
  if (!friendname.trim()) return; //friendname이 공백인 경우 그냥 return
  const res = await fetch('http://localhost:3001/api/auth/addfriend', { // 친구 추가 api
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ //json형태로 전송
      username: user.username,
      friendusername: friendname
    })
  });

  const data = await res.json(); //서버로 받은 값을 data에 저장
  if (res.ok) {
    alert(`${data.friend} 친구추가`);
    setFriendName('');
    fetchFriends(); //친구추가 이후 목록 갱신
  } else {
    alert(`${data.message}`);
  }
};

//페이지
if (!user) { //로그인 되어있는 경우

```

```

return (
  <div style={{ padding: '30px', textAlign: 'center' }}>
    <h2>{isRegister ? '회원가입' : '로그인'}</h2>
    <form onSubmit={handleAuth}>
      <input
        type="text"
        placeholder="아이디"
        value={loginform.username}
        onChange={(e) => setLoginForm({ ...loginform, username: e.target.value })} //loginform username과 password에
      /><br /><br />
      <input
        type="password"
        placeholder="비밀번호"
        value={loginform.password}
        onChange={(e) => setLoginForm({ ...loginform, password: e.target.value })}
      /><br /><br />
      <button type="submit">{isRegister ? '회원가입' : '로그인'}</button> { /*버튼을 통해 폼 전환이 되도록 한다*/ }
      <p
        style={{ marginTop: '10px', cursor: 'pointer', color: 'blue' }}
        onClick={() => setIsRegister(!isRegister)}
      >
        {isRegister ? '로그인 전환' : '회원가입 전환'}
      </p>
    </form>
  </div>
);
}

return(
  <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>
    <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
      <h2>실시간 채팅- {roomId}</h2> { /*방 이름 추가 */ }
      <button onClick={() => setUser(null)} style={{ padding: '6px 10px', fontSize: '12px' }}>
        로그아웃
      </button>
    </div>
    { /*방 생성*/ }
    <div style={{ marginBottom: '15px' }}>
      <input
        type="text"
        placeholder="새 방 이름 입력"
        value={newroomname} onChange={(e) => setnewroomname(e.target.value)} style={{ padding: '6px', width: '60%' }}
      <button onClick={handlecreateRoom} style={{ marginLeft: '10px' }}>방 생성 </button>
    </div>

    { /*방 목록 불러오기*/ }
    <div style={{ marginBottom: '20px' }}>
      <h4>방 목록</h4>
      {roomlist.map((room, idx) => ( <button key={idx} onClick={() => setRoomid(room)} style={{ margin: '5px' }}> {room

    </div>

    { /*친구추가 세팅*/ }
    <div style={{ marginBottom: '15px' }}>

```

```

<input type="text" placeholder="친구 아이디 입력" value={friendname} onChange={(e) ⇒ setFriendName(e.target
<button onClick={handleaddfriend} style={{ marginLeft: '8px', padding: '6px 10px' }}>
  친구추가
</button>

</div>

{/*친구 목록 */}
<div style={{ marginBottom: '20px' }}>
  <h4>친구 목록</h4>
  <ul style={{ paddingLeft: '20px' }}>
    {friends.map((f, i) ⇒ (
      <li key={i}>
        {f}
        <button onClick={() ⇒ handleRemoveFriend(f)} style={{ marginLeft: '10px', padding: '2px 6px', fontSize: '12px' }
          제거
        </button>

        <button onClick={() ⇒ handleEnterDM(f)} style={{ marginLeft: '6px', padding: '2px 6px', fontSize: '12px' }}>
          dm
        </button>
      </li> )))}
  </ul>

</div>

<form onSubmit={sendMessage}>
  <input
    type="text"
    value={message}
    onChange={(event)⇒setMessage(event.target.value)}
    placeholder="메시지 입력"
    style={{ width: '70%', padding: '8px'}}
  />
  <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
    메시지 전송
  </button>
</form>
{/* 채팅 출력 폼, user추가 + 자동 스크롤, 메시지 필터링 */}
<ul style={{ marginTop: '20px' }}>
  {chat
    .filter((msg) ⇒ msg.roomid === roomid)
    .map((msg, idx) ⇒ (
      <li key={idx} style={{ marginBottom: '5px', textAlign: msg.nickname === user.username ? 'right' : 'left' }}>
        <strong>{msg.nickname}</strong>: {msg.message}
        <div style={{ fontSize: '10px', color: '#888' }}>{new Date(msg.timestamp).toLocaleTimeString()}</div>
      </li>
    )))}
  <div ref={chatendRef} />
</ul>
</div>
);
}

export default App;

```

이때 backend의 rooms api를 확인해보면 공개 채팅방만 방 목록에 들어간다

```
router.get('/rooms', async (req, res) => {
  try {
    const rooms = await Room.find().sort({ createdAt: -1 }); //방 목록 받아오기기
    res.json({ rooms });
  } catch (err) {
    res.status(500).json({ message: '방 목록 조회 오류' });
  }
});
```

## 테스트

실시간 채팅- room1

로그아웃

새 방 이름 입력

방 생성

방 목록

친구 아이디 입력

친구추가

친구 목록

메시지 입력

메시지 전송

•

test: chat test-1  
2021-10-27 09:45

실시간 채팅- testroom2

로그아웃

새 방 이름 입력

방 생성

방 목록

testroom2 testroom-1

친구 아이디 입력

친구추가

친구 목록

메시지 입력

메시지 전송

실시간 채팅- testroom2

로그아웃

새 방 이름 입력

방 생성

방 목록

testroom2 testroom-1

친구 아이디 입력

친구추가

친구 목록

메시지 입력

메시지 전송

•

test: testmessage-testroom2  
2021-10-27 09:58

실시간 채팅- testroom-1 로그아웃

새 방 이름 입력 방 생성

방 목록

testroom2 testroom-1

친구 아이디 입력 친구추가

친구 목록

메시지 입력 메시지 전송

정상적으로 방을 만들고 해당 방을 통해서 메시지를 전송하는것이 가능했다

실시간 채팅- DM\_test\_test2 로그아웃

새 방 이름 입력 방 생성

방 목록

testroom2 testroom-1

친구 아이디 입력 친구추가

친구 목록

- test2 제거 dm

메시지 입력 메시지 전송

test: hello  
오전 10:00:43

또한 1대1로 특정 유저와 dm을 전송하고 확인하는것 또한 가능했다

추가적으로 다음과 같은 코드를 추가해 방을 제거할 수 있도록 하였다

```

{ /*방 목록 불러오기*/ }
<div style={{ marginBottom: '20px' }}>
<h4>방 목록</h4>
{roomlist.map((room, idx) => (
  <div key={idx} style={{ marginBottom: '5px' }}>
    <button onClick={() => setRoomid(room)}>{room}</button>
    <button onClick={() => handleDeleteRoom(room)} style={{ marginLeft: '8px', fontSize: '10px' }}>삭제</button>
  </div>
))}

</div>

```

**실시간 채팅- new room** 로그아웃

새 방 이름 입력 방 생성

**방 목록**

new room 삭제

방구 아이디 입력 방구추가

**친구 목록**

- test 제거 dm

메시지 입력 메시지 전송

- test: new room -test message  
시간 11:28:38

**실시간 채팅- DM\_test\_test2** 로그아웃

새 방 이름 입력 방 생성

**방 목록**

new room 삭제

hello test2 방구추가

**친구 목록**

- test2 제거 dm

메시지 입력 메시지 전송

- test: hello  
시간 11:28:42
- test2: hello test  
시간 11:28:38

## 이미지 추가

라이브러리 체크

```
npm install multer
```

```
const uploadDir = path.join(__dirname, 'uploads');
if (!fs.existsSync(uploadDir)) fs.mkdirSync(uploadDir);

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadDir),
  filename: (req, file, cb) => {
    const ext = path.extname(file.originalname);
    cb(null, `${Date.now()}~${file.originalname}${ext}`);
  }
})
```

```
const express = require('express'); //express 웹 프레임워크
const router = express.Router(); //라우터 -> register,login 생성
```

```
//이미지 추가용
const multer = require('multer');
const path = require('path');
const fs = require('fs');
```



```

const User = require('./usersetting');
const Room = require('./roomsetting');

//프로필 이미지 저장 위치
const uploadDir = path.join(__dirname, 'uploads');
if (!fs.existsSync(uploadDir)) fs.mkdirSync(uploadDir);

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadDir),
  filename: (req, file, cb) => {
    const ext = path.extname(file.originalname);
    cb(null, `${Date.now()}-${file.originalname}${ext}`);
  }
});

const upload = multer({ storage });

//https://velog.io/@jihukimme/React-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EB%A1%9C%EA%B7%
router.post('/register', async (req, res) => { //회원가입
  const { username, password } = req.body; //https://www.google.com/search?q=req.body&rlz=1C1GCEU_koKR1161KR116'
  try {
    const exist = await User.findOne({ username }); //이미 같은 id가 있는지 확인
    if (exist) return res.status(400).json({ message: '이미 저장된 id' }); //저장된 값이 있는 경우

    const user = new User({ username, password }); //user 정보 획득 및 저장
    await user.save();
    res.status(201).json({ message: '회원가입 성공' });
  } catch (err) {
    res.status(500).json({ message: '서버 오류', error: err });
  }
});

// 로그인
router.post('/login', async (req, res) => { //request,response
  const { username, password } = req.body; //request에서 id,password 추출
  const user = await User.findOne({ username, password }); //해당 요소 찾기
  if (!user) return res.status(401).json({ message: '로그인 실패' }); //user 데이터가 없는 경우

  res.status(200).json({ message: '로그인 성공', user: { username } }); //로그인 response 전송
});

router.post('/addfriend', async (req, res) => {
  const { username, friendusername } = req.body;
  try{
    const user = await User.findOne({ username }); //사용자
    const friend = await User.findOne({ username: friendusername }); //추가하려는 이름
    if (!user || !friend) { // 둘중 하나가 없는 유저인 경우
      return res.status(404).json({ message: '없는 유저' });
    }

    const alreadyfriend = user.friends.includes(friend._id); //이미 친구추가되어 있는지 체크
  }
}

```

```

    if (alreadyfriend) {
      return res.status(400).json({ message: '이미 친구추가됨' });
    }

    //친구 추가
    user.friends.push(friend._id);
    friend.friends.push(user._id);
    await user.save();
    await friend.save();
    res.status(200).json({ message: '친구 추가 완료', friend: friend.username }); //친구 추가 완료 체크

  } catch (err) { //error 확인
    console.error(err);
    res.status(500).json({ message: '서버 오류' });
  }
});

//친구 목록 확인용 api
router.post('/list', async (req, res) => {
  const { username } = req.body;

  try {
    const user = await User.findOne({ username }); //user찾기
    if (!user) return res.status(404).json({ message: '사용자 없음' });

    const friends = await User.find({ _id: { $in: user.friends }, 'username' }); //friend 확인
    res.json({ friends: friends.map(f => f.username) });
  } catch (err) {
    res.status(500).json({ message: '서버 에러' });
  }
});

//친구 제거 api
router.post('/removefriend', async (req, res) => {
  const { username, friendusername } = req.body;
  try {
    const user = await User.findOne({ username }); //user찾기 -> 기존과 동일
    const friend = await User.findOne({ username: friendusername });

    if (!user || !friend) { //둘중에 데이터가 없는 경우
      return res.status(404).json({ message: '없는 유저' });
    }

    //배열에서 제거
    user.friends = user.friends.filter(fld => !fld.equals(friend._id)); //친구 id를 제거해서 새로 확인
    friend.friends = friend.friends.filter(fld => !fld.equals(user._id));
    await user.save();
    await friend.save();

    res.status(200).json({ message: '친구 제거 완료', friend: friend.username });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: '서버 오류' });
  }
}

```

```

});

//방 설정
router.post('/createroom', async (req, res) => {
  const { name } = req.body;
  if (!name) return res.status(400).json({ message: 'room 이름 입력 필요' }); //방 이름 받아오기
  try {
    const exist = await Room.findOne({ name });
    if (exist) return res.status(400).json({ message: '이미 존재하는 방 이름입니다' });

    const room = new Room({ name }); //새 방
    await room.save();
    res.status(201).json({ message: '방 생성 완료', room }); //신규 room 생성 확인인
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: '서버 오류' });
  }
});

router.get('/rooms', async (req, res) => {
  try {
    const rooms = await Room.find().sort({ createdAt: -1 }); //방 목록 받아오기기
    res.json({ rooms });
  } catch (err) {
    res.status(500).json({ message: '방 목록 조회 오류' });
  }
});

//방 삭제 코드
router.post('/deleteroom', async (req, res) => {
  const { name } = req.body;
  try {
    const result = await Room.deleteOne({ name }); //이름 찾아서 지우기
    if (result.deletedCount === 0) {
      return res.status(404).json({ message: '방 확인 error' });
    }
    res.status(200).json({ message: '방 삭제 완료' });
  } catch (err) {
    res.status(500).json({ message: '서버 오류', error: err });
  }
});

//프로필 이미지 추가

router.post('/uploadprofile', upload.single('profile'), async (req, res) => {
  const { username } = req.body;
  if (!req.file) return res.status(400).json({ message: '이미지 파일 필요' }); //이미지 파일을 받아오지 못한 경우

  try {
    const user = await User.findOne({ username });
    if (!user) return res.status(404).json({ message: '사용자 없음' }); //유저체크
  }
});

```

```

    user.profileImage = `/uploads/${req.file.filename}`;
    await user.save();

    res.status(200).json({ message: '프로필 이미지 업로드 완료', imageUrl: user.profileImage });
  } catch (err) {
    res.status(500).json({ message: '서버 오류', error: err });
  }
});

module.exports = router;

```

다음과 같이 각 사용자에게 맞춰서 프로필 이미지를 추가한다

또한 index.js에도

`app.use('/uploads', express.static(path.join(__dirname, 'uploads')));` 을 추가하여 정적 파일에 연결될 수 있도록 하였다

해당 내용을 반영한 app.js

```

import React, { useState, useEffect, useRef } from 'react'; //리엑트 라이브러리
//https://hihiha2.tistory.com/19
import { io } from 'socket.io-client'; //socket 서버 연결

const socket = io('http://localhost:3001');

//참고 https://velog.io/@cychann/React-Class-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-vs-Function-%E
function App() {
  const [message, setMessage] = useState(""); //메시지
  const [chat, setChat] = useState([]); //기존 메시지 목록

  //로그인
  const [user, setUser] = useState(null);
  const [loginform, setLoginForm] = useState({ username: '', password: '' });
  const [isRegister, setIsRegister] = useState(false);
  const [roomid, setRoomid] = useState('room1');
  const chatendRef = useRef(null);
  const [friendname, setFriendName] = useState(""); //친구추가 입력값 상태
  const [friends, setFriends] = useState([]); //친구 목록

  const [profileFile, setProfileFile] = useState(null); // 이미지용 변수 추가

  //방 관련 설정
  const [roomlist, setRoomList] = useState([]);
  const [newroomname, setnewroomname] = useState("");

  //방 목록
  const fetchrooms = async () => {
    const res = await fetch('http://localhost:3001/api/auth/rooms');
    const data = await res.json();
    if (res.ok) setRoomList(data.rooms.map(r => r.name));
  };

```

```

//방 생성
const handlecreateRoom = async () => {
  if (!newroomname.trim()) return;

  const res = await fetch('http://localhost:3001/api/auth/createroom', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: newroomname })
  });

  const data = await res.json();
  if (res.ok) {
    alert('방 생성 완료');
    setnewroomname('');
    fetchrooms();
  } else {
    alert(data.message);
  }
};

//방 제거
const handleDeleteRoom = async (roomname) => {
  if (!window.confirm(`${roomname} ≡ 삭제할까`)) return; //확인창

  const res = await fetch('http://localhost:3001/api/auth/deleteroom', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: roomname })
  }); //json 형태로 확인

  const data = await res.json();
  if (res.ok) {
    alert('삭제 완료');
    fetchrooms();
    if (roomid === roomname) setRoomid('room1'); //삭제된 방의 경우 기본방 room1으로 이동
  } else {
    alert(data.message);
  }
};

//dm
const handleEnterDM = (friendname) => {
  // 친구 이름과 내 이름을 정렬해서 방 이름을 항상 고정된 순서로 설정
  const sorted = [user.username, friendname].sort();
  const dmRoom = `DM_${sorted[0]}_${sorted[1]}`;
  setRoomid(dmRoom); // 해당 DM 방으로 입장 -> dm으로 입장
};

```

```

//친구 목록 가져오기
const fetchFriends = async (u = user) => {
  if (!u || !u.username) return;
  const res = await fetch('http://localhost:3001/api/auth/list', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username: u.username })
  });

  const data = await res.json();
  if (res.ok) {
    setFriends(data.friends);
  } else {
    alert(`${data.message}`);
  }
};

//친구 제거 함수
const handleRemoveFriend = async (friendusername) => {
  const res = await fetch('http://localhost:3001/api/auth/removefriend', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username: user.username, friendusername }) //json형태로 사용
  });

  const data = await res.json();
  if (res.ok) {
    alert(`${data.friend} 친구 제거 완료`);
    fetchFriends(); // 목록 갱신
  } else {
    alert(data.message);
  }
};

// https://xiubindex.tistory.com/100
// 메시지 불러오기
//방 입장시 서버에 표시
useEffect(() => {
  socket.emit('join room', roomid);
}, [roomid]);

useEffect(() => { //화면이 그려지면 한번 시행

  //과거 메시지 수신
  socket.on('chat history', (messages) => { setChat(messages);});
  socket.on('chat message', (msg) => {setChat((prevChat) => [...prevChat, msg]);});
  //chat message 이벤트 발생시 수행
  //서버에서 전달된 msg를 setchat 호출
  //기존 메시지 배열 복사, 뒤에 msg 연결

  return () => { socket.off('chat message');
    socket.off('chat history');
  }; //중복 등록되지 않도록 제거
},[roomid]);

```

```

useEffect(() => {chatendRef.current?.scrollIntoView({ behavior: 'smooth' });}, [chat]);

useEffect(() => {if (user) {fetchFriends();fetchrooms();}}, [user]);

//로그인 처리
const handleAuth = async (e) => { //로그인/회원가입 처리 함수
  e.preventDefault();
  const url = `http://localhost:3001/api/auth/${isRegister ? 'register' : 'login'}`; //isRegister가 true인 경우 register 페이지로 0
  const res = await fetch(url, { //요청 전송
    method: 'POST',
    headers: { 'Content-Type': 'application/json' }, //json 형태
    body: JSON.stringify(loginform) //loginform과 같은 형태를 json으로 변환하여 전송
  });

  const data = await res.json(); //응답 처리
  if (res.ok) { //응답 성공한 경우
    const newUser = { username: loginform.username };
    setUser(newUser); // user 상태 설정
    fetchFriends(newUser); // user 값 넘겨주기
    fetchrooms();
  } else {
    alert(data.message);
  }
};

// 메시지 전송 처리
const sendMessage = (event) => { //https://programming119.tistory.com/100
  event.preventDefault(); //submit 되고 창이 돌아오는것을 방지
  if(message.trim()=== "")return; //빈 메시지 방지
  socket.emit('chat message',{ nickname: user.username, message,roomid}); // 입력받은 메시지 비우기, roomid 기능 추가
  setMessage(""); //메시지 비우기
};

const handleaddfriend = async () => {
  if (!friendname.trim()) return; //friendname이 공백인 경우 그냥 return
  const res = await fetch('http://localhost:3001/api/auth/addfriend', { // 친구 추가 api
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ //json형태로 전송
      username: user.username,
      friendusername: friendname
    })
  });

  const data = await res.json(); //서버로 받은 값을 data에 저장
  if (res.ok) {
    alert(`${data.friend} 친구추가`);
    setFriendName("");
    fetchFriends(); //친구추가 이후 목록 갱신
  } else {
    alert(`${data.message}`);
  }
};

```

```

// 이미지 추가용 함수
const handleUploadProfile = async () => {
  if (!profileFile || !user) return; //유저가 맞지 않은 경우 return, 이미지가 존재하지 않는 경우 return
  const formData = new FormData(); //multipart/form-data 형태로 전송하기위해 객체 생성
  formData.append('profile', profileFile);
  formData.append('username', user.username);
  const res = await fetch('http://localhost:3001/api/auth/uploadprofile', {
    method: 'POST',
    body: formData
  });
  const data = await res.json(); //전송 및 확인
  if (res.ok) {
    alert('프로필 이미지 업로드 성공');
    setUser({ ...user, profileImage: data.imageUrl });
  } else {
    alert(data.message);
  }
};

//페이지
if (!user) { //로그인 되어있는 경우
  return (
    <div style={{ padding: '30px', textAlign: 'center' }}>
      <h2>{isRegister ? '회원가입' : '로그인'}</h2>
      <form onSubmit={handleAuth}>
        <input
          type="text"
          placeholder="아이디"
          value={loginform.username}
          onChange={(e) => setLoginForm({ ...loginform, username: e.target.value })} //loginform username과 password에
        /><br /><br />
        <input
          type="password"
          placeholder="비밀번호"
          value={loginform.password}
          onChange={(e) => setLoginForm({ ...loginform, password: e.target.value })}
        /><br /><br />
        <button type="submit">{isRegister ? '회원가입' : '로그인'}</button> {/*버튼을 통해 폼 전환이 되도록 한다*/}
      <p
        style={{ marginTop: '10px', cursor: 'pointer', color: 'blue' }}
        onClick={() => setIsRegister(!isRegister)}
      >
        {isRegister ? '로그인 전환' : '회원가입 전환'}
      </p>
    </form>
  </div>
  );
}

return(
  <div style={{ padding: '20px', maxWidth: '500px', margin: 'auto' }}>

```



```

<div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
  <h2>실시간 채팅- {roomId}</h2> { /*방 이름 추가 */}
  <button onClick={() => setUser(null)} style={{ padding: '6px 10px', fontSize: '12px' }}>
    로그아웃
  </button>
</div>
{ /*방 생성*/}
<div style={{ marginBottom: '15px' }}>
  <input
    type="text"
    placeholder="새 방 이름 입력"
    value={newroomname} onChange={(e) => setnewroomname(e.target.value)} style={{ padding: '6px', width: '60%' }}
    <button onClick={handlecreateRoom} style={{ marginLeft: '10px' }}>방 생성 </button>
  </div>

  { /*방 목록 불러오기*/}
  <div style={{ marginBottom: '20px' }}>
    <h4>방 목록</h4>
    {roomlist.map((room, idx) => (
      <div key={idx} style={{ marginBottom: '5px' }}>
        <button onClick={() => setRoomid(room)}>{room}</button>
        <button onClick={() => handleDeleteRoom(room)} style={{ marginLeft: '8px', fontSize: '10px' }}>삭제</button>
      </div>
    ))}
  </div>

  { /*프로필 이미지 설정 */}
  <div style={{ marginBottom: '15px' }}>
    <h4>프로필 이미지</h4>
    {user.profileImage && (
      <div style={{ marginBottom: '10px' }}>
        <img src={`http://localhost:3001/uploads/${user.profileImage}`}
          alt="프로필" style={{ width: '80px', height: '80px', borderRadius: '50%' }} /> </div>

        <input type="file" accept="image/*" onChange={(e) => setProfileFile(e.target.files[0])} />
        <button onClick={handleUploadProfile} style={{ marginLeft: '10px' }}>이미지 업로드 </button>
      </div>

  { /*친구추가 세팅*/}
  <div style={{ marginBottom: '15px' }}>
    <input type="text" placeholder="친구 아이디 입력" value={friendname} onChange={(e) => setFriendName(e.target.value)}
    <button onClick={handleaddfriend} style={{ marginLeft: '8px', padding: '6px 10px' }}>
      친구추가
    </button>
  </div>

  { /*친구 목록 */}
  <div style={{ marginBottom: '20px' }}>
    <h4>친구 목록</h4>
    <ul style={{ paddingLeft: '20px' }}>

```

```

    {friends.map((f, i) => (
      <li key={i}>
        {f}
        <button onClick={() => handleRemoveFriend(f)} style={{ marginLeft: '10px', padding: '2px 6px', fontSize: '12px' }}>
          제거
        </button>

        <button onClick={() => handleEnterDM(f)} style={{ marginLeft: '6px', padding: '2px 6px', fontSize: '12px' }}>
          dm
        </button>
      </li> )))
  </ul>

</div>

<form onSubmit={sendMessage}>
  <input
    type="text"
    value={message}
    onChange={(event) => setMessage(event.target.value)}
    placeholder="메시지 입력"
    style={{ width: '70%', padding: '8px' }}
  />
  <button type="submit" style={{ padding: '8px 12px', marginLeft: '10px' }}>
    메시지 전송
  </button>
</form>
{/* 채팅 출력 폼, user추가 + 자동 스크롤, 메시지 필터링 */}
<ul style={{ marginTop: '20px' }}>
  {chat
    .filter((msg) => msg.roomid === roomid)
    .map((msg, idx) => (
      <li key={idx} style={{ marginBottom: '5px', textAlign: msg.nickname === user.username ? 'right' : 'left' }}>
        <strong>{msg.nickname}</strong>: {msg.message}
        <div style={{ fontSize: '10px', color: '#888' }}>{new Date(msg.timestamp).toLocaleTimeString()}</div>
      </li>
    ))}
  <div ref={chatendRef} />
</ul>
</div>
);
}

export default App;

```

다음과 같이 이미지를 받아서 저장한다.

```

const handleUploadProfile = async () => {
  if (!profileFile || !user) return; //유저가 맞지 않은 경우 return, 이미지가 존재하지 않는 경우 return

```

```
const formData = new FormData(); //multipart/form-data 형태로 전송하기위해 객체 생성
formData.append('profile', profileFile);
formData.append('username', user.username);
const res = await fetch('http://localhost:3001/api/auth/uploadprofile', {
  method: 'POST',
  body: formData
});
const data = await res.json(); //전송 및 확인
if (res.ok) {
  alert('프로필 이미지 업로드 성공');
  setUser({ ...user, profileImage: data.imageUrl });
} else {
  alert(data.message);
}
};
```

또한 이미지 업로드와 표시를 위한 코드를 추가하였다

```
<div style={{ marginBottom: '15px' }}>
  <h4>프로필 이미지</h4>
  {user.profileImage && (
    <div style={{ marginBottom: '10px' }}>
      <img src={`http://localhost:3001${user.profileImage}`}
        alt="프로필" style={{ width: '80px', height: '80px', borderRadius: '50%' }} /> </div> )}

  <input type="file" accept="image/*" onChange={(e) => setProfileFile(e.target.files[0])} />
  <button onClick={handleUploadProfile} style={{ marginLeft: '10px' }}> 이미지 업로드 </button>
</div>
```

## 테스트



정상적으로 프로필 이미지도 추가되는것을 확인할 수 있다

실시간 채팅- DM\_test\_test2

로그아웃

새 방 이름 입력

방 생성

방 목록

new room

방 목록

프로필 이미지

회원 삭제

선택된 파일 없음

이미지 업로드

친구 아이디 입력

친구 추가

친구 목록

test

제거

DM

메시지 입력

메시지 전송

- test: hello  
시간 10:20:43
- 
- test: hi test2  
시간 10:21:11

test2: hello test  
시간 10:20:48

도커 설정

backend docker

```
FROM node:18
RUN apt-get update && apt-get install -y netcat-openbsd
WORKDIR /app

# 설치
COPY package*.json ./
RUN npm install

# 전체 코드 복사
COPY . .

# wait-for-it.sh 복사 및 실행 권한 부여
COPY wait-for-it.sh /wait-for-it.sh
RUN chmod +x /wait-for-it.sh

EXPOSE 3001

# mongo:27017 이 열릴 때까지 기다렸다가 backend 실행
CMD ["/wait-for-it.sh", "mongo:27017", "--", "node", "index.js"]
```

frontend docker

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

doceker-compose.yml

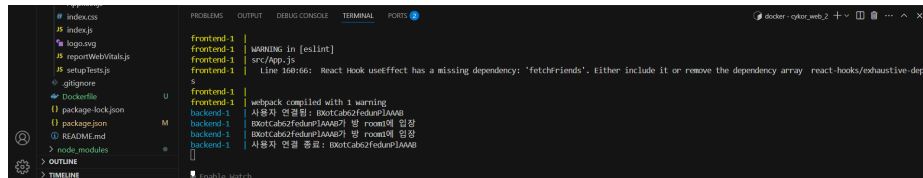
```
networks:
  appnet:
```



```

racon@DESKTOP-H67TV7J:/mnt/c/ku/cykor/2_1/web2/cykor_web_2$ sudo rm -rf ./data/db
[sudo] password for racon:
racon@DESKTOP-H67TV7J:/mnt/c/ku/cykor/2_1/web2/cykor_web_2$ mkdir -p ./data/db
racon@DESKTOP-H67TV7J:/mnt/c/ku/cykor/2_1/web2/cykor_web_2$ sudo chmod -R 777 ./data/db
racon@DESKTOP-H67TV7J:/mnt/c/ku/cykor/2_1/web2/cykor_web_2$

```




**실시간 채팅- room1** 로그아웃

새 방 이름 입력 방 생성

**방 목록**

프로필 이미지



파일 선택 download.jpg 이미지 업로드

친구 아이디 입력 친구추가

**친구 목록**

메시지 입력 메시지 전송

**실시간 채팅- newroom** 로그아웃

새 방 이름 입력 방 생성

**방 목록**

newroom 선택

프로필 이미지

파일 선택 선택한 파일 없음 이미지 업로드

친구 아이디 입력 친구추가

**친구 목록**

메시지 입력 메시지 전송

- test: test  
방명: 1234567
- test2: hi  
방명: 1234567

**실시간 채팅- DM\_test\_test2** 로그아웃

새 방 이름 입력 방 생성

**방 목록**

newroom 선택

프로필 이미지

파일 선택 선택한 파일 없음 이미지 업로드

친구 아이디 입력 친구추가

**친구 목록**

test 채팅 DM

메시지 입력 메시지 전송

- test2: hello test  
방명: 1234567

각 기능이 정상적으로 작동하는 것을 확인할 수 있다