

웹게시판 사이트 실습 가이드

사용기술

react+express+mysql

깃허브 저장소

<https://github.com/notecoding/reactProject>

node js와 vscode 설치

- node js 설치
- <https://tipbox.co.kr/node-js-%EC%84%A4%EC%B9%98-%EB%B0%A9%EB%B2%95/>
- vscode 설치
- <https://m.blog.naver.com/thomasworld/223661500899>

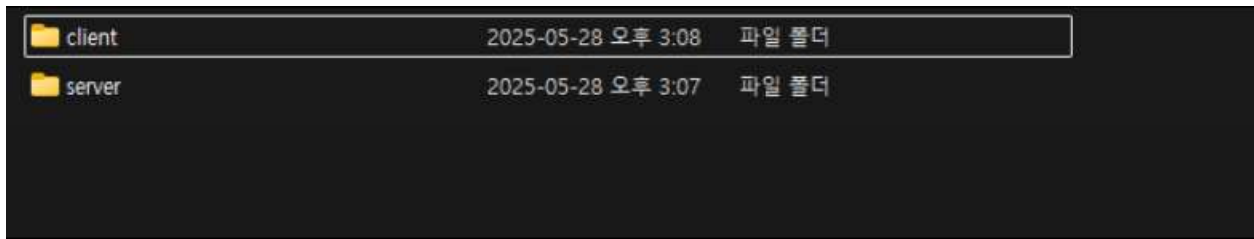
기능

- 로그인 회원가입
- 좋아요 댓글 게시판 글쓰기
- 카테고리별 분류
- 관리자 권한을 활용한 회원관리

1.작업할 폴더 생성 및 그 안에 리액트 파일 생성

```
npx create-react-app client
```

2.백엔드를 작업할 파일도 생성



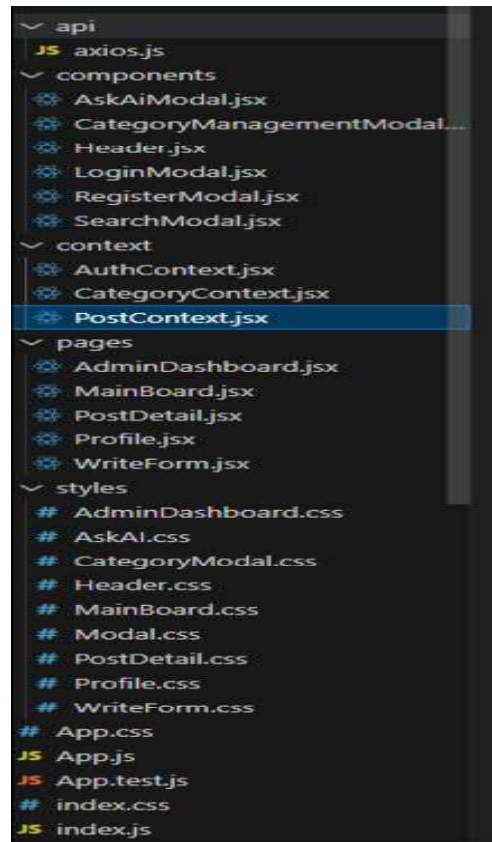
초기 패키지 설치(프론트)

- 1.http통신을 위한 axios
- 2.페이지 전환과 url기반 컴포넌트 렌더링을 위한 react-router-dom

```
C:\Users\User\Desktop\webSite\client>npm install axios@latest react-router-dom@latest
```

@latest를 쓰는 이유는 최신 버전 사용을 위해서

전체 프론트 구조



axios api

간단한 api 작성을 위해 필요

components

재사용 가능한 ui들(모달)

context

비즈니스 로직 관리

pages

각 라우팅 되는 페이지들

styles

각 컴포넌들을 스타일 정의

axios api

axios는 브라우저와 Node.js에서 HTTP 요청을 보낼 수 있게 해주는 js 라이브러리

```
import axios from 'axios';

✓ const instance = axios.create({
  baseURL: 'http://localhost:5000/api',
  headers: { 'Content-Type': 'application/json' },
});

// 요청 시 토큰 자동 포함
✓ instance.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

export default instance;
```

baseURL

모든 요청 url앞에 자동으로
기본 주소를 생성하게 함

headers

모든 요청에 기본으로 붙는
HTTP헤더를 서정(서버에JSON
형태로 데이터를 보냄)

interceptors.request.use()

서버로 요청을 보내기 전에
로컬스토리지에 토큰을 확인

존재한다면 토큰 문자열 자체
를 요청헤더에 실어서 서버로
보냄

axios 디렉토리를 만들었을 때와 안 만들었을 때

"사용자 정보"를 가져오는 동작에 대한 예시

axios 디렉토리를 안 만들었을 때

```
import axios from 'axios';

const token = localStorage.getItem('token');
const res = await axios.get('http://localhost:5000/api/user', {
  headers: {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`,
  },
});
```

axios 디렉토리를 만들었을 때

```
import axios from '../api/axios';

const res = await axios.get('/user');
```


프론트의 전체 흐름 예시(로그인)

/Context/AuthContext(로그인,회원가입,로그아웃)

```
// src/context/AuthContext.jsx
import React, { createContext, useState, useEffect } from 'react';
import axios from '../api/axios';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(false);

  const login = async ({ username, password }) => {
    setLoading(true);
    try {
      const res = await axios.post('/auth/login', { username, password });
      const { token, user } = res.data;

      localStorage.setItem('token', token);
      localStorage.setItem('user', JSON.stringify(user));
      setUser(user);
    } catch (err) {
      throw new Error('로그인 실패');
    } finally {
      setLoading(false);
    }
  };
};
```

axios api와 코드 작성에
필요한 훅들을 가져옴

AuthContext라는 이름의 컨텍스트(Context)를 생성
상태와 함수 공유 가능

상태 변수 선언

로딩 시작

서버에 username과
password 요청

로컬 스토리지에 저장

오류 발생 시 실행

useEffect는
컴포넌트가 화면에
처음 렌더링될 때
실행

```
useEffect(() => {  
  const stored = localStorage.getItem('user');  
  if (stored) setUser(JSON.parse(stored));  
}, []);  
  
return (  
  <AuthContext.Provider value={{ user, loading, login, register, logout }}>  
    {children}  
  </AuthContext.Provider>  
);
```

로컬 스토리지에
user정보가
남아있으면
불러와서
user상태로 세팅
새로고침해도
로그인
상태를
유지하기
위해

AuthContext.Provider로
감싸진 자식 컴포넌트에
user, loading, login,
register, logout 값을
전달
context를 사용하는 컴포넌트
어디서든 사용가능

/components/LoginModal

```
src/components/LoginModal.jsx

import React, { useContext, useState } from 'react';
import '../styles/Modal.css';
import { AuthContext } from '../context/AuthContext';

const LoginModal = ({ onClose, onSwitchToRegister }) => {
  const { login, loading } = useContext(AuthContext);

  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleLogin = async () => {
    setError('');
    try {
      await login({ username, password });
      onClose(); // 로그인 성공 시 모달 닫기
    } catch (err) {
      setError('로그인 실패: 아이디 또는 비밀번호를 확인하세요.');
```

모달 관련 CSS
스타일을 불러와서 이
컴포넌트에 적용

AuthContext를 가져와서
로그인 함수와 로딩 상태를
사용할 준비

context의 로그인 함수 호출

app.js에서
onClose함수
선언

```
{loginOpen && (
  <LoginModal
    onClose={() => setLoginOpen(false)}
    onSwitchToRegister={() => {
      setLoginOpen(false);
      setRegisterOpen(true);
    }}
  />
)}
```

```

<div className="auth-form">
  <div className="form-group">
    <label>아이디</label>
    <input
      type="text"
      value={username}
      onChange={(e) => setUsername(e.target.value)}
      placeholder="아이디 입력"
    />
  </div>
  <div className="form-group">
    <label>비밀번호</label>
    <input
      type="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      placeholder="비밀번호 입력"
    />
  </div>
</div>

<button
  className="auth-button"
  onClick={handleLogin}
  disabled={!username || !password || loading}
>
  {loading ? '로딩 중...' : '로그인'}
</button>

<div className="auth-links">
  아직 회원이 아니신가요?{' '}
  <a onClick={onSwitchToRegister}>회원가입</a>
</div>

```

```

<button className="close-btn" onClick={onClose}
  닫기
</button>
</div>

```

username, password
상태와 연동, 입력 시
상태 업데이트

아이디, 비밀번호 둘 중
하나라도 없거나, 로그인
중이면 버튼
비활성화(disabled)

로그인 요청 중이면 버튼
텍스트가 '로딩 중...' 으로
바뀜

pages/WriteForm

user데이터에서
이름만 가져옴

```
// src/pages/WriteForm.jsx
import React, { useContext, useEffect, useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { PostContext } from '../context/PostContext';
import { CategoryContext } from '../context/CategoryContext';
import { AuthContext } from '../context/AuthContext';
import '../styles/WriteForm.css';

const WriteForm = () => {
  const { user } = useContext(AuthContext);
  const { categories } = useContext(CategoryContext);
  const {
    createPost,
    updatePost,
    getPost,
  } = useContext(PostContext);
```

AuthContext를
가져와서 user데이터를
받아옴

user가 존재하는지
확인

```
<div className="author">작성자: {user?.name}</div>
```

```
// 로그인 여부 확인
useEffect(() => {
  if (!user) {
    alert('로그인이 필요합니다.');
```

↑

```
    navigate('/');
  }
}, [user, navigate]);

const handleSubmit = async (e) => {
  e.preventDefault();

  if (!title.trim() || !content.trim() || !category) {
    alert('모든 항목을 입력해주세요.');
```

↑

```
    return;
  }
}
```

app.js

```
import React, { useState } from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { Navigate } from 'react-router-dom';

// ✅ 페이지
import MainBoard from './pages/MainBoard';
import WriteForm from './pages/WriteForm';
import PostDetail from './pages/PostDetail';
import Profile from './pages/Profile';
import AdminDashboard from './pages/AdminDashboard'; // ✅ 관리자 페이지 import

// ✅ 컴포넌트
import AskAiModal from './components/AskAiModal'; // ✅ 추가
import Header from './components/Header';
import LoginModal from './components/LoginModal';
import RegisterModal from './components/RegisterModal';
import SearchModal from './components/SearchModal';
import CategoryManagementModal from './components/CategoryManagementModal';

// ✅ Context
import { AuthProvider } from './context/AuthContext';
import { PostProvider } from './context/PostContext';
import { CategoryProvider } from './context/CategoryContext';

function App() {
  const [loginOpen, setLoginOpen] = useState(false);
  const [registerOpen, setRegisterOpen] = useState(false);
  const [searchOpen, setSearchOpen] = useState(false);
  const [categoryOpen, setCategoryOpen] = useState(false);
  const [aiOpen, setAiOpen] = useState(false); // ✅ 상태 추가
```

← 각 컴포넌트들 import

← 모달 열기 닫기 상태들


```
<AuthProvider>
  <PostProvider>
    <CategoryProvider>
```

하위 컴포넌트들이 Auth, Post, Category 상태에 접근할 수 있도록 전역 상태를 제공

```
/* 상단 헤더 */
<Header>

  toggleLoginModal={() => setLoginOpen(true)}
  toggleRegisterModal={() => setRegisterOpen(true)}
  toggleSearchModal={() => setSearchOpen(true)}
  toggleCategoryModal={() => setCategoryOpen(true)}
  toggleAiModal={() => setAiOpen(true)}

/* 페이지 라우팅 */
<Routes>
  <Route path="/" element={<Navigate to="/board/all" />} />
  <Route path="/board/:categoryName" element={<MainBoard />} />
  <Route path="/write" element={<WriteForm />} />
  <Route path="/edit/:id" element={<WriteForm />} />
  <Route path="/post/:id" element={<PostDetail />} />
  <Route path="/profile" element={<Profile />} />
  <Route path="/ai" element={<AskAiModal />} />
  /* 관리자 페이지 라우트 */
  <Route path="/admin" element={<AdminDashboard />} />

  /* 기본 경로 리디렉션 또는 NotFound 페이지도 필요 시 추가 가능 */
  <Route path="*" element={<div>페이지를 찾을 수 없습니다.</div>} />
</Routes>
```

페이지 라우팅 경로 설정

찾을 수 없다면 오류 메시지 띄움

mysql 설치 및 데이터베이스 스키마 정의

해당 사이트 참고 후 mysql 설치
white63ser.tistory.com/16

1. mysql에서 sql파일을 클릭



2. 데이터베이스 스키마 정의

1. users (사용자)

사용자 정보 저장

id, name, username, password, is_admin, created_at

2. posts (게시글)

게시글 내용 저장

id, category, title, content, author, author_id, likes, created_at, updated_at

3. comments (댓글)

게시글의 댓글 저장

id, post_id, author, author_id, content, created_at, updated_at

4. categories (카테고리)

게시글 분류용 카테고리

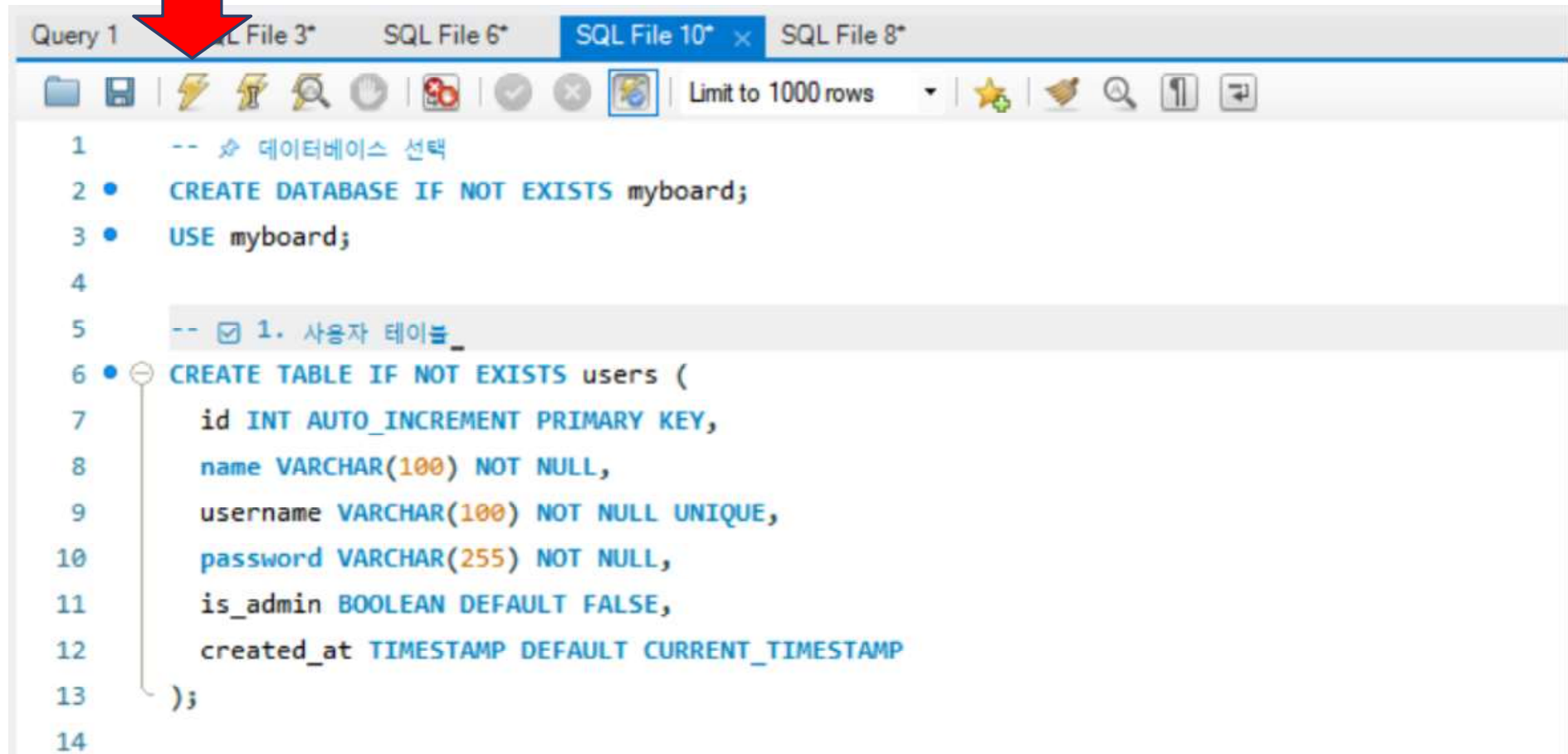
id, name, order, created_at

5. likes (좋아요)

사용자별 게시글 좋아요 정보

id, user_id, post_id, created_at

3.sql 명령어 입력 후 번개모양 클릭



The screenshot shows a SQL IDE window with multiple tabs: 'Query 1', 'SQL File 3*', 'SQL File 6*', 'SQL File 10*' (selected), and 'SQL File 8*'. The toolbar contains various icons, including a lightning bolt icon for executing the query. A red arrow points to this icon. The query editor displays the following SQL code:

```
1  -- ⚡ 데이터베이스 선택
2  • CREATE DATABASE IF NOT EXISTS myboard;
3  • USE myboard;
4
5  -- ☑ 1. 사용자 테이블_
6  • CREATE TABLE IF NOT EXISTS users (
7      id INT AUTO_INCREMENT PRIMARY KEY,
8      name VARCHAR(100) NOT NULL,
9      username VARCHAR(100) NOT NULL UNIQUE,
10     password VARCHAR(255) NOT NULL,
11     is_admin BOOLEAN DEFAULT FALSE,
12     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
13 );
14
```

3.5) 샘플코드도 삽입 후 번개모양 클릭

4. 나갔다 들어오면 적용 완료



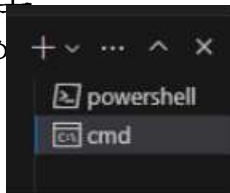
```
Query 1  SQL File 3*  SQL File 6*  SQL File 10*  SQL File 8* x
Limit to 1000 rows
1 • USE myboard;
2
3 -- ☑ 사용자 (관리자 + 일반 사용자) pw:admin1234
4 • INSERT INTO users (name, username, password, is_admin) VALUES
5 ('관리자', 'admin', '$2b$10$Xk01eA5Z88YDnHdRTzVN..CqsayGbHjea5p7WvFVU5ESz1yMfQx1K', TRUE);
6
7 -- ☑ 카테고리
8 • INSERT INTO categories (name, `order`) VALUES
9 ('자유게시판', 1),
10 ('질문답변', 2);
```

초기 패키지 설치(백엔드)

1.vs code에서 ctrl+~키를
누름

2.하단의 +키를 누른 cmd
접속

3.여기서 입력



```
C:\Users\User\Desktop\webSite>npm init -y
```

```
C:\Users\User\Desktop\webSite>npm install express cors dotenv mysql2 jsonwebtoken bcryptjs openai
```

명령어들 설명(패키지)

npm init -y

(기본 package.json 파일을 생성)

npm install express cors dotenv mysql2 jsonwebtoken bcryptjs openai

express (express 설치)

cors (프론트와 백엔드 통신을 위해 필요)<브라우저에서 자동 차단하기 때문>

dotenv (.env 파일 사용을 위해 필요)

mysql2 (mysql 연동)

jsonwebtoken(JWT생성 및 검증을 위해 필요)

bcryptjs(비밀번호 해싱 및 비교)

openai (ai api를 위해 필요)

(패키지 설치를 할 때는 반드시 실질적인 백엔드 디렉토리에서 할 것)

.env 파일 설정

.env파일을 사용(환경변수 관리 및 보안성 강화)

```
-----  
PORT=5000  
DB_HOST=localhost  
DB_USER=root  
DB_PASSWORD=yourKey  
DB_NAME=myboard  
JWT_SECRET=your_jwt_secret  
OPENAI_API_KEY=yourKey  
-----
```

주의 사항

api키와 db키 바꿀 것


JWT_SECRET은 복잡한 문자열로 바꿀 것

초기 패키지 설치를 설치하고 .env파일을 만들었다면 데이터 베이스와 서버 연결

```
const mysql = require('mysql2/promise');
require('dotenv').config();

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});

module.exports = pool;
```



process.env.~
라고 적혀있는 곳이 실제
env파일의 변수를
사용하는것

db 연결 및 .env파일 생성

- 1) db.js를 만듦
- 2) mysql라이브러리 불러오기
- 3) db의 호스트, user, 비밀번호, 이름을 삽입
- 3) exports후에 밖으로 내보냄
- 4).env파일에 위 db 정보들을 다 넣은 후 ai api key, jwt 해시 번호, port번호도 넣음

백엔드 전체 구조

Router

클라이언트 요청을 어떤 컨트롤러가 처리할지 연결해줌

Controller

실제 요청을 받아서 비즈니스 로직 처리 후 응답을 보냄

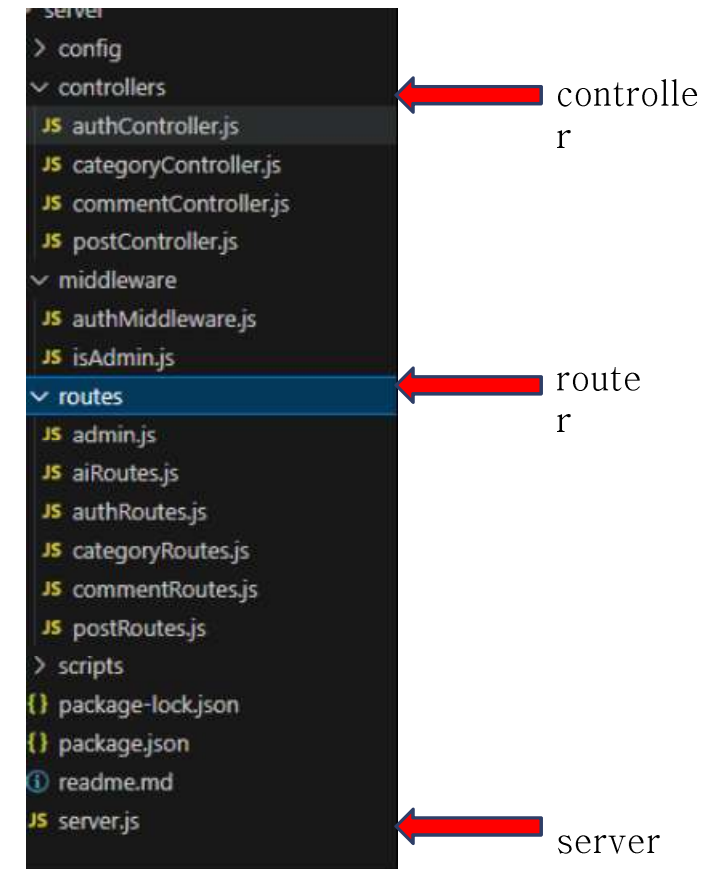
server

클라이언트 요청을 받아 처리하는 전체 시스템의 시작점

1)server에서 요청을 받음

2)요청을 받은 라우터가 어느 컨트롤러에 갈지 결정

3)해당 컨트롤러가 그에 맞는 로직을 실행하고 결과 반환



백엔드 기초: 요청, 응답, HTTP 메서드

요약

req, res

req:요청을 받는 객체

res:응답을 보내는 객체

```
exports.profile = async (req, res) => {  
  const userId = req.user.id;  
  try {  
    const [rows] = await pool.query('SELECT id, name, username FROM users WHERE id = ?', [userId]);  
    res.json(rows[0]);  
  } catch (err) {  
    res.status(500).json({ message: '서버 오류', error: err.message });  
  }  
};
```

get 데이터 조회

post 데이터 전송

put 데이터 수정/갱신

delete 데이터를 삭제

```
router.put('/change-password', verifyToken, changePassword);  
router.delete('/delete', verifyToken, deleteAccount);  
  
router.post('/register', register);  
router.post('/login', login);  
router.get('/profile', verifyToken, profile);
```

```
const [rows] = await pool.query('SELECT id, name, username FROM users WHERE id = ?', [userId]);  
res.json(rows[0]);
```

위 http 메서드를 활용할 때 sql 쿼리를 사용 ↑

클라이언트가 보낸 json파일 해석

```
app.use(express.json());
```

```
const jwt = require('jsonwebtoken');

const verifyToken = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader?.startsWith('Bearer ')) {
    return res.status(401).json({ message: '인증 토큰이 필요합니다.' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(403).json({ message: '토큰이 유효하지 않습니다.' });
  }
};

module.exports = verifyToken;
```



```
router.get('/profile', verifyToken, profile);
```

```
// 3. 사용자 정보 구성 (프론트 전달용)
const userData = {
  id: user.id,
  name: user.name,
  username: user.username,
  isAdmin: user.is_admin === 1 || user.is_admin === true // ✅ 핵심: 관리자 여부 포함
};

// 4. JWT 생성
const token = jwt.sign(userData, process.env.JWT_SECRET, { expiresIn: '1d' });
```



사용자 정보 구성



jwt 검사

JWT

jwt를 이용하여 인증 및 정보교환을 할 수 있음

위 코드에서 토큰을 확인하여 로그인이 되어있는지 여부와
토큰안에 정보를 이용하여 프로필을 확인 할 수 있음

next()문을 이용하여 다음으로 넘어감

auth기능 구현 예시

회원가입, 로그인, 프로필, 비밀번호변경, 회원탈퇴 구성

프로필 예시

컨트롤러

```
exports.profile = async (req, res) => {  
  const userId = req.user.id;  
  try {  
    const [rows] = await pool.query('SELECT id, name, username FROM users WHERE id = ?', [userId]);  
    res.json(rows[0]);  
  } catch (err) {  
    res.status(500).json({ message: '서버 오류', error: err.message });  
  }  
};
```


라우터

```
router.get('/profile', verifyToken, profile);
```

서버

```
app.use('/api/auth', authRoutes);
```

```
const {  
  register, login, profile,  
  changePassword, deleteAccount  
} = require('../controllers/authController');
```



관리자 생성 방법

1.mysql에 직접 삽입

```
- [x] 사용자 (관리자 + 일반 사용자) pw:admin1234
INSERT INTO users (name, username, password, is_admin) VALUES
('관리자', 'admin', '$2b$10$Xk01eASZ88YDnHdRTzVN..CqsayGbHjea5p7WvFVU5ESz1yMfQx1K', TRUE);
```

2.어드민을 만들기 위한js파일을 따로 만들어 삽입(cmd창에 입력)

```
if (!username || !password) {
  console.error('사용법: node make-admin.js [username] [password] [name]');
  process.exit(1);
}
```

관리자 계정을 따로
생성하는 이유는 보안,
유지보수, 자동화, 실수
방지 등의 이유 때문

관리자 전용 미들웨어

관리자가 아니라면 접근 불가

```
const isAdmin = (req, res, next) => {  
  if (!req.user) return res.status(401).json({ message: '인증 정보가 없습니다.' });  
  
  if (!req.user.isAdmin) return res.status(403).json({ message: '관리자 권한이 필요합니다.' });  
  
  next();  
};  
  
module.exports = isAdmin;
```


오류 코드

201 생성 완료

```
res.status(201).json({ message: '회원가입 성공' });  
} catch (err) {
```

400 요청 에러

```
const [existing] = await pool.query(`SELECT * FROM users WHERE username = ?`, [username]);  
if (existing.length > 0) return res.status(400).json({ message: '이미 존재하는 아이디입니다.' });
```

401 인증 필요

```
if (!authHeader?.startsWith('bearer ')) {  
  return res.status(401).json({ message: '인증 토큰이 필요합니다.' });  
}
```

403 권한 부족

```
res.status(403).json({ message: '토큰이 유효하지 않습니다.' });
```

500 서버 터짐

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

ai 프론트 코드

```
if (!question.trim()) return;
setLoading(true);
setAnswer('');

try {
  const res = await axios.post('http://localhost:5000/api/ai/ask', { question });
  setAnswer(res.data.answer);
} catch (err) {
  console.error(err);
  setAnswer('❌ AI 응답에 실패했습니다.');
```

```
} finally {
  setLoading(false);
}

return (
  <div className="modal-overlay">
    <div className="modal-content">
      <button className="modal-close" onClick={onClose}>X</button>
      <h2>🤖 AI에게 질문하기</h2>
      <textarea
        value={question}
        onChange={(e) => setQuestion(e.target.value)}
        placeholder="질문을 입력하세요..."
        rows={4}
        className="ask-ai-input">
      </textarea>
      <button onClick={handleAsk} disabled={loading} className="ask-ai-button">
        {loading ? '답변 생성 중...' : '질문하기'}
      </button>
      {answer && (
        <div className="ask-ai-answer">
          <h3>🔥 AI의 답변:</h3>
          <p>{answer}</p>
        </div>
      )}
    </div>
  </div>
);
```

질문이 비어있다면 반환

POST 요청을 보냄
요청 바디에 { question }
객체를 JSON 형식으로 전송

로딩 상태면 질문하기 버튼
막기

ai 백엔드 코드

```
// ✅ 최신 SDK 사용 방식
const express = require('express');
const router = express.Router();
require('dotenv').config();

const OpenAI = require('openai'); // ✅ 변경된 import

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
});

router.post('/ask', async (req, res) => {
  const { question } = req.body;

  if (!question) {
    return res.status(400).json({ error: '질문이 없습니다.' });
  }

  try {
    const chatCompletion = await openai.chat.completions.create({
      model: 'gpt-3.5-turbo',
      messages: [{ role: 'user', content: question }],
    });

    res.json({ answer: chatCompletion.choices[0].message.content });
  } catch (error) {
    console.error('AI 오류:', error);
    res.status(500).json({ error: 'AI 응답 실패' });
  }
});

module.exports = router;
```

최신 sdk방식

axios를 사용하면 에러가
많음

api key .env파일에서 가져옴

req.body.question:
클라이언트에서 보낸 질문

사용할 모델 (gpt-3.5-turbo)

GPT가 생성한 답변을
클라이언트에게 전송

출처

- <https://tipbox.co.kr/node-js-%EC%84%A4%EC%B9%98-%EB%B0%A9%EB%B2%95/>
- <https://m.blog.naver.com/thomasworld/223661500899>

white63ser.tistory.com/16

챗 gpt

감사합니다