기술 발표

발표자: 정세희

대상: 선행IP팀

일시: 2026-02-23

📋 목차

주제 1

GitLab CI/CD 기반

사내망 자동 배포 시도 결과 공유

시도 목적 / 목표 구조

목표

GitLab CI/CD로 사내망 개발 환경(PWC)에 자동 배포 구축

의도한 구조

```
GitLab (Repo)
    ↓
GitLab CI Pipeline
    ↓
공용 Runner (shared-kr-runner, HCloud)
    ↓
Docker 이미지 (빌드/배포 스크립트 실행)
    ↓
SSH로 PWC에 배포 (10.90.170.168)
```

진행한 작업 (여기까지는 완료)

1️⃣ CI/CD 구성 ✅

- `.gitlab-ci.yml`, `deploy.sh` 등 배포 스크립트 구성 완료
- GitLab Variables 설정 (키/환경변수 등)
- 공용 Runner(shared-kr-runner) 사용 설정

2️⃣ 네트워크/보안 절차 ✅

- **SRMS 방화벽 신청/승인 완료**
  - 출발지: `10.11.95.229` (HCloud 공용 Runner)
  - 목적지: `10.90.170.168:22` (PWC)
  - ⚠️ 특이사항: 테스트 목적 **6개월 한시 허용** 안내 받음

핵심 장애 (막힌 지점) ❌

PWC에서 SSH 수신 환경 구성 불가

문제점:

- PWC는 Windows PC (개인/팀 단말 성격)
- 보안/권한 정책(Defender, 백신, 로컬 관리자 권한 제한 등)으로
  OpenSSH 설치·활성화 시도 자체가 어려움

확인 결과:

```
netstat -an | findstr :22
```

- **22번 포트를 LISTEN 상태로 만들 수 없음**
- → 외부에서 SSH로 배포를 "받는" 구조가 성립하지 않음

왜 방화벽을 열어도 안 됐나? 🤔

한 문장 정리

> 방화벽은 열렸지만, 목적지(PWC)에 SSH 서비스 자체를 올릴 수 없어 접속 불가

요약

구분	상태	설명
네트워크 (길)	✅ 정상	SRMS 방화벽 승인 완료
엔드포인트 (문)	❌ 차단	PWC 정책/권한 제약으로 SSH 서비스 구동 불가

→ 길은 열렸지만, 문을 열 수 없음

구조적 한계 & 결론

구조적 한계

1. **PWC/팀 PC는 자동 배포 대상 서버 요건 미충족**
    - 상시 서비스/권한/운영 관리 불가
    - Windows PC 보안 정책 제약
2. **외부 공용 Runner → 사내 단말 직접 접근은 정책적으로 제약**
    - HCloud(10.11.x.x) → 사내망(10.90.x.x) 접근 제한

최종 결론

❌ **현 환경(PWC/팀 PC)에서는 사내망 CI/CD 자동 배포 불가**

대안 & 느낀점

향후 가능 방향 (대안)

정식 프로젝트화 필요

1. **GMDM 프로젝트 생성** (AI TFT 과제 등록)
2. **승인된 서버/VM 확보**
    - 운영팀 관리 하에 SSH/방화벽 설정
    - Linux 기반 서버 환경
3. **GitHub Self-hosted Runner 활용**
    - 사내망 내부에 Runner 구축
    - 표준 배포 구조로 진행

느낀점 (캐주얼)

💬 솔직한 후기

> 결론적으로 사내망에서 배포하기가 너무 힘들었습니다.

- 설정 문제보다 **정책/권한/보안 제약**이 더 큰 허들
- 다음에는 **정식 서버/프로젝트 선행이 필수**라고 느꼈습니다
- 다 막아놨으 ㅡㅅㅡ

교훈

✅ CI/CD 구성 자체는 완료했으나
✅ 배포 대상 환경 확보가 선행되어야 함
✅ 정식 프로젝트로 다시 시도 예정

주제 2

MY-TODOLIST 프로젝트

Vibe Coding으로 완성한 풀스택 웹 애플리케이션

Vibe Coding 교재 기반 실습 사례

1 Vibe Coding 개요

Vibe Coding이란?

AI 전문가 Andrej Karpathy가 제시한 개념

개발자가 생성형 AI의 도움을 받아 자연어 프롬프트를 통해 코드를 작성하는 프로그래밍 방식

- 개발자가 Project, Task를 LLM에 설명
- LLM이 프롬프트 기반으로 코드 생성
- 개발자는 결과를 평가하고 개선사항 요청
- **반복적인 실험에 집중**

Vibe Coding의 특징

1. 자연어 기반 개발

영어 혹은 모국어로 작성한 프롬프트로 원하는 결과 획득

2. AI 의존적 코딩

코드의 작동 원리를 완전히 이해하지 않아도 코드 작성 가능

3. 직감적 접근

엄밀한 논리나 설계보다 직감과 느낌에 의존

기존 코딩 vs Vibe Coding

구분	기존 코딩	Vibe Coding
개발자의 역할	설계자, 개발자, 디버거	프롬프트 작성, 가이드, 테스터
코드 생성	직접 개발자가 작성	AI가 코드 자동 생성
전문 지식	높은 프로그래밍 언어 지식 필요	상대적으로 낮은 기술적 지식
개발 속도	체계적이지만 느림	프로토타입 단계에서 빠름
디버깅	수동 디버깅 (코드 이해 필수)	AI 피드백 기반 반복적 자동 수정

Vibe Coding의 등장 배경

기술적 요인

- **LLM의 발전**: GPT, Claude, Gemini의 코드 생성 능력 비약적 발전
- **AI 코딩 도구**: Cursor AI, Windsurf, Claude Code 등장

사회적 요인

- **진입 장벽 완화**: 비개발자의 소프트웨어 개발 문턱 낮추기
- **빠른 프로토타이핑**: 스타트업과 개인 개발자의 구현 속도 향상 요구

VDD (Vibe-Driven Development)

전통적 SDLC와의 차이

전통적 SDLC

계획 → 설계 → 구현 → 테스트 → 배포

VDD

아이디어 → **프롬프트 기반 설계** → **프롬프트 기반 구현** → 테스트 → **개선 반복**

핵심: 빠른 피드백 루프와 점진적 개선

TDD vs VDT

TDD (Test-Driven Development)

- **Red → Green → Refactor** 사이클
- 테스트 코드를 먼저 작성
- 모든 요구사항을 안정적으로 검증

VDT (Vibe-Driven Testing)

- **Vibe → Test → Refine** 사이클
- 개발 흐름과 AI 피드백 활용
- 핵심 기능과 위험 지점 신속하게 검증

실제 프로젝트: VDT 방식으로 119개 테스트 작성 (커버리지 94~95%)

AI-DLC (AI-Driven Life Cycle)

AI를 SDLC의 '중심 동료'로 활용

모든 개발 단계에 AI를 일관되게 적용

- 요구사항 분석 → AI로 빠른 프로토타이핑
- 설계 → AI가 아키텍처 제안
- 구현 → 자연어 지시로 코드 생성
- 테스트 → AI가 테스트 케이스 자동 생성

목표: 속도와 품질을 동시에 향상 (수십배까지도)

CI/CD와 Vibe Coding

Vibe Coding은 CI/CD와 궁합이 매우 좋음

이유:

- ✅ 작은 단위의 변경이 많음
- ✅ 계속 바뀌는 요구사항
- ✅ AI 주도 테스트 작성
- ✅ 운영 피드백 반영 속도 향상

실제 적용: GitHub Actions + Vercel로 완전 자동화된 배포 파이프라인 구축

2 환경 설정

개발 환경 구성

필수 도구

- Node.js 18+
- PostgreSQL 17 (로컬 또는 Supabase)
- Git + GitHub
- VS Code 또는 Cursor AI
- Claude Code CLI

Claude Code 설정

설치

```
npm install -g @anthropic-ai/claude-code
```

MCP (Model Context Protocol) 설정

`.mcp.json` 파일로 필요한 서버 구성

```
{
  "mcpServers": {
    "postgresql-mcp": {
      "command": "npx",
      "args": ["-y", "@henkey/postgres-mcp-server"]
    }
  }
}
```

프로젝트 초기화

GitHub 리포지토리 생성

```
# 리포지토리 생성
gh repo create my-todolist --public

# 클론
git clone https://github.com/사용자명/my-todolist
cd my-todolist

# VS Code/Cursor로 열기
code .
```

3 프롬프트 엔지니어링

프롬프트의 중요성

명확하고 구체적인 프롬프트 = 정확한 결과

좋은 프롬프트의 조건:

- ✅ **명확한 컨텍스트** 제공
- ✅ **구체적인 요구사항** 전달
- ✅ **제약조건** 명시
- ✅ **예상 결과** 설명

프롬프트 전략: 계층적 컨텍스트

```
CLAUDE.md (전역 설정)
  └ 프로젝트 메모리 (MEMORY.md)
      └ 도메인 정의서
          └ PRD (요구사항)
              └ 설계 원칙
                  └ 실행 계획
```

각 계층이 다음 계층의 기반이 됨

CLAUDE.md 작성 예시

전역 지침

```
# 프로젝트 지침

## 코드 작성 원칙
- 간단한 코드는 주석 지정하지 않음
- 복잡한 코드일 때만 주석 부여

## 금지 사항
- 오버엔지니어링 금지
- 지침에 있는 기능만 정확하게 구현
```

하위 CLAUDE.md (백엔드)

```
# 백엔드 개발 지침

## 아키텍처 원칙
- SOLID 원칙 준수
- Clean 아키텍처 준수

## 기술 스택
- Node.js + Express + TypeScript
- PostgreSQL (pg 드라이버, Prisma 금지)
- JWT 인증

## 필수 사항
- 오버엔지니어링 금지
- 지침에 명시된 기능만 구현
```

4 Vibe Coding 도구

주요 도구

1. Claude Code CLI

터미널 기반 AI 코딩 어시스턴트

- 파일 읽기/쓰기, 코드 생성/수정
- Git 통합 (커밋, PR 생성)
- 서브에이전트 활용

2. MCP 서버

- PostgreSQL MCP
- Supabase MCP
- Playwright MCP

서브에이전트 (Subagent)

전문화된 AI 에이전트

역할별 분류:

- Plan Agent: 구현 계획 수립
- Explore Agent: 코드베이스 탐색
- Backend Developer: 백엔드 개발
- Frontend Developer: 프론트엔드 개발
- Test Runner: 테스트 실행

병렬 실행 가능

사용자 정의 Command

Custom Skill 작성

`.claude/commands/issue-resolver-backend.md`

```
# 백엔드 Issue 해결 Command

GitHub Issue 번호를 받아 해당 이슈를 해결

## 작업 내용
1. 이슈 확인 (gh CLI + docs 분석)
2. feature-${ISSUE_NUMBER} 브랜치 생성
3. 기존 코드 분석
4. 계획 수립
5. 테스트 작성
6. 문제 해결
7. 테스트 수행
```

5 프로젝트 기획 및 설계

도메인 정의서

특정 업무/제품이 다루는 문제 영역을 공통 언어로 정리

핵심 항목:

- 비즈니스 컨텍스트 (목적, 비전, 범위)
- 도메인 언어 (용어 사전)
- 도메인 모델 (엔티티, 값 객체, 애그리게이트)
- 비즈니스 규칙
- 업무 프로세스

도메인 정의서 작성 프롬프트

```
너는 최고의 비즈니스 도메인 전문가야.
"MY-TODOLIST" 할 일 관리 애플리케이션의
도메인 정의서 초안을 작성해줘.

# 문제 정의:
- 개인의 할 일이 흩어져 있고 관리가 어려움
- 마감일 관리 부족

# 주요 기능:
- 회원가입/로그인 (JWT)
- 할 일 CRUD
- 마감일 설정
- 상태별 필터

# 필수 사항:
- 핵심 내용만 포함
- docs/1-domain-definition.md로 저장
```

PRD (Product Requirements Document)

도메인 정의서 → PRD

도메인 정의서가 커버하는 요소:

- 제품 개요 및 목표
- 핵심 기능 요구사항
- 사용자 스토리/유스케이스

PRD에 추가되는 내용:

- 타겟 사용자 페르소나
- UI/UX 요구사항
- 기술 스택 선택 근거
- MVP 범위 정의

실행 계획 수립

단계별 Task 분할

M1: DB + 백엔드 API (1일)

- DB-01: PostgreSQL 초기화
- DB-02: DDL 적용
- BE-01~07: 백엔드 개발

M2: 프론트엔드 (2일)

- FE-01~07: 프론트엔드 개발

M3: 테스트 + 배포 (1일)

- QA-01~02: 품질 검증

GitHub Issue 생성

각 Task를 GitHub Issue로 관리

프롬프트 예시:

```
전문화된 서브에이전트를 이용해
@docs/7-execution-plan.md의 처리 단계를
유지하면서 GitHub 이슈를 생성해줘.

## 반드시 지켜야 할 것:
- Title: Stage 명시
- Label: 종류, 영역, 복잡도
- Todo: 해야 할 일 정보
- 완료 조건
- 기술적 고려사항
- 의존성 (선행/후행 작업)
```

6 백엔드 개발

PostgreSQL 설정

로컬 DB 구성

```
# PostgreSQL 17 설치
# macOS
brew install postgresql@17

# 데이터베이스 생성
createdb todolist_dev

# .env 파일 설정
DATABASE_URL=postgresql://postgres:password@localhost:5432/todolist_dev
```

PostgreSQL MCP 연결

.mcp.json 설정

```
{
  "mcpServers": {
    "postgresql-mcp": {
      "command": "npx",
      "args": ["-y", "@henkey/postgres-mcp-server"]
    }
  }
}
```

연결 테스트

```
/mcp
"postgresql-mcp를 이용해 로컬 데이터베이스에 연결되는지 테스트해줘"
```

스키마 적용

실행 계획 기반 진행

```
@docs/7-execution-plan.md의 1.2 스키마 적용 및 검증 단계를 진행해줘.
```

AI가 자동으로:

- `database/schema.sql` 실행
- `member`, `todo` 테이블 생성
- 인덱스 생성
- 제약조건 검증

백엔드 개발 태스크

BE-01 ~ BE-07

BE-01: 프로젝트 초기화 (Express + TypeScript)

BE-02: DB 연결 (pg Pool)

BE-03: Member Repository

BE-04: 인증 API (signup, login)

BE-05: JWT 미들웨어

BE-06: Todo Repository

BE-07: Todo API (CRUD + 상태 전이)

Custom Command로 Issue 해결

/issue-resolver-backend

```
/issue-resolver-backend 23
```

자동으로 수행되는 작업:

1. GitHub Issue #23 내용 확인
2. `feature-23` 브랜치 생성
3. 기존 코드 분석
4. 계획 수립
5. 테스트 작성 (커버리지 80%+)
6. 문제 해결
7. 테스트 수행

백엔드 개발 주의사항

로깅의 중요성

충실하게 로깅하도록 개발

- 개발 환경: 상세 로깅
- 프로덕션: 에러만 로깅

Logger 함수 작성 권장

```
const logger = {
  info: (msg: string) =>
    process.env.NODE_ENV === 'development' && console.log(msg),
  error: (msg: string) => console.error(msg)
};
```

디버깅 시 로그를 AI에게 제공하면 빠른 해결

완성된 백엔드 API

Swagger UI

총 35개 엔드포인트

- ✅ 회원 인증 (signup, login, refresh, logout)
- ✅ 할 일 CRUD
- ✅ 상태 전이 (complete, revert)
- ✅ JWT 미들웨어
- ✅ 에러 핸들링

7 프론트엔드 개발

스타일 가이드 생성

화면 캡처 → AI로 스타일 추출

프롬프트:

```
(이미지 첨부: Alt+V)

"이 화면 이미지 정보를 참조하여
tailwind를 적용할 때의 스타일 가이드를 생성하여
@docs/APP_STYLE_GUIDE.md 파일로 저장해줘"
```

결과: 색상, 폰트, 컴포넌트 스타일 자동 추출

프론트엔드 개발 준비

Custom Command 생성

`.claude/commands/issue-resolver-frontend.md`

백엔드와 유사하지만 추가 참조:

- `docs/8-wireframes.md`
- `swagger/swagger.json`
- `docs/APP_STYLE_GUIDE.md`

프론트엔드 개발 태스크

FE-01 ~ FE-07

FE-01: 프로젝트 초기화 (Vite + React 19)
FE-02: API Service 레이어
FE-03: 인증 페이지
FE-04: 할 일 목록
FE-05: 생성/수정 페이지
FE-06: Overdue 시각화
FE-07: 반응형 UI

Custom Command로 개발

```
/issue-resolver-frontend 45
```

자동 진행:

1. Issue 확인 + 문서 분석
2. `feature-45` 브랜치 생성
3. 코드베이스 분석 (병렬)
4. 계획 수립
5. 테스트 작성 (병렬)
6. 문제 해결
7. 테스트 수행

프론트엔드 개발 주의사항

환경변수 사용

```
// ❌ 나쁜 예
const API_URL = 'http://localhost:3000';

// ✅ 좋은 예
const API_URL = import.meta.env.VITE_API_URL;
```

기술 스택 명시

frontend/CLAUDE.md에 명시:

- API 호출: axios? fetch? tanstack query?
- 상태 관리: React hooks
- 스타일: Tailwind CSS

Chrome DevTools MCP 활용

브라우저 디버깅 자동화

```
"chrome-devtools mcp를 사용해
네트워크 탭에서 실패한 요청들을 확인하고
상태 코드와 함께 정리해줘"
```

```
"chrome-devtools mcp를 사용해
폼 제출 시 유효성 검사가 제대로 작동하는지 확인해줘"
```

8 배포

Supabase Database 설정

프로덕션 DB 준비

1. **Supabase 프로젝트 생성**
    - Project Name: my-todolist
    - Region: Northeast Asia (Seoul)
2. **연결 문자열 확인**
    - Transaction Pooler 사용
    - `postgresql://postgres.xxx:[PASSWORD]@aws-1-ap-northeast-2.pooler.supabase.com:6543/postgres`

로컬 → Supabase 마이그레이션

.mcp.deploy.json 설정

```
{
  "mcpServers": {
    "supabase": {
      "type": "http",
      "url": "https://mcp.supabase.com/mcp"
    },
    "postgresql-mcp": {
      "command": "npx",
      "args": ["-y", "@henkey/postgres-mcp-server"]
    }
  }
}
```

마이그레이션 실행

```
claude --mcp-config .mcp.deploy.json
```

프롬프트:

```
"postgresql-mcp와 supabase mcp를 이용해
로컬 데이터베이스의 스키마를
supabase로 마이그레이션 해줘"
```

결과: 테이블, 인덱스, 제약조건 자동 마이그레이션

Vercel 배포

Vercel for GitHub

특징:

- GitHub 연동 자동 배포
- main 브랜치 → Production
- 다른 브랜치 → Preview
- HTTPS 자동 적용
- CI/CD 파이프라인 불필요

백엔드 배포

Vercel 프로젝트 생성

1. Add New → Project
2. Import Git Repository
    - 프로젝트명: my-todolist-backend
    - Root Directory: `backend`
3. **환경변수 설정**
    - `DATABASE_URL` (Supabase 연결 문자열)
    - `JWT_SECRET`
    - `JWT_REFRESH_SECRET`
    - `CORS_ORIGIN` (프론트엔드 URL)

백엔드 배포 주의사항

Serverless 환경 제약

❌ 파일 시스템 쓰기 제한

- `/tmp` 만 쓰기 가능
- 로그를 파일에 저장하면 배포 실패

✅ 해결책: Console.log 사용

- Vercel 대시보드 Logs에서 확인 가능

프론트엔드 배포

Vercel 프로젝트 생성

1. Add New → Project
2. Import Git Repository
    - 프로젝트명: my-todolist-frontend
    - Root Directory: `frontend`
3. **환경변수 설정**
    - `VITE_API_URL` (백엔드 API URL)

배포 완료

서비스 URL

🌐 **프론트엔드:** https://my-todolist-app.vercel.app
🔧 **백엔드 API:** https://my-todolist-api.vercel.app
📚 **API 문서:** https://my-todolist-api.vercel.app/docs

CI/CD 파이프라인

GitHub Actions + Vercel

```
Git Push → GitHub
    ↓
Vercel Auto Deploy
    ↓
Build & Test
    ↓
Production/Preview Deploy
    ↓
Health Check
```

완전 자동화된 배포

9 실제 개발 성과

개발 기간 및 성과

총 개발 기간: 4일

단계	기간	성과
M1: 백엔드	1일	35개 엔드포인트, 35개 테스트 (커버리지 94%)
M2: 프론트엔드	2일	7개 페이지, 84개 테스트 (커버리지 95%)
M3: 배포	1일	Vercel + Supabase 프로덕션 배포

총 119개 자동화 테스트

성능 지표

지표	목표	실측값	결과
API 응답 시간	500ms 이하	평균 40ms	✅
할 일 목록 조회	1초 이하	평균 3.6ms	✅
테스트 커버리지	90% 이상	BE 94%, FE 95%	✅
자동화 테스트	100% 통과	119/119 통과	✅

모든 목표 달성! 🎯

구현된 기능

✅ 회원 인증

- 회원가입 (이메일 중복 검사, 비밀번호 정책)
- 로그인 (JWT Access Token + Refresh Token)
- 토큰 갱신 (Token Rotation)
- 로그아웃

✅ 할 일 관리

- CRUD (생성, 조회, 수정, 삭제)
- 상태 전이 (PENDING ↔️ DONE)
- 마감일 설정 (날짜 + 시간)
- 마감 초과 알림

구현된 기능 (계속)

✅ 사용자 경험

- 상태별 필터 (전체/진행중/완료)
- 다크모드 / 라이트모드 / 시스템 테마
- 다국어 지원 (한국어/영어/일본어)
- 반응형 UI (모바일 360px+)

✅ 품질 보증

- 119개 자동화 테스트
- 커버리지 94~95%
- CI/CD 파이프라인

🔟 회고 및 배운 점

Vibe Coding의 장점

1. 개발 속도 향상 🚀

- **4일** 만에 풀스택 애플리케이션 완성
- 보일러플레이트 자동 생성
- 테스트 코드 자동화

2. 품질 향상 ✨

- **119개** 자동화 테스트
- 커버리지 **94~95%**
- TypeScript strict mode

Vibe Coding의 장점 (계속)

3. 학습 효과 📚

- 최신 기술 스택 빠르게 습득
- 베스트 프랙티스 적용
- 아키텍처 패턴 이해

4. VDD 실천

- 빠른 피드백 루프
- 점진적 개선
- AI-DLC 적용

주요 도전 과제

1. 프롬프트 엔지니어링

- 명확한 요구사항 정의 필요
- 컨텍스트 관리의 중요성
- 단계별 검증 필요

2. AI 생성 코드 검토

- 보안 취약점 확인 필수
- 비즈니스 로직 검증
- 오버엔지니어링 방지

주요 도전 과제 (계속)

3. 통합 과정

- 프론트엔드-백엔드 API 연동
- CORS 정책 관리
- 환경변수 설정

4. 배포 환경

- Serverless 제약 사항
- 로깅 전략 변경
- 환경별 설정 관리

Vibe Coding 권장 사항

1. 📐 명확한 설계 먼저

- 도메인 정의서 작성
- PRD (요구사항 문서)
- 실행 계획 수립
- 아키텍처 다이어그램

2. 🔄 단계별 진행

- 작은 태스크로 분할
- GitHub Issue 관리
- 각 단계 검증

Vibe Coding 권장 사항 (계속)

3. 🧪 VDT 적용

- **Vibe → Test → Refine** 사이클
- 커버리지 80% 이상 유지
- E2E 시나리오 작성

4. 📚 문서화

- 프로젝트 메모리 (MEMORY.md)
- CLAUDE.md (계층적 지침)
- API 문서화 (Swagger)
- README 작성

Vibe Coding 권장 사항 (계속)

5. 🛠️ 도구 활용

- **서브에이전트** 적극 활용
- Custom Command 작성
- MCP 서버 연동
- CI/CD 자동화

6. 🔍 검토 필수

- AI 생성 코드 검토
- 오버엔지니어링 방지
- 보안 취약점 확인

교재의 핵심 교훈

VDD (Vibe-Driven Development)

빠른 피드백 루프와 점진적 개선

AI-DLC (AI-Driven Life Cycle)

모든 개발 단계에 AI를 일관되게 활용

VDT (Vibe-Driven Testing)

개발 흐름과 AI 피드백 기반 테스트

CI/CD 통합

Vibe Coding과 자동화의 시너지

핵심 메시지

Vibe Coding은...

✨ **개발 속도**를 높이고
✨ **코드 품질**을 향상시키며
✨ **학습 효과**를 극대화합니다

> "어떻게"보다 "무엇을"에 집중하면,
> 더 나은 제품을 만들 수 있습니다.

단, 명확한 설계와 검토는 필수입니다!

감사합니다! 🙏

질문이 있으시면 편하게 해주세요.

참고 자료

📚 교재

- 01_Vibe Coding 개요
- 02_환경설정
- 03_Vibe Coding을 위한 프롬프트 엔지니어링
- 04_Vibe Coding을 위한 도구
- 05_프로젝트 기획 및 설계
- 06_백엔드 개발
- 07_프론트엔드 개발
- 08_배포

참고 자료 (계속)

💻 프로젝트

- **GitHub:** github.com/sehee-jeong/my-todolist
- **프론트엔드:** https://my-todolist-app.vercel.app
- **백엔드:** https://my-todolist-api.vercel.app
- **API 문서:** https://my-todolist-api.vercel.app/docs

🛠️ 도구

- **Claude Code:** claude.ai/code
- **Vercel:** vercel.com
- **Supabase:** supabase.com