W04
지침과 개발 원칙

원하는 대로 에이전트를 동작시키려면

Cursor가 슬슬 내 말을 안듣기 시작한다.

- 하나씩 코드를 작성할 때마다 잘 작성했는지 확인하다 보면
- 이전에 분명히 어떻게 하라고 말을 했는데 자꾸 자기 맘대로 하는 경향이 생김
- 매번 프롬프트를 넣을때 마다 어떤 것을 고려하라고 말하지만 계속 까먹고 맘대로 한다

왜 이런 일이 생기는가?

- AI는 대화가 길어지면 처음 했던 말을 잊어버린다
- 새 대화를 시작하면 이전 대화 내용을 전혀 모른다
- 매번 같은 말을 반복해야 하는 건 비효율적이다

"AI에게 매번 같은 말을 반복해야하면 너무 시간 낭비 아닐까?"

LLM은 그저 답만 하는 기계일 뿐

LLM은 '인공 지능'일 뿐 정말 '지능'이 아니다

- 질문이 들어오면 답을 생성한다 - 그게 전부다
- 이전에 무슨 대화를 했는지 '기억'하지 않는다
- 당신의 프로젝트가 뭔지, 왜 만드는지 '이해'하지 않는다
- 매 질문마다 "처음 보는 사람"처럼 응답한다

"AI에게 '기억'은 없다 - 있는 건 '지금 보이는 정보'뿐"

LLM은 당신이 무엇을 알고있는지 모른다

당신의 머릿속은 AI에게 보이지 않는다

- 당신은 이 서비스를 왜 만드는지 안다
- 당신은 어떤 사용자를 위한 건지 안다
- 당신은 지난주에 어떤 결정을 했는지 안다

AI가 아는 것

- 지금 이 대화창에 적힌 내용
- 지금 열려 있는 파일 (Cursor의 경우)

결과적으로

- 당신이 "당연히 알겠지"라고 생각한 것을 AI는 모른다
- AI가 엉뚱한 답을 하면? 대부분 맥락을 모르기 때문이다

이 맥락을 유지하려면 어떻게 해야할까?

- AI는 기억이 없다
- 매번 같은 맥락을 전달하는 건 비효율적이다
- 대화가 길어지면 처음 말한 내용을 잊는다

해결 방향

- 반복해서 전달할 내용을 '파일'에 적어둔다
- AI가 매번 그 파일을 자동으로 읽게 한다
- 한 번 적어두면 모든 대화에서 참조된다

두가지 방법

1. agents.md: 이 프로젝트에만 적용되는 지침
2. User Rules: 모든 프로젝트에 적용되는 나의 원칙

첫번째 방법: agents.md

- https://agents.md/
- 프로젝트 루트에 AGENTS.md 파일이 있으면 Cursor가 자동으로 읽음(별도 설정 불필요)
- Agent 모드로 대화할 때, AGENTS.md 내용이 프롬프트 앞에 자동 삽입됨
- AI가 "이 프로젝트는 이런 맥락이구나" 알고 시작

실제 흐름

1. Agent 모드에서 질문
2. Cursor가 AGENTS.md 읽음
3. AI에게 전달하여 기존 질문에 삽입
4. 응답 생성

중첩 지원

"프로젝트가 커지면, 지침도 나눌 수 있다"

- 프로젝트 루트의 AGENTS.md → 전체 프로젝트에 적용
- 하위 폴더의 AGENTS.md → 해당 폴더 작업 시 추가 적용

예시 구조 my-project/

- AGENTS.md ← 전체 프로젝트 공통 지침
- frontend/
  - AGENTS.md ← 프론트엔드 작업 시 추가 지침
- backend/
  - AGENTS.md ← 백엔드 작업 시 추가 지침

언제 유용한가?

- 프로젝트가 커져서 영역별로 다른 규칙이 필요할 때
- 프론트엔드와 백엔드가 다른 기술 스택을 쓸 때
- 처음에는 루트에 하나만 만들어도 충분하다

코딩 에이전트별 파일 생성

각 코딩 에이전트들은 공통 프롬프트를 생성해 주는 기능을 제공한다.

- Claude Code: CLAUDE.md
- GPT Codex: AGENTS.md
- Gemini CLI: GEMINI.md
- 각 에이전트들은 /init 명령을 통해 이 프로젝트를 스스로 분석하여 초기 파일을 생성
- 단, 이 파일은 에이전트가 자동 생성한 파일인 만큼 자기가 넣고 싶은 대로 넣기 때문에 반드시 필요한 내용인지 확인해봐야 한다

(Cursor에서 AGENTS.md와 CLAUDE.md를 모두 참조하게 하는 기능을 제공)

무엇이 포함되어야 하나

agents.md에 보통 포함되는 내용

- 프로젝트 개요
- 핵심 문서 위치
- 기술 스택
- 작업 규칙
  - 코드 작성 시 지켜야 할 절차
  - 테스트 방법
  - 파일/폴더 구조 규칙
- 주의사항
  - 하지 말아야 할 것
  - 자주 발생하는 실수 방지

만들어놓은 문서를 연결할 수 있다

- 기존에 만들어 놓은 문서의 위치를 알려주면 항상 그 문서를 참고한다.
- 바이브 코딩의 핵심 문서
  - 서비스 개요
  - PRD
  - 서비스 구조(시스템, 유저 흐름, 데이터)
- 프로젝트 로드맵 문서
  - 마일스톤
  - 워크 패키지
  - 시나리오
- 이 문서를 어떻게 이해하고 사용해야 하는지 적어놓으면 매번 참고한다.

바이브 코딩 절차를 따르도록 한다

완전한 개발을 위해 지금까지 배워온 모든 절차들도 포함시킬 수 있다

- 매번 코드를 작성할 때 마다 이 서비스를 왜 만드는지에 대해 생각해 보고 작성해
- 작성이 끝나면 해당 코드가 의도한 대로 동작하는지 반드시 테스트를 실행해
- 코드 작성 전에 해당 시나리오의 목적을 먼저 확인해
- 한 번에 너무 많은 코드를 작성하지 말고 작은 단위로 나눠서 작업해
- 기존 코드를 수정할 때는 먼저 현재 동작을 확인하고 수정해
- 에러가 발생하면 원인을 분석하고 설명한 뒤에 수정해

"일하는 방식도 적어두면, AI가 그 방식을 따른다"

- AI가 "어떻게 일해야 하는지"를 매번 기억하게 한다
- 우리가 직접 감독하지 않아도 절차를 따르게 한다

두번째 방법: User Rules

- Cursor 전체에 적용되는 나의 개인 설정
- 어떤 프로젝트를 열어도 항상 적용된다
- 프로젝트와 무관하게 내가 원하는 코딩 방식을 지정
- 우측 상단 설정 버튼 → Rules and Commands → User Rules

agents.md와의 차이

- agents.md: 이 프로젝트에서만 적용 (프로젝트별 맥락)
- User Rules: 모든 프로젝트에 적용 (나의 코딩 원칙)

User Rules에 넣기 좋은 것

- 코드 품질 원칙 (DRY, KISS, YAGNI 등)
- 선호하는 코딩 스타일
- 항상 지켜야 할 규칙

좋은 코드를 위한 원칙

개발자들이 오랫동안 정립한 원칙들이 있다

- AI에게 "좋은 코드 작성해"라고 하면 모호하다
- 구체적인 원칙을 말해주면 AI가 따른다

이 원칙들을 User Rules에 넣으면

- 어떤 프로젝트에서든 AI가 같은 기준으로 코드를 작성한다
- 매번 "중복 코드 피해", "단순하게 작성해"라고 말할 필요 없다
- 한 번 설정하면 모든 작업에 적용된다

DRY

Don't Repeat Yourself - "반복하지 마라"

- 같은 코드를 여러 곳에 복사-붙여넣기 하지 말라
- 반복되는 로직은 하나로 만들어서 재사용하라

왜 중요한가?

- 같은 코드가 10군데 있으면, 수정할 때 10군데를 다 고쳐야 한다
- 하나라도 빠뜨리면 버그가 생긴다
- AI도 이 원칙을 모르면 복사-붙여넣기를 남발한다

User Rules 예시: "코드 작성 시 DRY 원칙을 따라 중복을 최소화해"

KISS

Keep It Short and Simple - "짧고 단순하게 유지하라"

- 복잡하게 만들 수 있어도, 단순하게 만들어라
- 나중에 읽고 이해하기 쉬운 코드가 좋은 코드다

왜 중요한가?

- AI는 때때로 불필요하게 복잡한 코드를 작성한다
- "더 좋아 보이는" 코드가 실제로는 유지보수하기 어렵다
- 단순한 코드가 버그도 적다

User Rules 예시: "KISS 원칙을 따라 가장 단순한 방법으로 구현해"

YAGNI

You Ain't Gonna Need It - "그거 필요 없을 거야"

- 지금 당장 필요하지 않은 기능은 만들지 마라
- "나중에 필요할 것 같아서"는 이유가 안 된다

왜 중요한가?

- AI는 종종 요청하지 않은 기능까지 추가한다
- "확장성을 위해", "나중을 대비해" 복잡해진다
- 실제로는 그 기능을 안 쓰는 경우가 대부분이다

User Rules 예시: "YAGNI 원칙을 따라 현재 요구사항에 필요한 것만 구현해"

SOLID

좋은 설계를 위한 5가지 원칙

- S - Single Responsibility (단일 책임): 하나의 코드는 하나의 일만 한다
- O - Open/Closed (개방/폐쇄): 새 기능 추가는 쉽게, 기존 코드 수정은 최소화
- L - Liskov Substitution (리스코프 치환): 기본 규칙을 따르면 어디서든 동작한다
- I - Interface Segregation (인터페이스 분리): 필요한 기능만 사용할 수 있게 나눈다
- D - Dependency Inversion (의존성 역전): 세부 구현에 의존하지 않고 추상화에 의존한다

각 원칙의 의미를 전부 이해할 필요는 없다 but AI는 이 용어들을 알고 있다

User Rules 예시: "SOLID 원칙을 따라 코드를 작성해"

질문 1. agents.md 파일 생성

"지금 진행되는 프로젝트에 공통 규칙 역할을 할 agents.md 파일을 생성하고, 서비스를 개발하는데 참고하는 서비스 개요, PRD, 시스템 구조, 유저 흐름 구조, 데이터 구조 등 여러 파일들을 쉽게 참조할 수 있도록 내용을 작성해줘"

생성 후 확인할 것

- 프로젝트 루트에 AGENTS.md 파일이 생성되었는가?
- 기존에 만든 문서들의 경로가 정확한가?
- 새 대화를 열고 질문했을 때 AI가 이 내용을 참조하는가?

질문 2. 바이브 코딩 프로세스 추가

"agents.md에 다음 프로세스가 필요한지 검토하고, 필요하다면 추가해줘. 기존 내용은 유지하면서 '작업 규칙' 섹션을 추가해줘"

포함할 내용:

- 코드 작성 전에 해당 단계의 목적을 먼저 확인할 것
- 작성 후에는 반드시 테스트를 실행하여 검증할 것
- 한 번에 너무 많은 코드를 작성하지 말고 작은 단위로 작업할 것
- 에러가 발생하면 원인을 분석하고 설명한 뒤에 수정할 것
- 등등등

실습 - User Rules 추가

User Rules에 다음 개발 원칙들을 추가:

- DRY 원칙을 따라 중복 코드를 피할 것
- KISS 원칙으로 가장 단순한 방법을 선택할 것
- YAGNI 원칙에 따라 현재 필요한 기능만 구현할 것
- SOLID 원칙을 따라 코드를 작성할 것

그럼에도 불구하고

이렇게 프롬프트를 열심히 넣어줘도 AI는 자기 멋대로 동작하는것이 기본

- AI는 지침을 "참고"할 뿐 "강제"되지 않는다
- 컨텍스트가 길어지면 지침의 일부를 놓칠 수 있다
- 같은 질문에도 매번 조금씩 다른 답을 한다

그래서 우리가 해야 할 것

- 지침을 만들어도 결과를 직접 확인해야 한다
- AI의 출력을 그대로 믿지 말고 검증해야 한다
- 문제가 생기면 지침을 더 구체적으로 수정해야 한다

그래도 지침이 있는 게 낫다

- 없을 때보다 일관성이 훨씬 높아진다
- 매번 같은 말을 반복하는 시간을 줄인다
- 문제가 생겼을 때 "지침을 어떻게 고칠까"로 접근할 수 있다

오늘 해결해야 할 문제

1. 프로젝트에 agents.md 파일을 생성하고, 기존에 만든 서비스 개요, PRD, 시스템 구조 문서를 참조하도록 작성합니다.
2. agents.md에 바이브 코딩 프로세스(작업 전 목적 확인, 작업 후 테스트 등)를 추가합니다.
3. Cursor의 User Rules에 개발 원칙(DRY, KISS, YAGNI, SOLID)을 설정합니다.
4. 설정 후 로드맵의 다음 시나리오를 진행하며, AI가 지침을 잘 따르는지 확인합니다.
5. (선택) 추가로 agents.md와 user rules를 보완할 수 있는 방안을 찾아보고 도움이 될 방안을 검토해 적용해 봅니다.

Q&A