Cursor 기능 사용법 : Context

1.1 Codebase Indexing

Codebase Indexing

AI가 프로젝트의 전체 코드를 **빠르게 이해하고 검색/참조**할 수 있도록 미리 분석하는 과정이다

1. Codebase Indexing의 개념

- **코드베이스 전체를 AI가 읽고 "색인(Index)"을 만드는 것**을 말한다.
- 단순히 파일 하나씩 보는 것이 아니라, 프로젝트 전체의 **파일 구조, 함수/클래스 정의, 주석, 의존성 관계**까지 분석한다.
- 인덱스를 만들어 두면 AI가 질의응답할 때 매번 전체 코드를 처음부터 읽을 필요 없이, 관련된 부분만 빠르게 찾아서 답변할 수 있다.

1.1 Codebase Indexing

2. 인덱싱 과정

1. **파일 수집**
   - Project 안의 모든 소스코드(.py, .js, .java, .cpp 등), 설정 파일, 문서 등을 스캔한다.
2. **파싱(Parsing)**
   - 코드 문법을 이해 가능한 단위로 쪼갠다. (함수, 클래스, 모듈 단위)
3. **임베딩(Embedding)**
   - 코드 조각을 벡터화(수학적 좌표로 변환)하여, 의미 기반 검색이 가능하도록 한다.
   - "이 함수가 무슨 역할인지" 같은 질문에도 대응 가능.
4. **저장(Indexing)**
   - 벡터 DB 또는 내부 인덱스 저장소에 보관한다.

1.1 Codebase Indexing

3. Codebase Indexing의 효과

- **빠른 검색**

  → "이 프로젝트에서 로그인 처리 부분 찾아줘" 하면 바로 관련 코드 보여줌.

- **맥락 이해**

  → 함수 호출 관계, 모듈 간 의존성을 AI가 파악하므로, "이 함수 수정하면 어디 영향?" 같은 질문에 답변 가능.

- **자동 문서화**

  → 인덱싱된 정보로 API 문서, 다이어그램, 요약본 등을 AI가 자동으로 생성할 수 있음.

- **Refactoring 지원**

  → 전체 구조를 이해하고 있기 때문에, 함수 분리·변경·최적화 제안이 정확해짐.

1.1 Codebase Indexing

4. Cursor AI에서의 활용

- **Chat with Codebase**: 인덱싱된 프로젝트 전체를 대상으로 AI와 대화할 수 있다.
- **폴더 단위 채팅**: 특정 폴더만 인덱싱 대상으로 삼아 대화 가능.
- **자동 업데이트**: 코드가 변경되면 인덱스도 자동으로 갱신된다.

✅ 정리

Codebase Indexing은 프로젝트 전체 코드를 AI가 빠르게 이해할 수 있게 색인화하는 과정이다. 이를 통해 **검색 속도 향상, 맥락 기반 답변, 문서 자동화, 리팩토링 지원**이 가능하다.

Codebase Indexing 실습 예제

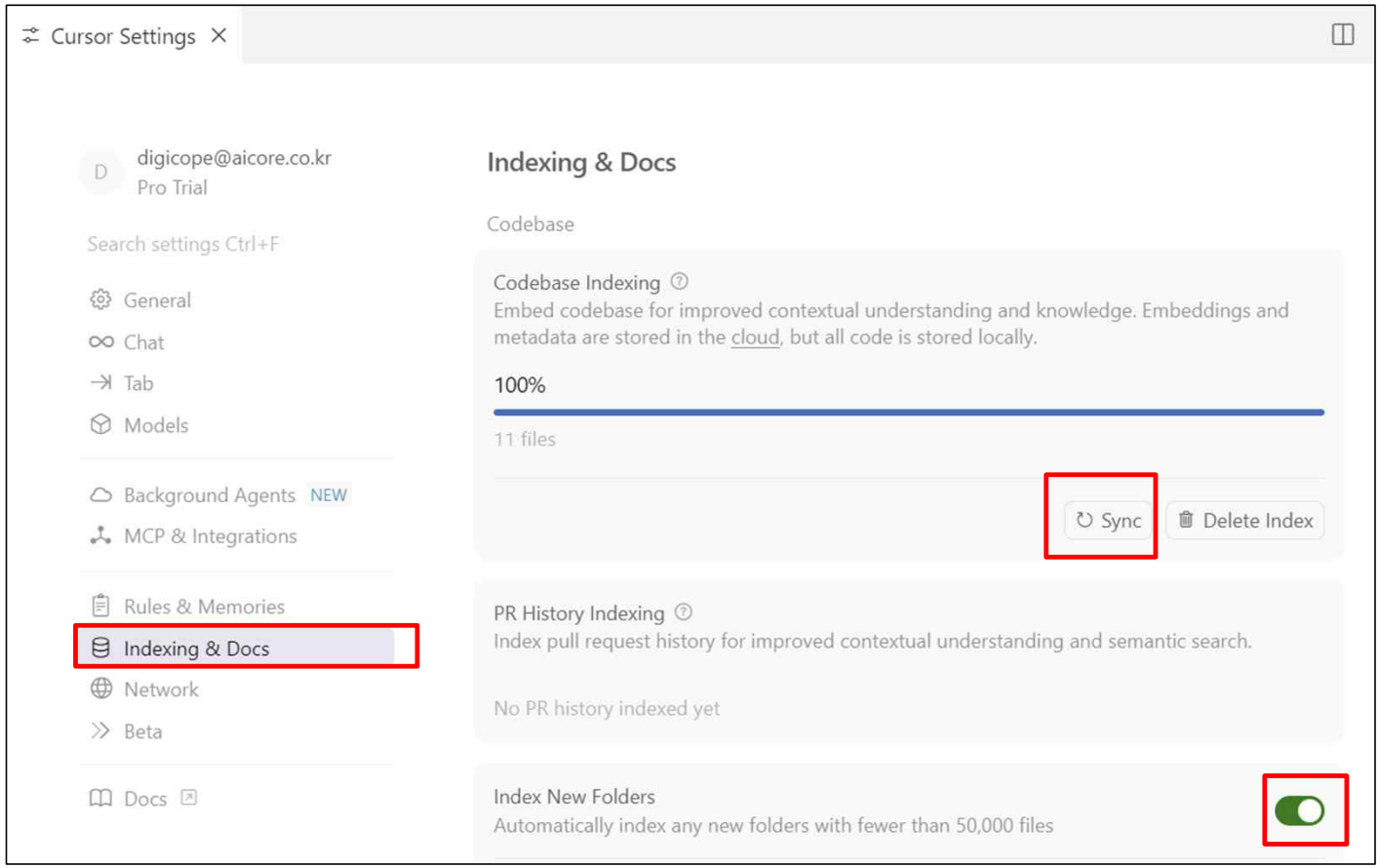
<실습 배포 예제 디렉토리 구조 : **계산기 + 유틸리티** 프로젝트>

my_indexing_project/

```
|
├── main.py
├── calculator/
|      ├──  __init__.py
|      ├──  add.py
|      ├──  subtract.py
|      ├──  multiply.py
|      └──  divide.py
|
├── utils/
|      ├──  __init__.py
|      ├──  logger.py
|      └──  validator.py
|
└── tests/
       ├──  test_calculator.py
       └──  test_utils.py
```

1.1 Codebase Indexing

Cursor Settings에 가보면 기본적으로 코드베이스의 모든 파일을 인덱싱하도록 설정 되어 있다. [Sync]버튼을 누르면 추가 수정 변경 소스를 인덱스에 추가 반영한다



1.1 Codebase Indexing

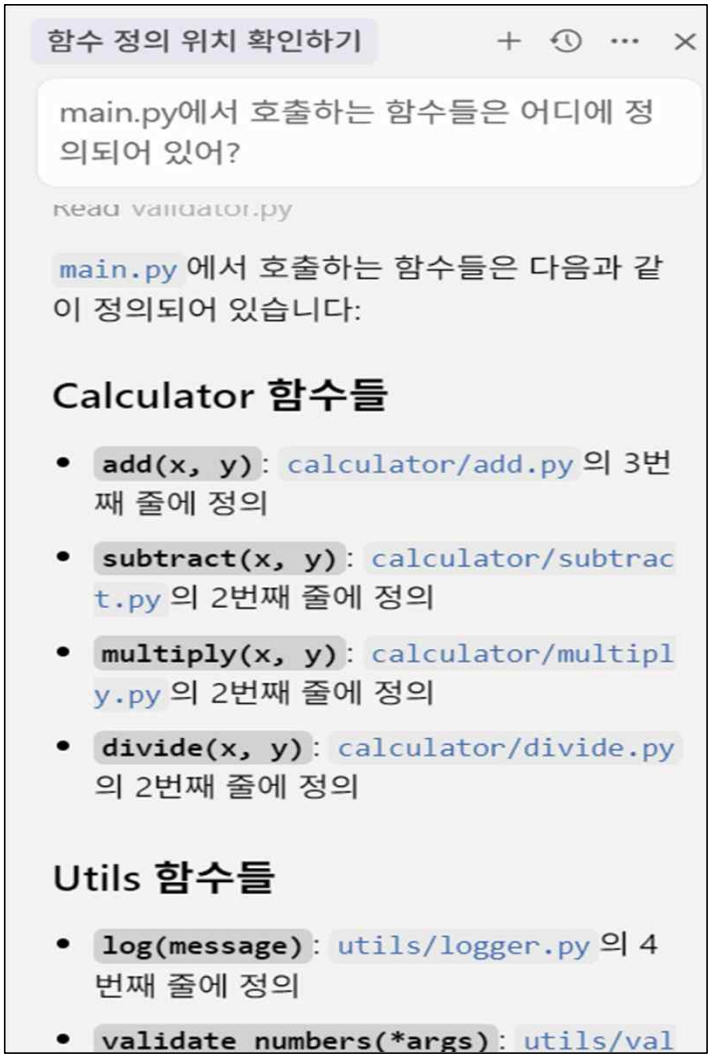
기본 검색 (indexing 검색 질의)

이 프로젝트에서 곱셈 기능이 구현된 부분 찾아줘

1.1 Codebase Indexing

의존성 추적 (indexing 검색 질의)

main.py에서 호출하는 함수들은 어디에 정의되어 있어?



1.1 Codebase Indexing

리팩토링 (indexing 검색 질의)

calculator 모듈의 각 연산을 하나의 Calculator 클래스로 합쳐줘

문서화 (indexing 검색 질의)

이 프로젝트 전체 API 문서 요약본 만들어줘

테스트 코드 생성 (indexing 검색 질의)

이 프로젝트 전체 API 문서 요약본 만들어줘

Cursor AI가 단순히 한 파일만 보는 게 아니라 **전체 프로젝트를 색인(indexing)** 해서, **검색 / 참조 / 리팩토링 / 문서화 / 테스트 생성**까지 지원한다는 걸 체험할 수 있다.

1.1 Codebase Indexing

구성

Cursor는 **ignore files** (예: `.gitignore`, `.cursorignore`)에 포함된 파일을 제외하고 모든 파일을 인덱싱해.

`Show Settings`를 클릭해서:

- 새 저장소 자동 인덱싱 사용
- 무시할 파일 설정

> **대용량 콘텐츠 파일을 무시**하면 답변 정확도가 올라가.

인덱싱된 파일 보기

인덱싱된 파일 경로를 보려면: `Cursor Settings` > `Indexing & Docs` > `View included files`

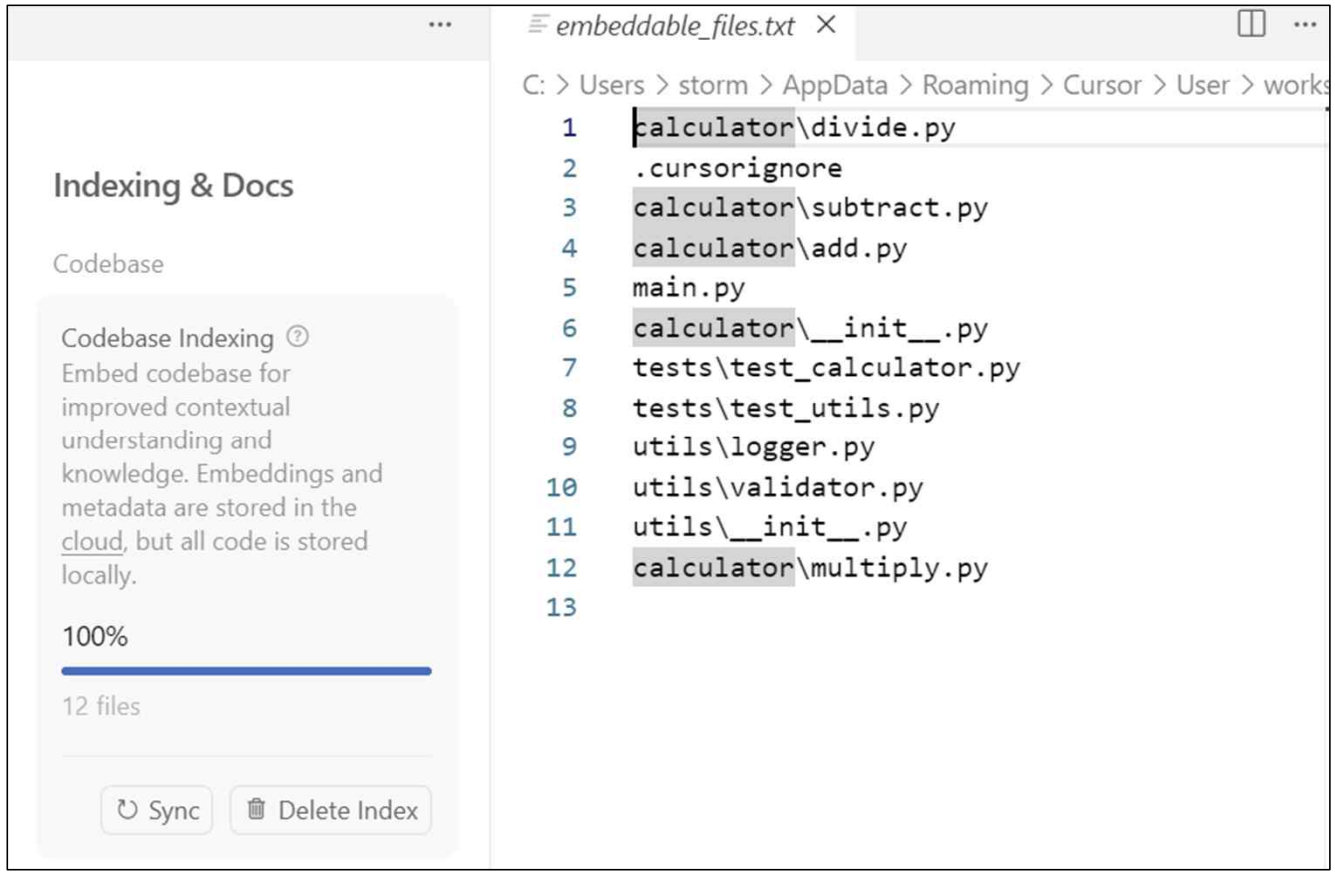
그러면 인덱싱된 모든 파일이 나열된 `.txt` 파일이 열려.

https://docs.cursor.com/ko/context/ignore-files

1.1 Codebase Indexing

Ignore Files in .cursorignore
Files to exclude from indexing in addition to .gitignore. View included files.

Edit



1.1 Codebase Indexing

Ignore files

.cursorignore와 .cursorindexingignore로 파일 접근 제어

개요

Cursor는 프로젝트의 코드베이스를 읽고 인덱싱해 기능을 제공해. 루트 디렉터리의 `.cursorignore` 파일로 Cursor가 접근할 수 있는 디렉터리와 파일을 제어해.

Cursor는 `.cursorignore` 에 포함된 파일에 대한 접근을 다음에서 차단해:

- 코드베이스 인덱싱
- **Tab**, **Agent**, **Inline Edit**에서 접근 가능한 코드
- **@ 기호 참조**를 통한 코드 접근

⚠ Agent가 실행하는 도구 호출(예: 터미널, MCP 서버)은 `.cursorignore` 적용 대상 코드에 대한 접근을 차단할 수 없어

https://docs.cursor.com/en/context/ignore-files

1.1 Codebase Indexing

왜 파일을 무시할까?

보안: API 키, 자격 증명, 시크릿에 대한 접근을 제한해. Cursor가 무시된 파일을 차단하더라도 LLM의 예측 불가능성 때문에 완전한 보호가 보장되진 않아.

성능: 대규모 코드베이스나 모노레포에선 인덱싱을 더 빠르게 하고 파일 탐색 정확도를 높이기 위해 관련 없는 부분을 제외해.

1.1 Codebase Indexing

Default patterns include:

- Environment files: `**/.env`, `**/.env.*`
- Credentials: `**/credentials.json`, `**/secrets.json`
- Keys: `**/*.key`, `**/*.pem`, `**/id_rsa`

Configuring `.cursorignore`

Create a `.cursorignore` file in your root directory using `.gitignore` syntax.

Pattern examples

```
config.json      # Specific file
dist/            # Directory
*.log            # File extension
**/logs          # Nested directories
!app/            # Exclude from ignore (negate)
```

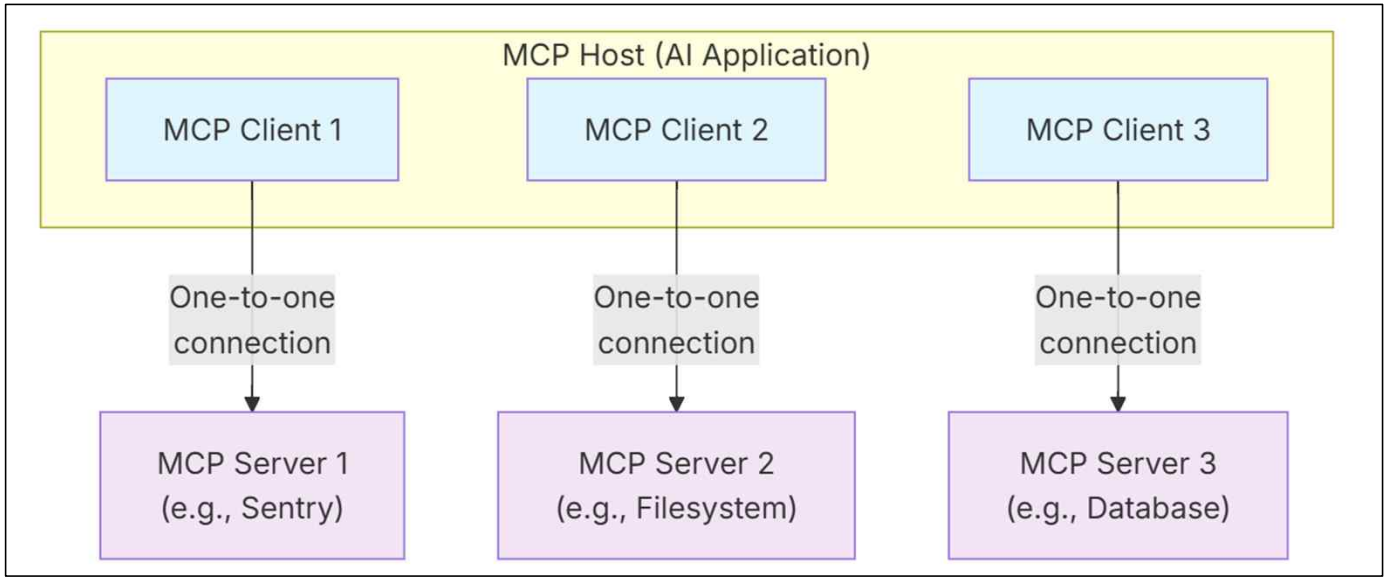
1.2 Model Context Protocol (MCP)

1.2 Model Context Protocol (MCP)

MCP(Model Context Protocol) 개요

1. 정의

- **MCP는 AI 모델과 외부 도구(툴), 데이터 소스, 서비스들을 안전하게 연결하는 표준 프로토콜**이다.
- 쉽게 말하면, AI(ChatGPT, Cursor AI 같은 LLM)가 **파일 시스템, API, 데이터베이스, 클라우드 서비스**와 대화할 수 있도록 해주는 "플러그인 시스템" + "표준 인터페이스"라고 보면 된다.



https://modelcontextprotocol.io/docs/learn/architecture

1.2 Model Context Protocol (MCP)

MCP 구조

https://dytis.tistory.com/112

- **MCP Host** : Cursor처럼 MCP를 통해 외부 데이터나 기능을 사용하려는 프로그램.

- **MCP Server** : 특정 기능을 제공하는 경량 프로그램으로, Cursor의 요청에 따라 외부 데이터나 도구에 접근한다 .

- **MCP Client** : Cursor와 MCP 서버 사이에서 1:1 연결을 유지하며 요청과 응답을 주고받는 역할을 한다.

- **Local Data Sources**: MCP 서버가 접근할 수 있는 컴퓨터 내 자원(예: 파일, 데이터베이스 등)

- **Remote Services** : Google Drive, Notion 등 인터넷을 통해 연결할 수 있는 외부 시스템으로, MCP 서버가 Cursor와 연결해 데이터를 가져올 수 있다.

1.2 Model Context Protocol (MCP)

2. Cursor에서 MCP의 역할

Cursor는 원래 코드 편집기+AI 기능을 제공하는 IDE잖아요?
여기에 MCP가 추가되면 AI가 다음을 할 수 있어요:

- **파일 읽기/검색**: 프로젝트 내 파일을 AI가 직접 탐색
- **실행 환경 접근**: 터미널 명령 실행, 빌드 로그 확인
- **외부 서비스 연동**: 예를 들어 GitHub, Postgres, API 서버 등과 직접 통신
- **보안 제어**: 허용된 MCP 도구만 연결 가능 → 무분별한 시스템 접근 방지

즉, AI가 "코드 편집기 안의 지능형 비서"에서 → "외부 환경과 상호작용하는 에이전트"로 확장되는 구조라고 보면 된다.

1.2 Model Context Protocol (MCP)

3. MCP 동작 방식 (흐름)

1. **Cursor (AI IDE)** 가 사용자 질문을 받음
2. AI가 답변 과정에서 "외부 정보 필요" → MCP로 요청 보냄
3. MCP는 등록된 **Provider(예: 파일 시스템, API, DB)**에게 작업을 위임
4. 결과를 다시 AI에 전달 → AI가 사용자에게 응답

➡️ 이 과정이 **표준화된 프로토콜**로 이루어져 있어서, 특정 IDE에 종속되지 않고 확장 가능함.

1.2 Model Context Protocol (MCP)

4. MCP의 장점

- **표준화**: 다양한 툴·데이터 소스에 일관된 방법으로 접근 가능
- **보안성**: 허용된 MCP Provider만 접근 가능, 권한 관리 용이
- **확장성**: 필요할 때마다 새로운 MCP Provider 추가 가능
- **에이전트화**: Cursor AI가 단순히 코드 작성 보조가 아니라, 테스트 실행/로그 분석/DB 질의 같은 자동화된 작업을 수행 가능

1.2 Model Context Protocol (MCP)

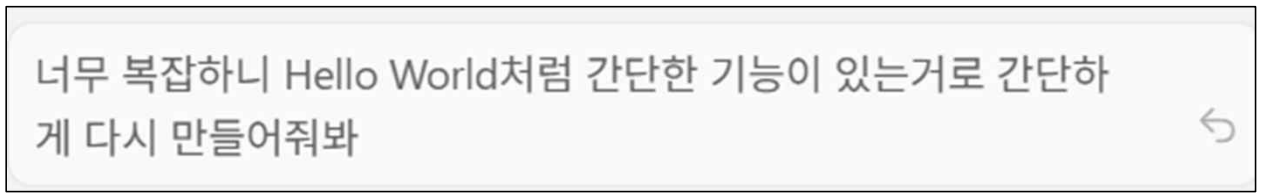
5. 예시 (Cursor에서 MCP 활용)

- 사용자: *"이 프로젝트에서 가장 많이 호출되는 함수 알려줘"*
- AI → MCP(File provider): 코드 인덱스/AST 분석 요청
- MCP: 결과 반환
- AI: 분석 결과 요약하여 답변
- 사용자: *"이 API 엔드포인트 응답 예시를 가져와"*
- AI → MCP(HTTP provider): `GET /api/v1/users` 요청
- MCP: JSON 응답 반환
- AI: 요약 후 사용자에게 답변

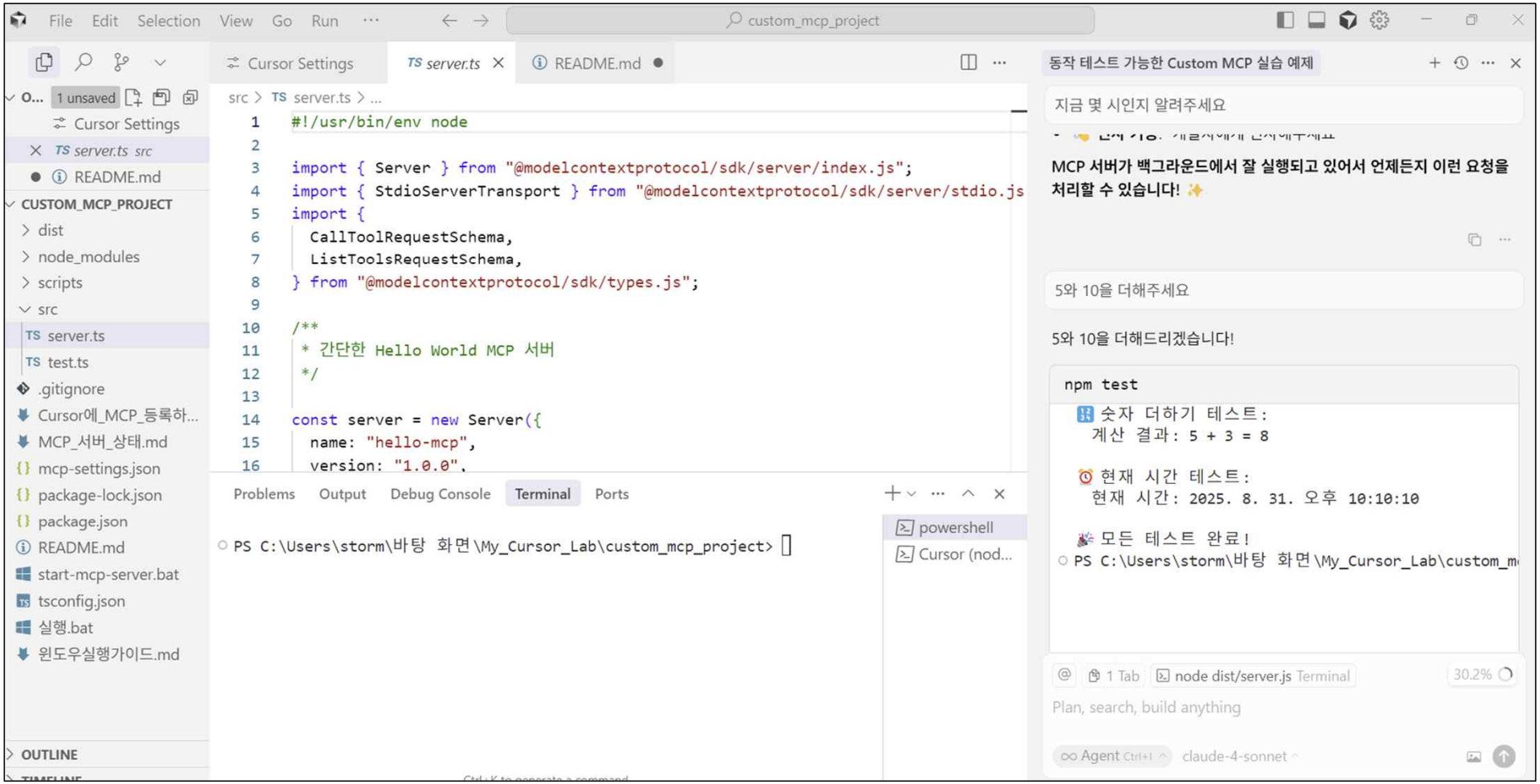
1.2 Model Context Protocol (MCP)

MCP 실습 예제

1. 프로젝트 준비 : custom_mcp_project/

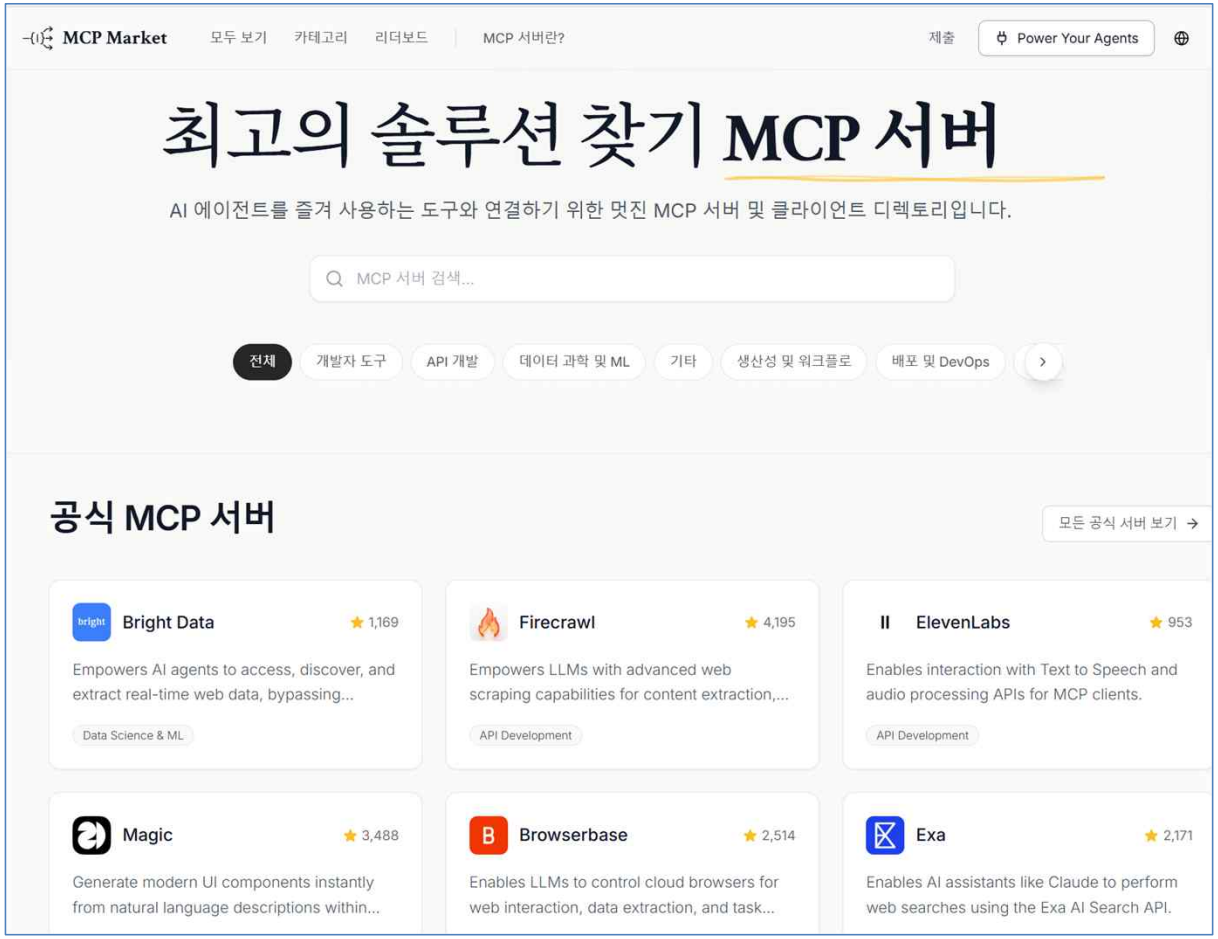


1.2 Model Context Protocol (MCP)



1.2 Model Context Protocol (MCP)

공식 MCP 서버 마켓



https://mcpmarket.com/

1.3 Rules

1.3 Rules

Rules

재사용 가능하고 스코프가 있는 지침으로 Agent 모델의 동작을 제어해.

Rules는 Agent와 Inline Edit에 시스템 레벨의 지침을 제공해. 프로젝트를 위한 지속적인 컨텍스트, 기본설정, 워크플로로 생각하면 돼.

Cursor는 네 가지 유형의 Rules를 지원해:

Project Rules

`.cursor/rules`에 저장되고, 버전 관리되며, 코드베이스 스코프로 적용돼.

User Rules

Cursor 환경 전역에 적용돼. 설정에서 정의되고 항상 적용돼.

AGENTS.md

마크다운 형식의 Agent 지침. `.cursor/rules`의 간단한 대안이야.

.cursorrules (Legacy)

아직 지원되지만 더는 권장하지 않아. 대신 Project Rules를 써.

1.3 Rules

1. Project Rules

- **저장 위치**: 프로젝트 내 `.cursor/rules` 폴더
- **특징**:
  - Git 버전 관리됨 → 팀 협업 시 공유 가능
  - 프로젝트 단위로만 적용됨
  - 코드베이스 스코프에 국한됨
- **활용 예시**:
  - "이 프로젝트에서는 ESLint 규칙 A, B, C를 따른다"
  - "React 컴포넌트는 항상 TypeScript로 작성"

1.3 Rules

2. User Rules

- **저장 위치**: Cursor 설정(Settings)
- **특징**:
  - Cursor **전체 환경(전역)** 에 적용
  - 어떤 프로젝트를 열든 항상 적용됨
- **활용 예시**:
  - "코드는 항상 변수명을 camelCase로 작성"
  - "주석은 영어로 작성"

> ⚡ 블로그 글에서는 `Global Rules` 라는 표현을 쓰기도 하지만, 공식 문서상 명칭은 **User Rules**입니다.
> (Global = User Rules)

3. AGENTS.md

- **저장 위치**: 프로젝트 루트 (보통 `.cursor/rules/AGENTS.md`)
- **특징**:
  - 마크다운 형식으로 Agent 지침 작성
  - Project Rules의 **간단 대안**
- **활용 예시**:
  - "이 Agent는 API 문서 기반 답변만 한다"
  - "테스트 코드는 Jest만 사용한다"

4. .cursorrules (Legacy)

- **저장 위치**: 프로젝트 루트에 단일 파일
- **특징**:
  - 예전 방식 → 지금도 작동하지만 권장되지 않음
  - 최신 기능(스코프 적용, 버전 관리 등)은 부족
- **활용 예시**:
  - 과거 프로젝트 호환성 유지 목적

1.3 Rules

User Rules 설정

User Rules는 **Cursor 설정(Settings)** 안에 직접 작성
별도 파일이 아니라 **Cursor → Setti ngs → Rules for A**I부분에 넣는 텍스트 규칙임.
프로젝트가 달라져도 항상 적용되므로 **Global 규칙처럼 동작**

 # User Rules (Global Scope) 예제

Code Style
- 변수명은 camelCase로 작성한다.
- 상수는 대문자 SNAKE_CASE로 작성한다.
- 함수는 동사+명사 조합으로 이름 짓는다. (예: getUserData)

Comments
- 주석은 반드시 영어로 작성한다.
- 복잡한 로직은 한 줄 요약 + 입력/출력 설명을 포함한다.

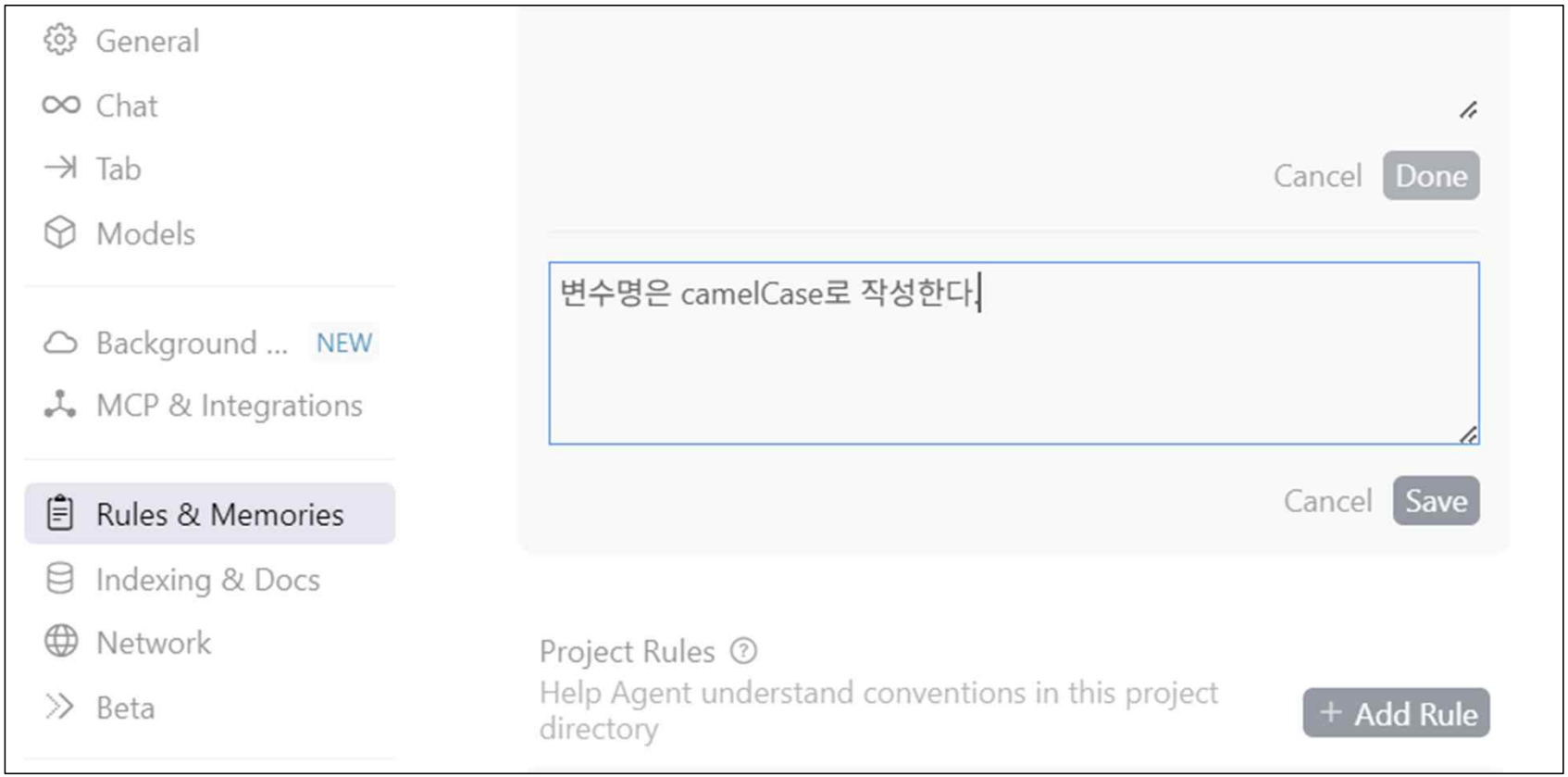
1.3 Rules

Documentation
- 모든 public 함수에는 JSDoc 주석을 작성한다.
- 클래스에는 최소한의 사용 예시(example)를 포함한다.

Testing
- 단위 테스트는 Jest를 사용한다.
- 새로운 함수 작성 시 최소 2개 이상의 테스트 케이스를 포함해야 한다.

Git / Collaboration
- 커밋 메시지는 영어로 작성하고, 형식은 다음을 따른다:
  - feat: 새로운 기능
  - fix: 버그 수정
  - docs: 문서 변경
  - refactor: 리팩토링

1.3 Rules


General
Chat
Tab
Models
Background ... NEW
MCP & Integrations
Rules & Memories
Indexing & Docs
Network
Beta
Cancel
Done
변수명은 camelCase로 작성한다
Cancel
Save
Project Rules
Help Agent understand conventions in this project directory
+ Add Rule

1.4 Memories

1.4 Memories

Cursor Memories란?

- **Memories**는 Cursor가 **특정 프로젝트, 파일, 대화 맥락에 대해 장기적으로 기억하는 정보 저장소**예요.
- Rules가 "AI가 따라야 하는 지침(규칙)"이라면,
  Memories는 "AI가 기억해야 할 사실/맥락(knowledge)"에 가깝습니다.

📊 Rules와 차이점

항목	Rules	Memories
성격	지침(규칙)	지식(사실/맥락)
적용 범위	User 전역 / 프로젝트별	주로 프로젝트별 (대화 맥락)
작성 방식	사용자가 명시적으로 작성	사용자가 추가하거나 AI가 학습
용도	스타일/정책 강제	컨텍스트 기억

1.4 Memories

Memories 저장/관리 위치

Cursor에서 Memories는 따로 파일로 관리되는 게 아니라, Cursor 앱 안에서 직접 관리된다.

1. **대화(Chat) 패널에서 저장**
   - AI와 대화 중 특정 정보를 **"이걸 기억해"**라고 입력하면 자동으로 Memories에 저장됩니다.
   - 또는 AI가 "이 내용을 Memory에 추가할까요?" 하고 물어볼 때 승인하면 저장됩니다.
2. **사이드바(Sidebar) → Memories 메뉴**
   - Cursor 왼쪽 사이드바(탐색기 영역)에서 ⚙ **Settings** 옆에 **Memories** 탭이 있습니다.
   - 여기서 프로젝트별 Memories를 확인, 수정, 삭제할 수 있습니다.
3. **프로젝트 단위 저장**
   - Memories는 기본적으로 **프로젝트 단위**로 묶입니다.
   - 예: `my-django-app` 프로젝트에서 저장한 Memories는 그 프로젝트를 열었을 때만 불러옵니다.
4. **User 단위(Global)**
   - 일부 정보(예: "나는 항상 영어 주석을 원한다")는 User Rules 쪽이지만,
   - Cursor는 **장기간 자주 반복되는 정보**를 User Memories로 올려둘 수 있습니다.
   - Settings 안의 Memories 관리 화면에서 확인 가능합니다.

1.4 Memories

메모리 저장과 결과 확인

이프로젝트는 MCP 커스텀 프로젝트야
이 사실을 Cursor Memory에 저장해

네! MCP 커스텀 프로젝트 정보를 Cursor Memory에 저장하겠습니다!

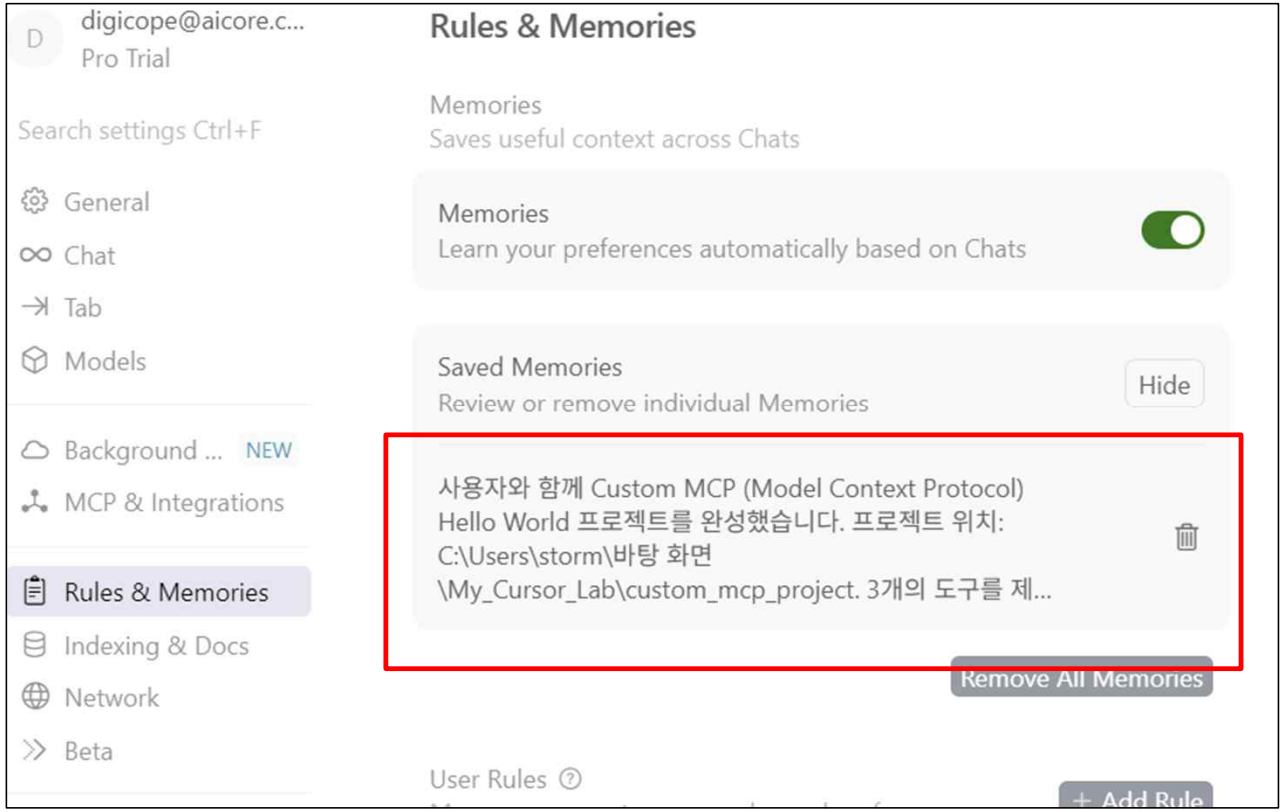
Memory created

✅ **Cursor Memory에 저장 완료!**

📝 저장된 정보

"Custom MCP Hello World **프로젝트**"로 다음 내용이 저장되었습니다:

- 🎯 **프로젝트 유형**: Custom MCP (Model Context Protocol)
- 📍 **프로젝트 위치**: `C:\Users\storm\바탕 화면\My_Cursor_Lab\custom_mcp_project`



Chat에서 직접 입력 :
이 프로젝트는 MCP 커스텀 프로젝트야
이 사실을 Cursor Memory에 저장해

1.4 Memories

📝 Cursor Memories 사용 예제

1. 프로젝트 기본 환경 저장

예를 들어, 새로운 프로젝트를 시작했을 때 아래처럼 입력합니다:

```
이 프로젝트는 Python 3.11을 사용하고, Django 5.0 기반으로 작성된다.
데이터베이스는 PostgreSQL이며, ORM으로는 Django ORM만 사용한다.
```

➡️ Cursor는 이 내용을 **Memories**에 저장합니다.
이후 "모델 생성 코드 작성해줘"라고 하면 AI가 **자동으로 Django ORM 기준**으로 코드를 제안합니다.

1.4 Memories

2. 스타일 가이드 기억

```
React 컴포넌트는 모두 함수형으로 작성한다.
CSS는 styled-components를 사용한다.
주석은 반드시 영어로 작성한다.
```

➡️ 다음에 "Button 컴포넌트 만들어줘"라고 요청하면,
클래스형이 아니라 함수형 + styled-components를 활용한 코드가 나옵니다.

1.4 Memories

3. API 스펙 기억

```
백엔드 API는 /api/v1/ 경로를 사용한다.
모든 요청은 JWT 토큰을 Authorization 헤더에 담아야 한다.
```

➡️ 이후 "로그인 API 호출 코드 만들어줘"라고 하면,
자동으로 `/api/v1/login` 엔드포인트와 JWT 인증 헤더를 붙여서 코드가 생성됩니다.

1.4 Memories

4. 팀별 컨벤션 기록

```
에러 핸들링은 반드시 try/catch 블록 안에서 logger.error로 남겨야 한다.
유닛 테스트 프레임워크는 Jest만 사용한다.
```

➡️ "API 호출 예시 만들어줘"라고 하면 자동으로 `logger.error()`를 활용합니다.

➡️ "테스트 코드도 같이"라고 하면 Jest로 기본 테스트 스켈레톤을 생성합니다.

Thank You!!