No Code & Low Code & Vibe Coding

엄진영

개요

No Code vs Low Code vs AI Coding

	No Code	Low Code	AI Coding(Vibe Coding)
개념	코딩 없이, 시각적 도구(UI/드래그앤드롭, 설정화면) 방식으로 앱/웹, 자동화 프로세스를 만드는 방식이다.	시각적 개발 + 최소한의 코드로 앱 개발	AI가 코드를 생성·보조하며 개발자가 수정/보완
대상자	비개발자, 기획자, 마케터, 현업 담당자	개발자 + 비즈니스 사용자 협업	주로 개발자 (기본 코딩 이해 필요)
도구	Bubble, Airtable, Zapier, Glide, Voiceflow, Aire 등	OutSystems, Retool, PowerApps, Google AppSheet, Mendix, Appian 등	GitHub Copilot, Cursor, Claude Code, Replit AI, Aider 등
장점	• 빠른 프로토타입 제작 • 기술 장벽 낮음	• 빠른 개발 속도 • No Code보다 복잡한 앱도 가능	• 자유도 가장 높음 • 최신 AI 활용 → 코드 생산성↑
단점	• 확장성/커스터마이징 제한 • 복잡한 로직 구현 어려움	• 플랫폼 종속성(비용, 유지보수) • 여전히 코드 작성 필요 • 완전 비개발자에게 진입 장벽 있음	• 품질·보안 한계 • AI 코드 이해/검증 필요
활용 사례	간단한 설문 앱, 업무 자동화, 데이터 관리	기업용 내부 앱, ERP/CRM 보완	웹/앱 개발, API 연동, 디버깅 지원
학습 곡선	가장 낮음	중간	다소 높음 (코딩 지식 필요)
통제권	플랫폼이 통제	사람 + 플랫폼 공동 통제	사람이 통제

개발 프로세스

No Code

아이디어

화면 구성

설정

배포

Low Code

아이디어

화면 구성

기본 로직

커스텀 로직

배포

AI Coding

아이디어

자연어 설명

AI가 코드 제안

사람이 검토·수정

테스트 · 배포

No Code

No Code 특징

- **코드 작성 없이** 시각적 인터페이스를 통해 애플리케이션을 개발하는 방식
- 비개발자, 현업 사용자 (업무 자동화·간단 앱 제작)
- **빠른 프로토타입 제작**,기술 장벽 낮음
- 확장성/커스터마이징 제한, 복잡한 로직 구현 어려움
- 간단한 설문 앱, 업무 자동화, 데이터 관리

No Code 대표 도구

Bubble – 웹 앱 개발 플랫폼

스타트업 MVP, SaaS 프로토타입, 내부 서비스

Airtable – 스프레드시트 + 데이터베이스

업무 관리 시스템, CRM, 간단한 백오피스

Glide – 스프레드시트 기반 모바일 앱

사내 모바일 앱, 간단한 고객용 앱, 설문·조회 앱

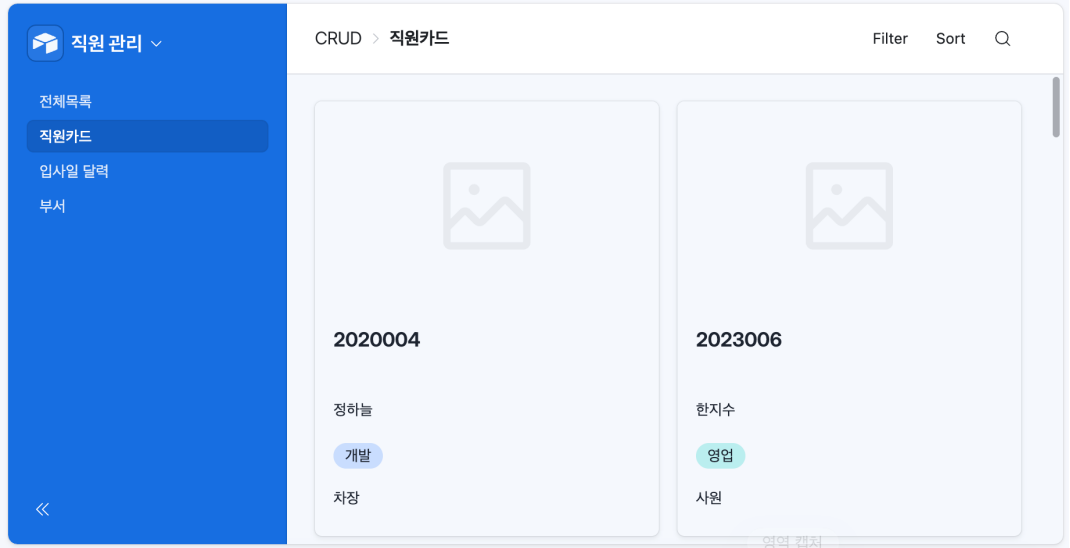
Zapier – 업무 자동화

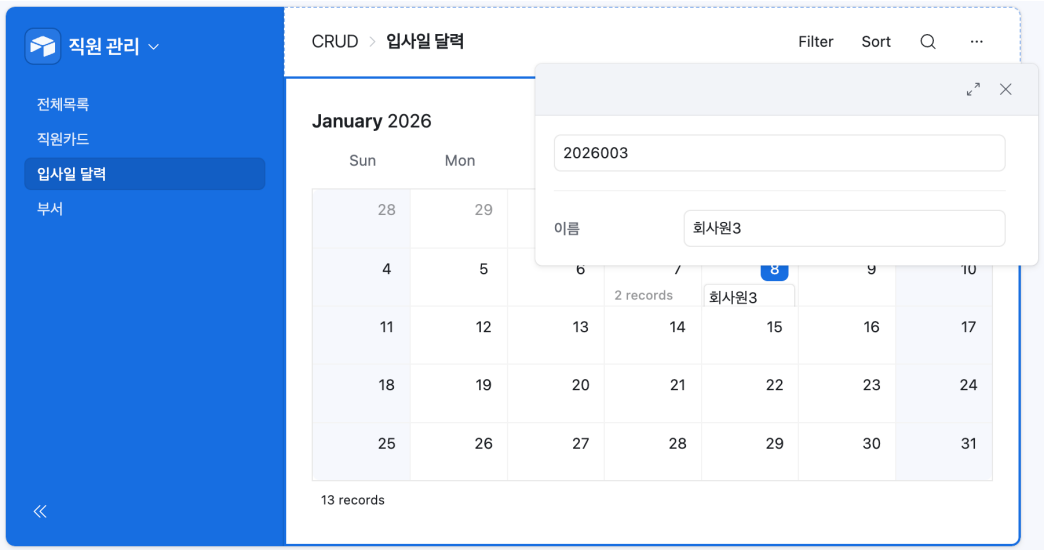
이메일·슬랙·시트 연동, 마케팅/운영 자동화, 데이터 동기화

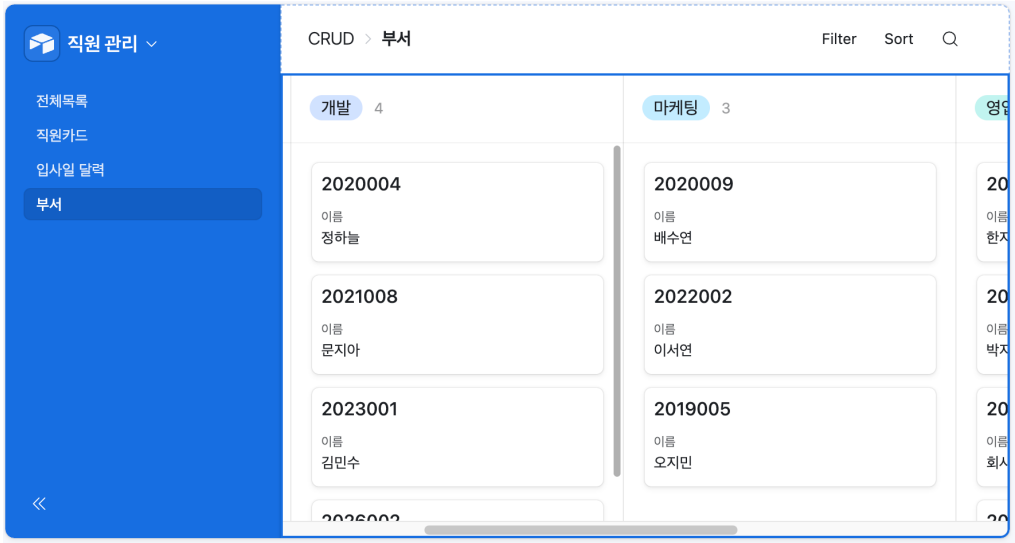
Webflow – 디자인 중심 웹사이트 제작

기업 홈페이지, 랜딩 페이지, 마케팅 페이지

Airtable 실습 - 간단한 직원관리 시스템 만들기



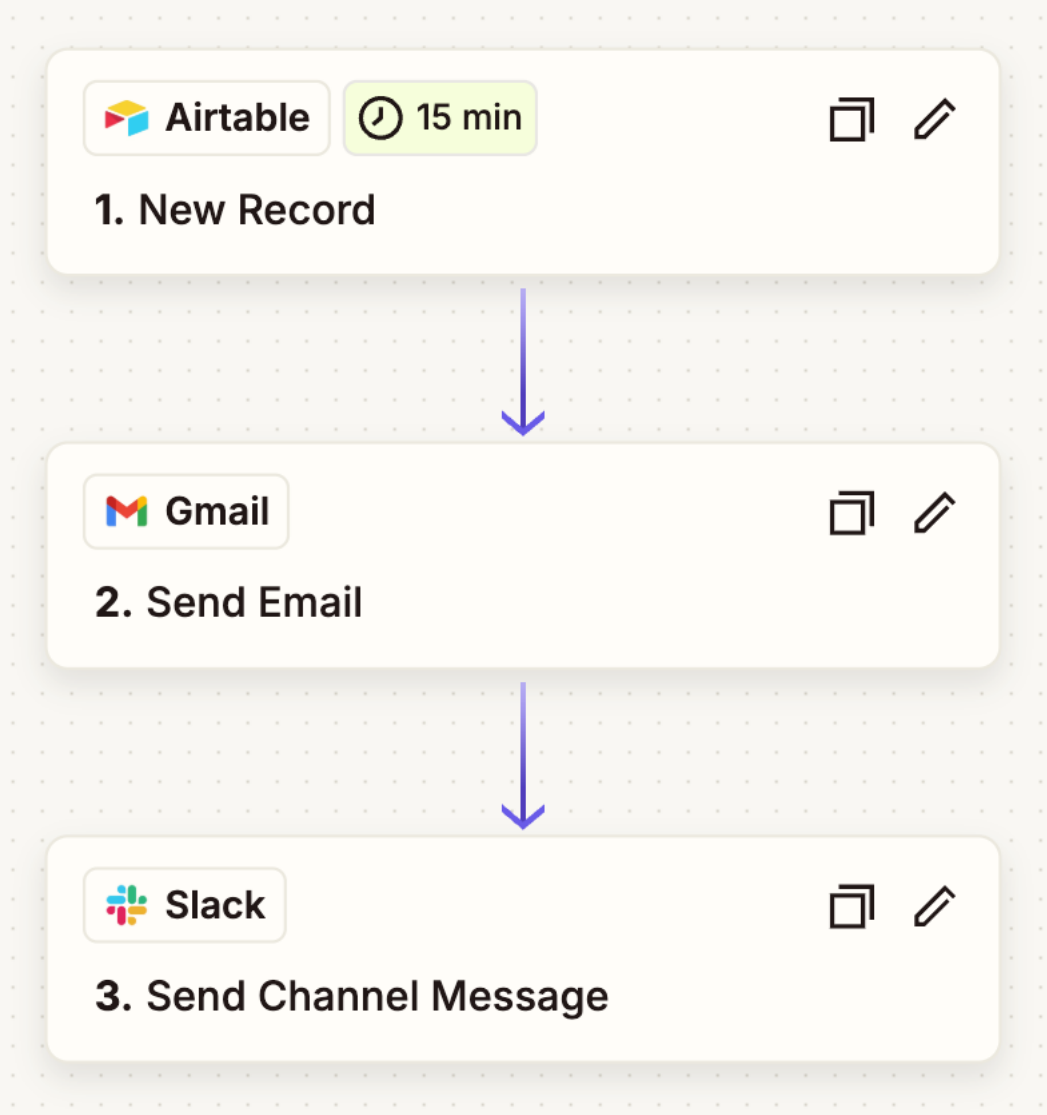




No Code

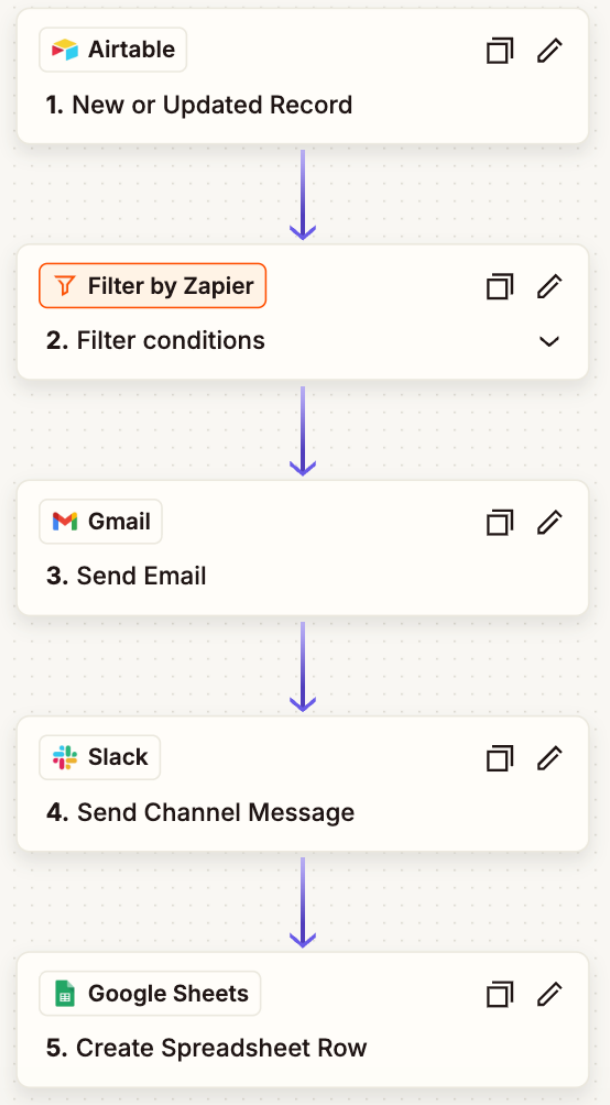
Zapier 실습 - 신규 직원 온보딩 업무 자동화

Zap 워크플로우



Zapier 실습 – 퇴사 처리 업무 자동화

Zap 워크플로우



Low Code

Low Code 특징

- 최소한의 코드 작성으로 애플리케이션을 개발하는 방식
- 개발자 + 비즈니스 사용자 협업
- 빠른 개발 속도, 복잡한 앱도 가능
- 플랫폼 종속성(비용, 유지보수), 일부 코딩 지식 필요 (SQL, JavaScript 등)
- 기업용 내부 앱, ERP/CRM 보완

Low Code 대표 도구

Microsoft Power Apps – Drag & Drop UI + 간단한 스크립트로 앱 제작 가능

기업 내부 업무 앱, CRM 보조 시스템 등

OutSystems – 시각적 개발 + 전문가용 코드 확장 가능

대기업용 포털, 모바일 웹/앱 개발

Mendix – UI 빌더 + 데이터 모델링 + API 연결까지 시각적으로 구성

대규모 엔터프라이즈 앱, 프로세스 자동화

Appian – 로우코드 + RPA 기능 결합, 업무 프로세스 자동화 + 케이스 관리 중심

금융·보험·헬스케어 등 복잡한 프로세스 자동화 프로젝트

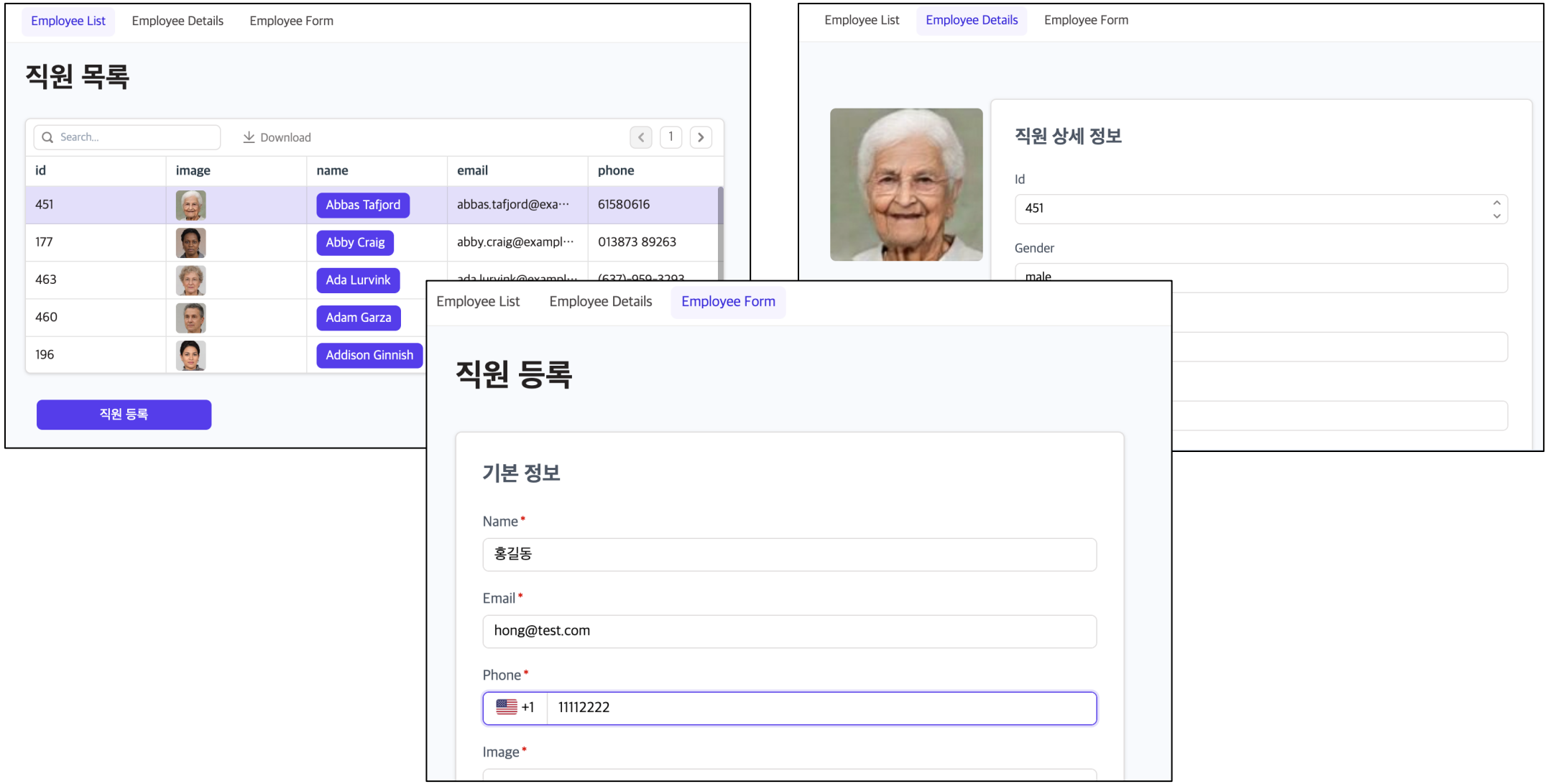
Salesforce Lightning Platform – CRM 기반 구축 플랫폼, 워크플로우, API 통합

CRM 커스터마이징, 고객 포털, 영업/서비스 앱

Appsmith – Drag & Drop UI + 커스텀 코드 확장

내부 도구/관리자 패널/대시보드 같은 비즈니스 앱을 빠르게 만들 수 있게 설계된 도구

Appsmith 실습: 직원 CRUD 앱 만들기



AI-assisted Coding

AI-assisted Coding 특징

- AI가 코드를 생성·보조하며 개발자가 수정/보완하는 방식
- 주로 개발자 (기본 코딩 이해 필요)
- 자유도 가장 높음, 최신 AI 활용 → 코드 생산성↑
- 품질·보안 한계, AI 코드 이해/검증 필요
- 웹/앱 개발, API 연동, 디버깅 지원

AI-assisted Coding 대표 도구

IDE 통합 코드 어시스턴트

- **GitHub Copilot**: 가장 널리 사용되는 AI 코드 완성 도구, OpenAI Codex 기반
- **Amazon CodeWhisperer**: AWS 서비스와 잘 통합되는 무료 코드 어시스턴트

대화형 AI 코딩 어시스턴트

- **ChatGPT**: 코드 설명, 디버깅, 알고리즘 구현에 널리 활용
- **Claude**: 긴 코드 분석 및 복잡한 프로그래밍 문제 해결에 강함
- **Google Bard/Gemini**: 구글의 AI 어시스턴트, 코딩 지원 포함
- **Perplexity**: 코딩 관련 정보 검색 및 코드 생성

전문 AI 코딩 플랫폼

- **Cursor**: AI 우선으로 설계된 코드 에디터, GPT-4 통합
- **Replit**: AI 기반 협업 코딩 환경
- **Claude Code CLI**: 커맨드 라인 인터페이스 기반의 코드 작성 및 수정 지원

Vibe Coding

정의

- **AI 코딩을 활용하는 새로운 개발 패러다임** 또는 **작업 방식**을 지칭한다.
- 단순히 AI가 코드를 작성하는 것을 넘어서,
  - **개발자**가 문제 정의, 아이디어 제공, 코드 리뷰 역할을 하고
  - **AI**가 세부 구현·자동화된 반복 작업을 담당하며
  - "**페어 프로그래밍**"처럼 맥락을 잃지 않고 자연스럽게 흐름을 이어가는 것.
- 즉, **AI와 함께 대화하듯 코딩하는 방식** 을 의미한다.
- 생산성과 창의성을 높이는 개발 경험(DX)의 변화에 더 초점을 둔다.
- 다시 말해,
  - AI 코딩 = 기술 (AI가 코드를 도와주는 기능)
  - **바이브코딩 = 문화/패러다임 (AI 코딩을 활용하는 방식)**

특징

개발 과정: AI와의 대화를 통해 코드를 생성하고 수정

- **인간**: “로그인 API 만들어줘” → **AI**: 로그인 API 코드 생성
- **인간**: “JWT 토큰도 추가해줘” → **AI**: JWT 로직 추가된 코드 생성

최종 결과물

- 실제 프로그래밍 코드 (Python, JavaScript, Java 등)
- 개발자가 이해하고 수정 가능한 텍스트 코드

개발자의 역할 변화

- **코드를 직접 짜는 사람 → AI와 협력하여 문제를 해결하는 사람**
- **AI 협력자**: AI와 대화하며 빠르게 프로토타입을 만드는 사람
- **문제 정의자**: 고객/사용자 요구를 구체화해 AI가 이해할 수 있게 번역하는 사람
- **검증자**: 결과물이 안전하고, 정확하며, 윤리적/규제적으로 문제없는지 확인하는 사람
- **AI 최적화자**: AI가 더 좋은 답을 내도록 프롬프트와 워크플로우를 설계하는 사람

장점

생산성 폭발적 향상

- 아이디어를 바로 코드로 옮길 수 있어 개발 속도가 수배 이상 빨라짐.
- 프로토타이핑 → 실험 → 수정 사이클이 극단적으로 짧아짐.

코딩 장벽 완화

- 복잡한 문법이나 API를 몰라도 자연어로 코드를 만들 수 있음.
- 비전문가(기획자, 디자이너, 연구자 등)도 직접 작은 툴이나 앱을 만들 수 있음.

창의성 확장

- “이거 가능할까?” 하는 아이디어를 바로 시도할 수 있음.
- 개발자가 아이디어 실현가(creator)로 더 많은 실험을 할 수 있음.

반복 업무 자동화

- 보일러플레이트 코드, CRUD, 테스트 코드 같은 단순·반복 업무를 AI가 대신 처리.
- 개발자는 핵심 로직과 전략적 문제 해결에 집중 가능.

빠른 학습 도구

- 초보 개발자에게는 코드를 일일이 배우기보다, AI가 만들어주는 코드를 보면서 학습하는 실시간 멘토 역할 가능.

단점

정확성 문제

- AI가 제시한 코드가 항상 올바른 건 아님.
- 버그, 보안 취약점, 성능 저하 등이 숨어 있을 수 있음.
- 따라서 “검증” 단계가 필수.

깊은 이해 부족

- 사용자는 코드 내부 원리를 잘 모른 채 결과만 쓰게 됨.
- 복잡한 문제 상황에서 AI가 틀린 답을 줘도 눈치 못 채고 넘어갈 위험.

의존성 증가

- 개발자가 점점 AI 없이는 코드를 못 짜는 상태가 될 수 있음.
- 문제 해결 능력(알고리즘, 최적화, 디버깅 등) 약화 가능성.

보안·윤리적 위험

- AI가 만든 코드에 오픈소스 라이선스 충돌, 데이터 유출, 보안 구멍이 있을 수 있음.
- 이를 제대로 검증하지 않으면 심각한 사고로 이어질 수 있음.

추상화의 추상화

- 이미 현대 개발은 프레임워크·라이브러리로 추상화가 많은데,
- AI 레이어까지 얹으면 개발자가 실제 동작 원리에 접근하기 더 어려워짐.

품질 관리 어려움

- AI가 만든 코드는 팀마다 일관성이 떨어질 수 있음.
- 코드 스타일, 설계 패턴, 문서화 관리가 복잡해짐.

바이브 코딩에 개발 지식이 필요한 이유

정답 기준 정의(스펙·제약 조건)

- LLM은 “가능해 보이는 답”을 내놓는다. 정합성, 경계조건, 비기능 요구(성능·보안·신뢰성)는 사람이 정의·관리해야 한다.

리스크 인지와 트레이드오프 판단

- 메모리/CPU 비용, 네트워크 실패, 일관성, 지연시간, 비용(유료 API 호출) 같은 현실 제약을 AI는 자동으로 고려하지 못한다.

품질·보안 검증 책임

- 취약점(OWASP Top 10), 라이선스/저작권, 개인정보 처리, 규제 준수는 사람이 점검해야 한다.

시스템 통합·운영

- 버전 호환, 트랜잭션, 동시성, 장애 복구, 로그/모니터링/알림 등 운영 요소는 아키텍처 지식 없이는 설계 불가.

지속 가능성(유지보수)

- 모듈화, 설계 원칙(SOLID/DRY), 테스트 전략 없이는 AI가 만든 코드가 스파게티로 변한다.

바이브 코딩을 위한 핵심 지식

요구분석·설계 - 유스케이스, 경계조건, API 계약서, 상태/에러 플로우, 트랜잭션·일관성 모델.

언어/런타임 기초 - 타입/메모리 모델, 비동기/동시성, 컬렉션·알고리즘·복잡도, I/O 모델.

웹 기본기 - HTTP/REST, CORS, 인증/인가(JWT/OAuth2), 세션/쿠키/토큰 보안, 캐싱.

데이터 - 스키마 설계, 인덱스, N+1 쿼리, 마이그레이션, 트랜잭션 고립 수준.

보안 - 입력검증, 인젝션, XSS/CSRF, 시크릿 관리, 권한 상승 방지, 로깅·감사.

테스트/품질 - 단위/통합/계약(E2E) 테스트, 커버리지, 린트·정적분석, 성능/부하 테스트.

운영 - CI/CD, 롤백 전략, 피처 플래그, 모니터링(메트릭·로그·트레이싱), SLO/에러버짓.

라이선스/컴플라이언스 - OSS 라이선스 호환, 데이터 프라이버시(PII), 감사 추적.

"바이브 코딩은 **개발 진입장벽을 낮추는 기술**이지만,
운영 가능한 소프트웨어를 만들려면 개발자의 지식과 판단이 필수다.
AI가 **쓰기**를 도와주고, **개발자**는 **설계·검증·운영**으로 가치를 완성한다"

현업 적용 시 고려사항 – 개발 프로세스 측면

기존 워크플로와의 통합

- Git Flow, CI/CD, 코드 리뷰 등 기존의 엔지니어링 절차에 vibe coding을 어떻게 녹여낼 것인지 명확히 할 것.
- 예: 개발자가 AI가 작성한 코드를 바로 커밋하기보다, 리뷰 단계를 강화하거나 테스트 코드 자동 생성과 연결.

속도 vs. 품질 균형

- 빠른 프로토타이핑에는 유리하지만, 대규모 서비스 코드에서는 버그·보안 취약점 가능성이 있음.
- → 린(Lean) 단계: 아이디어 검증, MVP(Minimum Viable Product; 최소 기능 제품) 개발에 집중 활용
- → 안정 단계: 엄격한 코드 품질 관리 프로세스 결합

현업 적용 시 고려사항 – 협업 및 조직 문화

개발자 역할 변화

- 기존의 "코드 작성자"에서 "AI 생성 코드의 검증자·설계자"로 역할 이동.
- → 팀 내 역할 정의 필요: 누가 AI 제안 품질을 점검하는지, 리뷰 기준은 무엇인지.

AI 의존도 관리

- 주니어 개발자는 학습 기회를 놓칠 수 있음.
- → 교육·멘토링 프로그램과 병행하여 "왜 이렇게 동작하는지"를 이해하도록 유도.

협업 도구와 통합

- VSCode, Cursor, Copilot, JetBrains 등 IDE 환경과 Slack/Jira/GitHub PR에 vibe coding 결과를 공유할 수 있는 협업 자동화 고려.

현업 적용 시 고려사항 – 보안 및 품질

데이터 보안

- 내부 코드/비밀키가 AI 훈련 데이터로 유출되지 않도록 프라이빗 LLM 또는 엔터프라이즈급 보안 제공 AI 서비스 사용.

코드 품질 및 표준화

- AI가 생성한 코드는 팀의 코딩 컨벤션·아키텍처 원칙을 벗어날 수 있음.
- → ESLint, Prettier, SonarQube, 테스트 자동화 등으로 자동 검증 체계 강화.

라이선스/저작권 이슈

- 생성된 코드가 오픈소스 라이선스 위반 소지를 만들 수 있으므로 법무·오픈소스 관리팀 검토 필요.

현업 적용 시 고려사항 – 생산성 측정과 ROI

정량적 지표 설정

- 개발 속도: 기능 개발 소요 시간 감소율
- 품질: 버그 발생률, 코드 리뷰에서 발견되는 오류 건수
- 팀 만족도: 개발자가 AI 코딩 도구를 통한 인지 부하 감소 체감도

부분적 도입 후 확장

- 초기에는 특정 팀/도메인(MVP, 데이터 파이프라인, 문서화)에 제한적으로 적용 → 효과 검증 후 전사 확대.

현업 적용 시 고려사항 – 교육 및 지속 개선

바이브 코딩 훈련

- 개발자들이 AI에게 잘 질문(prompt) 하고, AI 제안을 비판적으로 검토하는 습관을 길러야 함.
- → 사내 워크숍, 코드랩, 베스트 프랙티스 공유 세션 필요.

AI 피드백 루프

- 잘못된 코드/좋은 코드 예시를 지속적으로 수집하여 조직 맞춤 프롬프트 엔지니어링 가이드 축적.

현업 적용 시 고려사항 – 정리

균형

- **프로세스 적합성**
- **협업 문화**
- **보안·품질 보장**
- **ROI 검증**
- **개발자 교육**

"AI가코드를 대신 짜준다는 환상을 버리고,
AI + 사람의 협업 체계를 설계해야
성공적으로 현업에 안착시킬 수 있다"

바이브 코딩 도입 및 운영 전략 문서 예)

1) 개요

- 바이브 코딩(Vibe Coding): 개발자가 코드를 직접 작성하기보다 AI의 제안을 활용해 흐름과 느낌 위주로 개발하는 방식
- 목적: 생산성 향상, 빠른 프로토타이핑, 개발자 경험 개선
- 대상: 사내 개발팀(백엔드, 프론트엔드, 데이터 사이언스, DevOps 등)

2) 도입 배경

- AI 코딩 어시스턴트(Copilot, Cursor, ChatGPT 등)의 성숙
- 빠른 제품 개발 및 시장 검증 필요성 증가
- 개발자 업무 부담 완화 및 유지보수 효율화

3) 적용 시 고려사항

- 개발 프로세스
  - Git Flow 및 코드 리뷰 절차와의 정합성 유지
  - 테스트 코드 및 문서 자동화와 연계
  - 프로토타입/실서비스 단계 구분 적용
- 협업 및 조직 문화
  - 개발자 역할 변화: 작성자 → 검증자
  - AI 의존으로 인한 학습 부족 방지 (교육, 코드 리뷰 강화)
  - 협업 툴(Jira, Slack, GitHub PR 등)과 연계
- 보안 및 품질
  - 사내 코드/데이터 유출 방지 → 프라이빗 모델 또는 기업용 AI 사용
  - 팀별 코딩 컨벤션·아키텍처 표준 준수
  - 라이선스/저작권 검증 절차 포함

바이브 코딩 도입 및 운영 전략 문서 예)

4) 단계별 도입 전략

- 1단계: 파일럿 적용
  - 적용 범위: 프로토타입, PoC, MVP 개발
  - 성과 지표:
    - 개발 속도 (기능 릴리스 소요 시간)
    - 코드 리뷰 발견 오류 건수
    - 개발자 만족도 설문
- 2단계: 제한적 운영
  - 적용 범위: 일부 제품/팀
  - 운영 방법:
    - 바이브 코딩 가이드라인 공유
    - 사내 코드 리뷰 세션에서 AI 코드 활용사례 공유
    - 정기 피드백 루프 운영
- 3단계: 전사 확산
  - 적용 범위: 주요 서비스 전체
  - 추가 조치:
    - 자동화 툴링 강화 (ESLint, Prettier, SonarQube, Test Coverage)
    - 사내 맞춤형 프롬프트 템플릿 제작
    - 지속적인 교육 및 우수 사례 전파

바이브 코딩 도입 및 운영 전략 문서 예)

5) 교육 및 지원

- 개발자 교육
  - AI에게 올바른 프롬프트 작성법
  - AI 코드 제안의 검증 및 수정 방법
  - 코드 품질 도구 활용법

- 조직 지원
  - 정기 워크숍/세미나
  - 사내 베스트 프랙티스 문서화
  - 기술 리더/아키텍트의 품질 가이드라인 수립

6) 기대 효과

- 개발 속도 30~50% 향상
- 신규 인력의 온보딩 기간 단축
- 반복 업무(보일러플레이트 코드, 문서화) 자동화
- 개발자의 창의적 업무(설계, 문제 해결) 집중 가능

바이브 코딩 도입 및 운영 전략 문서 예)

7) 리스크 및 대응

리스크	대응 방안
품질 저하	자동화된 테스트 및 코드 리뷰 강화
데이터 유출	프라이빗 LLM/사내 배포형 모델 사용
AI 의존으로 인한 학습 부족	주니어 대상 코드 리뷰/멘토링 강화
라이선스 문제	오픈소스 라이선스 관리 툴 도입

8) 결혼

바이브 코딩은 단순히 코드를 빨리 쓰는 도구가 아니라, **조직의 개발 방식과 문화를 혁신하는 촉매제**이다. **사람 + AI 협업 체계**를 설계하고, 단계적·점진적 도입을 통해 현업에 안착시키는 것이 핵심이다.

최신 AI 트렌드 – Agent, RAG, MCP, 멀티모달, A2A

1) Agent (AI Agent)

정의: 단순한 질문-응답을 넘어서 스스로 판단하고 외부 도구를 호출하여 작업을 수행하는 AI 실행체

특징:

계획(Planning) + 실행(Execution) 구조

API, DB, 브라우저 등 다양한 툴 연동 가능

예: GitHub Copilot Agent, LangChain Agent, OpenAI o1-preview

장점: 자동화된 워크플로우, 반복 작업 제거

과제: 오작동 시 책임 소재 불명확, 신뢰성 보강 필요

최신 AI 트렌드 – Agent, RAG, MCP, 멀티모달, A2A

2) RAG (Retrieval-Augmented Generation)

정의: LLM + 검색 시스템을 결합하여 최신 정보나 사내 데이터를 불러와 답변하는 구조

특징:

DB, 벡터스토어(Vector DB, 예: Pinecone, Weaviate, Milvus) 사용

hallucination(환각) 줄이고 정확성 ↑

예: 사내 문서 QA, 법률/의료 지식 검색

장점: LLM을 훈련하지 않고도 실시간 최신 데이터 반영 가능

과제: 검색 품질에 의존 → 잘못된 데이터가 들어오면 잘못된 답변 생성

최신 AI 트렌드 – Agent, RAG, MCP, 멀티모달, A2A

3) MCP (Model Context Protocol)

정의: AI 모델과 외부 애플리케이션 간 표준 통신 프로토콜

특징:

OpenAI 주도로 표준화 시도

MCP 서버 ↔ MCP 클라이언트 구조 (IDE, ERP, CRM 연동 가능)

예: VSCode + MCP Client → ERP 서버(MCP Server) 연동

장점: 플러그인 호환성 확보, 다양한 시스템과 AI 연결 용이

과제: 아직 초기 생태계, 표준 확산 필요

최신 AI 트렌드 – Agent, RAG, MCP, 멀티모달, A2A

4) 멀티모달 (Multimodal AI)

정의: 텍스트, 이미지, 음성, 비디오 등 여러 모달리티를 동시에 처리하는 AI

특징:

ChatGPT-4o, Gemini 1.5, Claude 3.5 Sonnet 등 최신 모델이 대표

"이미지 설명 + 코드 생성 + 음성 답변" 통합 가능

장점: 인간과의 상호작용 방식이 자연스러움 (시각+청각 결합)

과제: 모달 간 동기화 문제, 학습 비용이 큼

최신 AI 트렌드 – Agent, RAG, MCP, 멀티모달, A2A

5) A2A (Agent-to-Agent)

정의: AI Agent끼리 상호작용하여 협업하는 구조\

특징:

Agent 간 역할 분담 (예: PM Agent, 개발자 Agent, QA Agent)

대규모 복잡한 프로젝트를 분산 수행

예: AutoGen, CrewAI

장점: 확장성 ↑, 사람 개입 최소화 가능

과제: 제어 불가능한 “AI 대화 루프” 발생 위험, 비용/리소스 관리 문제

최신 AI 트렌드 – Agent, RAG, MCP, 멀티모달, A2A

비교표

개념	핵심 목적	주요 활용	장점	한계
Agent	AI의 **행동 주체화**	자동화, 워크플로우 실행	반복작업 제거, 생산성 ↑	신뢰성 부족
RAG	최신/외부 데이터 반영	QA, 검색 기반 챗봇	정확도 ↑, 비용 효율	검색 품질 의존
MCP	**표준화된 연동**	IDE ↔ 시스템 ↔ AI 연결	생태계 확장성	초기 표준, 확산 부족
멀티모달	다양한 입력 처리	이미지 설명, 음성 인터랙션	UX 자연스러움	학습/운영 비용 ↑
A2A	AI 간 협업	프로젝트 관리, 멀티 에이전트	확장성 ↑	제어/비용 리스크

대표 도구

Cursor IDE

- 전체 코드베이스를 컨텍스트로 유지하며, PRD나 설명을 바탕으로 **여러 파일을 동시에 생성/변경** 가능
- 단순 코드 생성이 아니라 **계획** → **실행** → **코드 리뷰** → **검증까지** 이어주는 기능을 기본으로 지원
- AI가 제안하는 변경을 **한눈에 보고(diff로 확인)** 반영할 수 있어, 코드 품질/일관성 확보에 유리
- 기존 VS Code 익숙한 확장/디버깅/테스트/컨테이너 등 기능을 자연스럽게 활용 가능

Google Antigravity – Gemini 기반, 프리뷰 무료

Windsurf – Cursor 보다 저렴하고 세련된 UI

GitHub Codespaces – VSCode 기반 클라우드 IDE

Replit – 클라우드 기반 올인원, 로컬 환경 설정 불필요

Zed – 무료 및 오프 소스, 속도 빠름

실습

- ✓ 실습 1 - To-Do List 웹 앱 만들기
- ✓ 실습 2 – 날씨 조회 웹 앱 만들기
- ✓ 실습 3 - 쇼핑몰 백엔드 웹 애플리케이션 만들기 (SpringBoot 기반)