

Claude Code와 함께하는 바이브 코딩

AI 기반 개발 워크플로우 실습

RealWorld Java21 + Spring Boot 3 프로젝트로 배우는
AI Pair Programming

워크숍 개요

목표

- 바이브 코딩을 구체적으로 실행하는 방법 학습
- Claude Code를 활용한 효율적인 개발 워크플로우 구축
- 프로젝트 컨텍스트 관리와 자동화 기법 습득

대상

- AI 도구를 활용한 생산성 향상에 관심 있는 개발자
- 테스트 커버리지와 코드 품질 자동화를 배우고 싶은 분
- Claude Code의 실전 활용법을 알고 싶은 분

소개

- 황진성
- 로보코 컨설턴트
- 바이브코딩으로 풀스택 개발자가 되는 중
- ex. 컬리 백엔드 개발자




환경 설정

개발 환경 준비하기

Java 21, GitHub CLI, Claude Code CLI 설치

이번 세션의 목표

설치할 도구들

-  **Java 21** - 프로젝트 실행 환경
-  **GitHub CLI** - Git 작업 자동화
-  **Claude Code CLI** - AI 협업 도구

검증 사항

- 모든 도구가 정상 작동
- RealWorld 프로젝트 빌드 성공
- Claude Code 인증 완료

예상 소요 시간: 30분

Java 21 설치: macOS

방법 1: Homebrew (권장)

```
# OpenJDK 21 설치
brew install openjdk@21

# 심볼릭 링크 생성
sudo ln -sfn $(brew --prefix openjdk@21)/libexec/openjdk.jdk \
/Library/Java/JavaVirtualMachines/openjdk-21.jdk

# 환경 변수 설정
echo 'export JAVA_HOME=$(/usr/libexec/java_home -v 21)' >> ~/.zshrc
echo 'export PATH=$JAVA_HOME/bin:$PATH' >> ~/.zshrc
source ~/.zshrc

# 확인
java --version
```

Java 21 설치: macOS

방법 2: SDKMAN (버전 관리 필요 시)

```
# SDKMAN 설치
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"

# Java 21 설치
sdk install java 21.0.5-tem

# 기본 버전으로 설정
sdk default java 21.0.5-tem

# 확인
java --version
```

장점: 여러 Java 버전 관리 가능

Java 21 설치: Windows

Windows Subsystem for Linux (WSL) 권장

```
# 1. WSL 설치 (PowerShell 관리자 모드)
wsl --install

# 2. 시스템 재부팅

# 3. WSL Ubuntu 실행
wsl

# WSL 내부에서 Java 설치
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs
```

이유: Claude Code는 Unix 환경에 최적화

Java 21 설치: Windows

네이티브 Windows (대안)

```
# Scoop 패키지 매니저 사용
scoop install openjdk21

# 또는 winget
winget install Microsoft.OpenJDK.21

# 환경 변수 확인
$env:JAVA_HOME
java -version
```

참고: Git Bash에서 Claude Code 실행 가능

Java 설치 확인

필수 검증 단계

```
# 1. Java 버전 확인
java --version
# 출력 예시:
# openjdk 21.0.5 2024-10-15
# OpenJDK Runtime Environment (build 21.0.5+11)

# 2. Java 컴파일러 확인
javac --version
# 출력: javac 21.0.5





# 3. JAVA_HOME 확인
echo $JAVA_HOME
# 출력: /Library/Java/JavaVirtualMachines/openjdk-21.jdk/Contents/Home
```

모두 정상 출력되면 성공! 

GitHub CLI 설치

왜 GitHub CLI?

기능:

-  PR 생성/관리 CLI에서
-  Issue 작업 자동화
-  Repository 관리
-  GitHub Actions 제어

Claude Code 통합: Claude가 PR을 자동으로 생성 가능

GitHub CLI 설치: macOS

```
# Homebrew로 설치  
brew install gh  
  
# 설치 확인  
gh --version
```

GitHub CLI 설치: Windows

```
# Scoop 사용  
scoop install gh  
  
# 또는 winget  
winget install GitHub.cli  
  
# 확인  
gh --version
```

GitHub CLI 인증

로그인 과정

```
# 인증 시작
gh auth login

# 프롬프트 응답:
# 1. GitHub.com 선택
# 2. HTTPS 프로토콜 선택
# 3. Authenticate 선택
# 4. 웹 브라우저에서 권한 부여

# 인증 확인
gh auth status
```

Claude Code CLI 설치

시스템 요구사항

- **Node.js:** 18.0 이상
- **RAM:** 최소 4GB
- **네트워크:** 인터넷 연결 필수
- **셸:** Bash, Zsh, Fish 권장

Node.js 설치 (필요시)

```
# macOS  
brew install node  
  
# Windows (WSL 또는 Scoop)  
scoop install nodejs
```

Claude Code CLI 설치: 실행

npm으로 전역 설치

```
# Claude Code 설치
npm install -g @anthropic-ai/claude-code

# 설치 확인
claude --version

# 도움말 확인
claude --help
```

Claude Code 인증

인증 옵션

옵션 1: Claude Console

- <https://console.anthropic.com> 에서 API 키 생성
- 활성 청구 계정 필요
- API 사용량 기반 과금

옵션 2: Claude App (Pro/Max 플랜)

- Claude.ai Pro 또는 Max 구독자
- 웹 앱과 동일한 계정 사용

Claude Code 인증: 실행

초기 실행

```
# Claude Code 시작
claude

# 프롬프트가 표시되면:
# 1. API 키 입력 또는
# 2. Claude App 로그인 선택
```

환경 변수로 설정 (선택사항)

```
# API 키 설정
export ANTHROPIC_API_KEY="your_api_key_here"

# 영구 설정
echo 'export ANTHROPIC_API_KEY="your_key"' >> ~/.zshrc
source ~/.zshrc
```

RealWorld 프로젝트 준비

Fork 및 Clone

```
# 1. GitHub에서 Fork
# https://github.com/1chz/realworld-java21-springboot3
# Fork 버튼 클릭

# 2. Clone (your-username을 본인 계정으로)
git clone https://github.com/your-username/realworld-java21-springboot3.git

# 3. 디렉토리 이동
cd realworld-java21-springboot3
```

프로젝트 빌드 검증

첫 빌드 실행

```
# 전체 빌드 (테스트 포함)
./gradlew clean build

# 예상 소요 시간: 30초 ~ 1분
```

성공 출력:

```
BUILD SUCCESSFUL in 45s
```

설치 확인 체크리스트

모든 도구 테스트

```
# Java 확인  
java --version  
  
# GitHub CLI 확인  
gh --version  
  
# Claude Code 확인  
claude --version  
  
# Git 확인  
git --version
```

Claude Code 기본 사용법

프로젝트에서 시작

```
# 프로젝트 디렉토리에서  
cd realworld-java21-springboot3  
claude  
  
# 대화 시작  
> 이 프로젝트의 구조를 분석해줘  
  
# Claude가 파일을 읽고 분석 시작
```

종료

```
Ctrl+C 또는 /exit
```

Claude Code 편의 기능

Alias 설정 (선택사항)

```
# 승인 없이 계속 작업
echo 'alias claude-yolo="claude --dangerously-skip-permissions"' >> ~/.zshrc
source ~/.zshrc

# 사용
claude-yolo
```

주의: 외부 위험이 없는 격리된 환경에서만 사용!

프로젝트 상태 확인

```
# 현재 세션 상태
claude --status

# 설정 확인
claude config
```

환경 설정 완료 확인

최종 검증

```
# 1. Java 21 실행
java --version | grep "21"

# 2. 프로젝트 빌드
./gradlew clean build

# 3. 애플리케이션 실행
./gradlew realworld:bootRun
# (Ctrl+C로 종료)

# 4. GitHub CLI 인증
gh auth status

# 5. Claude Code 실행
claude --version
```

Tips & Best Practices

개발 환경 관리

- ✓ SDKMAN 사용 - 여러 Java 버전 관리
- ✓ Homebrew 업데이트 - 정기적으로 `brew update`
- ✓ 환경 변수 정리 - `.zshrc` 파일 깔끔하게 유지

Claude Code 사용

- ✓ 프로젝트별 세션 - 각 프로젝트에서 claude 실행
- ✓ 권한 확인 - 중요한 변경은 직접 검토
- ✓ 컨텍스트 유지 - 긴 작업은 한 세션에서

참고 자료

공식 문서

- [Java 21 Release Notes](#)
- [GitHub CLI 문서](#)
- [Claude Code 문서](#)

프로젝트

- [RealWorld Spec](#)
- [Spring Boot 3 문서](#)





Claude Code 프로젝트 설정

CLAUDE.md와 규칙 기반 협업

프로젝트 컨텍스트 관리의 모든 것

이번 세션의 목표

학습 목표

-  CLAUDE.md 파일의 역할과 구조 이해
-  .claude/ 디렉토리 활용법
-  커스텀 명령어 생성
-  규칙 기반 협업 시스템 구축

실습 목표

- 프로젝트 문서 자동 생성
- 작업 규칙 정의
- 반복 작업 자동화

문제 상황: AI와의 협업

전통적인 방식의 문제점

개발자: "UserService 코드를 리팩토링해줘"

Claude: "어떤 규칙을 따라야 하나요?"

"Lombok을 사용하나요?"

"테스트는 어떻게 작성하나요?"

"예외 처리는 어떻게 하나요?"

개발자: (매번 같은 설명 반복...)

비효율: 컨텍스트를 매번 설명해야 함

해결책: 프로젝트 문서화

CLAUDE.md의 역할

CLAUDE.md (프로젝트 가이드)

- 프로젝트 구조
- 빌드 명령어
- 아키텍처 패턴
- 개발 가이드라인



Claude Code가 자동으로 읽음



프로젝트 컨텍스트 이해



적절한 코드 생성

CLAUDE.md란?

공식 컨벤션

- 위치: 프로젝트 루트 디렉토리
- 이름: `CLAUDE.md` (대문자)
- 형식: Markdown
- 역할: Claude Code를 위한 프로젝트 가이드

자동 인식

```
cd your-project  
claude
```

Claude Code가 자동으로 CLAUDE.md 찾아서 내용 읽고 컨텍스트로 활용

CLAUDE.md 생성: 실습

1단계: Claude Code 시작

```
cd realworld-java21-springboot3  
claude
```

2단계: 프로젝트 분석 요청

```
/init
```

CLAUDE.md 생성: 결과

Claude가 생성한 파일

CLAUDE.md

이 파일은 Claude Code가 이 저장소에서 작업할 때 필요한 가이드를 제공합니다.

프로젝트 개요

Java 21, Spring Boot 3, H2 데이터베이스를 사용한 RealWorld API 구현체입니다.

빌드 시스템 & 명령어

애플리케이션 실행

```
./gradlew realworld:bootRun
```


CLAUDE.md 핵심 구조

필수 섹션

1. 프로젝트 개요

- 기술 스택, 주요 기능, 아키텍처 패턴

2. 빌드 시스템

- 실행 명령어, 테스트 명령어, 빌드 명령어

3. 모듈 구조

- 각 모듈의 역할, 의존성 관계

CLAUDE.md 핵심 구조

4. 개발 가이드라인

- 코딩 규칙, 네이밍 컨벤션, 아키텍처 원칙

5. 테스트 전략

- 테스트 작성 규칙, 실행 방법, 커버리지 기준

6. 문제 해결

- 일반적인 오류, 해결 방법

모듈별 CLAUDE.md

계층적 문서 구조

```
realworld-java21-springboot3/
├── CLAUDE.md                # 전체 프로젝트 가이드
├── module/
│   ├── core/               # Core 모듈 상세
│   │   └── CLAUDE.md
│   ├── persistence/       # Persistence 모듈 상세
│   │   └── CLAUDE.md
└── server/
    └── api/                # API 모듈 상세
        └── CLAUDE.md
```

장점: 각 모듈 작업 시 상세 컨텍스트 제공

모듈별 CLAUDE.md 실습

Core 모듈 문서 생성

> module/core 디렉토리를 분석하고 CLAUDE.md를 작성해줘.
이 모듈의 역할과 주요 클래스, 사용 패턴을 설명해줘.

생성 결과

Core Module – Domain Layer

역할

순수한 비즈니스 로직과 도메인 모델을 포함합니다.
다른 모듈에 의존하지 않습니다.

주요 패키지

- `model/`: 도메인 엔티티와 저장소 인터페이스
- `service/`: 비즈니스 로직 서비스

.claude/ 디렉토리 구조

전체 구조

```
.claude/  
├── README.md          # Claude 설정 개요  
├── SETUP.md           # 설정 가이드  
├── rules.md           # 작업 규칙 (핵심!)  
└── commands/          # 커스텀 명령어  
    ├── review.md      # /review 명령어  
    ├── test.md         # /test 명령어  
    ├── new-feature.md  # /new-feature 명령어  
    ├── fix-bug.md      # /fix-bug 명령어  
    └── refactor.md     # /refactor 명령어
```

rules.md: 작업 규칙 정의

CLAUDE.md vs rules.md

CLAUDE.md (What):

- 프로젝트가 무엇인지
- 구조가 어떻게 되어 있는지
- 어떤 도구를 사용하는지

rules.md (How):

- 어떻게 코드를 작성할지
- 어떤 규칙을 따를지
- 어떻게 테스트할지

rules.md 생성: 실습

작업 규칙 정의

> `.claude/rules.md` 파일을 생성하고 다음 규칙들을 정의해줘:

1. 코딩 표준 (Lombok 사용 규칙, 네이밍 규칙)
2. 아키텍처 규칙 (모듈 의존성, 레이어 분리)
3. 테스트 규칙 (특히 예외 메시지 검증 금지!)
4. 데이터베이스 규칙
5. API 설계 규칙
6. 보안 규칙

rules.md 핵심 내용

1. 코딩 표준

코딩 표준

Lombok 사용

- `@RequiredArgsConstructor` : 생성자 주입
- `@Getter` / `@Setter` : 필요시만 사용
- `@Builder` : 복잡한 객체 생성
- `@Value` : 불변 객체

네이밍 규칙

- 클래스: PascalCase
- 메서드/변수: camelCase
- 상수: UPPER_SNAKE_CASE

rules.md 핵심 내용

2. 아키텍처 규칙

아키텍처 규칙

모듈 의존성

api (compileOnly) → core ← persistence (implements)

레이어 분리

- Controller: HTTP 처리만
- Service: 비즈니스 로직
- Repository: 데이터 접근

예외 처리

- Not found: `NoSuchElementException``
- 검증 실패: `IllegalArgumentException``

rules.md 핵심 내용

3. 테스트 규칙

테스트 규칙

❌ 절대 금지: 예외 메시지 검증

// Bad - 메시지 변경 시 테스트 깨짐

```
val exception = shouldThrow<IllegalArgumentException> { ... }  
exception.message shouldBe "User not found"
```

✅ 올바른 방법: 예외 타입만 검증

// Good - 안정적인 테스트

```
shouldThrow<IllegalArgumentException> { ... }
```

이유: 예외 메시지는 변경 가능, 타입은 계약

규칙 참조 강제하기

문제: 규칙을 무시하는 경우

개발자: "UserService에 메서드 추가해줘"
Claude: (rules.md 보지 않고 작성)
→ 규칙 위반 코드 생성

해결: 다층 경고 시스템

레벨 1: 루트 CLAUDE.md에 경고

레벨 2: 각 모듈 CLAUDE.md에 경고

레벨 3: rules.md 자체에 경고

레벨 4: /check-rules 명령어

규칙 참조 강제: 구현

루트 CLAUDE.md 수정

CLAUDE.md

⚠ 필수 규칙

****모든 작업 시 반드시 ``.claude/rules.md`` 파일을 참조해야 합니다.****

이 파일에는 다음이 포함되어 있습니다:

- 코딩 표준
- 아키텍처 규칙
- 테스트 규칙 (예외 메시지 검증 금지!)
- ...

****작업 전 반드시 확인하세요!****

규칙 참조 강제: rules.md

rules.md 상단에 경고

   중요: 이 파일은 모든 작업의 기준입니다   

****이 규칙을 무시하고 작업하지 마세요!****

모든 코드 변경, 새 기능 추가, 버그 수정 시
이 규칙을 반드시 따라야 합니다.

불확실한 경우 `/check-rules` 명령어를 사용하세요.

커스텀 명령어: 개요

명령어란?

Slash 명령어:

```
/review      # 코드 리뷰  
/test        # 테스트 실행  
/new-feature # 새 기능 추가
```

위치: `.claude/commands/`

동작:

1. 사용자가 `/review` 입력
2. Claude가 `.claude/commands/review.md` 읽기
3. 명령어 내용 실행

커스텀 명령어: review.md

코드 리뷰 자동화

/review - 코드 리뷰

당신은 전문 코드 리뷰어입니다.
다음 순서로 코드를 검토하세요:

1. 코딩 표준 검증

- `.claude/rules.md`의 코딩 표준 확인

2. 아키텍처 규칙 검증

- 모듈 의존성 방향 확인
- 레이어 분리 원칙 준수

3. 테스트 규칙 검증

- 예외 메시지 검증 여부 확인 (금지!)
- 테스트 네이밍 규칙 확인

커스텀 명령어: test.md

테스트 자동화

```
# /test - 테스트 실행 및 검증

## 단계

### 1. 테스트 실행
./gradlew test

### 2. 커버리지 확인
./gradlew jacocoRootReport

### 3. 리포트 요약
- 전체 커버리지 퍼센트
- 개선이 필요한 영역
```


커스텀 명령어: check-rules.md

규칙 확인 명령어

/check-rules - 규칙 확인

`./claude/rules.md` 파일을 읽고 다음 작업과 관련된 규칙을 요약하세요:

입력

사용자가 수행하려는 작업 설명

출력

해당 작업에 적용되는 규칙 목록:

- 코딩 표준
- 아키텍처 규칙
- 테스트 규칙

예시

커스텀 명령어 사용: 실습

/check-rules 사용

> /check-rules 새 기능 추가

Claude 응답:

"새 기능 추가 시 다음 규칙을 따라야 합니다:

1. 모듈 순서: Core → Persistence → API
 2. Lombok 어노테이션 사용
 3. 예외 처리: IllegalArgumentException, NoSuchElementException
 4. 테스트: 예외 타입만 검증, 메시지 검증 금지
 5. 커밋 전: ./gradlew spotlessApply 실행
- ..."

커스텀 명령어 셋 사용: Super Claude

Super Claude 사용

https://github.com/SuperClaude-Org/SuperClaude_Framework

```
pip install SuperClaude && pip upgrade SuperClaude && SuperClaude install
```

"https://github.com/SuperClaude-Org/SuperClaude_Framework"에 나와있는 프레임워크 설치해줘!

커스텀 명령어 셋 사용: Super Claude

Super Claude 사용 (이미 잘 구현된 커스텀 명령어)

```
/sc:implement "@prd.md 보고 구현해줘." --type backend --focus performance
```

```
/sc:document "git 최신 커밋 3개의 변경사항을 보고 문서화해줘."
```

전체 시스템 아키텍처

Claude Code 협업 시스템

CLAUDE.md (프로젝트 구조)

- 빌드 명령어
- 모듈 아키텍처
- 개발 가이드



.claude/rules.md (작업 규칙)

- 코딩 표준
- 아키텍처 규칙
- 테스트 규칙



.claude/commands/ (워크플로우)

- /review, /test
- /new-feature, /fix-bug

실전 워크플로우 예시

시나리오: 새 기능 추가

```
# 1. 규칙 확인
> /check-rules 새 기능 추가

# 2. 기능 구현
> /sc:implement "게시글 북마크"




# Claude가 자동으로:
# - Core 모듈 작업, Persistence 모듈 작업, API 모듈 작업, 테스트 작성

# 3. 검증
> /review, /test



# 4. 빌드 확인
./gradlew build
```

핵심 성과



일관성 있는 협업

-  새로운 Claude 세션도 프로젝트 즉시 이해
-  동일한 규칙으로 일관된 코드 작성
-  컨텍스트 손실 최소화

자동화된 워크플로우

-  반복 작업을 명령어로 자동화
-  실수 가능성 감소

명확한 규칙 관리

-  프로젝트 구조와 작업 규칙 분리
-  규칙 변경 시 한 곳만 수정

Best Practices

CLAUDE.md 작성

- ✓ 명확하고 간결하게 - 핵심 정보만
- ✓ 예시 포함 - 명령어 실행 예시
- ✓ 최신 상태 유지 - 프로젝트 변경 시 업데이트

rules.md 작성

- ✓ 구체적으로 - 모호하지 않게
- ✓ 예시 코드 - Good/Bad 예시

커스텀 명령어

- ✓ 단일 책임 - 한 명령어는 한 가지 작업
- ✓ 명확한 이름 - /review, /test 등

고급 활용: 팀 협업

팀 규칙 공유

```
.claude/
├── rules.md           # 팀 전체 규칙
├── rules-frontend.md # 프론트엔드 규칙
├── rules-backend.md  # 백엔드 규칙
└── commands/
    ├── review.md     # 공통 리뷰
    ├── review-fe.md  # FE 전용 리뷰
    └── review-be.md  # BE 전용 리뷰
```

사용:

```
> .claude/rules-backend.md 규칙을 따라  
   새 API를 추가해줘
```

실습 체크리스트

완료 확인

- [] 루트 CLAUDE.md 생성
- [] 모듈별 CLAUDE.md 생성
- [] .claude/rules.md 작성
- [] 커스텀 명령어 3개 이상 생성
- [] 커스텀 명령어 테스트
- [] 새 Claude 세션에서 동작 확인

모두 완료하면 다음 세션으로!

참고 자료

문서

- [Claude Code 공식 문서](#)

예시 프로젝트

- [RealWorld Claude Config](#)
- [SuperClaude Framework](#)






Claude Code 고급 활용법

MCP, Plan 모드, YOLO 모드 마스터하기

생산성을 극대화하는 고급 기법들

이번 세션의 목표

학습 목표

-  MCP (Model Context Protocol) 이해 및 활용
-  Context7 MCP로 공식 문서 참조하기
-  Atlassian MCP로 Jira/Confluence 연동하기
-  Plan 모드로 구현 전 계획 수립하기
-  YOLO 모드로 빠른 실행하기

실습 목표

- MCP 서버 설치 및 설정
- Plan 모드로 기능 설계
- YOLO 모드로 신속한 개발

Part 1: MCP (Model Context Protocol)

MCP란?

Model Context Protocol: Claude Code가 외부 시스템과 통신하는 표준 프로토콜

```
Claude Code ←MCP→ Context7 (공식 문서)
               ←MCP→ Atlassian (Jira/Confluence)
               ←MCP→ GitHub (Issues/PRs)
               ←MCP→ Database (Query/Schema)
```

핵심 개념:

- 표준화된 통신 규약
- 플러그인 형태의 확장성
- 실시간 외부 데이터 접근

MCP의 필요성

기존 방식의 한계

개발자: "Spring Boot 3의 @ConfigurationProperties 사용법 알려줘"

Claude: (학습 데이터 기반 답변)

- 최신 버전 정보 부족
- 공식 문서와 다를 수 있음
- 프레임워크 업데이트 반영 불가

MCP 사용 시

개발자: "Spring Boot 3의 @ConfigurationProperties 사용법 알려줘"

Claude → Context7 MCP → Spring Boot 공식 문서

- 최신 공식 문서 직접 참조
- 정확한 코드 예시
- 버전별 차이점 명확

MCP 서버 종류

1. Context7 (문서 검색)

- 용도: 공식 라이브러리 문서 검색
- 지원: Spring, React, Vue, Angular, Python 등

2. Atlassian (협업)

- 용도: Jira 이슈, Confluence 문서
- 기능: 이슈 조회/생성, 문서 검색

3. GitHub

- 용도: Issues, Pull Requests, Code
- 기능: 이슈 관리, PR 리뷰, 코드 검색

Context7 MCP 설정

1단계: Smithery 가입 및 Claude code 선택

<https://smithery.ai/server/@upstash/context7-mcp>

2단계: 설치 스크립트 복사 및 붙여넣고 실행

```
claude mcp add --transport http upstash-context-7-mcp "https://server.smithery.ai/@upstash/context7-mcp/mcp"
```

3단계: 설치된 MCP 목록 확인

```
/mcp
```

Context7 MCP 사용법

기본 사용

```
# Claude Code에서 직접 요청
> Spring Boot 3의 최신 @ConfigurationProperties 사용법을
  공식 문서를 참조해서 알려줘

# Claude가 자동으로:
1. Context7 MCP 호출
2. Spring Boot 공식 문서 검색
3. 최신 문서 기반 답변
```

Context7 활용 시나리오

시나리오 1: 최신 API 사용

> Spring Boot 3.2에서 추가된 Virtual Threads 설정 방법을 공식 문서를 참조해서 알려줘

Claude → Context7 → Spring Boot 3.2 문서
→ application.yaml 설정
→ @Async 어노테이션 사용
→ ThreadPoolTaskExecutor 설정

시나리오 2: 버전별 차이

> React 17과 18의 useEffect 차이점을 공식 문서 기준으로 비교해줘

Claude → Context7 → React 17/18 문서
→ Automatic Batching 변경사항
→ StrictMode 동작 차이
→ 마이그레이션 가이드

Context7 활용 시나리오

시나리오 3: 실전 코드 생성

> Spring Data JPA의 Specification을 사용한 동적 쿼리를
공식 문서 예시를 참고해서 구현해줘

Claude → Context7 → Spring Data JPA 문서

- 정확한 인터페이스 구현
- 공식 권장 패턴
- 타입 안전성 보장

장점: 잘못된 API 사용 방지, 최신 베스트 프랙티스 적용

Atlassian MCP 설정

1단계: API 토큰 생성

<https://github.com/atlassian/atlassian-mcp-server> 를 보고 MCP 설치해줘! OAuth 인증방식을 사용한다.

2단계: OAuth 웹 로그인하기

Atlassian MCP 사용법

Jira 이슈 조회

> PROJ-123 이슈의 상세 내용을 확인해줘

Claude → Atlassian MCP → Jira

- 이슈 제목, 설명
- 현재 상태, 담당자
- 댓글, 첨부파일

Atlassian MCP 활용 시나리오

시나리오 1: 이슈 기반 개발

> PROJ-456 이슈의 요구사항을 읽고 구현 계획을 세워줘

Claude:

1. Atlassian MCP로 이슈 조회
2. Acceptance Criteria 분석
3. 구현 단계 계획 수립
4. 필요한 API 설계
5. 테스트 시나리오 작성

Atlassian MCP 활용 시나리오

시나리오 2: 이슈 자동 업데이트

> 이 기능 구현 완료했으니 PROJ-789 이슈를
"Done" 상태로 변경하고 구현 내용을 코멘트로 남겨줘

Claude:

1. 구현 내용 정리
2. Atlassian MCP로 이슈 업데이트
3. 상태 변경 (In Progress → Done)
4. 구현 상세 내용 코멘트 추가
5. 관련 커밋 링크 추가

효과: 개발과 프로젝트 관리 자동 연동

MCP 조합 활용

예시: 완벽한 워크플로우

```
# 1. Jira에서 요구사항 확인
> PROJ-100 이슈 확인

# 2. Confluence에서 설계 문서 참조
> "마이크로서비스 아키텍처" 문서 검색

# 3. Context7로 최신 프레임워크 문서 참조
> Spring Cloud Gateway 최신 설정 방법

# 4. 구현
> 위 정보를 바탕으로 API Gateway 구현

# 5. 이슈 업데이트
> PROJ-100을 "In Review"로 변경하고 PR 링크 추가
```


Plan 모드 활성화

방법 1: 명시적 요청

> Plan 모드로 게시글 북마크 기능을 설계해줘

방법 2: 복잡한 요청 시 자동 활성화

> 마이크로서비스로 전환하고 싶어.
현재 모놀리식 구조를 분석하고 마이그레이션 계획 세워줘

Claude가 자동으로 복잡도를 감지하고 Plan 모드 진입

Plan 모드 Best Practices

✅ 언제 사용하면 좋은가?

1. 복잡한 기능 추가

- 여러 모듈에 걸친 변경
- 데이터베이스 스키마 변경
- 새로운 기술 도입

2. 대규모 리팩토링

- 아키텍처 변경
- 레거시 코드 개선
- 모듈 재구성

3. 성능 최적화

Plan 모드 Best Practices

✓ 효과적인 사용법

1. 구체적인 컨텍스트 제공

Bad: "성능 개선해줘"
Good: "게시글 목록 API가 500ms 걸려.
N+1 쿼리 문제가 있는 것 같아.
Plan 모드로 최적화 계획 세워줘"

2. 제약사항 명시

"Plan 모드로 설계해줘. 단,
- 기존 API 스펙은 유지
- 데이터베이스 마이그레이션 최소화
- 테스트 커버리지 80% 이상 유지"

3. 단계별 피드백

Part 3: YOLO 모드 (권한 검사 하지 않음)

주의: 프로덕션 코드보다는 실험, 학습, 프로토타입에 적합

YOLO 모드 활성화

```
echo 'alias claude-yolo="claude --dangerously-skip-permissions"' >> .zshrc  
source .zshrc
```

claude-yolo

- Claude code 승인 없이 계속 작업하게 만듦
- 코드 작업과 같이 외부 위험한 작업과 격리됐을 때만 활용