

GitHub 모노레포 템플릿: Codex 및 Claude Code로 구현하는 완전 자동화 워크플로우

개요 및 목표

모노레포(Monorepo) 구조의 웹 앱 프로젝트(프론트엔드 + 백엔드)를 대상으로 **GitHub Actions**와 AI 도구(OpenAI Codex와 Anthropic Claude Code)를 활용하여 **완전 자동화된 개발 워크플로우**를 구축합니다. GitHub Issues와 Projects 보드를 통해 작업을 추적하면서, Pull Request(PR) 생성부터 코드 리뷰, 수정 반영, CI 빌드, 배포, 문서 업데이트까지 대부분의 단계를 자동화합니다. 이를 위해 **GitHub 내부 통합**(GitHub 앱 및 댓글 트리거) 방식으로 Codex와 Claude Code를 연동하며, 직접적인 API 호출은 사용하지 않습니다. 아래에서는 예시적인 모노레포 폴더 구조, 필요한 설정 파일, GitHub Actions 워크플로우 YAML 구성, 그리고 Codex/Claude 통합 설정 방법을 상세히 설명합니다.

모노레포 폴더 구조 예시

프로젝트의 폴더 구조는 프론트엔드와 백엔드를 한 레포지토리에서 관리할 수 있도록 설계합니다. 또한 문서와 GitHub 관련 설정도 포함합니다. 예를 들어:

```
repository-root/
├── frontend/           # 프론트엔드 소스 (예: React 앱)
│   ├── package.json
│   └── src/...
├── backend/           # 백엔드 소스 (예: Node/Express 또는 Python Flask 등)
│   ├── package.json or requirements.txt
│   └── src/...
├── docs/              # 프로젝트 문서 (API 명세서 등)
│   └── ... (.md 문서들)
├── README.md
├── .github/
│   ├── workflows/    # GitHub Actions 워크플로우 정의
│   │   ├── pr-review.yml      # PR 생성 시 자동 코드리뷰 (Codex)
│   │   ├── ai-comment.yml     # AI 댓글 트리거 처리 (Claude Code)
│   │   ├── frontend-ci.yml   # 프론트엔드 CI (빌드/테스트)
│   │   ├── backend-ci.yml    # 백엔드 CI (빌드/테스트)
│   │   ├── deploy.yml        # 메인 브랜치 배포 (프론트/Vercel, 백엔드/Docker)
│   │   ├── docs-sync.yml     # 문서 자동 업데이트 워크플로우
│   │   └── dependabot.yml    # Dependabot 설정 (의존성 업데이트)
│   ├── AGENTS.md       # (선택) Codex 코드리뷰 지침 파일 1
│   └── CLAUDE.md        # (선택) Claude Code 안내/규칙 파일 2
```

위 구조에서 `.github/workflows` 내 여러 YAML 파일로 각 기능별 Actions를 정의합니다. 또한 OpenAI Codex의 코드리뷰 기준을 커스터마이징하려면 `AGENTS.md`를, Claude Code의 행동 지침을 줄 때는 `CLAUDE.md`를 루트에 추가할 수 있습니다. Codex는 PR 코드리뷰 시 `AGENTS.md`의 **Review guidelines** 섹션을 자동으로 참고하며 ¹, Claude도 `CLAUDE.md`의 규칙과 프로젝트 패턴을 존중하여 동작합니다 ³ ².

OpenAI Codex 통합 - 자동 PR 코드 리뷰

OpenAI Codex(GPT 기반 코드 모델)을 GitHub에 연동하면, PR에 대한 **자동 코드 리뷰**를 수행할 수 있습니다. 먼저 OpenAI의 Codex Cloud 서비스를 세팅하고 해당 리포지토리에서 **Codex 코드리뷰** 기능을 활성화해야 합니다 ⁴. 설정 후에는 PR에서 `@codex review`라고 댓글을 남겨 Codex에게 리뷰를 요청할 수 있습니다 ⁵. Codex는 PR의 변경사항을 분석한 후 사람 코드리뷰어처럼 코멘트를 남겨주며, 발견 사항이 없으면 요약 코멘트를 남깁니다 ⁶.

자동 트리거: 매번 댓글을 달 필요 없이 PR 오픈 시 자동 리뷰를 원한다면, GitHub Actions로 Codex를 호출할 수 있습니다. OpenAI가 제공하는 공식 Action인 `openai/codex-action`을 사용하면 Codex를 워크플로우에서 실행하고, 결과를 PR 코멘트로 남길 수 있습니다 ⁷ ⁸. 예를 들어 `pr-review.yml` 워크플로우를 아래처럼 구성합니다:

```
name: AI Code Review (Codex)
on:
  pull_request:
    types: [opened] # PR 생성시에만 트리거
jobs:
  codex_review:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      pull-requests: write # PR 코멘트 작성을 위해 권한 필요
    steps:
      - uses: actions/checkout@v5
        with:
          # PR의 병합 커밋 기준으로 체크아웃 (Codex가 전체 diff를 보도록)
          ref: refs/pull/${{ github.event.pull_request.number }}/merge
      - name: Run Codex Review
        id: run_codex
        uses: openai/codex-action@v1
        with:
          openai-api-key: ${ secrets.OPENAI_API_KEY }
          prompt: |
            This is PR #${{ github.event.pull_request.number }} for $
            {{ github.repository }}.
            Review ONLY the changes introduced by this PR.
            Suggest improvements, potential bugs, or issues with specific
            feedback.
            Pull request title and description:
            ----
            ${{ github.event.pull_request.title }}
```

```

    ${{ github.event.pull_request.body }} 9 10
- name: Post Codex feedback
  if: ${{ steps.run_codex.outputs.final-message != '' }}
  uses: actions/github-script@v7
  with:
    github-token: ${{ secrets.GITHUB_TOKEN }}
    script: |
      await github.rest.issues.createComment({
        owner: context.repo.owner,
        repo: context.repo.repo,
        issue_number: context.payload.pull_request.number,
        body: process.env.CODEX_FINAL_MESSAGE
      });
  env:
    CODEX_FINAL_MESSAGE: ${{ steps.run_codex.outputs.final-message }}

```

위 YAML은 PR이 열리면 Codex를 실행하여 변경 코드를 검사하고, 결과를 PR에 댓글로 남깁니다 8 11. Codex Action은 Codex CLI를 사용하므로 OPENAI_API_KEY 시크릿 설정이 필요합니다 7. Codex가 생성한 리뷰 코멘트에는 개선사항, 잠재 버그, 스타일 위반 등을 구체적으로 지적하게 할 수 있으며, AGENTS.md에 팀별 코드 리뷰 규칙을 정의해두면 Codex가 이를 참고해 리뷰 품질을 높입니다 1.

Anthropic Claude Code 통합 - AI 코드 수정 및 자동 응답

Anthropic Claude Code는 GitHub에서 동작하는 AI 코딩 에이전트로, 댓글 명령을 인식하여 **코드 수정 커밋**, **새로운 PR 생성**, **Q&A 답변** 등을 수행할 수 있습니다 12 13. Codex가 리뷰어 역할이라면, Claude Code는 **제안 적용 및 구현자 역할**을 맡습니다. **GitHub 앱**으로 Claude를 설치하고 Actions에서 Anthropic API Key를 설정하면 연동이 가능합니다 14 15. 주요 설정 단계는 다음과 같습니다:

- **Claude GitHub App 설치:** Anthropic이 제공하는 GitHub App(**Claude**)을 레포지토리에 설치하고 권한을 부여합니다. 필요한 권한은 **Contents: Read/Write, Issues: Read/Write, Pull requests: Read/Write**입니다 16.
- **API Key 시크릿 추가:** Anthropic Claude API 키를 레포지토리 시크릿 ANTHROPIC_API_KEY로 추가합니다 17.
- **워크플로우 생성:** Claude Code 액션(anthropics/claude-code-action@v1)을 사용하는 YAML을 .github/workflows에 만들어 둡니다. 예시 ai-comment.yml에서는 PR이나 이슈에서 특정 댓글이 달릴 때 Claude를 실행하도록 합니다.

Claude Code 액션은 **지능형 모드 감지** 기능이 있어, 워크플로우 트리거 상황에 따라 자동으로 동작 모드를 선택합니다 18. 예를 들어 PR이나 이슈 댓글에 @claude가 언급되면 해당 맥락에서 질문에 답하거나 코드를 수정하는 작업을 수행합니다. 이슈가 assign되면 해당 이슈 구현을 도와주도록 동작할 수도 있습니다 12 13.

PR 수정 적용: 사용자가 PR 리뷰 후 "@claude apply"와 같이 댓글을 남기면, Claude가 앞서 나온 리뷰 코멘트를 반영하여 코드를 수정하고 커밋을 PR 브랜치에 푸시하도록 설정할 수 있습니다. 이를 위해 issue_comment 이벤트를 포착하는 워크플로우를 작성합니다:

```

name: Claude Apply Suggestions
on:
  issue_comment:
    types: [created] # 새로운 댓글이 달릴 때 트리거
jobs:
  apply_changes:
    if: |
      github.event.issue.pull_request && # 이슈가 PR에 연관된 경
      우 (PR의 댓글인 경우)
      contains(github.event.comment.body, '@claude apply') # 댓글 내용에
"@claude apply" 명령이 포함된 경우
    runs-on: ubuntu-latest
    permissions:
      contents: write # 브랜치에 커밋 푸시 권한
      pull-requests: write # PR 업데이트 권한
    steps:
      - name: Checkout PR branch
        uses: actions/checkout@v4
        with:
          ref: ${ github.event.issue.pull_request.head.ref } # PR의 헤드 브랜치
      - name: Claude Apply Changes
        uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "/apply"

```

위 워크플로우는 PR에 달린 댓글 중 `@claude apply`를 감지하면 실행됩니다 ¹⁹. `prompt: "/apply"`는 Claude에게 **이전 대화(Context)에서 나온 수정 제안을 코드에 적용하라**는 내장 명령입니다. Claude Code 액션은 PR의 변경사항과 대화맥락(이전에 Codex가 남긴 리뷰 코멘트 등)을 읽어들이고, 필요한 코드를 수정하고 커밋까지 수행합니다. 이러한 **Slash 명령** (`/review`, `/fix`, `/apply` 등)을 사용하면 복잡한 프롬프트 없이도 표준 동작을 실행할 수 있습니다 ²⁰ ²¹. 예를 들어 PR 생성 시 Claude에게 자동 리뷰를 시키고 싶다면 `prompt: "/review"`로 설정할 수 있습니다 ²² ²³. Claude Code는 커밋이나 PR 생성 등 GitHub 상의 액션도 수행할 수 있으며, `use_sticky_comment: true` 등의 옵션으로 한 PR의 동일한 댓글을 업데이트하는 등 세부 설정도 가능합니다 ²⁴ ²⁵.

참고: Claude Code 액션은 기본적으로 PR/이슈의 댓글 맥락에서 `@claude` 멘션을 찾으면 자동으로 응답을 댓글로 달아줍니다 ²⁶. 별도의 워크플로우 없이도, `@claude`로 시작하는 댓글에 Claude가 반응하여 답변하거나 코드를 수정하는 코멘트를 달 수 있습니다. 그러나 코드 커밋과 같은 **실제 코드 변경을 자동화**하려면 위의 워크플로우처럼 `contents: write` 권한을 주고 실행해야 합니다. Claude의 동작 범위와 안전을 관리하기 위해 `CLAUDE.md`에 코드 스타일, 아키텍처 지침 등을 기술하면 Claude가 이를 준수합니다 ².

이슈 생성 시 구현 계획 자동 댓글

새로운 이슈가 생성되면, AI가 이슈 내용을 분석해 **구현 계획**이나 해결 아이디어를 자동으로 댓글로 남겨줍니다. 이 기능은 Codex나 Claude 중 하나를 선택해서 구현할 수 있습니다. 여기서는 Claude를 활용하는 방법을 설명합니다 (Codex의 경우 OpenAI API를 비슷하게 활용 가능). Claude는 이슈 내용과 프로젝트 맥락을 고려하여 어떤 기능을 어떻게 구현할지 계획을 제안해줄 수 있습니다.

워크플로우 설정: `issues: opened` 이벤트를 잡아 Claude를 실행하고, 응답을 이슈 코멘트로 남깁니다. Claude Code 액션은 이슈 이벤트에 대해서도 동작하며, 별도 프롬프트를 주어야 합니다. 예시 YAML:

```
name: Issue Plan by Claude
on:
  issues:
    types: [opened]
jobs:
  plan_comment:
    runs-on: ubuntu-latest
    permissions:
      issues: write    # 이슈에 코멘트 작성 권한
      contents: read   # (필요시) 레포지토리 내용 읽기 권한
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: "${{ secrets.ANTHROPIC_API_KEY }}"
          prompt: |
            Analyze the issue titled "${{ github.event.issue.title }}".
            Issue description:
              "${{ github.event.issue.body }}"

            Provide a concise implementation plan as a bullet list, covering how to solve
            this issue.
```

위 설정은 새 이슈가 열리면 Claude가 해당 이슈의 제목과 본문을 읽고, 해결을 위한 구현 방안을 **요약된 bullet list** 형태로 작성하도록 프롬프트를 주고 있습니다. 권한상 `issues: write`가 있으므로 Claude 액션은 곧바로 해당 이슈에 댓글을 달 수 있습니다. (Claude Code 액션은 PR/Issue 컨텍스트에서 실행될 경우, 별도 지시가 없더라도 기본적으로 응답을 코멘트로 남기는 기능이 있습니다.) 이로써 팀원들이 이슈를 등록하면 AI가 자동으로 “어떤 접근으로 이 문제를 해결할 수 있을지” 계획을 제안해주므로, 개발 초기 방향을 잡는 데 도움이 됩니다.

Codex를 활용하는 대안: OpenAI Codex를 사용해도 유사한 일을 할 수 있습니다. `issues: opened` 시 Codex Action을 실행하고, `prompt` 에 이슈 내용 기반의 해결 계획 요청을 적은 뒤, `actions/github-script` 로 그 결과를 이슈 코멘트로 남기면 됩니다. 다만 Codex의 GitHub 통합은 주로 PR 맥락에 초점을 두고 있으므로, 이슈에 대한 자동 응답은 Claude로 구현하는 편이 더 간편합니다.

프론트엔드/백엔드 분리 CI 파이프라인

모노레포에서는 프론트엔드와 백엔드의 CI(빌드/테스트)를 각각 독립적으로 실행하여 불필요한 작업을 줄일 수 있습니다. GitHub Actions는 **path 필터링**을 지원하므로, 특정 디렉토리의 파일 변경이 있을 때만 해당 워크플로우를 트리거하면 됩니다 ²⁷. 설정 방법은 두 가지입니다:

- **워크플로우 별 분리:** `frontend-ci.yml` 과 `backend-ci.yml` 두 파일로 분리하고, 각 워크플로우의 `on:`에 경로 필터를 지정합니다. 예를 들어 프론트엔드 CI 워크플로우에서는:

```
on:
  push:
    branches: [main, develop]    # 예: main이나 develop 브랜치로 푸시될 때
    paths:
      - 'frontend/**'           # frontend/ 아래 파일이 수정된 커밋일 경우에만 실행
  pull_request:
    paths:
      - 'frontend/**'
jobs:
  frontend_tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v3
        with:
          node-version: 18
      - name: Install & Test (Frontend)
        working-directory: frontend
        run: |
          npm ci
          npm run build && npm test
```

백엔드의 경우 paths를 `'backend/**'`로 걸고, 언어에 맞는 세팅(Java의 Maven, Python의 pip 등)을 진행하면 됩니다. 위와 같이 설정하면 **frontend 디렉토리 내 변경사항이 있을 때만** 프론트엔드 CI가 동작합니다 ²⁷. 백엔드도 동일하게 설정합니다.

- **단일 워크플로우+조건부 실행:** 하나의 워크플로우 파일 안에 두 Job을 정의하고, 각 Job 단위에서 `if:` 조건으로 변경 감지(Action 컨텍스트의 `github.event` 활용)를 넣을 수도 있습니다. 예를 들어:

```
jobs:
  fe-ci:
    if: contains(github.event.head_commit.message, 'frontend/') # 커밋 메시지에 경로 포함 여부로 간단 체크
    ...
  be-ci:
```

```
if: contains(github.event.head_commit.message, 'backend/')
...
```

하지만 이 방식은 정확도가 떨어질 수 있어, 권장되는 방법은 각 서비스별로 워크플로우를 나누고 **paths** 필터를 사용하는 것입니다.

각 CI Job에서는 해당 부분의 빌드 및 테스트를 실행합니다. 예를 들어 프론트엔드는 Node 기반이라면 `npm install / npm test` 등을, 백엔드는 Python이라면 `pip install` / 테스트 명령 등을 수행합니다. 이렇게 분리하면 **한쪽 코드만 수정해도 전체 파이프라인이 도는 것을 방지하여** 효율을 높일 수 있습니다.

메인 브랜치 머지 시 자동 배포

메인(main) 브랜치에 코드가 머지되면 **프론트엔드와 백엔드의 배포**를 자동화합니다. 각 파트의 배포 환경이 다르므로 별도로 다룹니다:

- **프론트엔드 - Vercel 배포:** 프론트엔드는 정적 호스팅 또는 서버리스 환경에 배포됩니다. 일반적으로 Vercel을 사용한다면, **GitHub 연동**을 통해 main 브랜치에 푸시될 때 자동으로 프로덕션 배포가 일어나도록 설정할 수 있습니다²⁸. Vercel의 GitHub Integration을 활성화하면, PR마다 Preview Deploy를 생성하고 main 머지 시 Production에 배포해줍니다. 추가로 GitHub Actions에서 제어하고 싶다면 **Deploy to Vercel** 액션을 사용할 수 있습니다²⁹. 이 경우 Vercel의 토큰을 시크릿으로 저장하고, main 브랜치 푸시 이벤트에 다음과 같은 단계를 추가합니다:

```
- name: Deploy to Vercel
  uses: amondnet/vercel-action@v20
  with:
    vercel-token: ${ secrets.VERCEL_TOKEN }
    vercel-org-id: ${ secrets.VERCEL_ORG_ID }
    vercel-project-id: ${ secrets.VERCEL_PROJECT_ID }
    working-directory: frontend
    prod: true
```

이는 main 브랜치에 변화가 있고 `frontend/**` 경로에 변경이 있는 경우에 실행되도록 paths 필터를 함께 설정합니다. Vercel Action은 로컬에서 Vercel로 배포를 트리거하며, Vercel 대시보드에서도 배포 이력을 확인할 수 있습니다.

- **백엔드 - Docker 이미지 배포:** 백엔드는 Docker 이미지로 빌드하여 레지스트리에 push하고, 필요시 해당 이미지를 사용해 배포합니다. 예를 들어 GitHub Container Registry(ghcr.io)나 Docker Hub를 사용합니다. main 브랜치에 변경(특히 `backend/**` 경로)이 있을 때 아래 작업을 수행합니다:

```
on:
  push:
    branches: [main]
    paths:
```

```

- 'backend/**'
jobs:
  deploy-backend:
    runs-on: ubuntu-latest
    permissions:
      packages: write # GHCR 푸시 권한
    steps:
      - uses: actions/checkout@v4
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v2
        with:
          registry: ghcr.io
          username: ${ github.actor }
          password: ${ secrets.GITHUB_TOKEN } # GHCR은 GITHUB_TOKEN으로 푸
시 인증 가능
      - name: Build and Push Docker Image
        uses: docker/build-push-action@v4
        with:
          context: ./backend
          push: true
          tags: ghcr.io/<your-org>/<your-repo>/backend:latest

```

위 예시는 GitHub Container Registry에 `backend:latest` 이미지를 빌드/푸시합니다. Docker Hub를 쓰는 경우 `registry: docker.io` 및 Docker Hub 자격증명 시크릿을 사용하도록 설정하면 됩니다. 이렇게 이미지를 푸시한 후에는 쿠버네티스나 서버가 그 이미지를 받아 배포하도록 추가 파이프라인을 구성할 수 있습니다. (질문 범위를 벗어나므로 상세 생략)

배포 워크플로우 분리: 프론트엔드와 백엔드 배포를 각각 워크플로우로 분리하거나, 하나의 `deploy.yml` 안에 두 Job으로 정의할 수 있습니다. `paths` 필터를 활용하여 front만 변경됐을 땐 백엔드 Job을 `if:` 로 스킵하고, 반대의 경우 프론트 배포를 스킵하도록 제어하면 효율적입니다.

코드 변경 시 문서 자동 업데이트

코드 수정에 따라 문서(`docs/**` 디렉토리의 **Markdown 파일** 혹은 **주요 README**)를 최신 상태로 유지하는 작업도 자동화합니다. 이를 위해 AI 도구를 활용하여 코드 변경 내역을 분석하고, 문서에 반영해야 할 내용을 찾아 자동으로 수정하거나 수정 제안 PR을 생성합니다. Anthropic Claude Code를 CLI로 사용하면 이러한 **Documentation Sync** 작업을 구현할 수 있습니다 ³⁰ ³¹.

예를 들어, **메인 브랜치에 코드 변경이 푸시되면** 다음과 같은 워크플로우를 실행합니다 (`docs-sync.yml`):

```

name: Sync Documentation
on:
  push:

```

```

branches: [main]
paths:
  - 'backend/**'
  - 'frontend/**'
jobs:
  update-docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Generate documentation updates with Claude
        run: |
          # 코드 변경 diff 추출 (예: 마지막 커밋 기준)
          DIFF=$(git diff HEAD~1 HEAD -- 'backend/**' 'frontend/**')

npx @anthropic-ai/claude-code@latest -p "Update documentation based on these
code changes:
  ${DIFF}
  Update:
  - API documentation if any endpoints changed
  - README if major features added or changed
  - Relevant usage examples if needed
  Keep the documentation structure and tone consistent." --allowedTools
Edit,Write 32 33
  - name: Create docs PR
    uses: peter-evans/create-pull-request@v5
    with:
      title: "docs: sync with recent code changes"
      commit-message: "docs: automated documentation update"
      branch: docs/auto-update 34

```

위 단계에서는: 1. **코드 diff 수집**: `git diff` 로 최근 커밋의 코드 변경사항을 추출합니다. 2. **Claude를 사용한 문서 업데이트**: `claude-code` CLI(예: npm 패키지로 설치)에게 diff를 전달하며 프롬프트를 줍니다. "이 코드 변경에 기반하여 문서를 업데이트하라"고 지시하면 Claude가 `docs/` 내 관련 파일이나 `README.md` 를 수정합니다 ³² . `--allowedTools Edit,Write` 옵션은 Claude에게 파일 수정 권한을 부여합니다. 3. **PR 생성**: 수정된 문서를 새 브랜치에 커밋하고, PR을 자동 생성합니다 ³⁴ . 이렇게 하면 문서 변경을 리뷰하고 머지할 수 있습니다 (즉, 문서를 바로 main에 커밋하기보다 PR로 검토).

Claude Code 액션을 사용할 경우에도 유사하게 구현 가능합니다. Claude 액션 내부에서 `prompt` 로 diff를 주고 문서 파일들에 대한 수정을 요청하면, Claude가 곧바로 커밋까지 수행하도록 설정할 수도 있습니다 ³² ³³ . 다만 자동 커밋은 예기치 않은 수정을 할 수 있으므로, 위 예시처럼 PR을 생성하여 사람이 검토 후 머지하는 프로세스가 안전합니다.

이로써 개발자가 코드만 수정해도 관련된 문서가 뒤따라 업데이트되거나, 최소한 업데이트 제안이 이루어집니다. 특히 API 명세, 사용법 예시, 주요 변경 로그 등을 누락없이 관리할 수 있습니다.

의존성 업데이트 자동 PR (Dependabot)

프로젝트의 프론트엔드 및 백엔드 의존성을 주기적으로 검사하여 **업데이트 PR**을 자동으로 생성합니다. GitHub의 Dependabot 기능을 활용하면 별도 서버나 액션 없이도 설정 파일만으로 이 기능을 구현할 수 있습니다. `.github/dependabot.yml` 파일을 추가하고 각 패키지 매니저별로 설정합니다:

```
# .github/dependabot.yml
version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/frontend"
    schedule:
      interval: "weekly"
      day: "sunday"
      time: "04:00"
  - package-ecosystem: "npm"
    directory: "/backend"
    schedule:
      interval: "weekly"
      day: "sunday"
      time: "04:30"
```

위 설정은 매주 일요일 새벽에 프론트엔드와 백엔드 디렉토리의 `package.json` (또는 해당 언어의 의존성 파일)을 확인하여 업스트림 새 버전이 있으면 PR을 올립니다 ³⁵ ³⁶. 모노레포이므로 각 하위 프로젝트별로 `directory`를 지정하여 Dependabot이 각각 처리하도록 합니다 ³⁵. **자동 머지 금지:** Dependabot의 기본 동작은 생성된 PR을 자동 머지하지 않고 대기시키는 것입니다. 따라서 위 PR들은 리뷰/CI 후에 직접 머지해야 하므로, 안정성을 확보할 수 있습니다. (Dependabot으로 생성된 브랜치도 Actions CI를 거치므로 테스트를 통과하는지 확인 가능합니다.)

팁: Dependabot PR에는 기본적으로 `dependencies` 라벨이 붙고 변경 내역, 릴리즈 노트 등이 포함됩니다. 또한 필요에 따라 `allow`나 `ignore` 설정으로 특정 패키지 버전 범위를 지정하거나, PR당 변경 개수(예: 한 번에 하나씩 vs 그룹핑)를 조절할 수 있습니다 ³⁷ ³⁸. 2023년 이후 Dependabot에서 **복수 생태계 그룹화 (multi-ecosystem-group)** 및 **Cron 스케줄**도 지원되므로, 프로젝트 상황에 맞게 세부 조정이 가능합니다 ³⁹ ³⁷.

GitHub Projects 보드 자동화 (이슈 상태 업데이트)

프로젝트 보드를 활용해 이슈와 작업 현황을 **할 일** → **진행 중** → **완료**로 시각적으로 관리한다면, 여러 이벤트에 따라 **자동으로 상태를 갱신**하도록 설정할 수 있습니다. GitHub Projects (Beta 기준)는 기본 **Status 필드**를 통한 워크플로우 자동화 기능을 제공합니다. 다음과 같은 자동화 시나리오를 적용합니다:

- **이슈 생성 시 "Todo":** 새 이슈가 프로젝트에 추가되면 상태 필드를 `Todo`로 설정합니다. Projects 설정의 *Workflows*에서 해당 규칙을 활성화할 수 있습니다 ⁴⁰. 또한 *Add items automatically* 규칙을 사용하여 특정 레포 지토리의 이슈가 열릴 때 자동으로 프로젝트에 추가되도록 할 수 있습니다 ⁴¹. (예: 프로젝트 보드 설정에서 필터를 `repo:myorg/myrepo is:issue` 등으로 지정하면 신규 이슈 자동 추가).

- **PR 열리면 "In Progress"**: 누군가 이슈를 해결하기 위해 Branch를 만들거나 PR을 연결했을 때, 그 이슈를 진행 중으로 옮깁니다. 기본 내장 워크플로우에는 이 규칙이 없으므로, 두 가지 방법이 있습니다. (1) **수동**: 이슈에 연결된 PR이 열리면 관리자가 Projects 보드에서 해당 이슈의 Status를 In Progress로 수동 변경. (2) **자동화**: Actions으로 구현. 예를 들어 PR이 열릴 때 본문이나 연관 이슈를 확인하여, 연결된 이슈의 프로젝트 Status 필드를 업데이트 하는 GitHub Script를 실행할 수 있습니다. GitHub GraphQL API를 사용하면 이슈의 Project item을 찾아 `status` 값을 변경할 수 있습니다. 이 방법은 구현이 다소 복잡하지만 가능하며, GitHub CLI(`gh`)를 사용해 `gh project item update`로 상태 변경을 스크립트로 실행하는 것도 고려할 수 있습니다.

- **이슈 닫히거나 PR 머지 시 "Done"**: 프로젝트 보드에 있는 이슈가 닫히면 상태를 자동으로 Done으로 설정하는 기본 워크플로우가 있습니다 ⁴⁰. GitHub Projects의 기본 제공 자동화에 따라 **이슈 또는 PR이 Close/Merge되면 Status=Done으로 변경**됩니다 ⁴². (해당 프로젝트에 item으로 등록되어 있을 경우) 이는 프로젝트 설정에서 기본으로 활성화되어 있습니다. 예컨대 “When issues or pull requests in your project are closed, set status to Done” 와 “When pull requests in your project are merged, set status to Done”와 같은 규칙입니다 ⁴².

요약하면, **Issues → Todo, PR linked → In Progress, Closed/Merged → Done** 흐름을 이루도록 합니다. 기본 기능과 약간의 수동/액션 스크립트를 조합하여 완전 자동화에 가깝게 구현할 수 있습니다. 프로젝트 보드에 이러한 자동화가 적용되면 개발자들은 이슈 상태를 일일이 업데이트할 필요 없이 흐름에 따라 보드가 업데이트되므로, 관리 부담을 줄여줍니다.

추가 Tip: GitHub 이슈 코멘트의 **Slash Command** 기능으로도 프로젝트 Status를 업데이트할 수 있습니다. 예를 들어 이슈에서 `/status In Progress` 등의 명령을 치면 해당 이슈의 프로젝트 상태 필드를 바꾸는 기능이 베타로 제공되고 있습니다. 이러한 커맨드를 사용하면 굳이 Actions를 짜지 않고도 상태 변경을 어느 정도 자동화/간소화할 수 있으니, 팀의 문화에 맞게 선택하시기 바랍니다.

결론

위의 구성요소들을 모두 결합하면, **코드 작성 → PR 생성 → AI 코드리뷰 → AI 코드수정 반영 → CI 통과 → 머지 → 배포 → 문서/프로젝트보드 갱신**까지 상당 부분 자동화된 파이프라인이 완성됩니다. 개발자는 핵심 비즈니스 로직 구현에 집중할 수 있고, 반복적인 리뷰나 문서 작업, 의존성 관리, 배포 트리거 등은 자동으로 처리됩니다. 요약하면 다음과 같은 흐름이 이뤄집니다:

- **PR 생성**: Codex AI가 자동으로 PR의 변경사항을 리뷰하고 코멘트를 남김 ⁶.
- **AI 코드 수정**: Codex 리뷰를 참고하여 개발자가 `@claude apply` 댓글을 달면 Claude AI가 제안 사항을 코드에 적용, 커밋을 추가 ¹⁹ ⁴³.
- **CI 파이프라인**: 변경된 프론트엔드/백엔드 부분만 선택적으로 빌드/테스트하여 효율적 검증.
- **PR 머지**: main에 머지되면 프론트는 Vercel에, 백엔드는 Docker 이미지로 자동 배포.
- **문서 동기화**: 코드 변화에 맞춰 AI가 문서를 수정하고 PR로 제안하여 문서 최신성 유지 ³² ³⁴.
- **의존성 관리**: Dependabot이 주기적으로 업데이트 PR을 열어 안전한 버전 유지 ³⁵.
- **프로젝트 보드**: 이슈/PR 상태에 따라 Projects 보드의 상태 칼럼이 자동 갱신되어 팀 가시성 확보 ⁴⁰ ⁴².

모든 자동화에는 개발팀의 모니터링과 최종 승인 단계가 함께 해야 합니다. AI의 제안이나 수정은 팀원이 검토 후 받아들이고, Dependabot PR도 테스트 후 머지하며, 문서 PR도 확인 후 병합하는 식입니다. 이렇게 하면 **자동화의 편리함과 인간의 통제가 조화를 이루어, 개발 생산성을 극대화하면서 코드 품질과 최신 문서, 안전한 배포까지 달성**할 수 있습니다.

출처 및 참고: OpenAI Codex 공식 문서 [5](#) [11](#), Anthropic Claude Code GitHub Action 안내 [12](#) [44](#), GitHub Actions 및 Dependabot 활용 사례 [27](#) [35](#) 등을 참고하여 위 구성안을 도출하였습니다. 각 설정 예시는 실제 사용 사례에 기반한 것으로, 필요에 따라 환경에 맞게 수정 가능합니다.

[1](#) [4](#) [5](#) [6](#) **Code Review**

<https://developers.openai.com/codex/cloud/code-review/>

[2](#) [3](#) [14](#) [15](#) [16](#) [17](#) [20](#) [21](#) [22](#) [23](#) [26](#) **Claude Code GitHub Actions - Claude Docs**

<https://docs.claude.com/en/docs/claude-code/github-actions>

[7](#) [8](#) [9](#) [10](#) [11](#) **GitHub - openai/codex-action**

<https://github.com/openai/codex-action>

[12](#) [13](#) [18](#) [30](#) [31](#) **GitHub - anthropics/claude-code-action**

<https://github.com/anthropics/claude-code-action>

[19](#) [24](#) [25](#) [43](#) [44](#) **v1.0.5 runs but can't pick up the PR context nor post the output as a comment on the PR itself. · Issue #557 · anthropics/claude-code-action · GitHub**

<https://github.com/anthropics/claude-code-action/issues/557>

[27](#) **continuous integration - Deploy individual services from a monorepo using github actions - Stack Overflow**

<https://stackoverflow.com/questions/58136102/deploy-individual-services-from-a-monorepo-using-github-actions>

[28](#) **How can I use GitHub Actions with Vercel?**

<https://vercel.com/guides/how-can-i-use-github-actions-with-vercel>

[29](#) **Deploy to Vercel Action - GitHub Marketplace**

<https://github.com/marketplace/actions/deploy-to-vercel-action>

[32](#) [33](#) [34](#) **Pipeline Configuration | Cursor & Claude Code | Developer Toolkit**

<https://developertoolkit.ai/en/claude-code/lessons/ci-cd/>

[35](#) [36](#) [37](#) [38](#) [39](#) **Keeping dependencies in your GitHub projects up-to-date with Dependabot - DEV Community**

<https://dev.to/aloisseckar/keeping-dependencies-in-your-github-projects-up-to-date-with-dependabot-16bg>

[40](#) [41](#) [42](#) **Using the built-in automations - GitHub Docs**

<https://docs.github.com/en/issues/planning-and-tracking-with-projects/automating-your-project/using-the-built-in-automations>