신입 개발자 온보딩 가이드 **(Dev Onboarding Kit)**

문서번호**:** 2026-DEV-001

대상**:** 신규 입사자 및 수습 개발자

최종수정**:** 2026년 2월 8일

0. 환영합니다! 👋

메타코딩 개발팀에 합류하신 것을 환영합니다. 이 문서는 팀의 기술적 일관성을 유지하고, 여러분이 더 빠르고 안전하게 코드를 기여할 수 있도록 돕기 위해 작성되었습니다.

> **"**코드는 나를 위해 작성하는 것이 아니라**,** 내 코드를 읽을 동료를 위해 작성하는 것입니다**."**

1. 코드 컨벤션 **(Code Convention)**

우리는 가독성 **(Readability)** 을 최우선 가치로 둡니다. 언어별 세부 린터(Linter) 설정 외에 공통적으로 지켜야 할 핵심 규칙입니다.

1.1 네이밍 규칙 **(Naming)**

이름은 그 변수나 함수가 무엇을 하는지**,** 무엇을 담고 있는지 명확히 드러내야 합니다.

- 변수**/**함수명 **(camelCase):** 명사(변수) 또는 동사(함수)로 시작
  - userName (O), u_name (X)
  - getUserData() (O), userData() (X - 동사가 빠짐)
  - isValid, hasPermission (Boolean은 is, has 등으로 시작)
- 클래스**/**인터페이스명 **(PascalCase):**
  - UserService (O), userService (X)
- 상수 **(UPPER_SNAKE_CASE):**
  - MAX_LOGIN_RETRY, API_TIMEOUT

1.2 주석 **(Comments)**

- 무엇 **(What)** 을 하는지보다 왜 **(Why)** 그렇게 짰는지를 설명하세요.
- 코드로 충분히 설명되는 로직에는 주석을 달지 않습니다. (Clean Code)

```
// Bad: 나이가 18세 이상인지 확인
if (age >= 18) { ... }

// Good: 법적 성인 기준 (상수로 의미 부여)
const LEGAL_ADULT_AGE = 18;
```

if (age >= LEGAL_ADULT_AGE) { ... }

1.3 포맷팅 (Formatting)

- 모든 프로젝트에는 .prettierrc 또는 .editorconfig가 포함되어 있습니다.
- 저장 시 자동 포맷팅**(Format On Save)** 기능을 에디터에 반드시 설정하세요.
- 탭(Tab) 대신 스페이스**(Space) 2**칸 또는 **4**칸을 사용합니다. (프로젝트별 설정 따름)

2. 깃헙 (GitHub) 협업 전략

우리는 **Git Flow**를 단순화한 전략을 사용합니다. main 브랜치는 언제나 배포 가능한 상태여야 합니다.

2.1 브랜치 전략 (Branch Strategy)

작업을 시작할 때는 반드시 최신 develop (또는 main) 브랜치에서 새로운 브랜치를 생성해야 합니다.

브랜치 타입	설명
main (master)	실제 운영 서버에 배포되는 코드. 직접 커밋 금지(PR Merge만 허용).
develop	다음 버전을 위한 개발 브랜치. 모든 기능 브랜치는 여기서 시작하고 여기로 병합됨.
feature/	새로운 기능을 개발하는 브랜치.
fix/	버그를 수정하는 브랜치.
hotfix/	운영 중인 서버(main)의 긴급한 버그를 수정하는 브랜치.

2.2 브랜치 네이밍 규칙

- 형식: 타입/이슈번호-간략설명 (소문자, 하이픈 권장)
  - feat/102-login-page (O)
  - fix/header-layout (O)
  - kim/login (X - 개인 이름 사용 지양)

3. 커밋 메시지 규칙 (Commit Message)

커밋 메시지는 동료들이 변경 사항을 빠르게 파악할 수 있는 이정표입니다.

3.1 커밋 태그 (Tag)

커밋 메시지 제일 앞에 대괄호 [] 또는 태그명과 콜론:을 사용하여 성격을 명시합니다.

태그	설명	예시
Feat	새로운 기능 추가	Feat: 소셜 로그인 기능 추가
Fix	버그 수정	Fix: 결제 모달이 닫히지 않는 오류 수정
Refactor	코드 리팩토링 (기능 변경 없음)	Refactor: 회원가입 로직 함수 분리
Style	코드 포맷팅, 세미콜론 누락 등	Style: 메인 페이지 코드 포맷팅 적용
Docs	문서 수정	Docs: README.md 설치 가이드 업데이트
Test	테스트 코드 추가/수정	Test: 로그인 API 유닛 테스트 작성
Chore	빌드 설정, 패키지 매니저 설정 등	Chore: eslint 의존성 버전 업그레이드

3.2 메시지 작성 팁

- 제목**:** 50자 이내로 요약하며, 명령조로 작성합니다. (~함, ~수정)
  - 로그인 기능을 수정했음 (X) -> 로그인 유효성 검사 로직 수정 (O)
- 본문**:** (선택사항) 어떻게보다 무엇을**,** 왜 변경했는지 구체적으로 작성합니다.

4. 풀 리퀘스트 (PR) 및 코드 리뷰

코드를 합치기 전, 동료들의 검토를 받는 과정입니다. **"코드 리뷰는 비난이 아니라 성장을 위한 과정"**임을 명심해주세요.

4.1 PR 작성 규칙

1. **PR** 템플릿 준수**:** PR 생성 시 자동으로 뜨는 템플릿 양식을 채워주세요. (작업 내용, 테스트 방법, 스크린샷 등)
2. **Self-Review:** PR을 올리기 전, 본인이 먼저 Files changed 탭에서 실수는 없는지 확인합니다.
3. 작은 단위 유지**:** PR 하나가 너무 커지면(File 20개 이상) 리뷰하기 어렵습니다. 기능 단위로

쪼개주세요.

4.2 리뷰어 **(Reviewer)** 요청

- 최소 **1**명 이상의 승인 **(Approve)** 을 받아야 Merge가 가능합니다.
- 멘토나 팀 리더를 리뷰어로 지정하세요.
- 리뷰 코멘트가 달리면 적극적으로 토론하고 수정합니다. 수정이 완료되면 "Resolved" 처리를 하거나 대댓글로 알립니다.

5. 마치며 (체크리스트)

입사 첫 주, 다음 항목들을 세팅했는지 확인해보세요.

- [ ] Git 계정 설정 (git config user.name, user.email 확인)
- [ ] IDE(VS Code, IntelliJ) 포맷터 설정 완료
- [ ] 사내 GitHub Organization 초대 수락
- [ ] 프로젝트 Clone 및 로컬 빌드 성공 확인
- [ ] Hello World를 찍어보는 첫 번째 PR 올려보기

문의사항이 있다면 언제든 멘토나 슬랙 채널에 편하게 물어봐 주세요. 다시 한번 환영합니다! 🚀