**깃허브 활용 표준**



Document history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Ver | Description | Remark |
| 2024-12-27 | R0 | Originally issued. |  |
| 2025-02-25 | R1 | Add commit convention |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
|  | HANLA IMS  115, Hwajeonsandan 1(il)-ro, Gangseo-gu, Busan, Korea  www.hanlaims.com |

**CONTENTS**

0. 개요 3

1. 계정정보 3

**1.1. 회사 계정정보** 3

**1.2. 주의사항** 3

2. 레포지토리 및 프로젝트 생성 3

**2.1. 레포지토리 생성** 3

**2.2. 프로젝트 생성** 3

**2.3. 협업자(Collaborator)추가** 3

3. Git 작업 규칙 4

**3.1. 회사계정 사용규칙** 4

**3.2. Pull request 규칙** 4

4. 브랜치 관리 규칙 4

**4.1. 브랜치 기본 구조** 4

**4.2. 브랜치 이름 규칙** 4

**4.3. Merge 규칙** 5

5. 협업 시 권장사항 4

**5.1. 코드 리뷰 프로세스 준수** 4

**5.2. 작업 공유 및 소통** 4

**5.3. 로그 기록** 5

6. 일관성 유지 5

**6.1. 커밋 컨벤션** 5

**6.2. 커밋 메시지 템플릿** 6

**6.3. 풀 리퀘스트 템플릿** 5

**6.4. 브랜치 네이밍 컨벤션** 5

7. 요약 LT;DR; 5

**6.1. 요약** 5

**6.2. 시청각 자료** 6

Appendix 각종 템플릿 7

**0. 개요**

이 매뉴얼은 회사 GitHub 계정(github@hanlaims.com)을 이용한 프로젝트 및 레포지토리 관리, 협업 시의 규칙과 절차를 정의합니다. 회사 계정은 협업의 중심 계정으로 사용되며, 모든 작업은 명확한 규칙에 따라 이루어져야 함.

**1. 계정 정보**

**1.1. 회사 계정정보**

[1] 계정 이메일: github@hanlaims.com

[2] 비밀번호: 담당자에게 문의

**1.2. 주의사항**

[1] 회사 계정은 프로젝트 및 레포지토리 생성과 사용자 초대에만 사용될 것.

[2] 직접적인 코드 작업(Push, Merge 등)은 개인 계정을 통해 이루어져야 함.

[3] 레포지토리 임의삭제 절대 금지.

**2. 레포지토리 및 프로젝트 생성**

**2.1. 레포지토리 생성**

[1] 회사 계정으로 로그인 후, 새 레포지토리를 생성할 것.

[2] 레포지토리 생성 시 기본 설정

* **Visibility**: Private 설정 권장.
* **Branch Protection Rule** 설정 권장.
* **ReadMe.md 파일 생성.**

[3] 레포지토리 생성 시 이름 규칙

* [프로젝트번호]-[유형]-[(프로그램)이름 or 목적]
* 예시: RD0098-SW-원가절감형\_ESDS

**2.3. 프로젝트 생성**

[1] 레포지토리를 생성한 후, 프로젝트의 이름, 설명, 기본 브랜치를 설정합니다.

[2] ReadMe 파일 작성: 프로젝트 개요 및 주요 정보를 작성하여 협업자들이 참고할 수 있도록 할 것.

**2.3. 협업자(Collaborator) 추가**

[1] 새로 생성된 레포지토리의 Settings 🡪 Collaborators 메뉴에서 협업할 팀원을 초대할 것.

[2] 협업자는 각자의 개인 GitHub 계정을 사용해야 하며, 회사 계정은 직접 Push, Merge 작업을 하지 않음.

**3. Git 작업 규칙**

**3.1. 회사계정 사용규칙**

[1] 회사 계정으로는 절대로 아래 작업을 하지 않음.

* Push
* Merge

[2] 개인 계정 작업 원칙.

* 모든 Push 및 Merge 작업은 협업자로 등록된 개인 계정을 통해 이루어짐.
* Pull Request**(PR)**를 통해 검토 및 승인을 받은 후 Merge를 진행할 것.

**3.2. Pull Request(PR) 규칙**

[1] Pull Request(PR) 생성

* 모든 작업결과는 새 브랜치에서 진행한 후, 메인 브랜치로 Merge하기 위해 PR을 생성할 것.
* PR제목과 설명에는 작업 내용 및 변경 사항을 명확히 작성할 것.

[2] Auto Merge

* PR 승인 후 자동 병합(auto merge)을 활성화합니다.
* 협업 시 코드 리뷰어는 PR 내용을 충분히 검토한 후 승인할 것.

[3] 충돌 해결

* 충돌 발생 시 로컬 환경에서 충돌을 해결한 후 변경 사항을 Push하고 PR을 다시 제출할 것.

**4. 브랜치 관리 규칙**

**4.1. 브랜치 기본 구조**

[1] Main Branch: 배포 가능한 안정적인 코드만 유지.

[2] Develop Branch: 모든 개발 작업의 기본 브랜치.

[3] Feature Branch: 새로운 기능 개발 또는 작업 단위를 위해 생성.

[4] Hotfix Branch: 긴급한 버그 수정 시 사용.

**4.2. 브랜치 이름 규칙**

[1] 브랜치 이름은 명확하고 작업 목적을 나타내야 합니다.

[2] 예시:

* Feature: feature/add-login-page
* Bugfix: bugfix/fix-login-error
* Hotfix: hotfix/critical-bug-fix
* Main: main
* Develop: dev

**4.3. Merge 규칙**

[1] Feature 브랜치는 Develop 브랜치로만 병합.

[2] Main 브랜치는 Develop 브랜치를 통해서만 변경 사항을 반영.

[3] 모든 Merge는 Pull Request를 통해 이루어져야 함.

**5. 협업 시 권장사항**

**5.1. 코드 리뷰 프로세스 준수**

[1] PR생성 후, 코드 리뷰어가 내용을 확인하고 승인 또는 피드백을 제공함.

[2] 코드 품질 향상을 위해 리뷰 프로세스를 준수 하는게 좋음.

**5.2. 작업 공유 및 소통**

[1] 작업 진행 상황과 문제는 팀 내에서 공유하며, 명확한 소통을 통해 효율적인 협업을 유지할 것.

[2] 필요 시 GitHub Issues를 활용하여 작업 사항을 정리하고 논의할 것.

**5.3. 로그 기록**

[1] 모든 커밋 메시지는 작업 내용을 명확히 표현 해야할 것.

[2] 예시:

* Good: 로그인 페이지 MFA기능 추가
* Bad: 수정함. 진짜진짜진짜최신본!!

**6. 일관성 유지**

**6.1. 커밋 컨벤션**

[1] 개념 : Git에서 커밋 메시지를 작성할 때 일정한 규칙을 정해 일관성을 유지하는 방식

[2] 일관성 유지 : 팀원 간 코드 히스토리가 통일된 형식으로 기록

[3] 변경 사항 추적 용이 : 나중에 특정 기능 추가, 버그 수정 등을 쉽게 검색 가능.

[4] 릴리즈 관리 최적화 : 커밋 메시지에서 변경 사항을 자동으로 추출하여 릴리즈 노트 생성등 자동화 기능 사용가능.

**6.2. 커밋 메시지 템플릿**

**Angular 커밋 (가장 많이 사용됨)**

|  |
| --- |
| <타입>: <커밋 메시지 제목> // 제목과 본문의 분리를 위해 한 칸 띄움. <본문 설명 (선택적)> //어떻게 보다는 왜 무엇을 추가했는지  <footer 이슈번호(선택적)> //깃허브 이슈관리 Close #10 이슈 PR의 Merge시 #10의 상태가 Close됨. |

**예시**

|  |
| --- |
| feat: 사용자 로그인 기능 Password 암호화 추가  - 비밀번호 암호화가 필요해서 RFC2898추가  - salt 32bit, iteration 310,000회 (OWASP 권장사항)  **Close #10** |

**타입**

**# ✨feat : 새로운 기능 추가**

**# 🐛fix : 버그 수정**

**# 📝docs : 문서 수정**

**# 🧪test : 테스트 코드 추가**

**# ♻️refact : 코드 리팩토링**

**# 🎨style : 코드 의미에 영향을 주지 않는 변경사항(포맷, 들여쓰기)**

**# ⚡️perf : 성능개선**

**# 🚀chore : 빌드 부분 혹은 패키지 매니저 수정사항(코드변경 없음)**

**# 🚀ci : 배포관련**

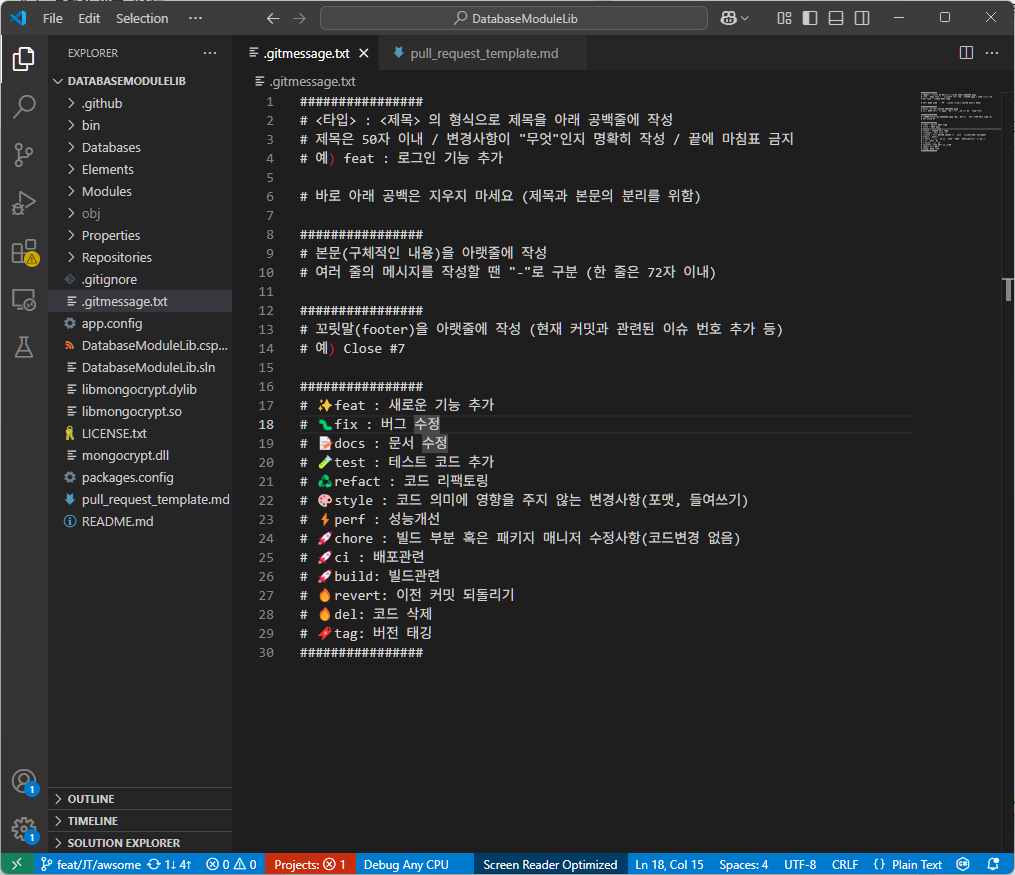
**# 🚀build: 빌드관련**

**# 🔥revert: 이전 커밋 되돌리기**

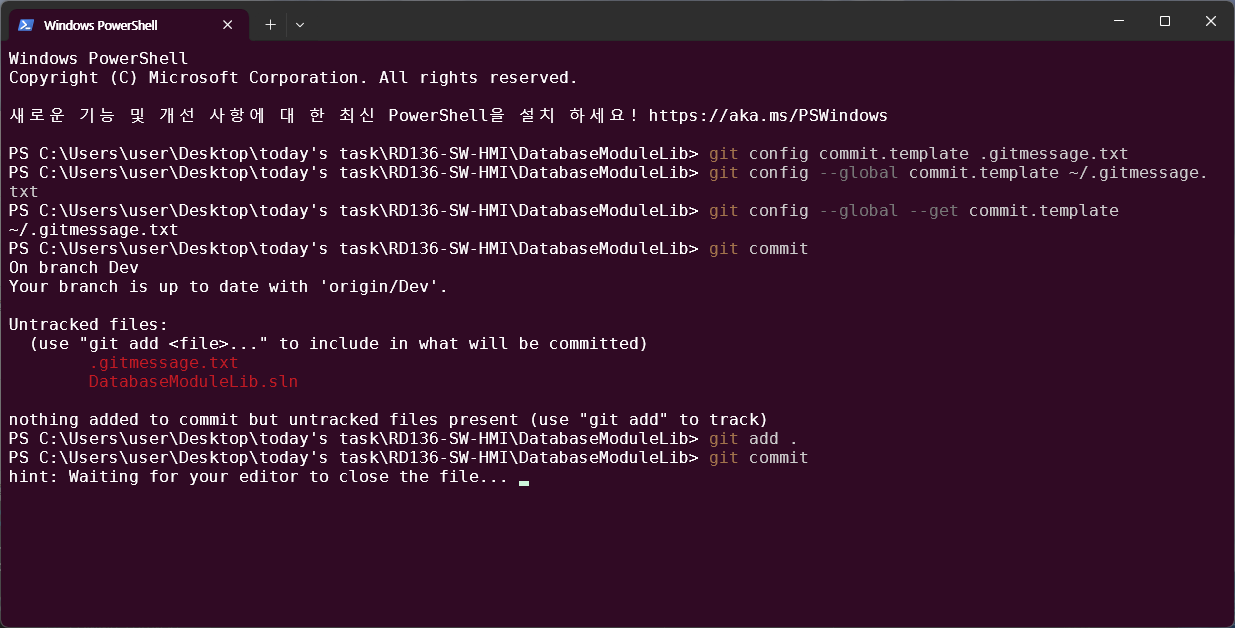
**# 🔥del: 코드 삭제**

**# 🔖tag: 버전 태깅**

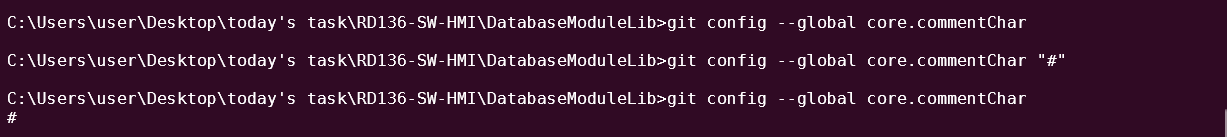
**설정방법**



[1] .gitmessage.txt 루트경로에 추가 후 템플릿 작성 (#은 주석처리됨.)

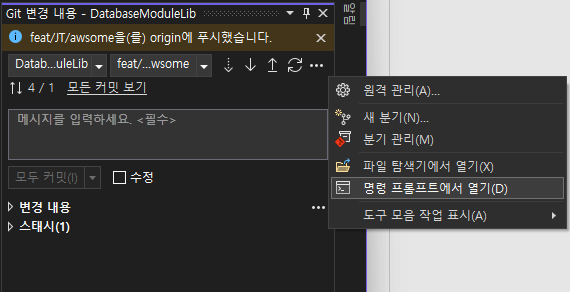


[2] 루트경로에서 powershell 실행. –global 설정으로 커밋템플릿 설정.

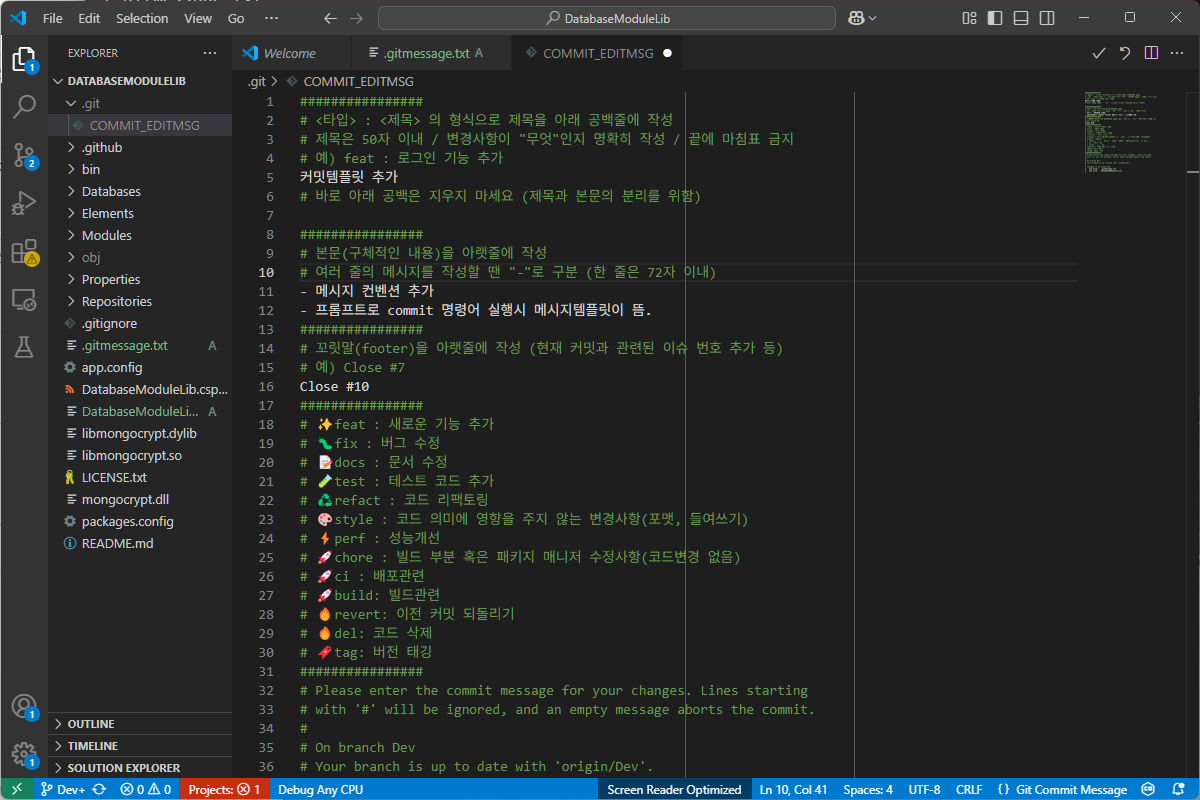


[3] “#”가 주석처리 되도록 설정.

[4] 해당 브랜치 commit 후 dev로 PR하여 병합할 것.



[5] 템플릿 사용을 위해서는 변경사항 저장 후. 명령 프롬프트에서 git commit 명령어 엔터



[6] 템플릿 규칙대로 작성하고, 저장 후 탭 종료.



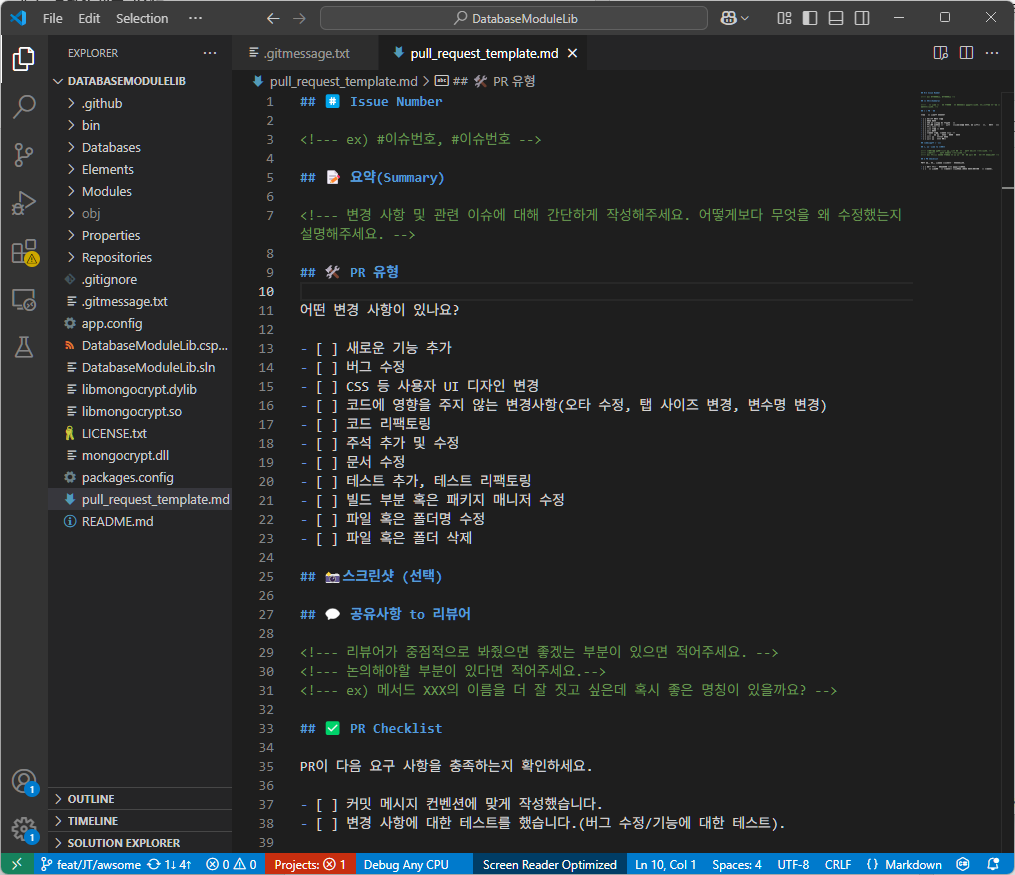
[7] git log에서는 #이 생략되어 angular commit 컨벤션 양식으로 저장됨

[8] 숙련되면 굳이 템플릿을 이용하여 작성하는 수고를 들일 필요 없을 정도로 간단한 규칙임.

[9] PR머지시 자동으로 이슈를 닫는 기능은 (Close, Fix, Resolve)뒤에 #이슈번호

[10] 단순 참조는 (see, refs, references) #이슈번호

**6.3. 풀 리퀘스트 템플릿**



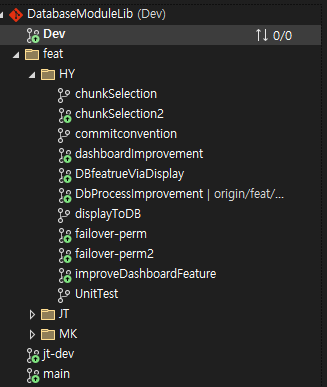
[1] pull\_request\_template.md 루트경로에 추가 후 템플릿 작성 (#은 주석처리됨.)

[2] main 브랜치까지 병합해야 깃허브에서 PR merge시 템플릿 동작.

[3] 깃허브 웹 인터페이스를 이용해 main 브랜치에 직접 추가하면 전역으로 적용되기도 함.

**6.4. 브랜치 네이밍 컨벤션**

[1] ex) feat/HY/login\_encryption : 로그인 암호화 추가, 작업자 주호연



[2] 브랜치 목적을 명확히 구분 가능하여 가독성이 향상되며, directory 트리 구조로 관리

[3] CI/CD 적용 가능. 테스트용 PC로 자동배포 설정 (고도화 계획중)

**7. 요약 TL;DR;**

**7.1. 로그 기록**

[1] 회사 계정으로 레포지토리 및 프로젝트 생성.

[2] 협업자로 개인 계정을 초대.

[3] 새 브랜치를 생성하여 작업 진행.

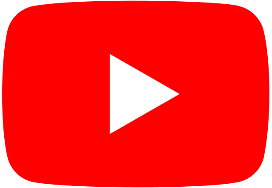
[4] Pull Request(PR)로 작업 반영 요청.

[5] 코드 리뷰 및 승인 후 Merge.

[6] 충돌 발생 시 로컬에서 해결 후 Pull Request 다시 제출.

[7] 메인 브랜치 변경 사항은 항상 리뷰를 거친 후 반영.

**7.2. 시청각 자료**

[](https://youtu.be/9t3YxAAKcec)

**Appendix** 각종 템플릿

|  |
| --- |
| .gitmessage.txt  ################  **# <타입> : <제목> 의 형식으로 제목을 아래 공백줄에 작성**  **# 제목은 50자 이내 / 변경사항이 "무엇"인지 명확히 작성 / 끝에 마침표 금지**  **# 예) feat : 로그인 기능 추가**  **# 바로 아래 공백은 지우지 마세요 (제목과 본문의 분리를 위함)**  ################  **# 본문(구체적인 내용)을 아랫줄에 작성**  **# 여러 줄의 메시지를 작성할 땐 "-"로 구분 (한 줄은 72자 이내)**  ################  **# 꼬릿말(footer)을 아랫줄에 작성 (현재 커밋과 관련된 이슈 번호 추가 등)**  **# 예) Close #7**  ################  **# ✨feat : 새로운 기능 추가**  **# 🐛fix : 버그 수정**  **# 📝docs : 문서 수정**  **# 🧪test : 테스트 코드 추가**  **# ♻️refact : 코드 리팩토링**  **# 🎨style : 코드 의미에 영향을 주지 않는 변경사항(포맷, 들여쓰기)**  **# ⚡️perf : 성능개선**  **# 🚀chore : 빌드 부분 혹은 패키지 매니저 수정사항(코드변경 없음)**  **# 🚀ci : 배포관련**  **# 🚀build: 빌드관련**  **# 🔥revert: 이전 커밋 되돌리기**  **# 🔥del: 코드 삭제**  **# 🔖tag: 버전 태깅**  ################ |
| **pull\_request\_template.md**  **## #️⃣ Issue Number**  <!--- ex) #이슈번호, #이슈번호 -->  **## 📝 요약(Summary)**  <!--- 변경 사항 및 관련 이슈에 대해 간단하게 작성해주세요. 어떻게보다 무엇을 왜 수정했는지 설명해주세요. -->  **## 🛠️ PR 유형**  어떤 변경 사항이 있나요?  - [ ] 새로운 기능 추가  - [ ] 버그 수정  - [ ] CSS 등 사용자 UI 디자인 변경  - [ ] 코드에 영향을 주지 않는 변경사항(오타 수정, 탭 사이즈 변경, 변수명 변경)  - [ ] 코드 리팩토링  - [ ] 주석 추가 및 수정  - [ ] 문서 수정  - [ ] 테스트 추가, 테스트 리팩토링  - [ ] 빌드 부분 혹은 패키지 매니저 수정  - [ ] 파일 혹은 폴더명 수정  - [ ] 파일 혹은 폴더 삭제  **## 📸스크린샷 (선택)**  **## 💬 공유사항 to 리뷰어**  <!--- 리뷰어가 중점적으로 봐줬으면 좋겠는 부분이 있으면 적어주세요. -->  <!--- 논의해야할 부분이 있다면 적어주세요.-->  <!--- ex) 메서드 XXX의 이름을 더 잘 짓고 싶은데 혹시 좋은 명칭이 있을까요? -->  **## ✅ PR Checklist**  PR이 다음 요구 사항을 충족하는지 확인하세요.  - [ ] 커밋 메시지 컨벤션에 맞게 작성했습니다.  - [ ] 변경 사항에 대한 테스트를 했습니다.(버그 수정/기능에 대한 테스트). |