** 주의사항

1. 자신이 작업한 파일 꼭 ! 따로 버전별로 관리 할 것 !!
2. commit, pull, push 하기 전에 git branch 로 현재 브랜치 위치 꼭 확인하기 !!
3. 다른 팀원의 업무 영역 수정 시 꼭 먼저 말한 뒤 수정 하기 !!
4. merge 하기 전에 pull 명령어로 버전 맞추기 !!
5. 마지막에 develop 브랜치에 merge 후 push까지 하면 자신이 작업했던 브랜치 삭제 !! git branch -d <로컬 브랜치 이름> git branch -D <로컬 브랜치 이름> → 강제 삭제 git push origin -d <원격 브랜치 이름>

가이드 1) 브랜치 clone 후 작업 시작 ~ 자신의 브랜치에 commit, push

1. 폴더 하나 생성 후 터미널 열기
2. git clone --branch {branchname develop} {remote-repo-url} → 로컬레포에 웹 레포 내려받기 git remote add origin {주소}
3. 그 폴더 안에 로컬레포 생성된 것 확인 → 다시 생성된 로컬레포로 터미널 열기
4. git checkout -b {생성할 브랜치 feature/기능요약} {복제할 브랜치 develop} : 로컬 레포에 브랜치 생성과 동시에 이동됨
5. git push origin {생성한 브랜치 feature/기능요약} : 웹 레포에 {복제할 브랜치}의 파일을 그대로 가진채로 feature/기능요약 브랜치가 생성됨
6. 폴더 생성 후 git clone --branch {클론 할 브랜치 이름 - feature/기능요약} {remote-repo-url}
7. 그 폴더 안에 로컬레포 생성된 것 확인
8. ================== 코딩 진행 ==================
9. 생성된 로컬레포로 터미널 열기
10. git init
11. git branch → 현재 브랜치 확인!! (feature/기능요약)
12. git add .
13. git commit -m "가이드 4) 커밋메시지 작성" → 커밋
14. git push origin {자신의 브랜치 - feature/기능요약}
15. 작업했던 {feature/기능요약} 브랜치 삭제 git branch -d <로컬 브랜치 이름> git branch -D <로컬 브랜치 이름> → 강제 삭제 git push origin -d <원격 브랜치 이름>

가이드 2) merge 순서

1. git add .
2. git commit -m “커밋 메시지”
3. git pull origin {가져올 브랜치 명} -> develop 브랜치를 가져와서 버전 맞추기 !
4. git checkout {머지 할 브랜치 명} -> develop 브랜치로 이동
5. git merge {자신이 작업한 브랜치 명} -> feature/기능요약 브랜치
6. git push origin develop -> develop 브랜치에서 머지 후 push 해주기 !

*** merge 테스트 시나리오..

** A 팀원 feature/1 → 프로젝트 내 a, b 파일 수정

1) 프로젝트 폴더 터미널 열기 → git add .

2) git commit -m “커밋메시지”

3) git pull origin develop
- 내 로컬 레포의 **feature/1** 브랜치에 develop 브랜치 merge 된다.
- 현재 내 로컬 파일에 반영됨

4) git checkout develop

5) git merge feature/1
- Fast-Forward merge

6) git push origin develop → 원격저장소에 올리기!

* * B 팀원 feature/2 → 프로젝트 내 c 파일 수정

1) 프로젝트 폴더 터미널 열기 → git add .

2) git commit -m “커밋메시지”

3) git pull origin develop
- 내 로컬 레포의 **feature/2** 브랜치에 develop 브랜치 merge 된다.
- 현재 내 로컬 파일에 반영됨

4) git checkout develop

5) git merge feature/2
- Fast-Forward merge

6) git push origin develop → 원격저장소에 올리기!

* * C 팀원 feature/3 → 프로젝트 내 a, c 파일 수정 → 다른 라인 수정

1) 프로젝트 폴더 터미널 열기 → git add .

2) git commit -m “커밋메시지”

3) git pull origin develop
- 내 로컬 레포의 **feature/3** 브랜치에 develop 브랜치 merge 된다.
- 현재 내 로컬 파일에 반영됨

4) git checkout develop

5) git merge feature/3
- Fast-Forward merge

6) git push origin develop → 원격저장소에 올리기!

가이드 3) merge 충돌 해결

**** develop 에서 merge 하기 전에 pull 했을 때 충돌 상황**
- 내가 수정한 파일을 다른 팀원이 수정 후 이미 develop에 merge한 상태

```
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git commit -m "feat: NoticeCon
troller logger삭제 및 AuthController 수정

test4
test44

footer 생략 가능"
[test4 513436b] feat: NoticeController logger삭제 및 AuthController 수정
 2 files changed, 5 insertions(+), 8 deletions(-)
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git pull origin develop
Enter passphrase for key '/Users/jeongtaeyeon/.ssh/id_rsa':
remote: Enumerating objects: 68, done.
remote: Counting objects: 100% (68/68), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 59 (delta 22), reused 53 (delta 17), pack-reused 0
오브젝트 묶음 푸는 중: 100% (59/59), 5.20 KiB | 171.00 KiB/s, 완료.
github.com:jungtaeyeon/demoGit URL에서
 * branch            develop    -> FETCH_HEAD
   b6977b9..c39076e  develop    -> origin/develop
자동 병합: src/main/java/com/example/demo/controller/AuthController.java
충돌 (내용): src/main/java/com/example/demo/controller/AuthController.java에 병
합 충돌
자동 병합이 실패했습니다. 충돌을 바로잡고 결과물을 커밋하십시오.
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit %
```

1. git status → 충돌난 파일 확인

```
-zsh
Command
오브젝트 묶음 푸는 중: 100% (59/59), 5.20 KiB | 171.00 KiB/s, 완료.
github.com:jungtaeyeon/demoGit URL에서
 * branch            develop    -> FETCH_HEAD
   b6977b9..c39076e  develop    -> origin/develop
자동 병합: src/main/java/com/example/demo/controller/AuthController.java
충돌 (내용): src/main/java/com/example/demo/controller/AuthController.java에 병합 충돌
자동 병합이 실패했습니다. 충돌을 바로잡고 결과물을 커밋하십시오.
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git status
현재 브랜치 test4
브랜치가 'origin/test4'보다 1개 커밋만큼 앞에 있습니다.
  (로컬에 있는 커밋을 제출하려면 "git push"를 사용하십시오)

병합하지 않은 경로가 있습니다.
  (충돌을 바로잡고 "git commit"을 실행하십시오)
  (병합을 중단하려면 "git merge --abort"를 사용하십시오)

커밋할 변경 사항:
        수정함:         src/main/java/com/example/demo/controller/MemberController.java
        삭제함:         src/main/java/com/example/demo/controller/RestNoticeController.java
        새 파일:        src/main/java/com/example/demo/controller/Test3_add.java

병합하지 않은 경로:
  (해결했다고 표시하려면 "git add <파일>..."을 사용하십시오)
        양쪽에서 수정:  src/main/java/com/example/demo/controller/AuthController.java

jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit %
```

2. 충돌난 파일 들어가서 확인

<<<<<<< HEAD

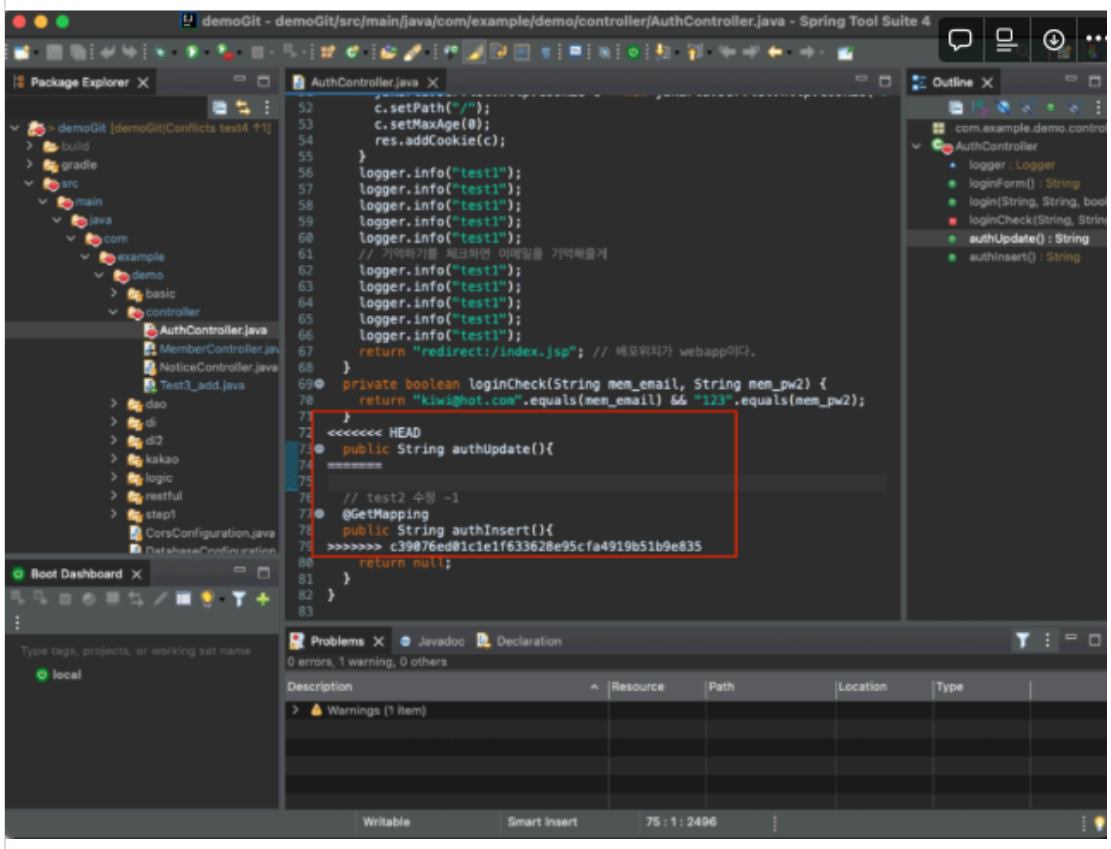
{ 이 안에 있는 코드가 pull 되기 전 내가 현재 브랜치에서 작업한 코드 내역 }

=======

{이 안에 있는 코드가 pull 당겨온 develop 브랜치의 코드}

>>>>>>> c39076ed01c1e1f633628e95cfa4919b51b9e835

위에 두 코드 중 남길부분 제외하고 전부 삭제 후 저장
<<<< >>>> ==== 이런것도 전부 삭제!!



3. 수정한 파일 add, commit, pull, merge

→ git add src/main/java/com/example/demo/controller/AuthController.java
→ 다시 commit, pull **### 이 때 충돌 해결한 파일 commit 메시지 작성 유의!**
→ 그 후 develop으로 이동 후 merge

```
-zsh
Command
힌트: 사용해 해결 표시하고 커밋하십시오.
fatal: 해결하지 못한 충돌 때문에 끝납니다.
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % code .
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git add AuthController.java
fatal: 'AuthController.java' 경로명세가 어떤 파일과도 일치하지 않습니다
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git add src/main/java/com/example/demo/cont
roller/AuthController.java
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git commit -m "충돌 해결"
[test4 68170b5] 충돌 해결
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git pull origin develop
Enter passphrase for key '/Users/jeongtaeyeon/.ssh/id_rsa':
github.com:jungtaeyeon/demoGit URL에서
 * branch            develop    -> FETCH_HEAD
이미 업데이트 상태입니다.
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git branch
* test4
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git checkout develop
branch 'develop' set up to track 'origin/develop'.
새로 만든 'develop' 브랜치로 전환합니다
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit % git merge test4
업데이트 중 c39076e..68170b5
Fast-forward
 src/main/java/com/example/demo/controller/AuthController.java   | 7 +++----
 src/main/java/com/example/demo/controller/NoticeController.java | 8 --------
 2 files changed, 3 insertions(+), 12 deletions(-)
jeongtaeyeon@jeongtaeyeon-ui-MacBookPro demoGit %
```

4. git push origin develop

가이드 4) 커밋 메시지 규칙

1. 제목과 본문을 **빈 행으로 구분**한다.
2. 제목은 **50글자** 이내로 제한한다.
3. 제목의 첫 글자는 대문자로 작성한다.
4. 제목 끝에는 마침표를 넣지 않는다.
5. 제목은 명령문으로 사용하며 과거형을 사용하지 않는다.
6. 본문의 각 행은 72글자 내로 제한한다.
7. 어떻게 보다는 **무엇과 왜를 설명**한다.

**** 커밋 메시지 구조**

// Header, Body, Footer는 빈 행으로 구분한다.
타입(스코프): 주제(제목) // Header(헤더)

본문 // Body(바디)

바닥글 // Footer

Header는 **필수**이며 스코프는 생략 가능하다.

타입은 해당 커밋의 성격을 나타내며 아래 중 하나여야 한다.

타입 이름	내용
feat	새로운 기능에 대한 커밋
fix	버그 수정에 대한 커밋
build	빌드 관련 파일 수정 / 모듈 설치 또는 삭제에 대한 커밋
chore	그 외 자잘한 수정에 대한 커밋
ci	ci 관련 설정 수정에 대한 커밋
docs	문서 수정에 대한 커밋
style	코드 스타일 혹은 포맷 등에 관한 커밋
refactor	코드 리팩토링에 대한 커밋
test	테스트 코드 수정에 대한 커밋
perf	성능 개선에 대한 커밋

Body는 Header에서 표현할 수 없는 **상세한 내용**을 적는다.

Header에서 충분히 표현할 수 있다면 **생략 가능**하다.

Footer는 바닥글로 어떤 이슈에서 왔는지 같은 **참조 정보들을 추가**하는 용도로 사용한다.

예를 들어 특정 이슈를 참조하려면 Issues #1234 와 같이 작성하면 된다.

Footer는 **생략 가능**하다.

**** 작성 예시**

→ 메모장에 양식 적어두고 복붙해서 사용하는데 더블 쿼테이션(””) 은 직접 작성하고 안에 내용 글만 복붙!!

git commit -m "fix: Safari에서 모달을 띄웠을 때 스크롤 이슈 수정

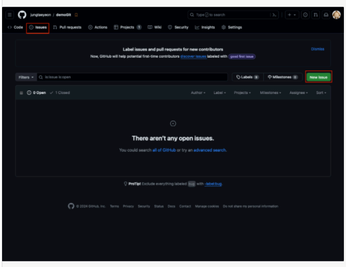
모바일 사파리에서 Carousel 모달을 띄웠을 때, 모달 밖의 상하 스크롤이 움직이는 이슈 수정.

resolves: #1137”

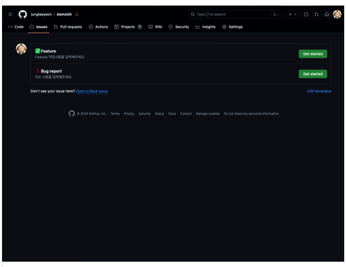
가이드 5) 깃 이슈 등록

** 이슈 등록

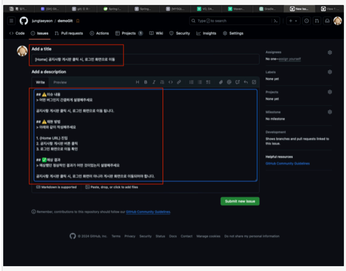
1. GNB > ISSUES탭 클릭 후 New issue 버튼 클릭



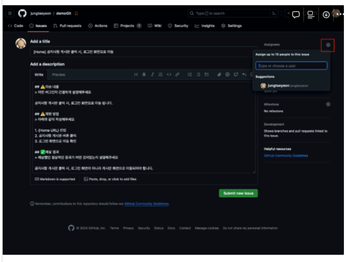
2. 등록해둔 템플릿 중 Bug report 템플릿 Get started 버튼 클릭



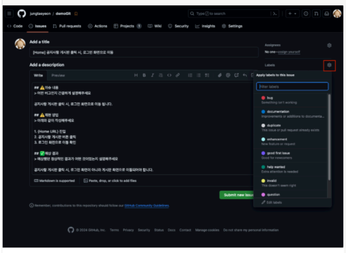
3. 이슈 작성 화면 예시
title → [업무 영역?] 이슈 사항 간단히 작성
description → 첫 줄 띄고 내용 간단히 작성 !!



4. Assigness → 이슈 해결해야 할 담당자로 지정할지 이슈 업 한 사람을 담당자로 지정할지 정하기

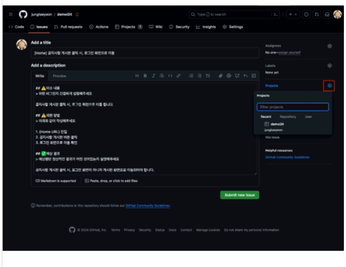


5. Lables → 아래 사진을 참고하여 알맞은 라벨 선택

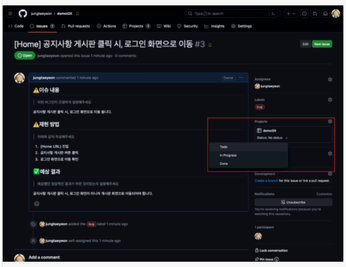


keyword	의미
bug	예기치 않은 문제 또는 의도하지 않은 동작(버그)을 나타냅니다.
documentation	문서를 개선하거나 추가 할 필요가 있음을 나타냅니다.
duplicate	해당이슈 또는 PR이 기존에 있음을 나타냅니다.
enhancement	새로운 기능 요청을 나타냅니다.
good first issue	처음 기여해볼 사람에게 좋은 문제를 나타냅니다.
help wanted	관리자가 문제 또는 PR 요청에 대한 도움을 원함을 나타냅니다.
invalid	이슈 또는 PR 요청이 더 이상 관련이 없음을 나타냅니다.
question	이슈 또는 풀 요청에 추가 정보가 필요함을 나타냅니다.
wontfix	문제 나 PR 요청에서 작업이 계속되지 않음을 나타냅니다.

6. Project 선택 후 Submit new issue 버튼 클릭



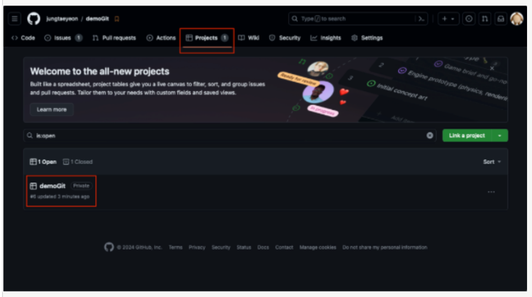
7. 업로드한 이슈 확인 → 프로젝트에 Status: Todo 로 설정 → 이슈 등록 완료!



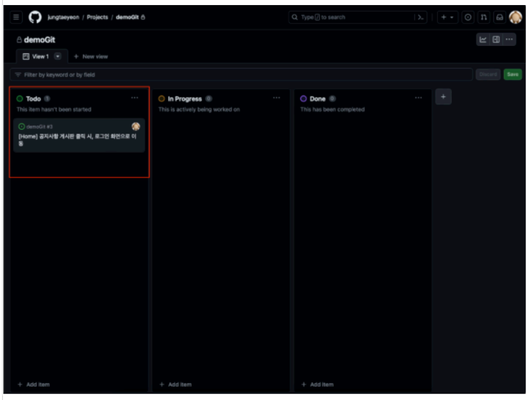
가이드 6) 깃 프로젝트 관리

** 프로젝트 관리

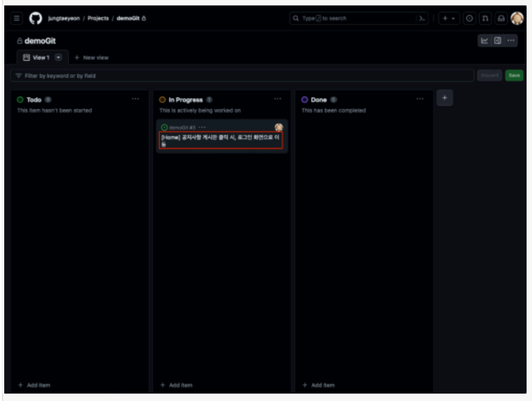
1. GNB > Projects탭 클릭 후 현재 진행중인 프로젝트 클릭



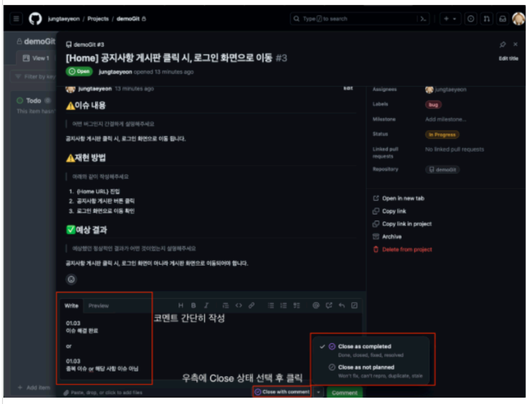
2. Todo → 이슈 등록 후 아무도 가져가지 않은 상태
: 자신이 수정 및 해결해야할 이슈 In Progress로 드래그 해서 가져오기 !!



3. In Progress 상태에서 이슈 title 클릭해서 이슈 상세 팝업 노출



4. 이슈 상세 팝업 → 코멘트 작성 후 상황에 맞게 Close상태 선택 → Close with comment 버튼 클릭
: 중복 이슈 또는 이슈 수정사항이 아닌 경우, Lables 도 수정 필요! - Won't fix | duplicate



5. 이슈 Close 시, Project에서 자동으로 상태가 Done으로 상태 변경됨