

Coding Standard – UOSense

(시대생 맛집 지도)



2019920034 이도권

2019920015 김세윤

2019920018 김준호

2019920045 장규민

2021930028 최수아

목차

1. Code: Branch Description.....	4
1.1 브랜치 설명.....	4
2. PR Strategy	5
2.1 PR 제목 양식.....	5
2.2 PR 내용 양식.....	5
2.3 Reviewer로 상대방 지정.....	6
2.4 지정된 Reviewer는 코드 확인.....	6
2.5 코멘트 작성.....	7
2.6 수정 및 재검토	7
2.7 리뷰 과정을 거친 후 작업 브랜치를 dev 브랜치에 merge	7
2.8 PR을 닫고 작업 브랜치를 삭제	7
3. Coding Standard: Naming Convention	8
3.1 상수	8
3.1.1 설명.....	8
3.2 클래스와 인터페이스 이름	8
3.2.1 설명.....	8
3.3 변수 이름	8
3.3.1 설명.....	8
3.4 메소드 이름	8
3.4.1 설명.....	8
3.5 패키지 이름	9
3.5.1 설명.....	9
3.6 Json 내 파라미터 이름	9
3.6.1 설명.....	9
3.7 블록	9
3.7.1 설명.....	9
3.8 문장과 수식	9
3.8.1 설명.....	9
3.9 오류 처리.....	9
3.9.1 설명.....	9
3.10 주석	10
3.10.1 설명.....	10
3.11 메서드 주석	10
3.11.1 설명.....	10
3.12 Swagger 문서 주석.....	10
3.13 컨트롤러 및 서비스의 CRUD 메서드	11
3.13.1 설명.....	11
3.14 API Path.....	12
3.14.1 설명.....	12

Revision History(변경 이력)

버전	일자	변경 내역	작 성 자
Ver 1.0	2024/11/19	Initial version	김세윤
Ver 2.0	2024/12/14	문서 통합	최수아

1. Code: Branch Description

1.1 브랜치 설명

1. Github에서 organization을 새로 생성한다.
2. repository를 프론트엔드는 **Front-End**, 백엔드는 **Back-End**로 각각 명명한다.
3. 해당 repository에서 배포용 브랜치 이름은 **main**, 작업 내용을 병합할 개발용 브랜치는 **dev**로 명명한다.
4. Github에서 이슈 생성, 번호 확인하고 **issue/#이슈번호**로 작업 브랜치를 생성한다.
 - A. 이슈에 적은 목표를 달성한다.
 - B. 로컬 환경에서 테스트한다.
 - C. 수정 사항을 **commit**하고 원격 저장소로 **push**한다.
 - D. PR을 생성하고 **Reviewer**로 상대방을 지정한다.
 - E. 지정된 **Reviewer**는 코드를 보고 코멘트를 남긴다.
 - F. 리뷰 과정을 거친 후 작업 브랜치를 **dev** 브랜치에 **merge**한다.
 - G. **issue**를 닫고 작업 브랜치를 삭제한다.
5. 4번을 반복해 수행한다. □ 기능 구현 완료 시 까지
6. 배포는 각각의 repository를 **clone**해서 한다.
7. 최종 완성본 코드는 교수님이 계신 **software-engineering_1** repository에 올린다.

2. PR Strategy

2.1 PR 제목 양식

1. 일반 커밋 제목은 `type: subject`, PR 제목은 `[type] subject` 이런 식으로 구분한다.

2. 타입 종류

A. `feat`: 새로운 기능 추가

B. `fix`: 버그 수정

C. `docs`: 문서 수정

D. `style`: 코드 포매팅, 세미콜론 누락, 코드 변경이 없는 이유

E. `refactor`: 코드 리팩토링

F. `test`: 테스트 코드, 리팩토링 테스트 코드 추가

G. `chore`: 빌드 업무 수정, 패키지 매니저 수정

3. 예시 □ `feat`: 관리자 페이지 기능 추가

2.2 PR 내용 양식

1. `body`: 작업 요약, 왜 이렇게 했는지, 팀원에게 공유 필요한 정보, 팀원이 코드를 이해하는데 도움이 되는 정보 등

2. `footer`: 꼬리말, `resolves`, `fixes` 등과 함께 `issue/#`이슈번호를 붙인다.

3. 예시

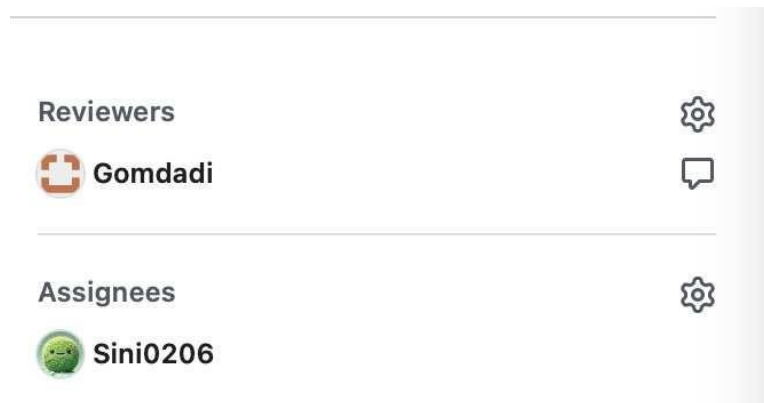
A. `body`

- 메뉴 정보 수정 제안 삭제 API 추가
- Restaurant 메서드명 `naming convention`에 맞게 포매팅
- `PurposeRestImgRepository`에서 JPA가 쿼리를 생성하지 못해 `nativeQuery`로 변경

B. `footer`

- `resolves: issue/#107`

2.3 Reviewer로 상대방 지정

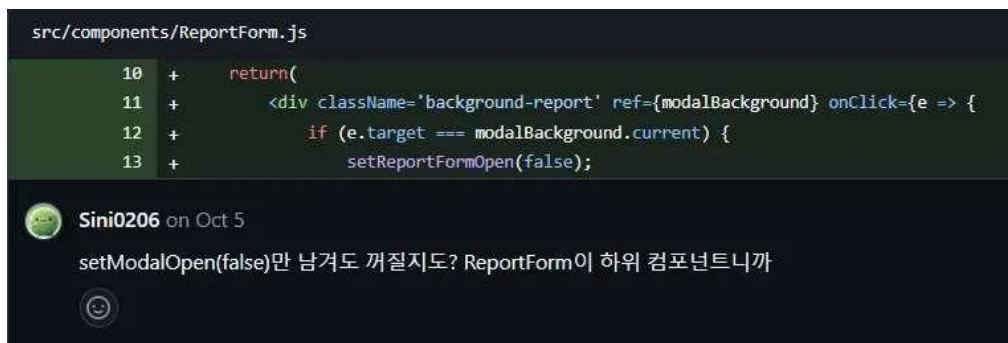
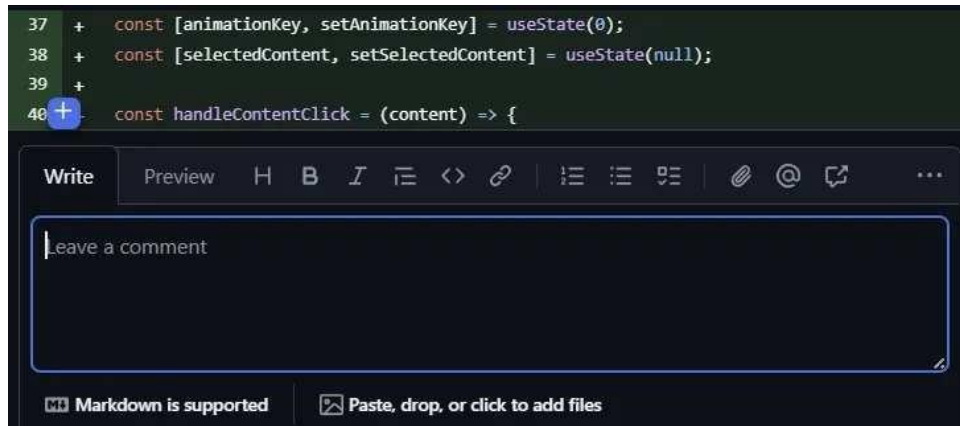


2.4 지정된 Reviewer는 코드 확인

1. 기능 확인: 요구 사항을 충족하는지
2. 코드 품질: 코드가 읽기 쉽고, 유지보수가 가능한지
3. 버그 검출: 잠재적인 버그, 논리적 오류, 예외 처리 누락 여부 확인
4. 스타일 준수: Coding Standard를 잘 준수했는지
5. 성능: 비효율적인 코드가 있는지

2.5 코멘트 작성

1. 특정 코드에 대한 코멘트를 원할 경우, “+” 버튼 클릭



2.6 수정 및 재검토

1. 리뷰어의 피드백에 따라 코드를 수정한 뒤, 다시 push □ 2.4로 이동
2. 코멘트가 더이상 없을 때까지 반복

2.7 리뷰 과정을 거친 후 작업 브랜치를 dev 브랜치에 merge

2.8 PR을 닫고 작업 브랜치를 삭제

3. Coding Standard: Naming Convention

3.1 상수

3.1.1 설명

1. 모두 대문자
2. 단어 구분 필요할 경우 Snake Case

3.2 클래스와 인터페이스 이름

3.2.1 설명

1. 대문자로 시작(명사 또는 명사구)
2. Pascal Case

3.3 변수 이름

3.3.1 설명

1. 일반적으로 소문자 시작
2. 최소 1 자, 최대 20 자 길이의 제한을 둘 것
3. 모호한 이름 사용하지 않기

3.4 메소드 이름

3.4.1 설명

1. Camel Case
2. 일반적으로 소문자 시작
3. 매개변수는 전치사구로 표현
 - A. 웬만하면 By로 통일
 - B. e.g., findByCategory
4. 조건 묻는 boolean 함수 이름은 "is"로 시작
5. 메소드명의 경우 글자 수 제한 삭제

3.5 패키지 이름

3.5.1 설명

1. Camel Case

3.6 Json 내 파라미터 이름

3.6.1 설명

1. Camel Case

3.7 블록

3.7.1 설명

1. 일반적인 명명 규칙을 따름
2. if-else문 같은 경우 아래 예시와 같이 블록을 이어서 작성

```
if (flag) {  
  
    validate();  
  
    update();  
  
} else {
```

3.8 문장과 수식

3.8.1 설명

1. 문장이나 수식은 메소드나 클래스로 패키지화
2. 블록 문장
3. 제어구조 중첩되었을 때 발생하는 혼란 방지
4. 수식
 - A. 괄호를 이용하여 오퍼레이션의 순서를 명확히

3.9 오류 처리

3.9.1 설명

1. 오류 발생 시 주석 처리

3.10 주석

3.10.1 설명

1. `/**` 혹은 `//` 추가

3.11 메서드 주석

3.11.1 설명

1. 매개 변수와 리턴 값, 예외 표시 가능
2. 메소드 상단에 `/* */` 사용
3. 필수 작성 사항 아님
4. 예시

```
/*  
  
 *      @param username  
  
 *      @throws IllegalArgumentException  
  
 *      @return UserEntity  
  
*/
```

3.12 Swagger 문서 주석

3.12.1 설명

1. 아래 예시와 같이 tag, operation, API Response 작성
2. Tag : 컨트롤러 역할 표기
 - A. `@Tag(name = "댓글 관리")`
3. operation : 컨트롤러 메소드 기능 표기
 - A. `@Operation(summary = "댓글 삭제", description = "기존 댓글을 삭제합니다.")`
4. APIResponse : 응답 코드에 따른 설명 표기
 - A. 같은 코드는 설명 형식 통일
 - B. 예시1
 - i. 200: ~이 성공적으로 ~되었습니다
 - ii. 400: 잘못된 요청입니다.
 - iii. 500: 서버 오류입니다 / 네트워크 연결에 문제가 있습니다.

C. 예시 2

```
@ApiResponses(value = {  
    @ApiResponse(responseCode = "200", description = "댓글이  
성공적으로 저장되었습니다."),  
  
    @ApiResponse(responseCode = "404", description = "해당  
댓글을 찾을 수 없습니다."),  
  
    @ApiResponse(responseCode = "400", description = "잘못된  
요청입니다.")  
})
```

3.13 컨트롤러 및 서비스의 CRUD 메서드

3.13.1 설명

1. Controller

A. 컨트롤러 클래스 안에서 메서드명을 작성할 때는 아래와 같은 동사로 시작

B. create: 등록 유형의 controller 메서드

C. get: 조회 유형의 controller 메서드

i. 전체목록일 경우 뒤에 list를 붙임. e.g. getListByDoorType

D. update: 수정 유형의 controller 메서드

E. delete: 삭제 유형의 controller 메서드

2. Service

A. 서비스 클래스 안에서 메서드명을 작성할 때는 아래와 같은 동사로 시작

B. register: 등록 유형의 service 메서드

C. find: 조회 유형의 service 메서드

i. 특정 파라미터로 조회할 경우 findBy{파라미터} 형식으로 명명

ii. 단, 파라미터가 기본적인 값일 경우 By~ 생략

D. edit: 수정 유형의 service 메서드

E. remove: 삭제 유형의 service 메서드

3. 행위 대상이 default 하위일 경우 동사 뒤에 붙이고 default 일 경우 생략

A. e.g. saveImage(사진 저장), remove(식당 삭제)

3.14 API Path

3.14.1 설명

1. **default path:** /api/v1/domain
2. HTTP 요청 방식 유형 별로 아래와 같은 형식을 따름
3. GET
 - A. /domain/get
4. POST
 - A. /domain/create
5. PUT
 - A. /domain/update
6. DELETE
 - A. /domain/delete

