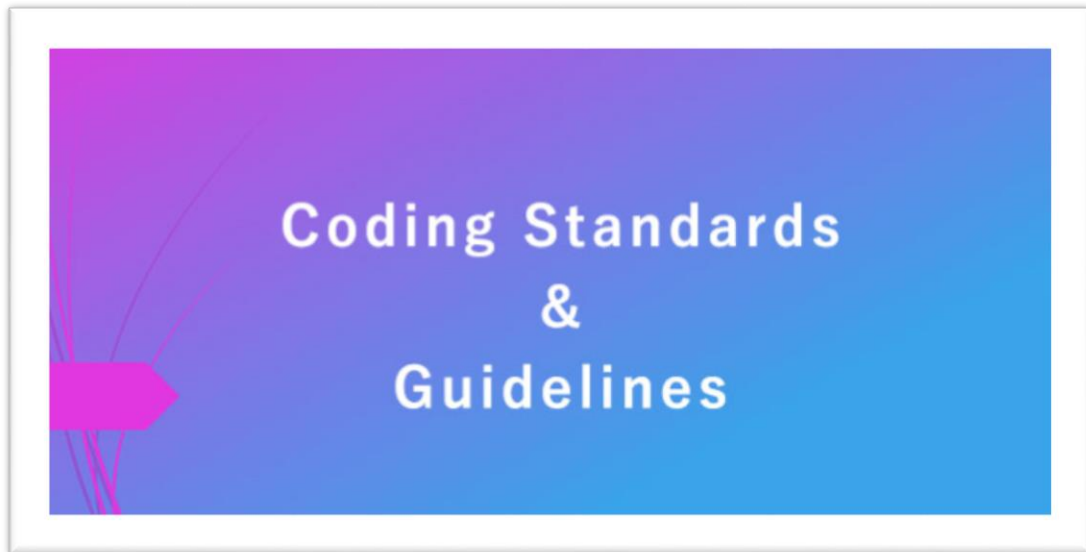


Assignment 3: Coding Standard, Repository Management and Review Process (UOSense)



본 소프트웨어 시스템의 목적은 서울시립대학교 학생들을 위해 학생들만의 솔직한 리뷰와 추천으로 구성된 서울시립대 주변 맛집 탐색 지도를 제공하는 것입니다. 이 시스템은 학생들이 맛집 선택 시 겪는 고민을 덜어주고, 다양한 의견과 정보를 바탕으로 최적의 선택을 할 수 있도록 돕습니다. 학생들이 작성한 솔직하고 다양한 관점의 리뷰를 바탕으로 맛집에 대한 신뢰도 높은 정보를 제공하여, 사용자가 자신의 취향에 맞는 맛집을 쉽게 찾을 수 있게 합니다.

일반적인 명명 규칙을 토대로 한 UOSense 명명 규칙

1. 상수는 모두 대문자
2. 클래스와 인터페이스 이름
 - 대문자로 시작(명사 또는 명사구)
 - Upper Camel case
3. 변수 이름
 - 일반적으로 소문자 시작
 - 최소 1 자, 최대 20 자 길이의 제한을 둘 것
 - 모호한 이름 사용하지 않기
4. 메소드 이름
 - Lower camel case
 - 일반적으로 소문자 시작
 - 매개변수는 부사구로 표현
 - e.g. 주소변환 with 좌표 혹은 주소변환 By 좌표, 주소변환 From 좌표 etc.
 - e.g., findByCategory
 - 조건 묻는 boolean 함수 이름은 "is"로 시작
5. 패키지 이름
 - 소문자
6. Json 내 파라미터 이름
 - Camel Case
7. 블록은 일반적인 명명 규칙을 따름
8. 문장과 수식

- 문장이나 수식은 메소드나 클래스로 패키지화
- 블록 문장
 - 제어구조 중첩되었을 때 발생하는 혼란 방지
- 수식
 - 괄호를 이용하여 오퍼레이션의 순서를 명확히

9. 오류 처리

- 오류 발생 시 주석 처리

10. 주석

- `/* */` 사용

11. 메서드 주석

- `/*`
 - `* @param username`
 - `* @return UserEntity`
 - `*/`
- 이런 방식으로 매개 변수와 리턴 값 표시

12. Swagger 문서 주석

- 아래 예시와 같이 tag, operation, ApiResponse 작성
- tag : 컨트롤러 역할 표기
 - `@Tag(name = "댓글 관리")`
- operation : 컨트롤러 메소드 기능 표기
 - `@Operation(summary = "댓글 삭제", description = "기존 댓글을 삭제합니다.")`
- ApiResponse : 응답 코드에 따른 설명 표기

- e.g. 200, 404, 500 일 시 설명 형식 통일 (~이 성공적으로 저장되었습니다, ~를 찾을 수 없습니다, 잘못된 요청입니다)
- @ApiResponses(value = {

 @ApiResponse(responseCode = "200", description = "댓글이 성공적으로 저장되었습니다."),

 @ApiResponse(responseCode = "404", description = "해당 댓글을 찾을 수 없습니다."),

 @ApiResponse(responseCode = "500", description = "잘못된 요청입니다.")

 })

13. 컨트롤러 및 서비스 메서드

○ Controller

- 컨트롤러 클래스 안에서 메서드 명을 작성할 때는 아래와 같은 접두사를 붙인다.
- create: 등록 유형의 controller 메서드
- get: 조회 유형의 controller 메서드
 - 전체목록일 경우 뒤에 list

 → e.g., getListByDoorType
- update: 수정 유형의 controller 메서드
- delete: 삭제 유형의 controller 메서드
- path parameter 로 id 를 넣을 때, 어떤 엔티티의 id 인지 명확히 한다
- {restaurantId}, {menuId} (o), {id} (x)

○ Service

- 서비스 클래스 안에서 메서드 명을 작성할 때는 아래와 같은 접두사를 붙인다.

- register: 등록 유형의 service 메서드
- find: 조회 유형의 service 메서드
 - find 일 경우 뒤에 +By 매개변수
- edit: 수정 유형의 service 메서드
- remove: 삭제 유형의 service 메서드
- 행위 대상이 default 하위일 경우 동사 뒤에 붙이고 default 일 경우 생략한다
 - e.g. remove(식당 삭제), removeMenu

Github Branch(Naming) Rules

1. Github 에서 organization 을 새로 판다
2. repository 를 프론트엔드는 Front-End, 백엔드는 Back-End 로 각각 명명한다.
3. 해당 repository 에서 배포용 브랜치 이름은 main, 작업 내용을 병합할 개발용 브랜치는 dev 로 명명한다.
4. Github 에서 이슈 생성, 번호 확인하고 issue/#이슈번호로 작업 브랜치를 생성한다.
5. 배포는 각각의 repository 를 clone 해서 한다.
6. 최종 완성본 코드는 교수님 계신 software-engineering_1 repository 에 올린다.

Code Review Process

1. PR 제목 양식
 - [type] subject
 - 일반 커밋 제목은 type: subject

- PR 제목은 [type] subject

- 타입 종류

- feat : 새로운 기능 추가
- fix : 버그 수정
- docs : 문서 수정
- style : 코드 포매팅, 세미콜론 누락, 코드 변경이 없는 경우
- refactor : 코드 리팩토링
- test : 테스트 코드, 리팩토링 테스트 코드 추가
- chore : 빌드 업무 수정, 패키지 매니저 수정

- e.g. feat: 관리자 페이지 기능 추가 ← 소문자로 시작

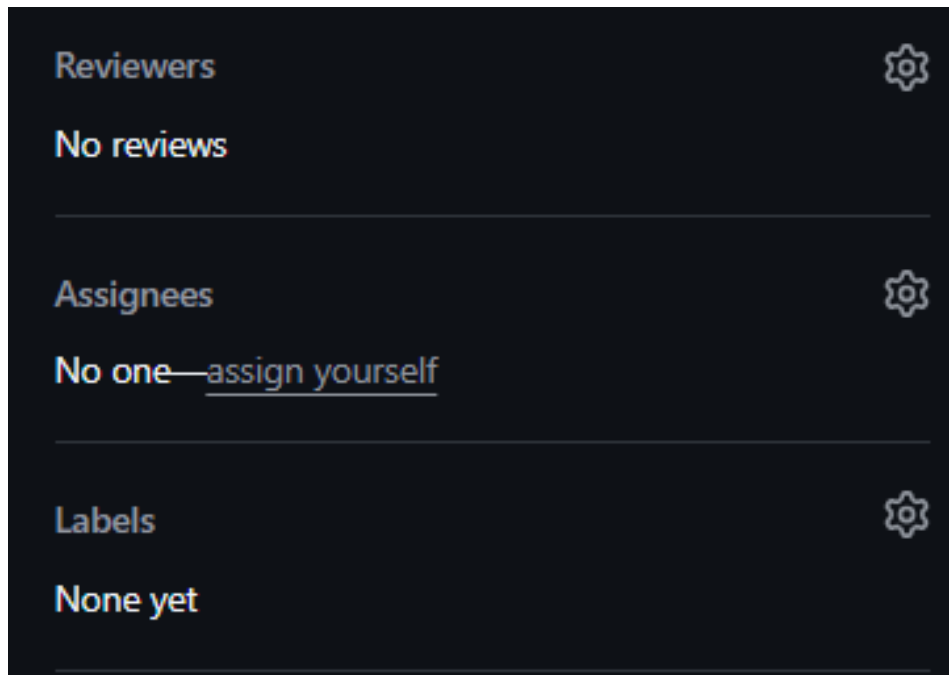
2. PR 내용 양식

body // 한 일 요약, 왜 이렇게 했는지, 팀원에게 공유 필요한 정보 등등

footer // 꼬리말 e.g. resolves, fixes

- footer resolves: #이슈번호 (이슈 해결 했을 경우)

3. Reviewer 로 상대방 지정



4. 지정된 Reviewer 는 코드 확인

- 기능 확인: 요구 사항 충족하는지
- 코드 품질: 코드가 읽기 쉽고, 유지보수 가능한지
- 버그 검출: 잠재적인 버그, 논리적 오류, 예외 처리 누락 여부
- 스타일 준수: Coding Standard 를 잘 준수했는지
- 성능: 비효율적인 코드가 있는지

5. 코멘트 작성

- 특정 코드에 대한 코멘트를 원할 경우 + 클릭


```
37 + const [animationKey, setAnimationKey] = useState(0);
38 + const [selectedContent, setSelectedContent] = useState(null);
39 +
40 + const handleContentClick = (content) => {
```

Write Preview H B I ≡ <> 🔗 ≡ ≡ ≡ 📎 @ ↗ ...


Leave a comment

📄 Markdown is supported 📁 Paste, drop, or click to add files

```
src/components/ReportForm.js
10 + return(
11 +   <div className='background-report' ref={modalBackground} onClick={e => {
12 +     if (e.target === modalBackground.current) {
13 +       setReportFormOpen(false);
```

 Sini0206 on Oct 5

setModalOpen(false)만 남겨도 꺼질지도? ReportForm이 하위 컴포넌트니까



6. 수정 및 재검토

- 리뷰어의 피드백에 따라 코드를 수정한 뒤, 다시 제출 → 4 번(반복)

7. 리뷰 과정을 거친 후 작업 브랜치를 base 브랜치에 merge

8. PR 을 닫고 작업 브랜치 삭제