Project 이룸오더
Coding Standard, Repository Management and Review Process - v2

<팀명: 이룸핑>

<팀원: 2022920028 박수빈

2022920050 이희진

2022920062 최진영

2022920040 양나슬

2022920016 김은지

2022920058 주영은>

변 경 이 력

버전	일자	변경 내역	작 성 자
V1	2024-11-20	coding standard, review guideline doc	김은지, 박수빈, 양나슬, 이희진, 주영은, 최진영
V2	2024-12-13	버전 관리, 커버 수정, TOC 수정	주영은

Table of Contents

1. Coding Standard

    a. naming - Camel Case(카멜 표기법)을 기본 명명 규칙으로 사용
        i. 상수 - 대문자로 작성하며, 단어는 밑줄(_)로 구분
            ex) MAX_LIMIT, DEFAULT_TIMEOUT
        ii. 패키지명 - 소문자로 작성
            ex) member.managing, category
        iii. 클래스와 인터페이스명 - 명사 또는 명사구로 표현하며, 첫 글자는 대문자로 시작
            ex) CategoryManaging, Order
        iv. 메서드명 - 동사구로 표현하며, 첫 글자는 소문자로 시작
            ex) calculateTotal(), addMenu()
        v. 파일 및 디렉터리 구조
            1. 파일 이름: 클래스명 또는 인터페이스명과 동일하게 설정 (첫 글자는 대문자)
                ex) Order.java, CategoryManaging.java
            2. 디렉터리 구조: 패키지 명명 규칙에 따라 계층적으로 구성
                ex) src.main.java.irumping.irumOrder
        vi. 변수명: 명확하고 간결하게 작성하며, 카멜 표기법 사용.
            1. boolean 변수: 긍정적 의미를 담는 이름으로 작성.
                ex) isAvailable

    b. 접근 제어자 사용: 접근 범위를 최소화하기 위해 기본적으로 private 사용
        i. 필요 시 protected 또는 public으로 확장.
        ii. 클래스 내부에서도 getter/setter를 통해 접근하도록 유도.

    c. 에러 처리: 예외 처리를 위해 try-catch 사용 시, 로그를 남기고 필요 시 메시지를 클라이언트에 반환

    d. 코드 정렬 및 들여쓰기
        i. 들여쓰기: 탭 사용
        ii. 중괄호 배치
            1. 여는 중괄호는 선언문과 같은 줄에 작성
            2. 닫는 중괄호는 중괄호로 시작된 블록의 끝에 같은 수준에서 작성(아래 예시)

            ```
            class(){

            }
            ```

e. 코드 길이 제한: 한 줄에 120자 이상 사용하지 않도록 제한
    i. 길어지는 경우 줄 바꿈을 통해 가독성 유지
    ii. 줄 바꿈 시 연속된 줄은 들여쓰기 정렬

f. 주석사용
    i. 코드의 특정 부분에 대한 보충 설명이 필요한 경우, 간략하게 인라인 주석 사용
        1. // 간략한 설명
    ii. 가장 도움이 될 외부 참조에 대한 링크를 인라인 주석을 사용해 나타냄
    iii. 아직 완전하지 않은 구현을 표시하기 위해서는 인라인 주석 사용
    iv. 파일/클래스 수준 주석 처리
        1. 파일/클래스의 역할과 목적 설명
        2. 작성자 정보, 수정 이력 기록

```
/**
 * 클래스 설명: ...
 * 작성자: ...
 * 마지막 수정일: xxxx-xx-xx
 */
```

    v. 메서드 수준 주석 처리
        1. JavaDoc 스타일 적용

```
/**
 * 메서드 역할 설명
 *
 * @param param1 매개변수1에 대한 간략한 설명
 * @param param2 매개변수2에 대한 간략한 설명
 * @return return 값에 대한 간략한 설명
 */
```

g. api 스웨거 규칙
    i. swagger는 OpenAPI 3.0 이상 사양을 준수해야한다.
    ii. 문서화 내용에는 다음 사항이 반드시 포함되어야 한다.
        1. API 설명: 기능 및 목적을 명시
        2. 요청 데이터: 필요한 매개변수, 데이터 타입, 예시 포함
        3. 응답 데이터: 응답 형식, HTTP 상태 코드별 설명
        4. 오류 메시지: 가능한 오류와 원인을 문서화
    iii. API 문서는 `docs/swagger.yaml`에 저장하며, 변경 사항 발생 시 반드시 업데이트한다.

iv. application.properties:

1. springdoc.api-docs.enabled=true # Swagger 문서화를 활성화
2. springdoc.api-docs.path=/v3/api-docs # API 문서의 기본 경로
3. springdoc.swagger-ui.path=/swagger-ui.html # Swagger UI 접근 경로
4. springdoc.swagger-ui.operations-sorter=method # API 메서드 기준으로 정렬

v. docs/swagger.yaml 파일 관리

1. http://localhost:8080/v3/api-docs :접근해서 json 형태로 swagger 명세를 반환한다.
2. JSON to YAML Online Converter : 접근해서 json형태의 파일을 yaml 파일로 변환하여 자신의 브랜치에 docs/swagger.yaml 파일 저장한다.
3. swagger editor를 사용하여 yaml 파일로 변환이 잘 되었는지 검토한다.
4. 프론트에서 합병할 branch 안의 docs/swagger.yaml 파일에서 스웨거 형식의 Open API 사용한다.

2. Github Branch (Naming) Rules

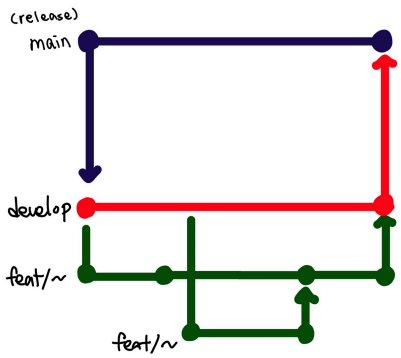
    1. Feature branches

       기능 구현에 쓰이는 branch 이름 앞에는 'feat/'이 붙어야 한다.
       프론트엔드 화면 관련 작업 branch 이름 끝에는 '-ui'를 추가한다.
       기능이 명확하게 드러나도록 branch 이름을 작성하고, 단어는 dash(-)로 구분한다.

    2. Merge Workflow

       유사 기능을 다루는 'feat/~' branches를 병합하여 test 후 'develop' branch에 병합한다.

    3. Git branch Strategy



    4. Current branch status

Main Branch

- **develop**
  - feat/order
  - feat/payment
  - feat/sales-view
  - feat/sign-up
  - feat/login-signup-ui
  - feat/user-main-ui
  - feat/order-management
  - feat/routine-alarm
  - feat/pickup-alarm
  - feat/menu
  - feat/routine-setting
  - feat/menu-management
  - feat/employee

Legend

11월 20일 이전의 Git graph를 확인하시면, 초기에 브랜치를 생성할 때 일부 feat/~ 브랜치들이 잘못 생성되어 develop 브랜치가 아닌 main 브랜치에서 시작된 것을 확인하실 수 있습니다.

이후 해당 브랜치들은 develop 브랜치로 병합될 것이며, 최종적으로 main 브랜치로 release될 예정입니다.

이 점 양해 부탁드리며, 참고해주시면 감사하겠습니다.

reference : https://inpa.tistory.com/entry/GIT-%E2%9A%A1%EF%B8%8F-github-flow-git-flow-%F0%9F%93%88-%EB%B8%8C%EB%9E%9C%EC%B9%98-%EC%A0%84%EB%9E%B5

3. Code Review Process
    a. 커밋 메시지

    < 구조 >

```
// Header, Body는 빈 행으로 구분한다.
타입(스코프): 주제(제목) // Header(헤더)

본문 // Body(바디)
```

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

    b. 필수 리뷰어(최소 2인) : 김은지, 이희진
    c. 코드 리뷰 시 체크리스트
        i. uml 다이어그램에 맞게 구현되었는가
        ii. 유지보수성
            1. 중요한 값을 하드코딩하고 있지 않은가

2. 주석이 코드가 하는 일이 아닌, 코드에 담긴 의도를 설명하고 있는가
3. 코드를 쉽게 이해할 수 있는가
4. 스타일 가이드에 맞게 작성되어 있는가
5. 하나의 함수가 10라인을 넘어간다면 너무 많은 관심사를 갖고 있는 건 아닌가

iii. 재사용성
1. 중복된 코드는 없는가

iv. 안정성
1. 예외 처리를 제대로 하고 있는가

v. 확장성
1. 새로운 기능을 추가하기 쉽게 작성되어 있는가
2. 함수나 클래스가 두 개 이상의 관심사를 갖고 있진 않은가

vi. 테스트
1. 테스트 코드가 작성되어 있는가

d. 코드 리뷰 과정
i. 코드 작성자가 커밋 메시지 형식을 따라 Pull request를 보낸 후 필수 리뷰어에게 코드 리뷰를 요청한다.
ii. 요청받은 리뷰어는 코드 내용을 코드 리뷰 시 체크리스트를 참고하여 검토한 후 피드백을 남긴다.
iii. 코드 작성자가 이를 반영한 후 다시 리뷰를 받는다.
iv. 필수 리뷰어 2명을 포함한 3명의 리뷰어에게 승인을 받으면 merge한다.

reference : https://github.com/meshkorea/front-end-engineering/blob/main/conventions/code-review/index.md