**Development GuideLine**

Unicircle-서울시립대 동아리 플랫폼



Team Universe

**2024.11.25. Version:2**

2020920006 김동하

2020920008 김민회

2020920023 박세환

2020920029 배유찬

2020920061 주재원

2022920041 오승민

변 경 이 력

|  |  |  |  |
| --- | --- | --- | --- |
| **버전** | **일자** | **변경 내역** | **작성자** |
| 1.0 | 2024-11-19 13:00 | 최초작성 | Universe |
| 2.0 | 2024-11-25 15:00 | [수정] Content - 디렉토리 구조 | 주재원 |
|  |  |  |  |
|  |  |  |  |

목 차

목차

[Coding Standard 3](#_Toc183441125)

[Naming Convention 3](#_Toc183441126)

[파일 및 디렉토리 구조(Next.js/FrontEnd) 5](#_Toc183441127)

[파일 및 디렉토리 구조(Spring/BackEnd) 6](#_Toc183441128)

[Repository Management 7](#_Toc183441129)

[Code Review Process: Github PR Rule 7](#_Toc183441130)

# Coding Standard

## Naming Convention

1. 상수 및 변수

* 상수 및 열거형: 대문자 + 언더스코어(\_) 사용

예: MAX\_VALUE, DEFAULT\_TIMEOUT

* 변수: Camel Case 사용

예: userName, userEmail

2. 클래스와 인터페이스 이름

* Pascal Case 사용 + 명사구

예: UserService, UserRepository

3. 메소드 이름

* Camel Case 사용 + 소문자 동사로 시작

예: getUser(), deleteUser()

4. 부울 함수 이름

* 조건을 묻는 함수는 is로 시작

예: isAvailable(), isLoggedIn()

5. 인터페이스 구현 클래스 이름

* Interface 이름 + 구현 클래스의 역할 추가

예: UserService의 구현 클래스 이름 -> UserServiceImpl, DefaultUserService

6. 테스트 클래스 이름

* 테스트 대상 클래스 이름 + Test

예: UserService의 테스트 -> UserServiceTest

7. 데이터베이스 관련 네이밍

* Entity: 대문자로 시작하는 단수형 명사. 필요할 경우 Pascal Case 적용

예: User, AdmissionForm

* Column: Snake Case 사용

예: user\_id, created\_at

8. 컴포넌트 관련 네이밍

* **리액트 컴포넌트:** Pascal Case사용, function 말고 const 방식 사용

예: const UserProfile = () => { ... }

9. 이벤트 처리 함수(callback)

* 매개 변수인 경우 (on{EventName})
* 지역 변수인 경우 (handle{EventName})

10. 훅(Hook)

* use로 시작하는 Camel Case사용

예: function useAuth() { ... }

## 파일 및 디렉토리 구조(Next.js/FrontEnd)

**1. 컴포넌트 파일(src/components)**

* Pascal Case사용
* 예: Header.js, UserProfile.js
* 클래스 명 동일

**2. 페이지 파일 (src/app)**

* nextjs의 app router 규칙에 따름

**3. 스타일 파일(src/styles)**

* 해당 컴포넌트와 동일한 이름을 사용하고, 확장자로 .module.css 사용
* 예: Header.module.css, UserProfile.module.scss

**4. 유틸리티 및 헬퍼 함수 파일(src/utils)**

* Camel Case 사용
* 예: formatDate.js, calculateTotal.js

**5. 서비스 파일(src/services)**

* Pascal Case 사용

**6. 사용자 정의 훅(src/hooks)**

* use로 시작하는 Camel Case사용
* 예: useAuth.js

**7. 사용자 정의 타입(src/define)**

* types로 끝나는 Pascal Case
* 예: ApplyTypes.js, UserTypes.js

## 파일 및 디렉토리 구조(Spring/BackEnd)

**1. 애플리케이션 파일 (src/BackendApplication.java)**

* Spring Boot 애플리케이션의 시작점이 되는 클래스

**2. 설정 파일 (src/config)**

* 보안과 관련된 설정 포함

**3. 컨트롤러 파일 (src/controller)**

* REST API 요청을 처리하는 클래스 포함

**4. 데이터 전송 객체 (DTO) (src/dto)**

* 클라이언트와 데이터 전송을 위한 객체 정의

**5. 요청 DTO 파일 (src/request)**

* 클라이언트 요청 데이터를 처리하기 위한 DTO

**6. 응답 DTO 파일 (src/response)**

* 서버 응답 데이터를 처리하기 위한 DTO

**7. 엔티티 파일 (src/entity)**

* 데이터베이스와 매핑되는 클래스 포함

**8. 저장소 파일 (src/repository)**

* 데이터베이스 작업을 수행하는 인터페이스 포함

**9. 서비스 파일 (src/service)**

* 비즈니스 로직을 처리하는 클래스 포함

**10. 유틸리티 파일 (src/util)**

* 공통적으로 사용되는 유틸리티 클래스 포함

# Repository Management

**main**

* 제품으로 출시될 수 있는 브랜치
* 사용자에게 배포 가능한 상태만을 관리

**develop**

* 다음 출시 버전을 개발하는 브랜치
* 기능 개발을 위해 브랜치들을 병합하기 위해 사용, 즉 모든 기능이 추가되고 버그가 수정되어 배포 가능한 안정적인 상태라면 develop 을 main 으로 병합
* 평소에는 이 브랜치 기반으로 개발을 진행

**Feature**

* 새로운 기능 개발 및 버그 수정이 필요할 때마다 develop 브랜치로부터 분기
* 개발이 완료되면 develop 브랜치로 merge 하여 다른 사람들과 공유
* feature/front/기능요약 형태로 사용
* 예: feature/front/login, feature/back/login-api

feature/front/{issue-number}-기능요약 ex) feature/front/#77-login, feature/back/#77-login-api

# Code Review Process: Github PR Rule

main, develop 브랜치의 경우 pull request를 merge하기 위해서는 요청한 사람 이외의 한 명의 approval이 있어야 함