

# 프로젝트 구조와 규칙

## 기술 스택 / 버전

| 구분              | 기술                          | 버전    | 역할 / 설명                                     |
|-----------------|-----------------------------|-------|---|
| JDK             | Temurin JDK                 | 21    | 최신 LTS, Spring Boot 3.x 호환, 성능/보안 안정성 확보    |
| Spring Boot     | Spring Boot                 | 3.5.4 | 애플리케이션 프레임워크, 의존성 관리, DI, MVC 지원            |
| JPA / Hibernate | Spring Data JPA + Hibernate | 3.x   | ORM 기반 DB 매핑, Entity 관리, Repository CRUD 지원 |
| Database        | MariaDB                     | 11.x  | 관계형 DBMS, MySQL 호환, 개발용 Docker 컨테이너 운영 가능   |
| Build Tool      | Gradle                      |       | 의존성 관리, 빌드, 배포, 테스트 관리                      |
| Version Control | Git / GitHub                | 최신    | 브랜치 전략 (Feature/Develop/Main), 협업, 코드 리뷰    |
| API 테스트         | Postman                     | 최신    | REST API 테스트 및 요청/응답 확인                     |

빌드툴 Maven or Gradle 상의해서 정합시다

메이븐(Maven)과 그라들(Gradle)의 개념 및 비교

스프링과 스프링부트를 공부 하려던 중 maven과 gradle을 알게되었습니다. maven과 gradle이 빌드관리도구인 것은 알고있지만 자세한 개념은 모르기에 maven과 gradle 각각의 개념과 정확한 차이점을 알

 <https://dev-coco.tistory.com/65>



## 라이브러리

| 라이브러리명                                    | 설명                        | 비고            |
|---|---------------------------|---------------|
| <code>spring-boot-starter-web</code>      | Spring MVC 기반 웹 애플리케이션 개발 | REST API 기본   |
| <code>spring-boot-starter-data-jpa</code> | JPA + Hibernate ORM 사용    | 엔티티, 레포지토리 구현 |

| 라이브러리명                                      | 설명                            | 비고   |
|---|-------------------------------|--|
| <code>spring-boot-starter-validation</code> | Bean Validation(JSR-380) 지원   | DTO 유효성 검사   |
| <code>lombok</code>                         | Getter, Setter, Builder 자동 생성 | 개발 편의성 ↑, <code>@Builder</code> , <code>@Getter</code> 등 |
| <code>mariadb-java-client</code>            | MariaDB 연결 드라이버               | DB 연결 필수   |
| <code>spring-boot-starter-test</code>       | JUnit5 + MockMvc + AssertJ 포함 | 테스트 환경 구성  |

추가 고려 사항

## Spring Boot Starter Security

### [Spring Boot] 24. 스프링 시큐리티(Spring Security)

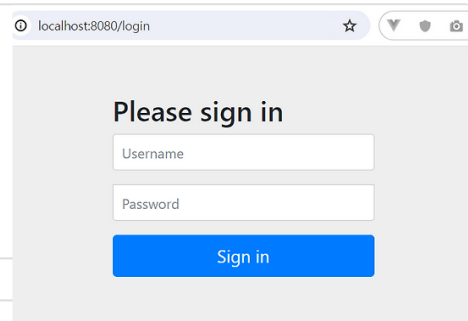
스프링 시큐리티란? 스프링 부트에서는 회원가입 및 로그인에 도움을 주는 스프링 시큐리티를 사용할 수 있다. 스프링 시큐리티는 스프링 기반 웹 애플리케이션의 인증과 권한을 담당하는 스프링의 하위 프레임워크

<https://exuzii.tistory.com/entry/Spring-Boot-24-%EC%A4%ED%94%84%EB%A7%81-%EC%8B%9C%ED%81%90%EB%A6%AC%ED%8B%B0Spring-Security>

### Getting Spring Security :: Spring Security

This section describes how to get the Spring Security binaries. See Source Code for how to obtain the source code.

<https://docs.spring.io/spring-security/reference/getting-spring-security.html>

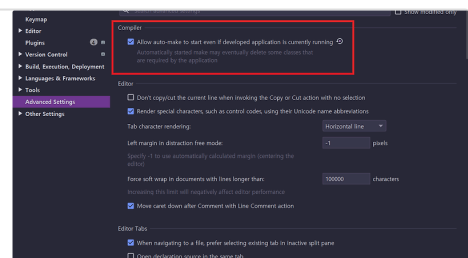


## Developer Tools

### Spring Boot Devtools 사용법

Devtools란? devtools는 Spring boot에서 제공하는 개발 편의를 위한 모듈이다. 개발을 하다보면, 코드 수정시 브라우저에서 보여주는 내용도 수정하려면 어플리케이션을 재시작해야 하기 때문에 불편한 점

<https://barbera.tistory.com/47>



### Developer Tools :: Spring Boot

Spring Boot includes an additional set of tools that can make the application development experience a little more pleasant.

The spring-boot-devtools module can be included in any project to provide additional development-

<https://docs.spring.io/spring-boot/reference/using/devtools.html>

## 도메인 패키징형 구조

---

예)

src/main/java/com/example/project

```
├── user
│   ├── User.java          # Entity
│   ├── UserController.java # REST Controller
│   ├── UserService.java    # Service
│   ├── UserRepository.java # Repository
│   └── dto
│       ├── UserRequest.java
│       └── UserResponse.java
```

```
├── order
│   ├── Order.java
│   ├── OrderController.java
│   ├── OrderService.java
│   ├── OrderRepository.java
│   └── dto
│       ├── OrderRequest.java
│       └── OrderResponse.java
```

```
└── common    #common: 전역 설정, 예외 처리, 공통 유틸
    ├── config    # Security, CORS, Jackson, WebMvc, OpenAPI, JPA
    └── Auditing
        ├── exception    # ErrorCode, ErrorResponse, GlobalExceptionHandler
        └── util
```

### 공통 모듈

- `common.config` → 전역 설정
- `common.exception` → 전역 예외 처리
- `common.util` → 재사용 가능한 유틸리티 클래스

### 담당자의 작업 범위 명확히 분리

예:

- `user` 도메인 담당자는 `user/` 폴더만 작업.
- `order` 담당자가 `order/` 폴더만 작업.

## Controller

- 클라이언트(웹/앱) 요청을 받아 처리
- 요청을 서비스 계층에 전달하고, 응답을 반환
- HTTP 요청/응답, DTO 변환, 검증 처리 담당
- 비즈니스 로직 거의 없음
- 요청과 응답만 처리

## DTO

- 도메인 객체와 혼동되지 않도록 `dto` 하위 패키지로 분리
- 계층 간 데이터 전달용 객체
- 주로 Controller와 Service, Service와 Repository 사이에서 데이터 전달
- 외부 요청(Request)이나 응답(Response)에 사용

## Service

- interface 기반, 구현은 Impl로 명확히 분리  
예: `UserService` (인터페이스), `UserServiceImpl` (구현체)
- 서비스 구현체의 메서드에 `@Transactional` 명시
- 핵심 비즈니스 로직 처리
- 여러 레포지토리 호출, 도메인 엔티티 처리
- 컨트롤러와 레포지토리 사이의 **조정자** 역할

## Entity

- 빌더 패턴 기반 생성
- 불변성 유지

- 비즈니스 로직 일부 가질 수 있음 (예: 상태 변경 메소드)

## Repository

- DB와 직접 통신
- JPA를 통해 Entity를 조회/저장/삭제
- 쿼리 작성 (JPQL, QueryDSL 등)
- 비즈니스 로직 없음
- DB 작업만 담당
- 보통 인터페이스로 선언하고 Spring Data JPA가 구현

## JPA

### 엔티티 규칙

| 규칙   | 설명                                 | 예시   |
|--|------------------------------------|--|
| Setter 금지                                    | 무분별한 값 변경 방지, 생성자/메서드로 값 변경        | 업데이트와 같은 수정이 필요할 시 <code>updateName()</code> 메서드 사용                |
| <code>@Builder</code> + 생성자                  | 필수 필드만 받도록 생성자 설계                  | <code>@Builder public User(String name, String email) {...}</code> |
| 컬럼명 명시                                       | DB 변경에도 안전하게                       | <code>@Column(name = "user_name")</code>                           |
| <code>@Enumerated(EnumType.STRING)</code> 필수 | ENUM 저장 시 기본 값(ORDINAL) 금지 → 버그 예방 | <code>@Enumerated(EnumType.STRING)</code>                          |

### 연관관계 매핑 규칙

| 규칙             | 설명   | 예시  |
|----------------|--|---|
| 지연 로딩(LAZY) 기본 | <code>@ManyToOne</code> , <code>@OneToOne</code> 은 LAZY 기본 | <code>@ManyToOne(fetch = FetchType.LAZY)</code> |
| 양방향 최소화        | 불필요한 양방향 관계 지양, 필요 시 주인 명확히                                | Order ↔ User 중 주인은 Order                        |
| 단방향 우선         | 가능하면 단방향 매핑 → 유지보수 쉬움                                      | User → Order만                                   |

## 쿼리 작성 규칙

| 규칙             | 설명                          | 예시  |
|----------------|-----------------------------|---|
| 단순 조회 → 메서드 쿼리 | findByName, existsByEmail 등 | List<User> findByName(String name)                |
| 복잡 조회 → @Query | 복잡한 조건은 JPQL 사용             | @Query("SELECT u FROM User u WHERE u.age > :age") |

## 트랜잭션 규칙

| 규칙        | 설명                                 |
|-----------|------------------------------------|
| 서비스 계층에서  | @Transactional 적용                  |
| 읽기 전용 조회는 | @Transactional(readOnly = true) 적용 |

## 네이밍 규칙

| 대상             | 규칙                    | 예시                        |
|----------------|-----------------------|---------------------------|
| Entity         | 단수형, PascalCase       | User, Order               |
| Repository 메서드 | find + By조건           | findByEmail, existsById   |
| DTO            | Request, Response 접미사 | UserRequest, UserResponse |

JPA Method 기본사용법(한국어) / 공식문서(영어)

학원에서 받은 책에도 어느 정도 적혀있는 것으로 알고 있음 참고 바람

### [Spring Data JPA] 기본 사용법 정리

전에 공부했던 Spring Data JPA의 기본 사용법을 정리하고자 한다.

✓ Dependency build.gradle 파일의 dependencies 부분에 다음을 추가하자. implementation

🔗 <https://wisdom-cs.tistory.com/66>

Spring Data JPA

### JPA Query Methods :: Spring Data JPA

This section describes the various ways to create a query with Spring Data JPA.

🔗 <https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html>

## 코드 규칙

- 클래스명: PascalCase ( UserService, OrderController )

- 메서드명 & 변수명: `camelCase` ( `findUserId` , `orderList` )
- 상수: `UPPER_SNAKE_CASE` ( `MAX_LOGIN_ATTEMPTS` )
- 패키지명: 모두 소문자, 복합어는 점( `.` )으로 구분 ( `com.example.project.domain.user` )
- DTO: 요청 → `SomethingRequest` , 응답 → `SomethingResponse`
  - 행위 + 도메인 + 요청/응답 형식
- 예외: `SomethingNotFoundException` , `InvalidSomethingException`
- 테스트 클래스: `클래스명Test` ( `UserServiceTest` )
- 메서드는 어떤 행위를 수행하는지 정확한 의도 전달  
예) `createUser`, `updateUser`, `deleteUser`

## 참고


### Google Java Style Guide

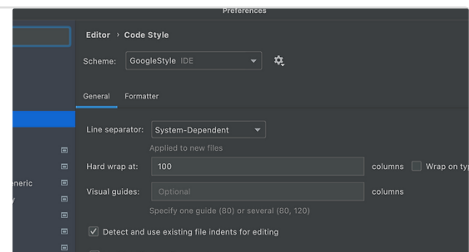
This document serves as the complete definition of Google's coding standards for source code in the Java™ Programming Language. A Java source file is described as being in Google Style if and only if it adheres to the rules herein.

 <https://google.github.io/styleguide/javaguide.html>

### [IntelliJ] Google Java auto-formatting 적용

1) 서로 여러 사람이 함께 코드를 작성할 때는 다양한 스타일의 코드가 작성됩니다. 같은 목적의 코드라도, 개인의 스타일은 다를 수밖에 없는데요. 만약 큰 회사 혹은 조직이라면 통일된 코드 스타일을 만들수도

 <https://yeon-kr.tistory.com/197>



## Git 브랜치 전략

디폴트 브랜치는 **develop**으로 변경

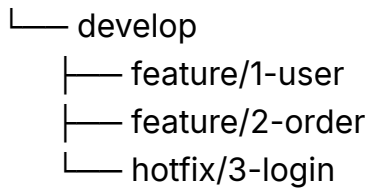
### Git Flow

main → 운영 배포용  
develop → 개발 통합 브랜치  
feature/\* → 기능 개발 브랜치  
hotfix/\* → 운영 긴급 수정

## 브랜치명 예시

이슈번호-도메인

main



공통 로직( **common** ) 건드릴 경우 미리 팀원들에게 해당 내용 공유

## Commit 메시지 규칙

### 형식

[타입] 내용

예

[feat] 회원가입 API 구현

[fix] 로그인 시 인증 버그 수정

[refactor] OrderService 결제 로직 리팩토링

[docs] README에 API 사용법 추가

### 타입 규칙

- **feat** : 새로운 기능 추가
- **fix** : 버그 수정
- **refactor** : 코드 리팩토링 (기능 변경 없음)
- **style** : 코드 포맷 변경, 세미콜론 누락, 공백 등
- **test** : 테스트 코드 추가/수정
- **chore** : 빌드/환경설정 변경, 라이브러리 추가
- **docs** : 문서 수정 (README 등)

## PR 규칙

- PR 제목: **[feat] 회원가입 API 구현** 처럼 Commit 타입과 동일
- PR 템플릿 필수 항목:




- 작업 내용
  - 관련 이슈 번호
  - 최소 1명 이상 코드 리뷰 후 머지
  - default branch는 develop  
develop 과 main 브랜치는 **무조건 PR로만 머지**, 직접 push 금지
  - 이슈와 PR은 템플릿 이용해서 사용 예정
- 템플릿 예시

git\_test/.github at main · wjdvl5456/git\_test  
Git 원격 저장소 테스트. Contribute to wjdvl5456/git\_test development by creating an account on GitHub.  
[https://github.com/wjdvl5456/git\\_test/tree/main/.git](https://github.com/wjdvl5456/git_test/tree/main/.git)  
hub

wjdvl5456/git\_test

Git 원격 저장소 테스트



1 Contributor

1 Issue

0 Stars

0 Forks