
SpringBoot 샘플 가이드

서비스 개발 3팀

문서 유형	가이드 문서
버전	V1.1
작성일	2024-07-31

1. 프로젝트 생성 (기본 설정)

The image shows the Spring Initializr web form for creating a new project. It includes sections for Project, Language, Spring Boot, Project Metadata, Packaging, and Java version. Numbered orange circles (1-6) highlight specific settings: 1. Project (Gradle - Groovy), 2. Language (Java), 3. Spring Boot (3.3.2), 4. Project Metadata (entire section), 5. Packaging (Jar), and 6. Java (21).

Project 1
☒ Gradle - Groovy ☐ Gradle - Kotlin ☐ Maven

Language 2
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot 3
☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M1) ☐ 3.3.3 (SNAPSHOT) ☒ 3.3.2
☐ 3.2.9 (SNAPSHOT) ☐ 3.2.8

Project Metadata 4

Group	kr.co.milkt
Artifact	demo
Name	demo
Description	현재교과서 SpringBoot 샘플
Package name	kr.co.milkt.demo

Packaging 5
☒ Jar ☐ War

Java 6
☐ 22 ☒ 21 ☐ 17

스프링부트는 <https://start.spring.io>에서 솔루션을 구성할 수 있습니다.

1. 프로젝트 빌드 유형을 선택합니다.
Gradle-Groovy를 선택합니다. Groovy와 Kotlin은 큰 차이가 없지만, Maven은 개발 환경을 저해하는 몇 가지 이슈가 있습니다.
 2. JAVA를 선택합니다.
 3. 스프링부트 프레임워크 버전은 기본적으로 선택되어 있는 3.3.2를 선택합니다.
 4. 프로젝트 기본 Package를 구성합니다.
 5. 패키징 방식은 JAR를 선택합니다.
 6. JDK 21을 선택합니다.
- 패키징이란?
개발된 소스파일들을 JAR 또는 WAR 하나의 파일로 묶어서 도커구성이나 배포를 편리하게 할 수 있도록 합니다. JAR의 경우 JVM에서 바로 실행 가능하도록, WAS (Tomcat)가 내부에 포함되어 있으므로 별도 WAS를 구성할 필요가 없습니다.

1. 프로젝트 생성 (Gradle 구성)

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Thymeleaf TEMPLATE ENGINES

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MyBatis Framework SQL

Persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations.

화면 좌측의 **ADD DEPENDENCIES** 버튼을 클릭하여 솔루션 구성요소를 추가할 수 있습니다.

Spring Web

REST API나 MVC 패턴의 웹 프로젝트를 구성

Lombok

어노테이션 기반의 코드 자동 생성 지원 (생성자, getter, setter 등 자동 구현을 지원)

Thymeleaf

HTML 템플릿 엔진 (asp.net mvc cshtml, Razor 등과 동일한 개념)

Spring Data Jpa

Java Persistence API (C#의 ENTITY FRAMEWORK와 비슷한 개념)

MyBatis Framework

쿼리 매핑 프레임워크

위 항목을 기본적으로 추가 합니다.

1. 프로젝트 생성 (솔루션 다운로드)

Packaging ☒ Jar ☐ War

Java ☐ 22 ☒ 21 ☐ 17

mappings. MyBatis couples objects with stored |
descriptor or annotations.

화면 하단의 GENERATE 버튼을 누르면 솔루션이 ZIP파일로 압축되어 다운로드 됩니다.

Click


GENERATE CTRL + ↵

EXPLORE CTRL + SPACE

SHARE...

1. 프로젝트 생성 (JDK21 다운로드)

Java downloadsTools and resourcesJava archive

 Looking for other Java downloads?

OpenJDK Early Access BuildsJRE for Consumers

Java 22, Java 21, and Java 17 available now

JDK 21 is the latest long-term support release of Java SE Platform.

Learn about Java SE Subscription

JDK 22JDK 21JDK 17GraalVM for JDK 22GraalVM for JDK 21GraalVM for JDK 17

JDK Development Kit 21.0.4 downloads

JDK 21 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

JDK 21 will receive updates under the NFTC, until September 2026, a year after the release of the next LTS. Subsequent JDK 21 updates will be licensed under the Java SE OTN License (OTN) and production use beyond the limited free grants of the OTN license will require a fee.

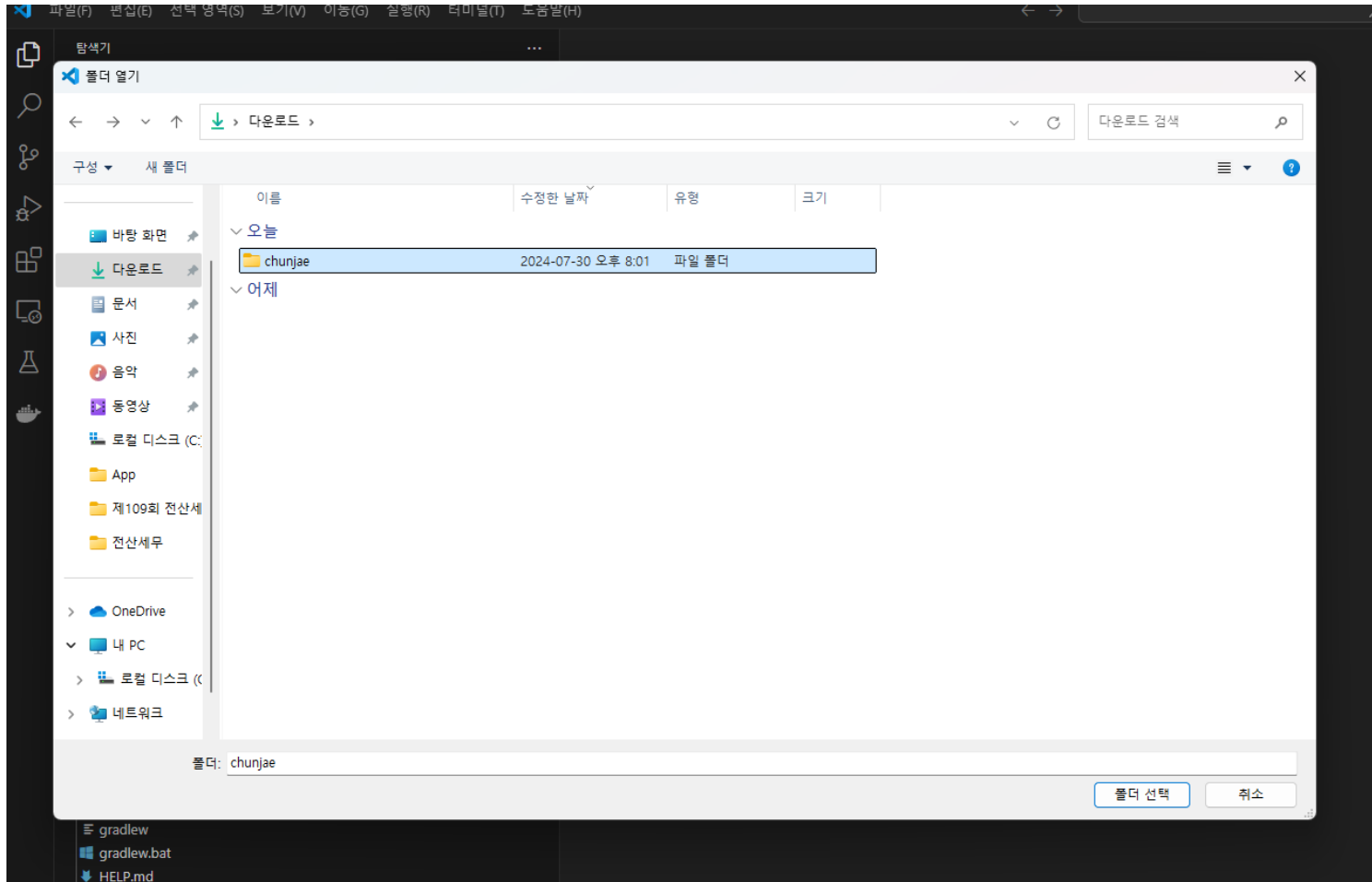
LinuxmacOSWindows

Product/file description	File size	Download
x64 Compressed Archive	185.84 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.zip (sha256)
x64 Installer	164.23 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe (sha256)
x64 MSI Installer	162.97 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.msi (sha256)

<https://www.oracle.com/kr/java/technologies/downloads/#java21>

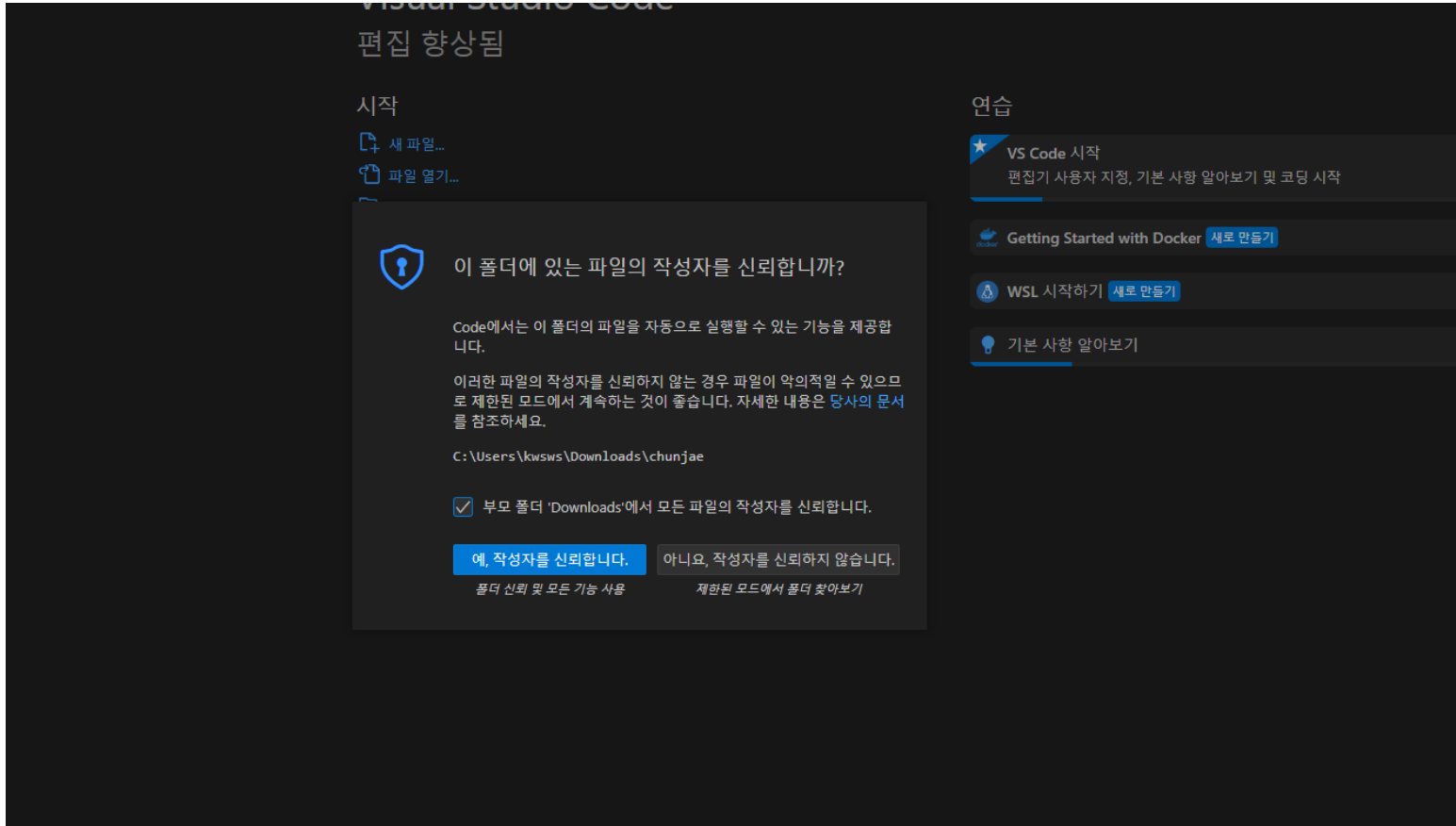
오라클에서 제공하는 JDK21 무료버전을 다운로드 받습니다.
다운로드 받은 후 설치합니다. (설치는 매우 간단하므로 자세한 설명은 생략)

1. 시작하기 (폴더 열기 1)



1. 다운로드 받은 zip파일의 압축을 푼 후..
2. VS CODE 를 실행
3. 파일 → 폴더 열기를 클릭하여 압축을 푼 폴더를 선택합니다.

1. 시작하기 (폴더 열기 1)



그러면 폴더에 있는 파일을 신뢰하는지 여부를 묻는 CONFIRM창이 나타납니다.

우리는 당연히 신뢰버튼을 클릭!!

2. 패키지 관리자 설정 (build.gradle)

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.3.2'
    id 'io.spring.dependency-management' version '1.1.6'
}

group = 'kr.co.milkt'
version = '0.0.1-SNAPSHOT'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(21)
    }
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-jdbc'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.2'
    implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.5.0'
    implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.3'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'
    // runtimeOnly 'com.mysql:mysql-connector-j'
    // runtimeOnly 'com.microsoft.sqlserver:mssql-jdbc'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter-test:3.0.2'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
    testRuntimeOnly 'com.h2database:h2'

    // QueryDSL
    implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
    annotationProcessor 'com.querydsl:querydsl-apt:5.0.0:jakarta'
    annotationProcessor "jakarta.annotation:jakarta.annotation-api"
    annotationProcessor "jakarta.persistence:jakarta.persistence-api"
}

tasks.named('test') {
    useJUnitPlatform()
}

// plain.jar 파일 생성하지 않음
jar {
    enabled = false
}
```

build.gradle에 내용을 추가해주세요

Actuator

헬스 체크, 메트릭 수집 등을 지원하는 스프링 부트 기본 기능입니다.
~/health 로 확인 가능합니다.

Swagger 사용

Swagger는 API 명세를 작성해주는 좋은 서드파티입니다.
~/swagger-ui/index.html 로 확인 가능합니다.

QueryDSL 사용

SpringDataJpa의 JPQL이나, MyBatis는 문자열로 쿼리를 작성하기 때문에, 런타임 이전까
지 문제를 발견할 수 없습니다.
QueryDSL은 컴파일 레벨에서 Q클래스를 생성하고 소스단에서 쿼리를 작성하여, 문제를
빌드 이전에 발견 가능하도록 돕습니다.

2. 솔루션 설정 (application.properties)

Application.yaml을 다음과 같이 작성해주세요

```
server:
  port: 8080
spring:
  application:
    name: demo
  datasource:
    # # MSSQL
    # driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
    # url: jdbc:sqlserver://localhost:1433;databaseName=mytest
    # # MY-SQL
    # driver-class-name: com.mysql.cj.jdbc.Driver
    # url: jdbc:mysql://localhost:3306/mytest?serverTimezone=UTC&characterEncoding=UTF-8
    # # 계정
    # username: root
    # password: 1234!
    # H2
    url: jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
  jpa:
    # # MSSQL
    # database-platform: org.hibernate.dialect.SQLServer2012Dialect
    # # MY-SQL
    # database-platform: org.hibernate.dialect.MySQL8Dialect
    # DDL 자동 생성
    generate-ddl: true
    # DDL 자동 생성 하이버네이트 옵션 (create, create-drop, update, validate, none)
    hibernate:
      ddl-auto: update
  mybatis:
    # /src/main/resources/mapper/*.xml
    mapper-locations: classpath:mapper/*.xml
    configuration:
      map-underscore-to-camel-case: true
```

port

애플리케이션이 포트 될 포트 번호를 지정합니다.

Datasource 설정

데이터소스를 설정합니다.

- MSSQL과 MySQL, H2 설정에 대해 각각 작성했습니다.
- 계정 정보는 모두 공통적입니다.
- H2의 경우, 인메모리 디비로써,
 - 실제 인프라 없이도 스프링부트를 바로 실행할 수 있습니다.

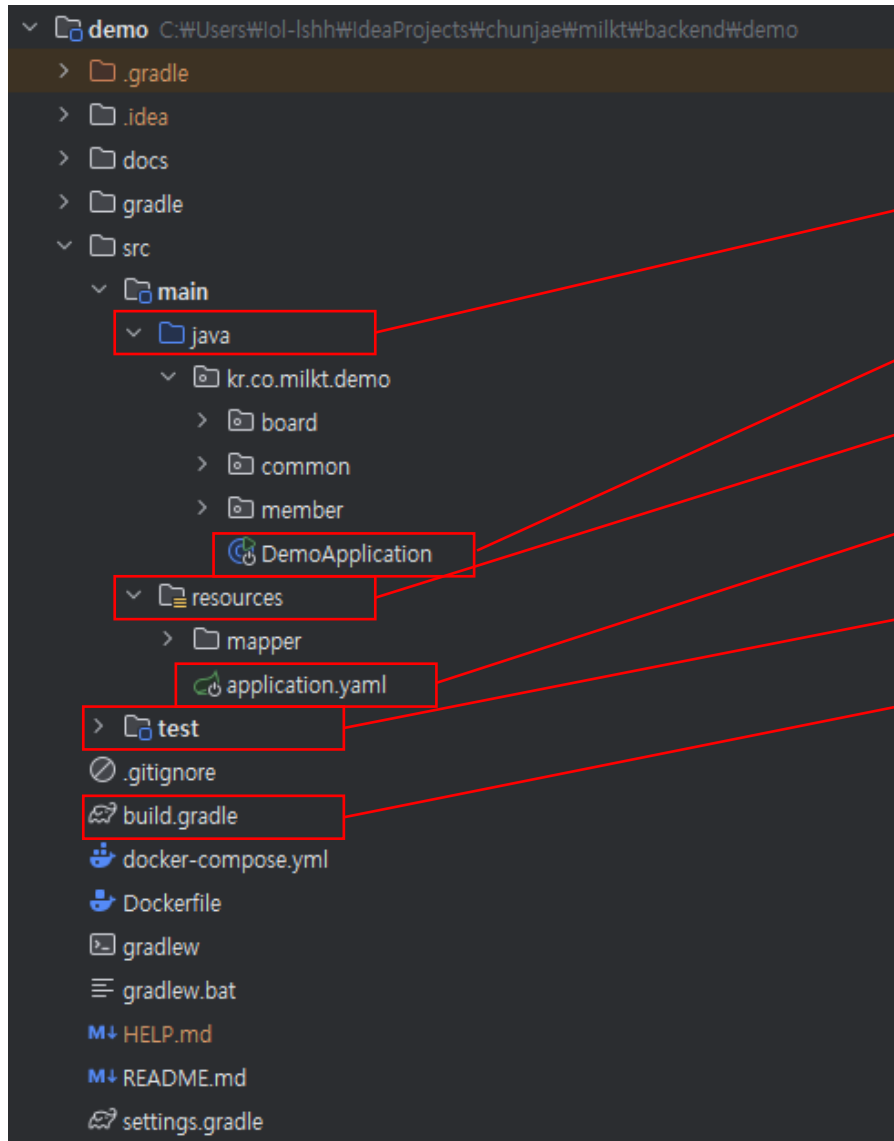
JPA 설정

- 데이터베이스마다 목표에 맞는 방안을 지정해줍니다.
- JPA의 구현체 hibernate는 방언에 맞게, 쿼리를 작성해줍니다.
- ddl 설정을 통해, ddl 지원 여부를 설정합니다.

MyBatis 설정

- MyBatis 인터페이스가 구현될 xml 파일 위치를 지정할 수 있습니다.
- 자동으로 언더바 카멜케이스로 매핑하는 설정을 할 수 있습니다.

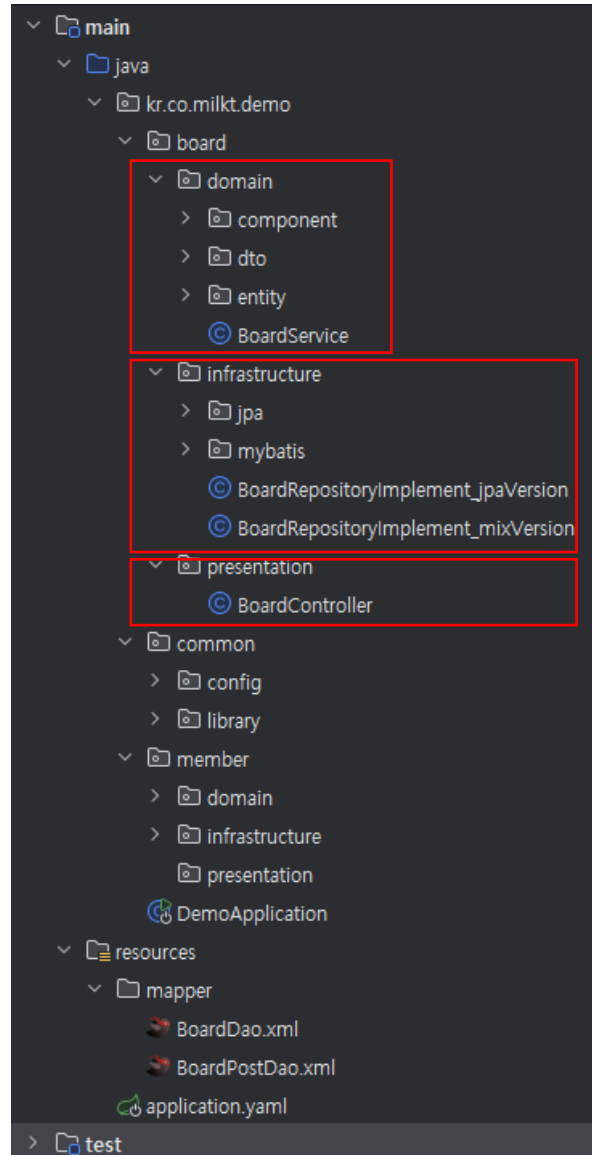
2. 솔루션의 기본구조 (폴더 구성)



탐색기에서 기본적인 폴더의 구성을 확인할 수 있습니다.

1. Java class가 저장되는 폴더 (패키지)
폴더 경로 자체가 package(C#의 namespace와 동일)로 구성
2. DemoApplication.java
SpringBootApplication을 실행하는 main 함수가 있는 클래스.
Launcher 개념 (C# Winform의 Program.cs와 동일한 역할)
3. resources
정적인 파일들을 보관
4. application.yaml
application.properties를 대신 작성해도 된다.
해당 파일들을 이용하여, 스프링 프로파일 구성이 가능하다.
글로벌 설정 파일(asp.net의 web.config와 동일한 역할)
5. 테스트 폴더
테스트를 한 곳에 작성하기 좋게 구성해준다.
6. build.gradle
설치된 패키지들을 관리하는 파일(C#의 Nuget Package와 동일함)

2. 솔루션의 기본구조 - Layered Architecture (MVC pattern)



src/main의 java.kr.co.milkt.demo 위치에 패키지를 추가

common : 공통 적용

└ **config** : 전역으로 적용되는 설정들

└ **library** : 공통 사용되는 라이브러리성 요소들

board : 게시판(board) 도메인에 관련된 요소들

└ **presentation** : Controller 클래스 또는 외부 인터페이스들
- 구) controller 패키지

└ **domain** : 애플리케이션 외부 인프라 구조와 연결되는 요소들
- 구) service 패키지

└ **infrastructure** : 애플리케이션 외부 인프라 구조와 연결되는 요소들

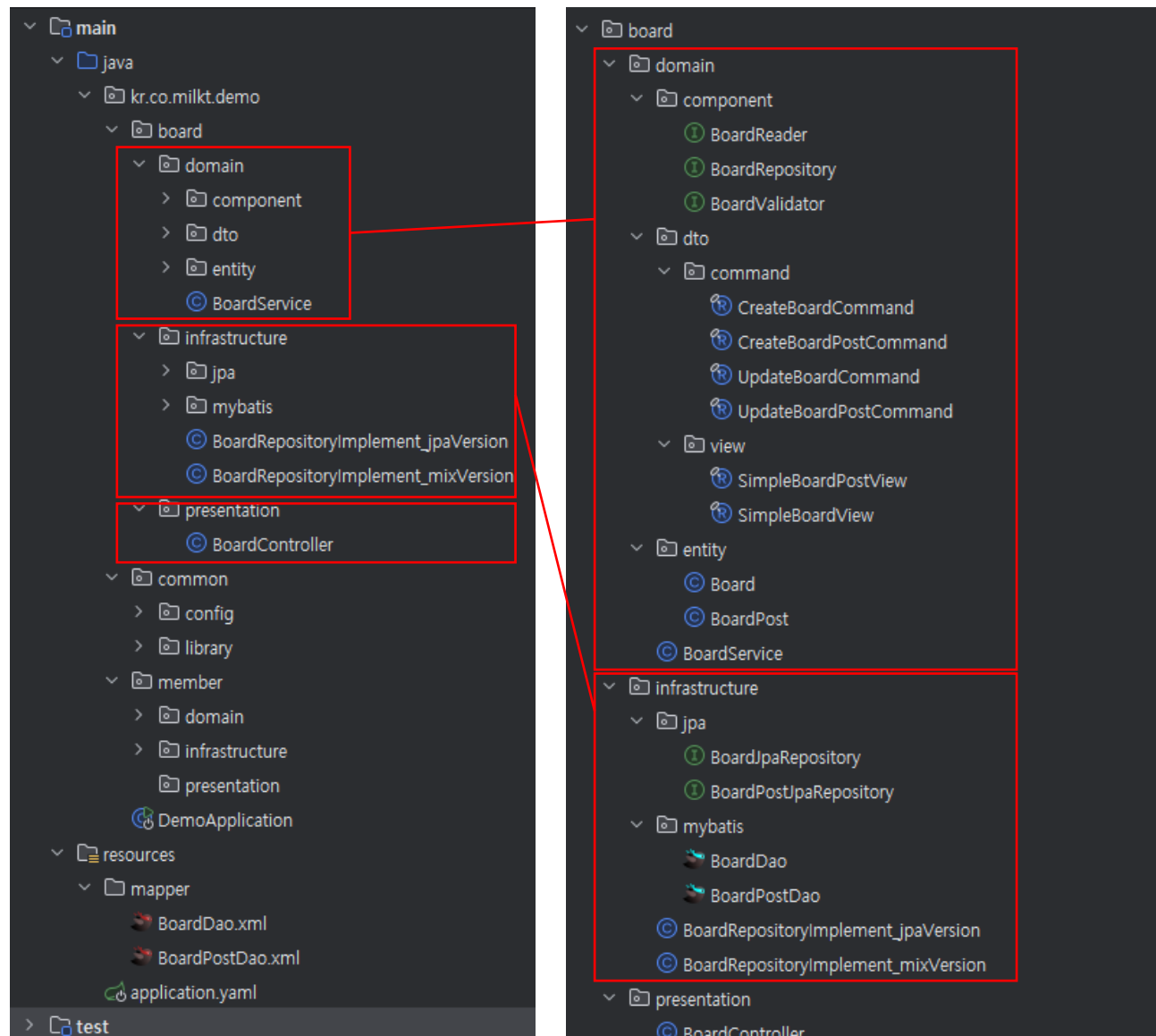
src/resource에 폴더 추가

mapper : mybatis XML를 저장

static/script : 자바스크립트 라이브러리(JS) 저장

templates : html 파일 저장

2. 솔루션의 기본구조 - Layered Architecture (MVC pattern)



src/main의 java.kr.co.milkt.demo 위치에 패키지를 추가

presentation : 컨트롤러 모음

domain : 비즈니스 처리

└ **component** : 비즈니스를 돕는 요소들

└ **BoardReader** : dto를 가져오는 읽기 전용 컴포넌트

└ **BoardRepository** : entity를 읽고 쓰는 jpa repository

└ **BoardValidator** : 비즈니스 검증 로직을 처리해주는 컴포넌트

└ **entity** : 도메인 엔티티

└ **dto** : 도메인 밖과 전달되는 객체

└ **view** : presentation으로 전달하는 데이터 객체 (Projection)

└ **command** : presentation에서 전달되는 명령 객체

└ **BoardService** : Board 도메인에 관한 서비스

infrastructure : 인프라 구조체

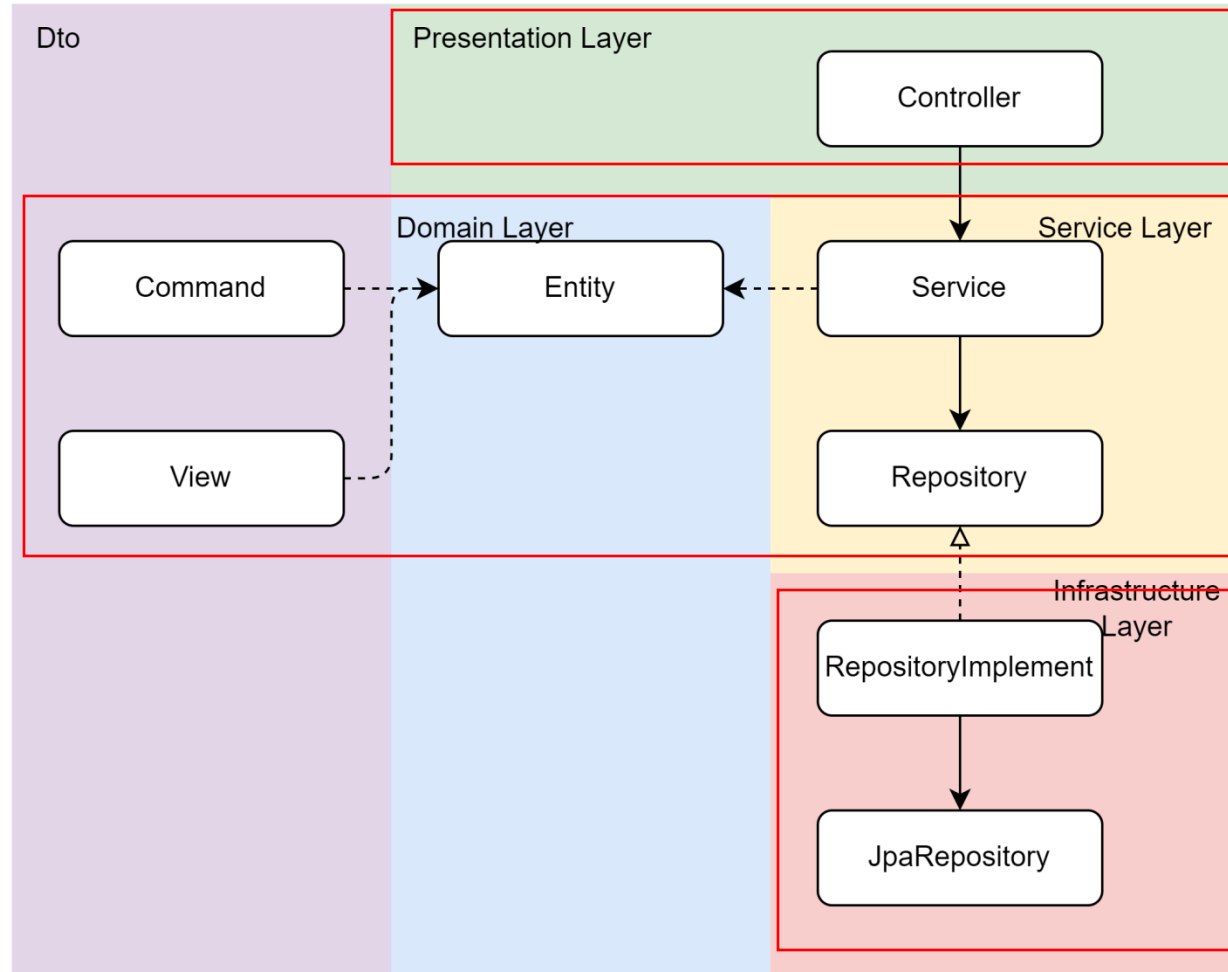
└ **BoardRepositoryImplement_jpaVersion** : Jpa와 QueryDsl 로 구현된 Repository

└ **BoardRepositoryImplement_mixVersion** : Jpa와 MyBatis 로 구현된 Repository

└ **jpa** : SpringDataJpa의 JpaRepository 인터페이스들을 모아둔 패키지

└ **mybatis** : MyBatis의 Mapper DAO 인터페이스들을 모아둔 패키지

2. 솔루션의 기본구조 - Layered Architecture (MVC pattern)



src/main의 java.kr.co.milkt.demo 위치에 패키지를 추가

common : 공통 적용

└ **config** : 전역으로 적용되는 설정들

└ **library** : 공통 사용되는 라이브러리성 요소들

board : 게시판(board) 도메인에 관련된 요소들

└ **presentation** : Controller 클래스 또는 외부 인터페이스들
- 구) controller 패키지

└ **domain** : 애플리케이션 외부 인프라 구조와 연결되는 요소들
- 구) service 패키지

└ **infrastructure** : 애플리케이션 외부 인프라 구조와 연결되는 요소들

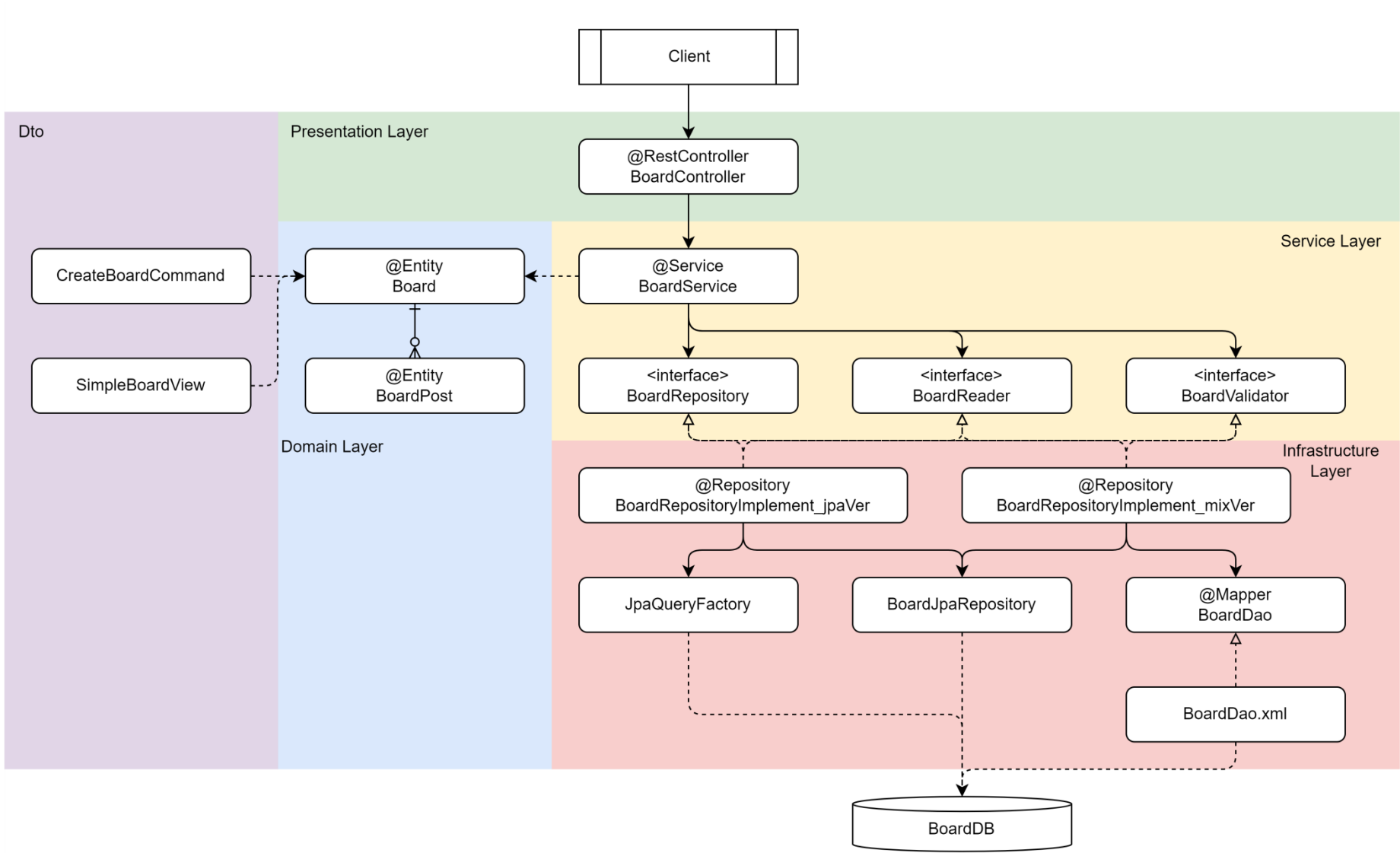
src/resource에 폴더 추가

mapper : mybatis XML를 저장

static/script : 자바스크립트 라이브러리(JS) 저장

templates : html 파일 저장

2. 솔루션의 기본구조 - Layered Architecture (MVC pattern) - 상세도



5. 소스코드 작성(Service)

```
@RequiredArgsConstructor 3 usages lshh
@Service
public class BoardService {
    private final BoardRepository boardRepository;
    private final BoardValidator boardValidator;
    private final BoardReader boardReader;
    private final ClockManager clockManager;

    @Transactional(readOnly = true) 3 usages lshh
    public Result findAllBoards() {...}

    @Transactional 5 usages lshh
    public Result createBoard(CreateBoardCommand command) {...}

    @Transactional 1 usage lshh
    public Result updateBoard(UpdateBoardCommand command) {...}

    @Transactional 1 usage lshh
    public Result deleteBoard(Long boardId) {...}

    public Result findAllBoardPosts(Long boardId) {...}

    @Transactional 2 usages lshh
    public Result createBoardPost(CreateBoardPostCommand command) {...}

    @Transactional 1 usage lshh
    public Result updateBoardPost(UpdateBoardPostCommand command) {...}

    @Transactional 1 usage lshh
    public Result deleteBoardPost(Long boardId, Long postId) {...}
}
```

/board/domain/entity 패키지에 작성

@Service :

스프링부트 애플리케이션의 시작 시에, 서비스 속성의 컴포넌트 빈을 생성해주고, 이를 스프링 컨테이너 안에 보관하게 한다.

@RequiredArgsConstructor :

필드는 private final로 작성한다. 생성자에서 주입받도록 한다.

(@Autowired는 리플렉션을 이용하여, 필드에 값을 강제로 주입하기 때문에, 스프링 재단에서 지양하라는 권고가 있는 방식이다.)

@RequiredArgsConstructor는 필수적인 필드들로 생성자를 만들어주는 롬복의 어노테이션이다.

@Transactional :

해당 방법은 어노테이션을 통해 트랜잭션을 지원하는 방법이다.

readOnly 속성은 해당 트랜잭션이 읽기만 하는 트랜잭션임을 명시적으로 알려주며, 이는 커스텀 설정을 통해 CQRS 달성을 돕기도 한다.

5. 소스코드 작성(Entity 생성)

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Getter
@Entity
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @Setter
    private LocalDateTime deleted;

    @OneToMany(mappedBy = "board", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<BoardPost> posts;

    public void addPost(CreateBoardPostCommand command) {...}

    public void update(UpdateBoardCommand command) { this.name = command.name(); }

    public Optional<BoardPost> findPostById(Long postId) {...}

    public void updatePost(UpdateBoardPostCommand command) {...}

    public void deletePost(Long postId, LocalDateTime now) {...}
}
```

/board/domain/entity 패키지에 작성

@Entity:

JPA(Jakarta.persistence API)는 @Entity 어노테이션을 통해, JPA Entity를 지원한다.

@Id :

해당 어노테이션은 엔티티의 식별자를 의미하는 JPA 어노테이션이다. JPA는 이 식별자를 통해, 엔티티가 동일한 엔티티인지 확인한다. (값의 동일함과 별개)

@GeneratedValue :

해당 어노테이션은 엔티티 아이디에 같이 쓰인다. 아이디 자동 생성 규칙을 더해줄 수 있다.

@OneToMany :

해당 어노테이션은 엔티티의 관계를 나타내는 JPA 어노테이션이다.

- mappedBy: 대응하는 엔티티가 어떤 속성명으로 대응되는지를 지정한다.
- fetch: 엔티티를 불러올때, 무조건 바로 불러올지(EAGER), 트랜잭션 안에서 필요할때만 불러올지(LAZY) 설정할 수 있다. N+1 문제를 방지한다.
- cascade: 연관 관계의 엔티티의 상태를 같이 저장할지 여부를 설정할 수 있다. 예컨데, ALL 타입은 Board 엔티티를 저장할 때, 연관된 BoardPost의 상태 변경이 있을 때, 같이 저장해준다.

@Builder, @NoArgsConstructor, @AllArgsConstructor :

모두 롬복 어노테이션이다.

@Builder 어노테이션은 객체를 builder 패턴으로 생성할 수 있게 해준다.

@Builder를 사용하기 위해서는 전체 필드를 받는 생성자, 필드를 받지 않는 생성자 두가지가 필요하다.

@NoArgsConstructor와 @AllArgsConstructor는 각각 기본 생성자, 모든 필드를 받는 생성자이다.

@Getter, @Setter :

롬복 어노테이션이다.

@Getter는 getter 메서드를 만들어주며, @Setter는 setter 메서드를 만들어준다.

@Setter는 취약점을 만들어낼 수 있으므로, 필요한 곳에만 사용한다.

@Getter, @Setter는 클래스와 메서드 모두 사용할 수 있다. 클래스에 사용할 경우, 해당 클래스의 모든 필드들의 getter setter를 만들어준다.

5. 소스코드 작성(RestController)

```
@RequiredArgsConstructor
@RequestMapping("/board")
@RestController
public class BoardController {
    private final BoardService boardService;

    @GetMapping("/all")
    public Response findAllBoards(){...}

    @PostMapping("/")
    public Response createBoard(CreateBoardCommand command){...}

    @PostMapping("/{boardId}")
    public Response updateBoard(UpdateBoardCommand command){...}

    @DeleteMapping("/{boardId}")
    public Response deleteBoard(Long boardId){...}

    @GetMapping("/{boardId}/post")
    public Response findAllBoardPosts(Long boardId){...}

    @PostMapping("/{boardId}/post")
    public Response createBoardPost(CreateBoardPostCommand command){...}

    @PostMapping("/{boardId}/post/{postId}")
    public Response updateBoardPost(UpdateBoardPostCommand command){...}

    @DeleteMapping("/{boardId}/post/{postId}")
    public Response deleteBoardPost(Long boardId, Long postId){...}
}
```

/board/presentation/ 패키지에 작성

@RestController:

스프링부트 애플리케이션의 시작 시에, RestController 속성의 컴포넌트 빈을 생성해주고, 이를 스프링 컨테이너 안에 보관하게 한다.

@RequestMapping :

해당 어노테이션은 모든 요청의 메소드에 대해 매핑한다.
값은 경로를 의미한다. /board

@GetMapping :

해당 어노테이션은 get 요청의 메소드에 대해 매핑한다.
값은 경로를 의미한다.

{ }는 path parameter를 의미한다. 이를 메서드의 파라미터에서 동일한 이름으로 받을 수 있다.

@PostMapping :

해당 어노테이션은 post 요청의 메소드에 대해 매핑한다.
값은 경로를 의미한다.

파라미터에서는 RequestBody를 해당 DTO로 받을 수 있다.

5. 소스코드 작성(Repository)

```
// entity 전용
public interface BoardRepository { 8개 사용 위치 2개 구현 ① lshh
    Board save(Board newBoard); 6개 사용 위치 2개 구현 ① lshh

    List<Board> findAll(); 1개 사용 위치 2개 구현 ① lshh
    Optional<Board> findByName(String name); 2개 사용 위치 2개 구현
    Optional<Board> findById(Long id); 5개 사용 위치 2개 구현 ① lshh
    List<BoardPost> findPostByTitle(String postName); 1개 사용 위

} ①
```

```
// projection dto 읽기 전용
public interface BoardReader { 6개 사용 위치 2개 구현 ① lshh

    List<SimpleBoardView> findAllView(); 1개 사용 위치 2개 구현 ①

    List<SimpleBoardPostView> findAllBoardPostView(); 1개 사용 위

} ①
```

/board/domain/component 패키지에 작성

Repository:

Repository 인터페이스는 일반적으로 JPA Entity를 넣고 꺼내는 역할을 담당시킨다.

Reader :

Reader 인터페이스는 일반적으로 DTO를 읽어오는 역할을 담당시킨다.

5. 소스코드 작성(Repository – JPA version)

```
// entity 전용
public interface BoardRepository { 8개 사용 위치 2개 구현 ㄱ lshh
    Board save(Board newBoard); 6개 사용 위치 2개 구현 ㄱ lshh

    List<Board> findAll(); 1개 사용 위치 2개 구현 ㄱ lshh
    Optional<Board> findByName(String name); 2개 사용 위치 2개 구현
    Optional<Board> findById(Long id); 5개 사용 위치 2개 구현 ㄱ lshh
    List<BoardPost> findPostByTitle(String postName); 1개 사용 위

} =
```

```
// projection dto 읽기 전용
public interface BoardReader { 6개 사용 위치 2개 구현 ㄱ lshh
```

```
@Repository ㄱ lshh
@RequiredArgsConstructor
public class BoardRepositoryImplement_jpaVersion implements BoardRepository, BoardReader, BoardValidator {
    private final BoardJpaRepository boardJpaRepository;
    private final BoardPostJpaRepository boardPostJpaRepository;
    private final JPAQueryFactory queryFactory;

    @Override 6개 사용 위치 ㄱ lshh
    public Board save(Board newBoard) {
        return boardJpaRepository.save(newBoard);
    }

    @Override 1개 사용 위치 ㄱ lshh
    public List<Board> findAll() { return boardJpaRepository.findAll(); }

    @Override 2개 사용 위치 ㄱ lshh
    public Optional<Board> findByName(String name) {
        return boardJpaRepository.findByName(name);
    }
}
```

/board/infrastructure/ 패키지에 작성
- Persistence에 접근하는 방법을 구현하는 구현체이다.

@Repository:

스프링부트 애플리케이션의 시작 시에, 레포지토리 속성의 컴포넌트 빈을 생성해주고, 이를 스프링 컨테이너 안에 보관하게 한다.

BoardRepositoryImplement_jpaVersion은 BoardRepository, BoardReader, BoardValidator를 모두 구현하는 JPA 방식만을 사용하는 구현체이다.

- BoardRepository는 BoardJpaRepository로 동작된다.
- BoardReader는 JPAQueryFactory라는 QueryDSL에서 지원하는 인스턴스로 동작된다.

(다음 슬라이드에서 각각의 동작 구현을 설명한다.)

5. 소스코드 작성(Repository – JPA version : JpaRepository)

```
public interface BoardJpaRepository extends JpaRepository<Board, Long> { 4개 사용 위치
    @Query("SELECT b FROM Board b WHERE b.deleted IS NULL") 1개 사용 위치
    List<Board> findAll();

    @Query("SELECT b FROM Board b WHERE b.id = :id AND b.deleted IS NULL") 1개 사용 위치
    Optional<Board> findById(Long id);

    @Query("SELECT b FROM Board b WHERE b.name = :name AND b.deleted IS NULL") 2개 사용 위치
    Optional<Board> findByName(String name);

    boolean existsByName(String name); 2개 사용 위치 1개 사용 위치

    boolean existsById(Long id); 4개 사용 위치 1개 사용 위치
}
```

/board/infrastructure/jpa 패키지에 작성

JpaRepository<Entity, Class>:

JpaRepository는 SpringDataJPA에서 지원하는 인터페이스이다.

이를 상속함으로써, 해당 레포지토리는 JPA 기능이 자동으로 구현된다.

JpaRepository는 몇 가지 메서드를 기본적으로 지원한다.

- save: 저장 메서드로 update insert를 지원한다.
- find: 조회 메서드로 By를 통해 컬럼을 추가할 수 있다.
 - Null Exception을 피하기 위해, Optional로 반환하도록 하자.
- exists: 조회 메서드이나, 있는지 여부를 확인해준다.

@Query :

해당 어노테이션은 JPQL 기능으로 복잡한 쿼리 등을 메서드에 오버라이딩 할 수 있다.

두가지 방식이 가능하다.

- JPA식 쿼리: JPA 구현체(Hibernate)가 지원하여, 방언에 따라 자동으로 쿼리를 다르게 작성해준다.
- Native 쿼리: 날 쿼리를 작성할 수 있다. 지원이 안되므로, 디비에 맞춰 작성해야 한다.

5. 소스코드 작성(Repository – JPA version : JpaRepository)

```
public interface BoardJpaRepository extends JpaRepository<Board, Long> { 4개 사용 위치
    @Query("SELECT b FROM Board b WHERE b.deleted IS NULL") 1 Lshh
    List<Board> findAll();

    @Query("SELECT b FROM Board b WHERE b.id = :id AND b.deleted IS NULL") 1 Lshh
    Optional<Board> findById(Long id);
```

```
@Repository 1 Lshh
@RequiredArgsConstructor
public class BoardRepositoryImplement_jpaVersion implements BoardRepository, BoardReader, BoardValidator {
    private final BoardJpaRepository boardJpaRepository;
    private final BoardPostJpaRepository boardPostJpaRepository;
    private final JPAQueryFactory queryFactory;

    @Override 6개 사용 위치 1 Lshh
    public Board save(Board newBoard) {
        return boardJpaRepository.save(newBoard);
    }

    @Override 1개 사용 위치 1 Lshh
    public List<Board> findAll() { return boardJpaRepository.findAll(); }

    @Override 2개 사용 위치 1 Lshh
    public Optional<Board> findByName(String name) {
        return boardJpaRepository.findByName(name);
    }
}
```

/board/infrastructure/ 패키지에 작성

@Repository:

스프링부트 애플리케이션의 시작 시에, 레포지토리 속성의 컴포넌트 빈을 생성해주고, 이를 스프링 컨테이너 안에 보관하게 한다.

@Query :

해당 어노테이션은 JPQL 기능으로 복잡한 쿼리 등을 메서드에 오버라이딩 할 수 있다.

두가지 방식이 가능하다.

- JPA식 쿼리: JPA 구현체(Hibernate)가 지원하여, 방언에 따라 자동으로 쿼리를 다르게 작성해준다.
- Native 쿼리: 날 쿼리를 작성할 수 있다. 지원이 안되므로, 디비에 맞춰 작성해야 한다.

5. 소스코드 작성(Repository – JPA version : QueryDsl)

```
@Configuration  @lshh
public class QueryDslConfig {

    @PersistenceContext 1개 사용 위치
    private EntityManager entityManager;

    @Bean  @lshh
    public JPAQueryFactory jpaQueryFactory() {
        return new JPAQueryFactory(entityManager);
    }
}
```

/common/config 패키지에 작성

@Configuration: 스프링부트 애플리케이션의 시작 시에, 설정 속성의 컴포넌트 빈을 생성해주고, 내부 설정 로직을 실행시켜준다.

@PersistenceContext: EntityManager를 Bean으로 주입받기 위한 어노테이션

@Bean: 스프링부트 애플리케이션의 시작 시에, 스프링 컨테이너에 해당 클래스의 빈 객체를 생성하여 넣는다.

5. 소스코드 작성(Repository – JPA version : QueryDsl)

```
@Configuration
public class QueryDslConfig {

    @PersistenceContext
    private EntityManager entityManager;

    @Bean
    public JPAQueryFactory jpaQueryFactory() {
        return new JPAQueryFactory(entityManager);
    }
}
```

이를 Repository 구현체에서 Dependency Injection받아 사용

Projections: QueryDsl에서 제공하는 기능으로, 제공하는 클래스로 데이터를 반환 받을 수 있다.

```
@Repository
@RequiredArgsConstructor
public class BoardRepositoryImplement_jpaVersion implements BoardRepository, BoardReader, BoardValidator {

    private final BoardJpaRepository boardJpaRepository;
    private final BoardPostJpaRepository boardPostJpaRepository;
    private final JPAQueryFactory queryFactory;

    @Override
    public List<SimpleBoardView> findAllView() {
        return queryFactory
            .select(Projections.constructor(SimpleBoardView.class,
                board.id, board.name))
            .from(board)
            .fetch();
    }

    @Override
    public List<SimpleBoardPostView> findAllBoardPostView() {
        return queryFactory
            .select(Projections.constructor(SimpleBoardPostView.class,
                boardPost.id, boardPost.title, boardPost.content, boardPost.board.id))
            .from(boardPost)
            .fetch();
    }
}
```

5. 소스코드 작성(Repository – MyBatis version)

```
// entity 전용
public interface BoardRepository { 8개 사용 위치 2개 구현 ㄱ lshh
    Board save(Board newBoard); 6개 사용 위치 2개 구현 ㄱ lshh

    List<Board> findAll(); 1개 사용 위치 2개 구현 ㄱ lshh
    Optional<Board> findByName(String name); 2개 사용 위치 2개 구현
    Optional<Board> findById(Long id); 5개 사용 위치 2개 구현 ㄱ lshh
    List<BoardPost> findPostByTitle(String postName); 1개 사용 위
```

/board/infrastructure/ 패키지에 작성

- Persistence에 접근하는 방법을 구현하는 구현체
- DTO를 가져오던 QueryDsl의 QueryFactory가 BoardDao로 바뀌었다.

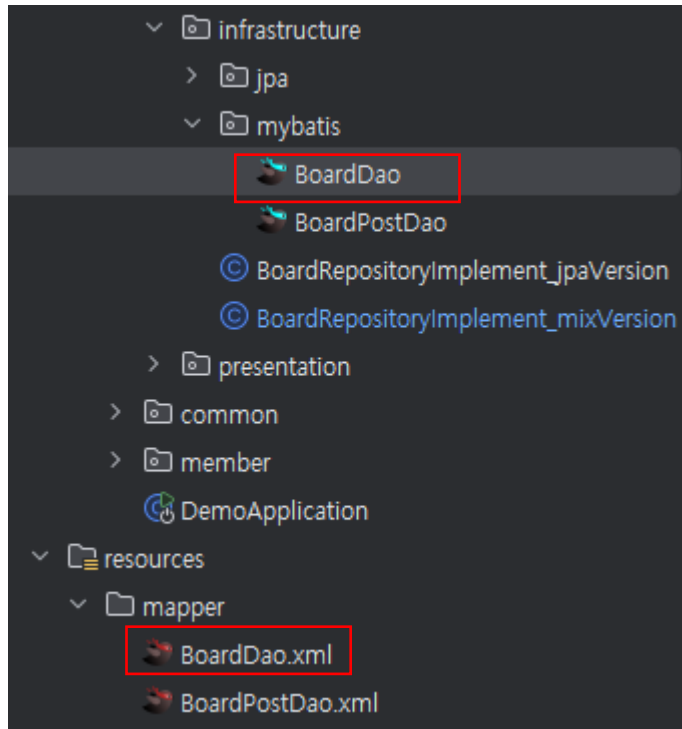
(다음 슬라이드에서 각각의 동작 구현을 설명한다.)

```
@Repository ㄱ lshh
@RequiredArgsConstructor
public class BoardRepositoryImplement_mixVersion implements BoardRepository, BoardReader, BoardValidator {
    private final BoardJpaRepository boardJpaRepository;
    private final BoardPostJpaRepository boardPostJpaRepository;
    private final BoardDao boardDao;
    private final BoardPostDao boardPostDao;

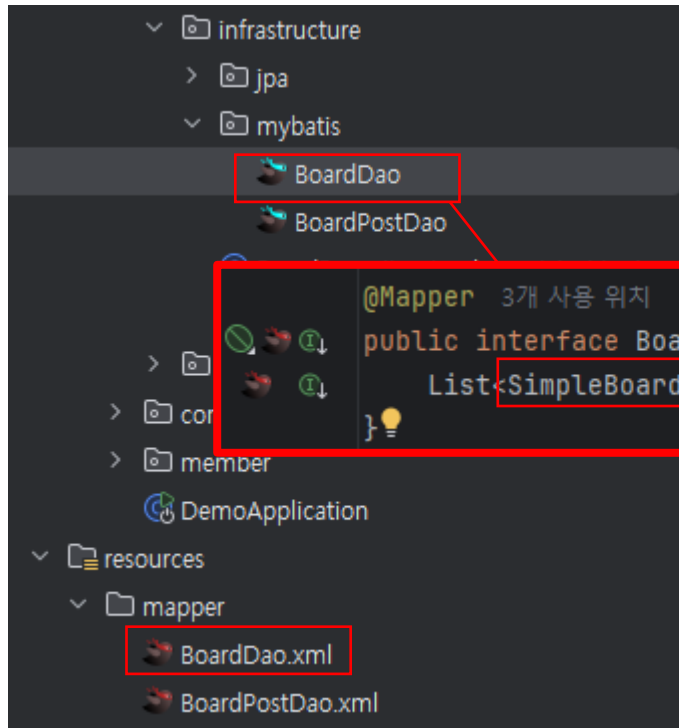
    @Override 1개 사용 위치 ㄱ lshh
    public List<SimpleBoardView> findAllView() {
        return boardDao.findAllBoards();
    }

    @Override 1개 사용 위치 ㄱ lshh
    public List<SimpleBoardPostView> findAllBoardPostView() {
        return boardPostDao.findAllBoardPosts();
    }
}
```


5. 소스코드 작성(Repository – MyBatis version)



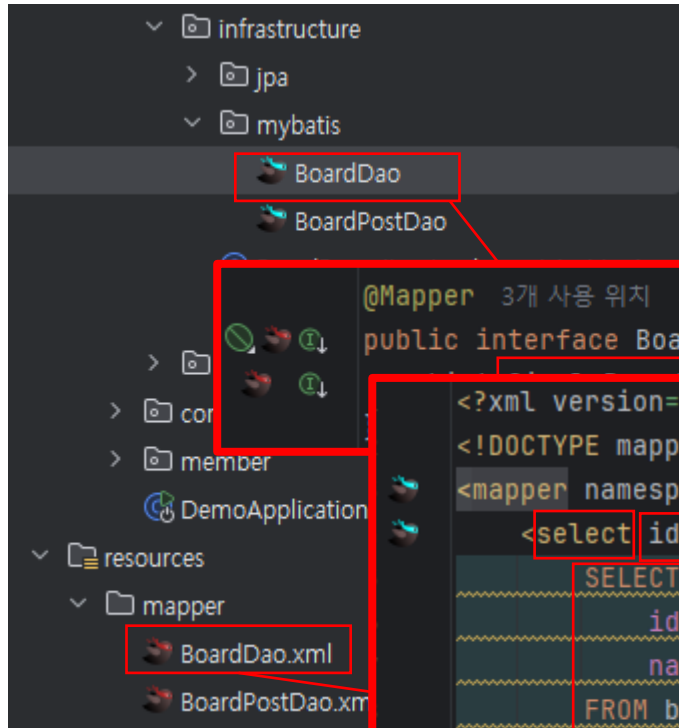
5. 소스코드 작성(Repository – MyBatis version: BoardDao.java)



/board/infrastructure/mybatis 패키지에 작성

Mapper: ibatis 제공 어노테이션. MyBatis에 의해, 매핑되는 xml로 구현된다.

5. 소스코드 작성(Repository – MyBatis version: BoardDao.java)



resources의 mapper 디렉토리 안에 작성

select: select 쿼리가 보내질 것을 암시

id: 메서드명과 일치

resultType: 레코드가 매핑되어 반환될 객체의 클래스

parameterType: 전달할 파라미터의 클래스. Java의 Primitive Type은 따로 필요 없습니다.

```
@Mapper 3개 사용 위치 Ishh
public interface BoardDao {

    <?xml version="1.0" encoding="UTF-8" ?>
    <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <mapper namespace="kr.co.milkt.demo.board.infrastructure.mybatis.BoardDao">
        <select id="findAllBoards" resultType="kr.co.milkt.demo.board.domain.dto.view.SimpleBoardView">
            SELECT
            ~~~~~
            id,
            ~~~~~
            name
            ~~~~~
            FROM board
            ~~~~~
        </select>

        <select id="findAllBoardPosts" resultType="kr.co.milkt.demo.board.domain.dto.view.SimpleBoardPostView">
            SELECT
            ~~~~~
            id,
            ~~~~~
            title,
            ~~~~~
            content,
            ~~~~~
            boardId
            ~~~~~
            FROM board_post
            ~~~~~
            WHERE board_id = #{boardId}
            ~~~~~
        </select>
    </mapper>
```

https://github.com/lol-lshh/board4guide_be 에서 모든 코드를 받아보실 수 있습니다.

- `git clone https://github.com/lol-lshh/board4guide_be.git`