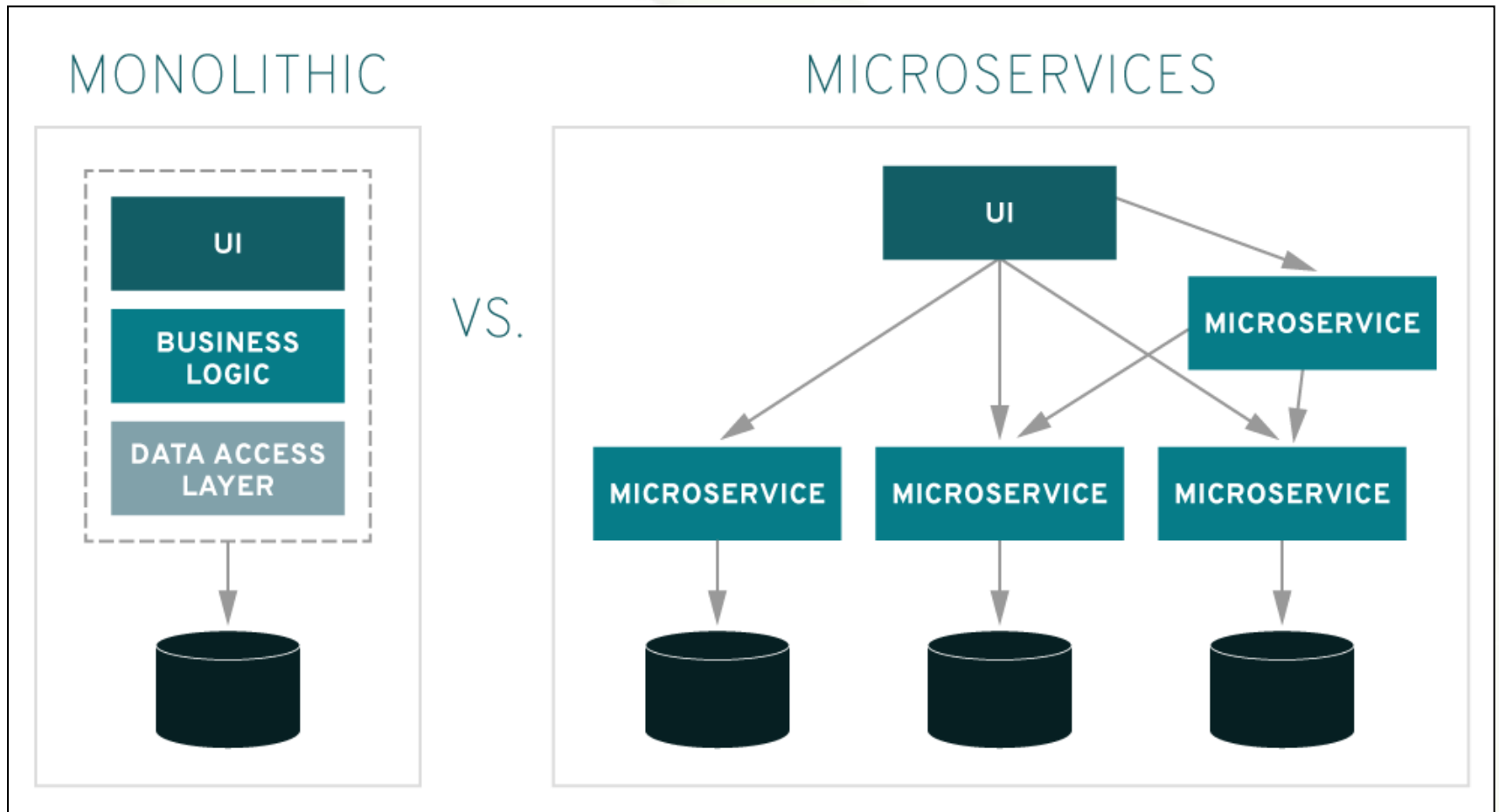


The background features a series of flowing, wavy green lines that create a sense of movement. A solid dark green horizontal bar is positioned at the very bottom of the frame. The text "Spring Boot" is centered in the upper half of the image.

Spring Boot

Micro Service

- 독립적으로 서비스를 개발하고 배포할 수 있는 아키텍처
- 각 서비스는 독립적인 프로세스에서 실행되고 이를 통해 경량 애플리케이션 모델 구현



Micro Service 장점

- 출시 주기 단축
 - 개발 주기 단축으로 보다 민첩한 배포 및 업데이트 지원
- 높은 확장성
 - 특정 서비스에 대한 수요가 증가할 경우 여러 서버 및 인프라에 쉽게 배포
- 뛰어난 복구 능력
 - 제대로 구현된 독립적인 서비스는 서로에게 영향을 주지 않기 때문에 한 부분에서 장애가 발생하더라도 전체 애플리케이션이 다운되지 않습니다.

Micro Service 장점

- 손쉬운 배포
 - 모듈화 되고 규모가 작아졌기 때문에 배포에 대한 우려 사항 감소
- 편리한 액세스 → 큰 애플리케이션을 작은 조각으로 분할했기 때문에 개발자들이 각 조각을 파악하고 개선하기가 용이해짐
- 개방성 향상
 - 다중 언어 지원 API 사용 → 필요한 기능에 맞는 최적의 언어와 기술 선택 가능

Spring Boot

- 마이크로 서비스 구축을 위한 자바 기반 오픈 소스 프레임워크
 - 즉시 실행 가능한 제품 수준의 독립 실행형 자바 애플리케이션 개발 플랫폼
 - 전체 스프링 설정을 구성하지 않고 최소한의 설정으로 시작 가능
- 장점
 - 상대적으로 쉬운 스프링 애플리케이션 개발
 - 생산성 향상
 - 개발 시간 단축
- 목표
 - 복잡한 XML 설정 제거
 - 제품 수준의 스프링 애플리케이션을 쉽게 개발
 - 개발 시간을 단축 및 독립적인 애플리케이션 실행
 - 쉬운 애플리케이션 개발의 시작점 제공

주요 구성

▪ @EnableAutoConfiguration

- 프로젝트에 추가한 의존성을 기반으로 자동으로 설정 완료

▪ @SpringBootApplication

- 스프링 부트 애플리케이션의 진입점
- 자동으로 패키지를 검색해서 다양한 @Annotation 적용

```
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

@SpringBootApplication

- 대상 클래스는 반드시 main메서드 정의
- @EnableAutoConfiguration, @ComponentScan, @SpringBootConfiguration Annotation 자동 등록

```
public class DemoApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(DemoApplication.class, args);  
    }  
}
```

주요 구성

■ Spring Boot Starters

- spring-boot-start-* 형식의 의존성 세트
- 복잡한 의존성 관리를 일련의 의존성 세트를 통해 자동화 → 개발자 편의성 강화

■ 의존성 세트 사례 → Spring Security

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

■ 의존성 세트 사례 → Spring Web

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```

Spring Initializer

- 웹 페이지에서 프로젝트 구성 → <http://start.spring.io/>

The screenshot shows the Spring Initializr web application interface. The browser address bar displays <https://start.spring.io>. The page features a sidebar with a hamburger menu icon and the Spring Initializr logo. The main content area is divided into several sections for project configuration:

- Project:** Includes radio buttons for **Maven Project** (selected) and **Gradle Project**.
- Language:** Includes radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for versions 2.3.1 (SNAPSHOT), **2.3.0** (selected), 2.2.8 (SNAPSHOT), 2.2.7, 2.1.15 (SNAPSHOT), and 2.1.14.
- Project Metadata:** A form with input fields for **Group** (com.example), **Artifact** (demo), **Name** (demo), **Description** (Demo project for Spring Boot), and **Package name** (com.example.demo).
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Java:** Includes radio buttons for versions 14, 11, and **8** (selected).

On the right side, there is a **Dependencies** section with a button **ADD DEPENDENCIES... CTRL + B**. Below this, several dependencies are listed with category tags:

- Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- 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_INITIALIZER

■ STS에서 프로젝트 구성

The screenshot illustrates the process of creating a new Spring Starter Project in Spring Tool Suite 4. The main menu bar is visible at the top, with the 'New' option highlighted. A red box highlights the 'Spring Starter Project' option in the 'New' submenu. An orange arrow points from this option to the 'New Spring Starter Project' dialog box.

The 'New Spring Starter Project' dialog box contains the following fields and options:

- Service URL: `https://start.spring.io`
- Name: `spring-boot-demo`
- ☒ Use default location
- Location: `D:\workspace\spring-fx-one\spring-boot-demo` (with a 'Browse' button)
- Type: `Maven` (dropdown), Packaging: `Jar` (dropdown)
- Java Version: `8` (dropdown), Language: `Java` (dropdown)
- Group: `com.example`
- Artifact: `demo`
- Version: `0.0.1-SNAPSHOT`
- Description: `Demo Project for Spring Boot`
- Package: `com.demo`
- Working sets: ☐ Add project to working sets (with 'New...' and 'Select...' buttons)

An orange arrow points from the 'Next >' button in the 'New Spring Starter Project' dialog to the 'New Spring Starter Project Dependencies' dialog box.

The 'New Spring Starter Project Dependencies' dialog box contains the following fields and options:

- Spring Boot Version: `2.3.0` (dropdown)
- Frequently Used:
 - ☒ Lombok
 - ☐ MyBatis Framework
 - ☐ MySQL Driver
 - ☒ Spring Boot DevTools
 - ☒ Spring Web
- Available: `Type to search dependencies` (text input)
- Selected:
 - ☒ Spring Boot DevTools
 - ☒ Lombok
 - ☒ Spring Web
- Available list (scrollable):
 - Alibaba
 - Amazon Web Services
 - Developer Tools
 - Google Cloud Platform
 - I/O
 - Messaging
 - Microsoft Azure
 - NoSQL
 - Observability
 - Ops
 - Pivotal Cloud Foundry
- Buttons: `Make Default`, `Clear Selection`

Controller 구현

▪ HomeController 구현

```
package com.example.demo.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {

    @GetMapping(path = { "/" })
    public String home() {
        return "Hello, Spring Boot Web Service !!!!!";
    }
}
```

프로젝트 빌드 및 애플리케이션 시작

■ 빌드

```
관리자: 명령 프롬프트
D:\workspace\spring-fx-one\spring-boot-demo>mvn clean package
[INFO] Scanning for projects...
[INFO] -----< com.example:spring-boot-demo >-----
[INFO] Building spring-boot-demo 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
```

■ 실행

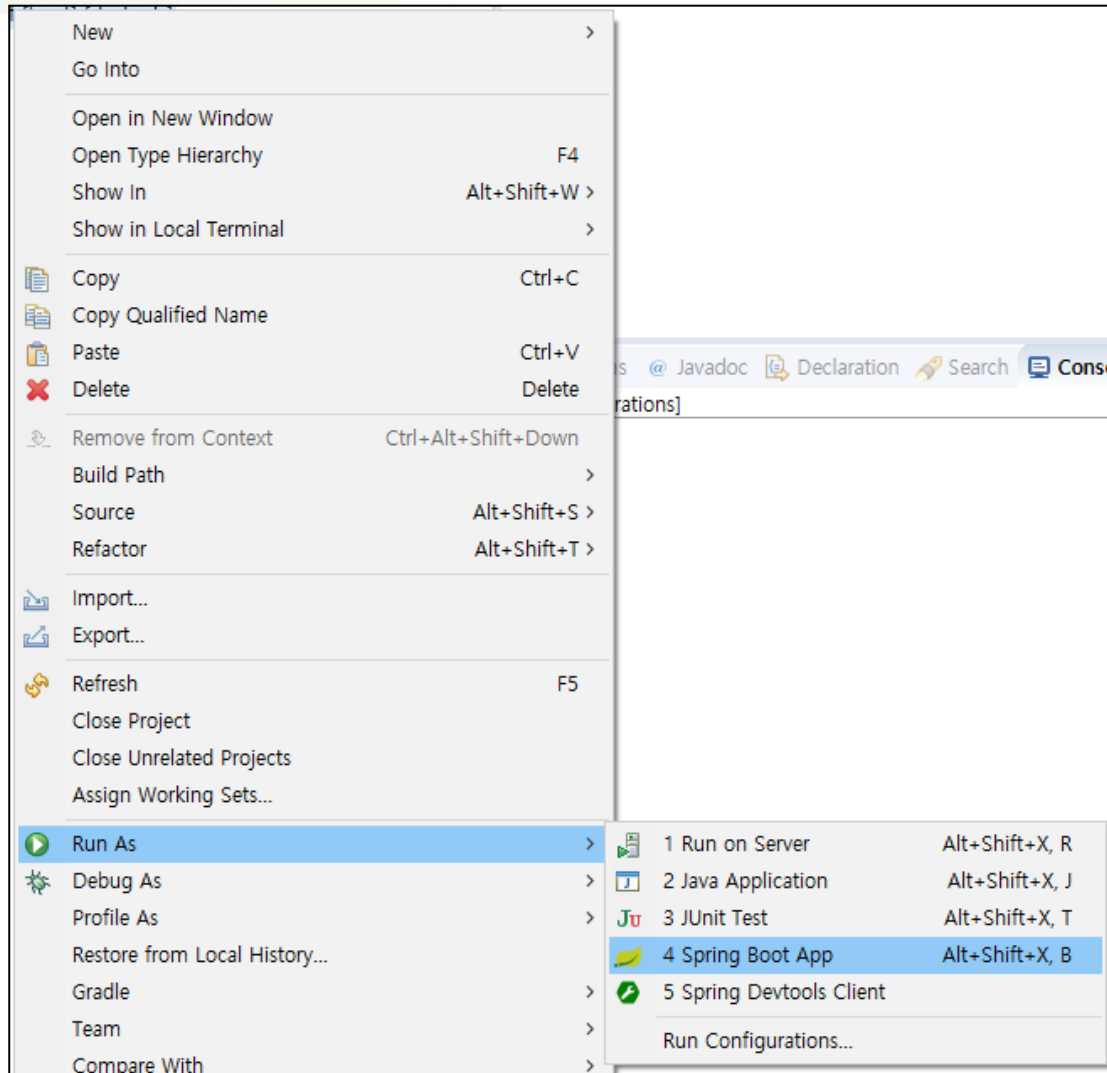
```
관리자: 명령 프롬프트 - java -jar spring-boot-demo-0.0.1-SNAPSHOT.jar
D:\workspace\spring-fx-one\spring-boot-demo>cd target
D:\workspace\spring-fx-one\spring-boot-demo\target>java -jar spring-boot-demo-0.0.1-SNAPSHOT.jar

  ____ _
 / ___ \| | | |
/ /___ \| |_| |
\____ \|_____|_|
   ____ _
  / ___ \| | | |
 / /___ \| |_| |
\____ \|_____|_|

:: Spring Boot ::                (v2.3.0.RELEASE)
```

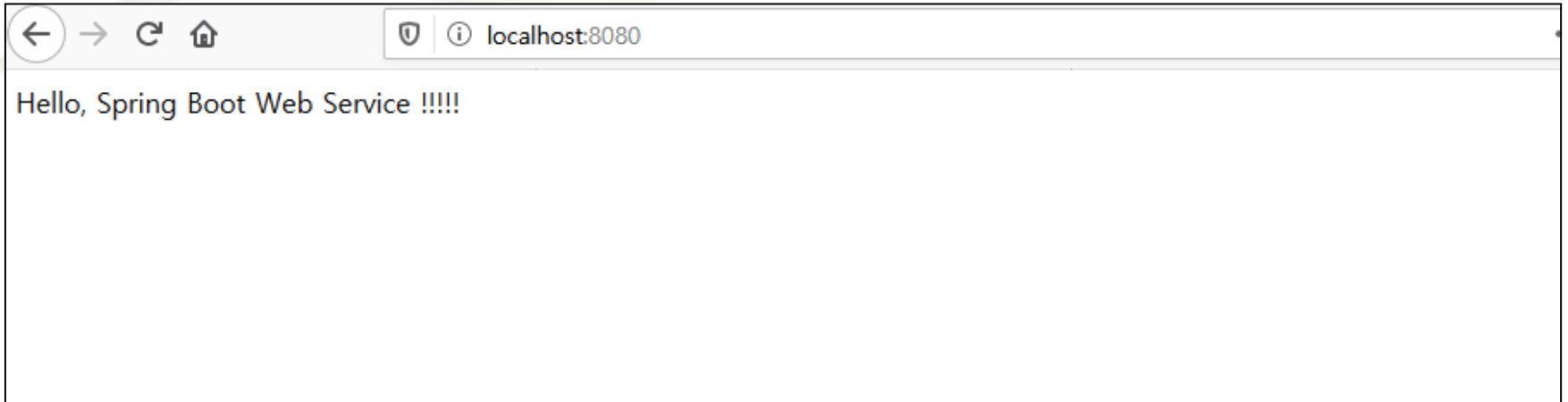
프로젝트 빌드 및 애플리케이션 시작

■ STS에서 시작



요청 실행

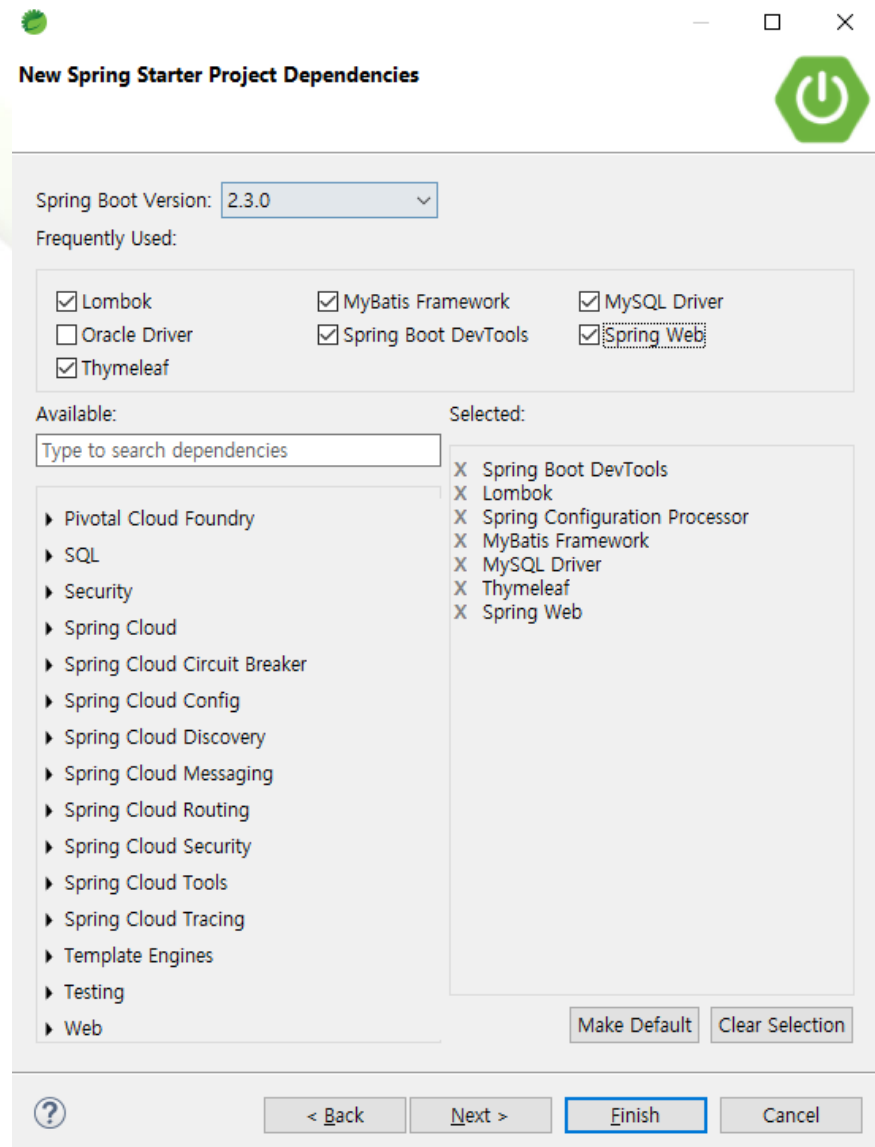
- 브라우저에서 웹 애플리케이션 사용



게시판 웹 프로젝트 기본 구성

■ 의존성 선택

- DevTools,
- Lombok,
- Spring Configuration Processor
- MySQL Driver,
- MyBatis Framework
- Thymeleaf
- Web



게시판 웹 프로젝트 기본 구성

■ 데이터 소스 설정

- src/main/resources/application.properties

```
application.properties
1 server.port=8081
2
3 spring.datasource.hikari.driver-class-name=com.mysql.cj.jdbc.Driver
4 spring.datasource.hikari.jdbc-url=jdbc:mysql://localhost:3306/boot_board2?serverTimezone=UTC
5 spring.datasource.hikari.username=devuser
6 spring.datasource.hikari.password=mariadb
7 spring.datasource.hikari.connection-test-query=SELECT 1
```

- DatabaseConfiguration 클래스 구현

```
@Configuration
@PropertySource("classpath:/application.properties")
public class DatabaseConfig {

    @Bean
    @ConfigurationProperties(prefix="spring.datasource.hikari")
    public HikariConfig hikariConfig() {
        HikariConfig config = new HikariConfig();

        return config;
    }

    @Bean
    public DataSource dataSource() {
        DataSource dataSource = new HikariDataSource(hikariConfig());
        return dataSource;
    }
}
```

게시판 웹 프로젝트 기본 구성

■ MyBatis 연동

- application.properties

```
mybatis.configuration.map-underscore-to-camel-case=true
```

- DatabaseConfiguration.java

```
@Bean
@ConfigurationProperties(prefix="mybatis.configuration")
public org.apache.ibatis.session.Configuration mybatisConfig() {
    return new org.apache.ibatis.session.Configuration();
}

@Bean
public SqlSessionFactory sqlSessionFactory() throws Exception {

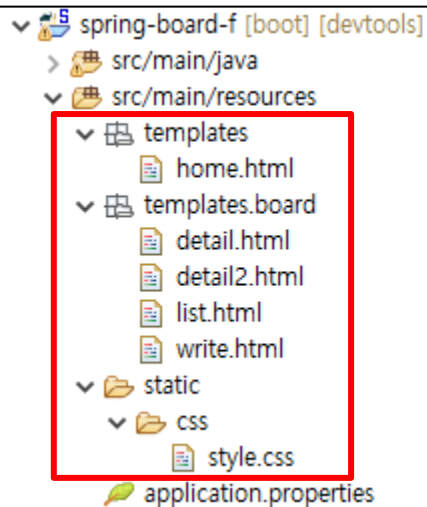
    SqlSessionFactoryBean factoryBean = new SqlSessionFactoryBean();
    factoryBean.setConfiguration(mybatisConfig());
    factoryBean.setDataSource(dataSource());
    return factoryBean.getObject();
}
```


게시판 웹 프로젝트 기본 구성

▪ Thymeleaf 뷰 템플릿 사용

```
<dependency>  
  <groupId>org.thymeleaf</groupId>  
  <artifactId>thymeleaf</artifactId>  
  <version>3.0.11.RELEASE</version>  
</dependency>
```

pom.xml



```
spring-board-f [boot] [devtools]  
├── src/main/java  
└── src/main/resources  
    ├── templates  
    │   ├── home.html  
    │   └── templates.board  
    ├── static  
    │   └── css  
    │       └── style.css  
    └── application.properties
```

resource files

게시판 웹 프로젝트 기본 구성

■ JSP 뷰 템플릿 사용

```
<dependency>
  <groupId>javax.servlet.jsp.jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jasper</artifactId>
  <version>9.0.35</version>
</dependency>
```

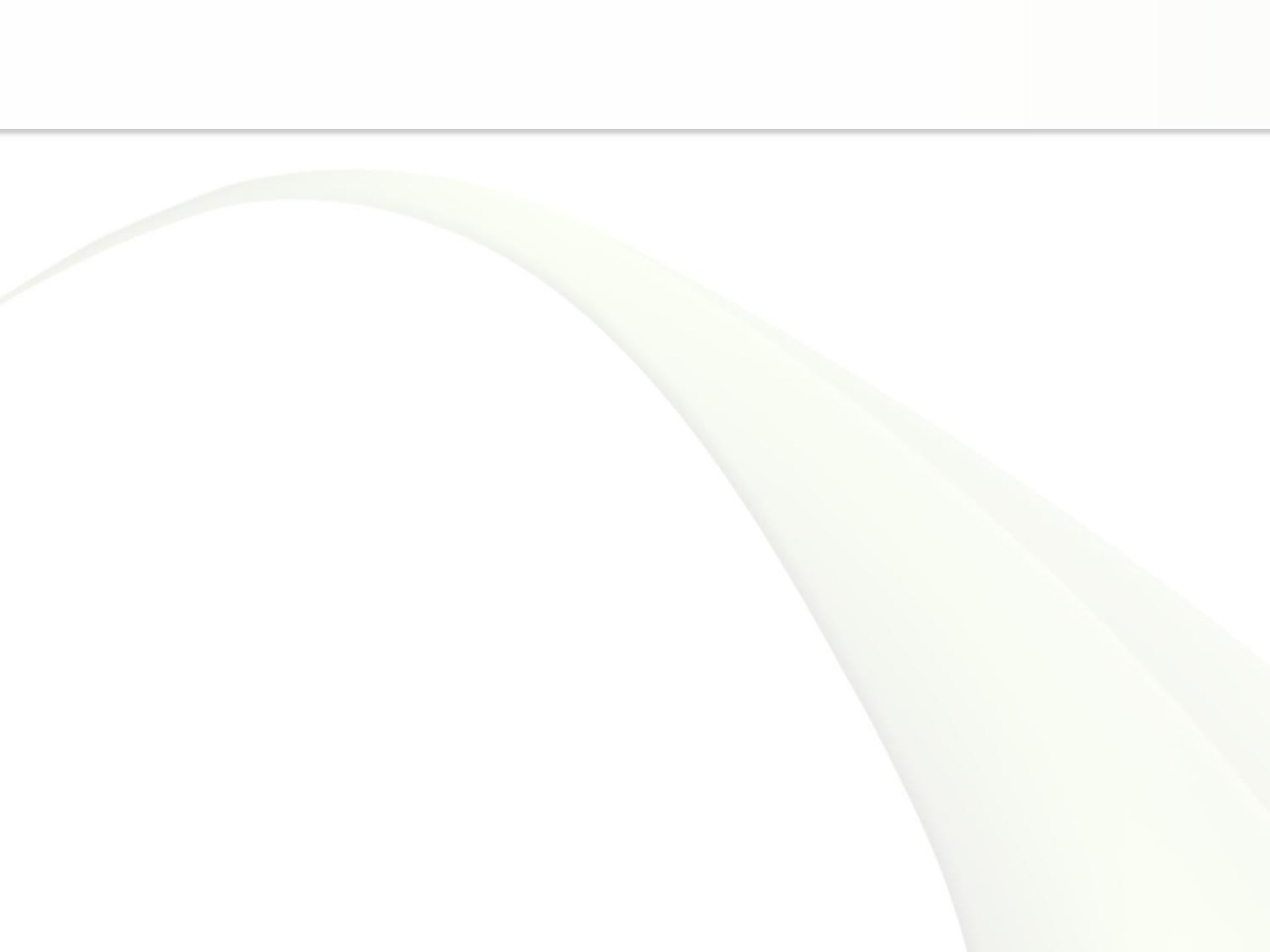
pom.xml

```
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

application.properties

```
src
├── main
│   ├── webapp
│   │   ├── resources
│   │   │   ├── css
│   │   │   │   └── style.css
│   │   └── WEB-INF
│   │       ├── views
│   │       │   └── board
│   │       │       ├── detail.jsp
│   │       │       ├── list.jsp
│   │       │       ├── write.jsp
│   │       │       └── home.jsp
```

resource files



REST(Representational State Transfer)

- 월드 와이드 웹과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식
- 자원을 정의하고 자원에 대한 주소를 지정하는 방법 전반을 의미하는 네트워크 아키텍처 원리의 모음
- 간단한 의미로 웹 상의 자료를 HTTP위에서 별도의 전송 계층 없이 전송하기 위한 아주 간단한 인터페이스

REST(Representational State Transfer)

■ 리소스

- 서비스를 제공하는 시스템의 자원으로 URI로 정의

■ 리소스 URI 설계 규칙

- URI는 명사 사용
- 슬래시로 계층 관계 표현
- 마지막에 슬래시 사용 금지
- 소문자로 작성
- 가독성을 위해 하이픈(-)은 사용하지만 밑줄(_)는 사용 금지

■ HTTP 메서드

| HTTP 메서드 | 의미 | 역할 |
|----------|--------|--------|
| POST | CREATE | 리소스 생성 |
| GET | READ | 리소스 조회 |
| PUT | UPDATE | 리소스 수정 |
| DELETE | DELETE | 리소스 삭제 |

Json 응답 구현

- 의존성 라이브러리 등록 (pom.xml)

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.4</version>
</dependency>
<!-- java8 date/time -->
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jsr310</artifactId>
  <version>2.9.4</version>
</dependency>
<!-- java8 Optional, etc -->
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-jdk8</artifactId>
  <version>2.9.4</version>
</dependency>
```

Json 응답 구현

- @Controller + @ResponseBody
 - 요청 처리 메서드에 @ResponseBody를 선언하고 객체를 반환하면 json으로 변환
- @RestController
 - 컨트롤러 클래스에 @Controller 대신 @RestController를 선언하면 해당 컨트롤러 클래스에 포함된 모든 요청 처리 메서드에 자동으로 @ResponseBody가 선언된 효과

```
@RestController
public class RestMemberController {
    private MemberDao memberDao;
    private MemberRegisterService registerService;

    @GetMapping("/api/members")
    public List<Member> members() {
        return memberDao.selectAll();
    }
}
```

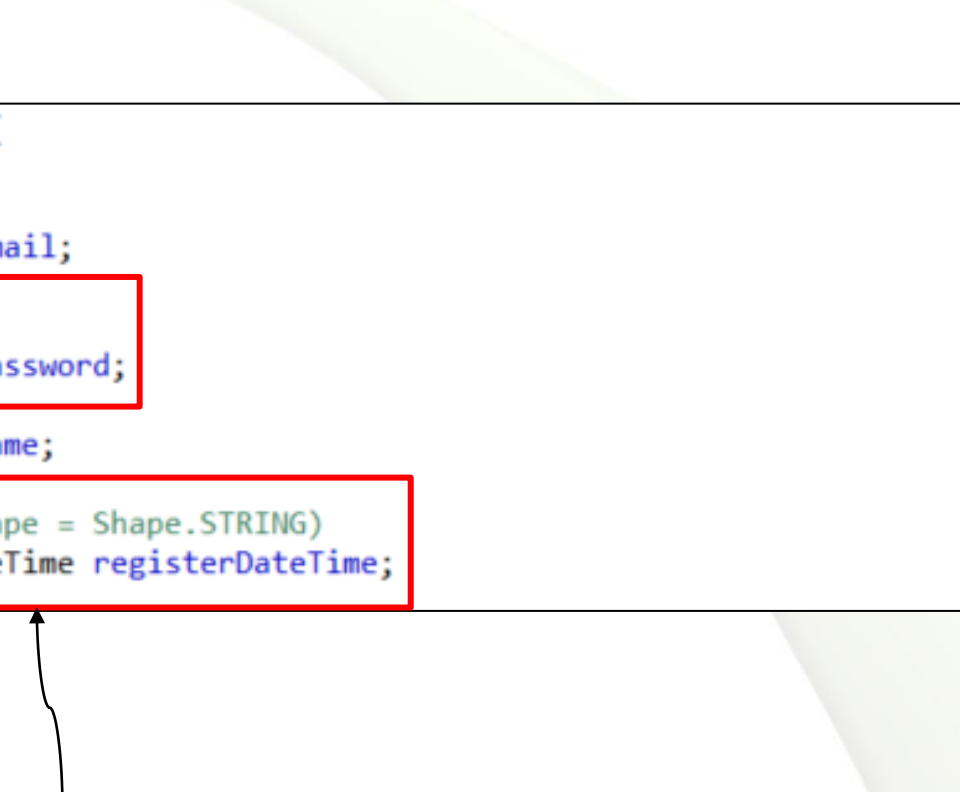
```
@GetMapping("/api/members/{id}")
public ResponseEntity<Object> member(@PathVariable Long id) {
    Member member = memberDao.selectById(id);
    if (member == null) {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(new ErrorResponse("no member"));
        // return ResponseEntity.notFound().build();
    }
    return ResponseEntity.ok(member);
}
```

Json 응답 구현

■ @JsonIgnore

- 패스워드와 같은 데이터의 노출을 막기 위해 필드에 선언

```
public class Member {  
  
    private Long id;  
    private String email;  
  
    @JsonIgnore  
    private String password;  
  
    private String name;  
  
    // @JsonFormat(shape = Shape.STRING)  
    private LocalDateTime registerDateTime;  
}
```



■ 날짜 형식 지정

- 기본 구현은 밀리초 단위의 타임스탬프 값 반환
- @JsonFormat으로 날짜 표현 형식 변경

Json 응답 구현

- @EnableWebMvc가 선언된 WebMvcConfigurer 구현 클래스에 모든 날짜 형식 데이터 처리에 공통으로 적용할 기본 구현 설정 가능

```
@Configuration
@EnableWebMvc
public class MvcConfig implements WebMvcConfigurer {

    @Override
    public void extendMessageConverters(List<HttpMessageConverter<?>> converters) {
        //DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        ObjectMapper objectMapper = Jackson2ObjectMapperBuilder
            .json()
            .featuresToDisable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS)
            //.serializerByType(LocalDateTime.class, new LocalDateTimeSerializer(formatter))
            //.deserializerByType(LocalDateTime.class, new LocalDateTimeDeserializer(formatter))
            //.simpleDateFormat("yyyy-MM-dd HH:mm:ss")
            .build();
        converters.add(0, new MappingJackson2HttpMessageConverter(objectMapper));
    }
}
```

Json 응답 구현

- HttpServletResponse 객체를 통해 응답 코드를 설정할 경우 404와 같이 서버에서 정한 응답 콘텐츠가 전송될 수 있음
- 오류와 같은 특수한 상황에서도 일관되게 Json 형식의 데이터 전송 필요
- ResponseEntity 객체를 통해 응답 코드와 메시지를 직접 관리할 수 있음

```
@PostMapping("/api/members")
public ResponseEntity<Object> newMember(
    @RequestBody @Valid RegisterRequest regReq /*,
    Errors errors */) {
    /*
    if (errors.hasErrors()) {
        String errorCodes = errors.getAllErrors()
            .stream()
            .map(error -> error.getCodes()[0])
            .collect(Collectors.joining(", "));
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(new ErrorResponse("errorCodes = " + errorCodes));
    }
    */
    try {
        Long newMemberId = registerService.regist(regReq);
        URI uri = URI.create("/api/members/" + newMemberId);
        return ResponseEntity.created(uri).build();
    } catch (DuplicateMemberException dupEx) {
        return ResponseEntity.status(HttpStatus.CONFLICT).build();
    }
}
```

Json 요청 처리

- 요청 처리 메서드의 전달인자에 `@RequestBody`를 선언해서 Json 요청 데이터를 객체로 변환

```
@PostMapping("/api/members2")
public void newMember2(
    @RequestBody RegisterRequest regReq,
    Errors errors,
    HttpServletResponse response) throws IOException {
    try {
        new RegisterRequestValidator().validate(regReq, errors);
        if (errors.hasErrors()) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST);
            return;
        }
        Long newMemberId = registerService.regist(regReq);
        response.setHeader("Location", "/api/members/" + newMemberId);
        response.setStatus(HttpServletResponse.SC_CREATED);
    } catch (DuplicateMemberException dupEx) {
        response.sendError(HttpServletResponse.SC_CONFLICT);
    }
}
```

웹 브라우저 요청 HTTP 메서드 활성화

- 웹 브라우저의 HTML은 POST와 GET 메서드만 지원
- 스프링은 HiddenHttpMethodFilter를 통해 POST, GET 뿐만 아니라 PUT, DELETE 메서드를 사용할 수 있도록 지원
- 메서드 활성화를 위해 필터 등록

```
@Bean
public HiddenHttpMethodFilter methodFilter() {
    HiddenHttpMethodFilter filter = new HiddenHttpMethodFilter();
    return filter;
}
```

웹 브라우저 요청 HTTP 메서드 활성화

- HTML에 hidden method field 포함

```
<!-- HttpHiddenMethodFilter에게 전달할 데이터 (method 정보) -->  
<input type="hidden" id="_method" name="_method">  
</form>
```

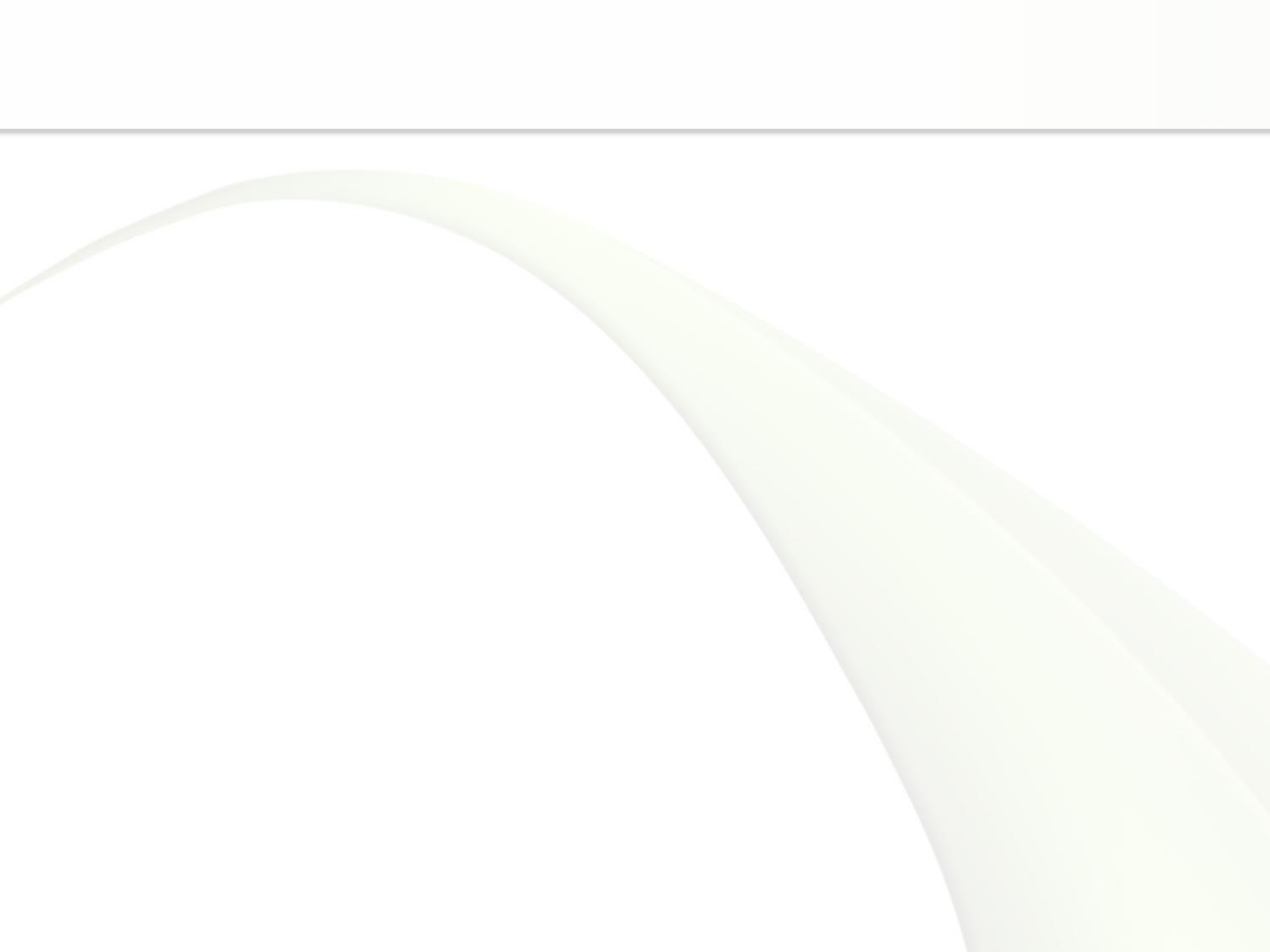
- 각 컨트롤러에서 메서드 매핑

```
@GetMapping(path = { "/list" })  
public String showList(Model model) {
```

```
@PostMapping(path = { "/write" })  
public String doWrite(BoardEntity board, MultipartHttpServletRequest req) {
```

```
@PutMapping(path = { "/update" })  
public String updateBoard(BoardEntity board) {
```

```
@DeleteMapping(path = { "/delete/{boardIdx}" })  
public String deleteBoard(@PathVariable int boardIdx, BoardEntity board) {
```



JPA (Java Persistence API)

- 자바 객체와 데이터베이스 테이블 간의 매핑을 처리하는 ORM 기술 표준
- JAP 구현체
 - JPA 기술 명세를 구현한 제품 → JPA Provider
 - 하이버네이트, 이클립스링크 등
- 장점
 - 개발이 편리하다
 - 데이터베이스에 대한 종속성 제거 → 데이터베이스 제품 의존성 없음
 - 유지보수가 쉽다
- 단점
 - 학습곡선이 가파르다
 - 특정 데이터베이스의 특화된 기능 사용 제한
 - 객체지향 설계 필요

Spring Data JPA

- 스프링에서 JPA를 쉽게 사용할 수 있도록 지원하는 라이브러리
- Spring Data JPA가 제공하는 Repository 인터페이스를 상속해서 정해진 규칙에 맞게 메서드를 작성하면 자동으로 데이터 연동 코드 완성
- 내부에서는 실제 기능을 담당하는 JPA 구현체로 하이버네이트 사용
- 하이버네이트와 같은 ORM 사용에 필요한 다양한 요구사항을 추상화해서 상대적으로 쉽게 JPA 사용 가능

Spring Data JPA 구현

- application.properties 설정

```
spring.jpa.database=mysql  
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect  
spring.jpa.generate-ddl=true  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.hibernate.use-new-id-generator-mappings=false
```

- DatabaseConfig.java 변경

```
@Bean  
@ConfigurationProperties(prefix="spring.jpa")  
public Properties hibernateConfig() {  
    return new Properties();  
}
```

- Application 클래스 수정

```
@EntityScan(basePackages = { "com.springdemo.bootboard" })  
@SpringBootApplication  
public class SpringBoardBApplication {
```

Spring Data JPA 구현

▪ Entity 클래스 정의

```
@Entity
@Table(name = "tbl_users")
@Data
public class UsersEntity {

    @Id
    private String userName;

    @Column
    private String passwd;

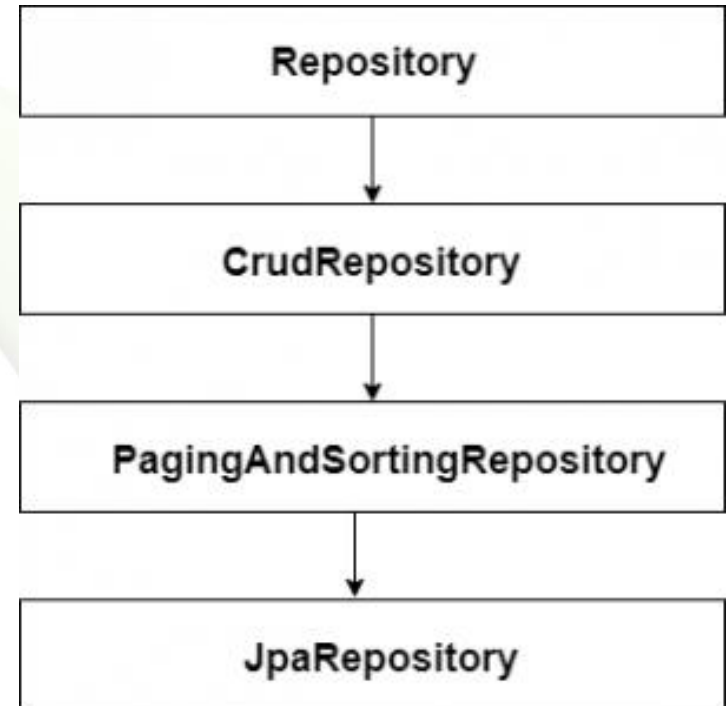
    @Column
    private boolean enabled = true;

    @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name = "user_name")
    private Collection<UsersRolesEntity> usersRoles;

}
```

Repository interface

- 스프링 데이터 JPA가 제공하는 인터페이스로 기능 수준에 따라 4가지 인터페이스 제공
- 각 인터페이스를 상속하는 인터페이스를 정의하면 일정한 패턴에 따라 데이터베이스 연동 메서드를 자동 생성
- 자동으로 생성된 메서드로 충족되지 않는 기능은 일정한 규칙에 따라 메서드를 선언할 수 있음
- JPA 에서 제공하는 JPQL 이 나 데이터베이스에서 사용하는 SQL 을 사용해서 메서드를 만드는 것도 가능



Repository interface method

▪ CrudRepository 인터페이스 기준

| 메서드 | 설명 |
|---------------------|------------------------------|
| save(entity) | 지정된 엔티티 저장 |
| saveAll(entities) | 지정된 엔티티 목록 저장 |
| findById(id) | 지정된 아이디로 식별된 엔티티 반환 |
| existsById(id) | 지정된 아이디로 식별된 엔티티 존재 여부 반환 |
| findAll() | 모든 엔티티 반환 |
| findAllById(ids) | 지정된 아이디 목록에 의해 식별된 모든 엔티티 반환 |
| count() | 사용 가능한 엔티티 개수 반환 |
| deleteById(id) | 지정된 아이디로 식별된 엔티티 삭제 |
| delete(entity) | 지정된 엔티티 삭제 |
| deleteAll(entities) | 지정된 엔티티 목록 삭제 |
| deleteAll() | 모든 엔티티 삭제 |

Query Method Pattern

- 메서드 이름은 `find...By`, `read...By`, `query...By`, `count...By`, `get...By`로 시작
- 첫 번째 `By` 뒤쪽은 컬럼 이름으로 구성 → 검색 조건
- `And` 또는 `Or` 키워드를 사용해서 두 개 이상의 속성을 조합
- 다양한 비교 연산자를 조합해서 필요한 쿼리 작성
- 쿼리 메서드 연산자

| 키워드 | 사용사례 |
|------------|--|
| And | <code>findByLastnameAndFirstname</code> |
| Or | <code>findByLastnameOrFirstname</code> |
| Is, Equals | <code>findByFirstname</code> <code>findByFirstnameIs</code> <code>findByFirstnameEquals</code> |

Query Method Pattern

■ 쿼리 메서드 연산자 (계속)

| 키워드 | 사용사례 |
|--------------------|--------------------------------------|
| Between | findByStartDateBetween |
| LessThan | findByAgeLessThan |
| LessThanEqual | findByAgeLessThanEqual |
| GreaterThan | findByAgeGreaterThan |
| GreaterThanEqual | findByAgeGreaterThan |
| After | findByStartDateAfter |
| Before | findByStartDateBefore |
| IsNull | findByAgeIsNull |
| IsNotNull, NotNull | findByAgeIsNotNull, findByAgeNotNull |
| Like | findByFirstnameLike |
| NotLike | findByFirstnameNotLike |
| StartingWith | findByFirstnameStartWith |
| EndingWith | findByFirstnameEndingWith |

Query Method Pattern

■ 쿼리 메서드 연산자 (계속)

| 키워드 | 사용사례 |
|------------|---------------------------|
| Containing | findByFirstnameContaining |
| OrderBy | findByAgeOrderByLastname |
| Not | findByLastnameNot |
| In | findByAgeIn |
| NotIn | findByAgeNotIn |
| True | findByActiveTrue |
| False | findByActiveFalse |
| IgnoreCase | findByFirstnameIgnoreCase |

■ @Query

- 메서드 이름이 복잡하거나 쿼리 메서드로 표현하기 힘든 경우 직접 SQL을 작성할 때 사용
- JPQL 또는 데이터베이스에서 지원하는 SQL 사용

Spring Data JPA 구현

▪ Repository 인터페이스 정의

```
public interface BoardRepository
    extends CrudRepository<BoardEntity, Integer> {

    //find : select,
    //AllBy : 모든 데이터,
    //OrderByBoardIdxDesc : order by board_idx desc
    List<BoardEntity> findAllByOrderByBoardIdxDesc();

    @Query("SELECT file FROM BoardFileEntity file WHERE idx = :idx")
    BoardFileEntity findBoardFileByIdx(@Param("idx") int idx);

    @Transactional
    @Modifying
    @Query("UPDATE BoardEntity b " +
        "SET b.title = :#{#board.title}, " +
        "    b.contents = :#{#board.contents}, " +
        "    b.updatedDatetime = :#{#board.updatedDatetime} " +
        "WHERE b.boardIdx = :#{#board.boardIdx}")
    void updateBoardOnly(@Param("board") BoardEntity board);
}
```