



Spring Boot의 이해와 활용

강사 : 백현숙

목차

- 1일차 : 스프링부트의 기본구조, MVC, Lombok, thymeleaf, crud
- 2일차 : Restful API 서버, JPA 연동
- 3일차 : Mybatis, Ajax, 회원가입과 로그온
- 4일차 : Aop, Transaction, logback, vue와 연동
- 5일차 : 보안(Spring Security), 인증과 인가,
Security 설정 , 토큰 인증방식, Vue와 연동(토큰방식)

스프링부트란

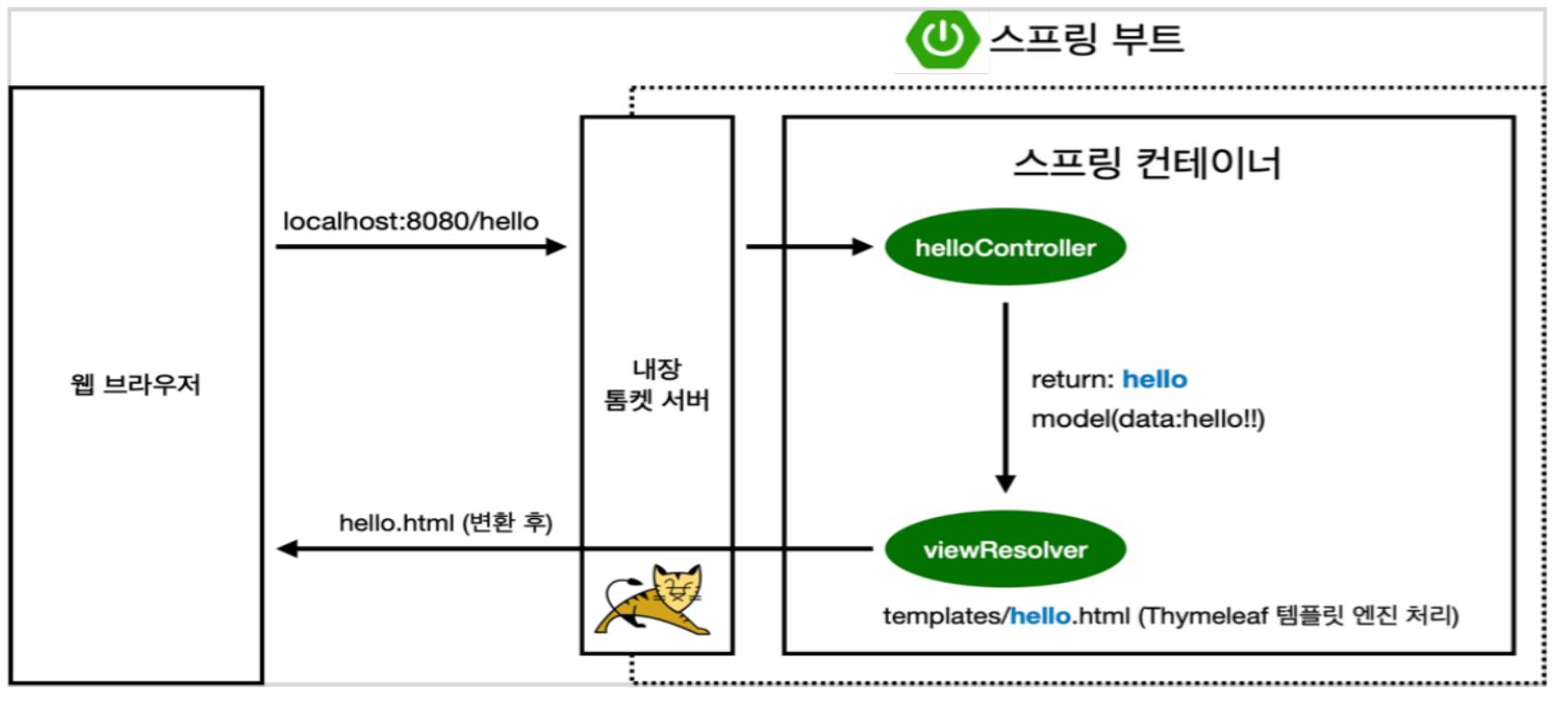
- 스프링부트(Spring Boot)는 **스프링 프레임워크**를 기반으로 한 자바 애플리케이션 개발을 쉽게 할 수 있도록 도와주는 프레임워크입니다.
- 스프링프레임워크는 강력하고 유연한 프레임워크이지만, 설정이 복잡하고 많은 설정 파일이 필요하다는 단점이 있습니다.
- 스프링부트는 이런 복잡한 설정 과정을 단순화하고, 빠르게 애플리케이션을 개발할 수 있도록 **자동 설정 기능**과 여러 편의 기능을 제공합니다.

스프링부트의 특징

- 자동 설정(Auto Configuration) : 설정을 대부분 자동으로 해줍니다. 설정은 properties 나 yaml(야물)파일을 사용합니다.
- 내장 WAS사용(예: Tomcat, Jetty 등) : 외장 was를 별도로 설치할 필요가 없습니다.
- 스타터(Starter): 자주 사용되는 의존성 세트를 하나의 그룹(spring-boot-starter-web)으로 묶어 제공합니다.
- 프로덕션 제공 : 모니터링, 보안, 로깅 등과 같은 기능을 손쉽게 통합할 수 있으며, 애플리케이션을 운영 환경에서 쉽게 관리할 수 있는 도구를 제공합니다.
- Spring Boot CLI: 스프링부트는 명령줄 도구를 제공하여 그루비(Groovy) 스크립트로 애플리케이션을 빠르게 개발하고 실행할 수 있습니다.
- JSP 템플릿 배제 : thymeleaf(타임리프), mustache(머스티치)등을 이용해야 합니다 jsp 사용이 가능하나 권장하지 않습니다.
- RestController등 Restful API 작성이 용이합니다.

스프링 구조

동작 환경 그림



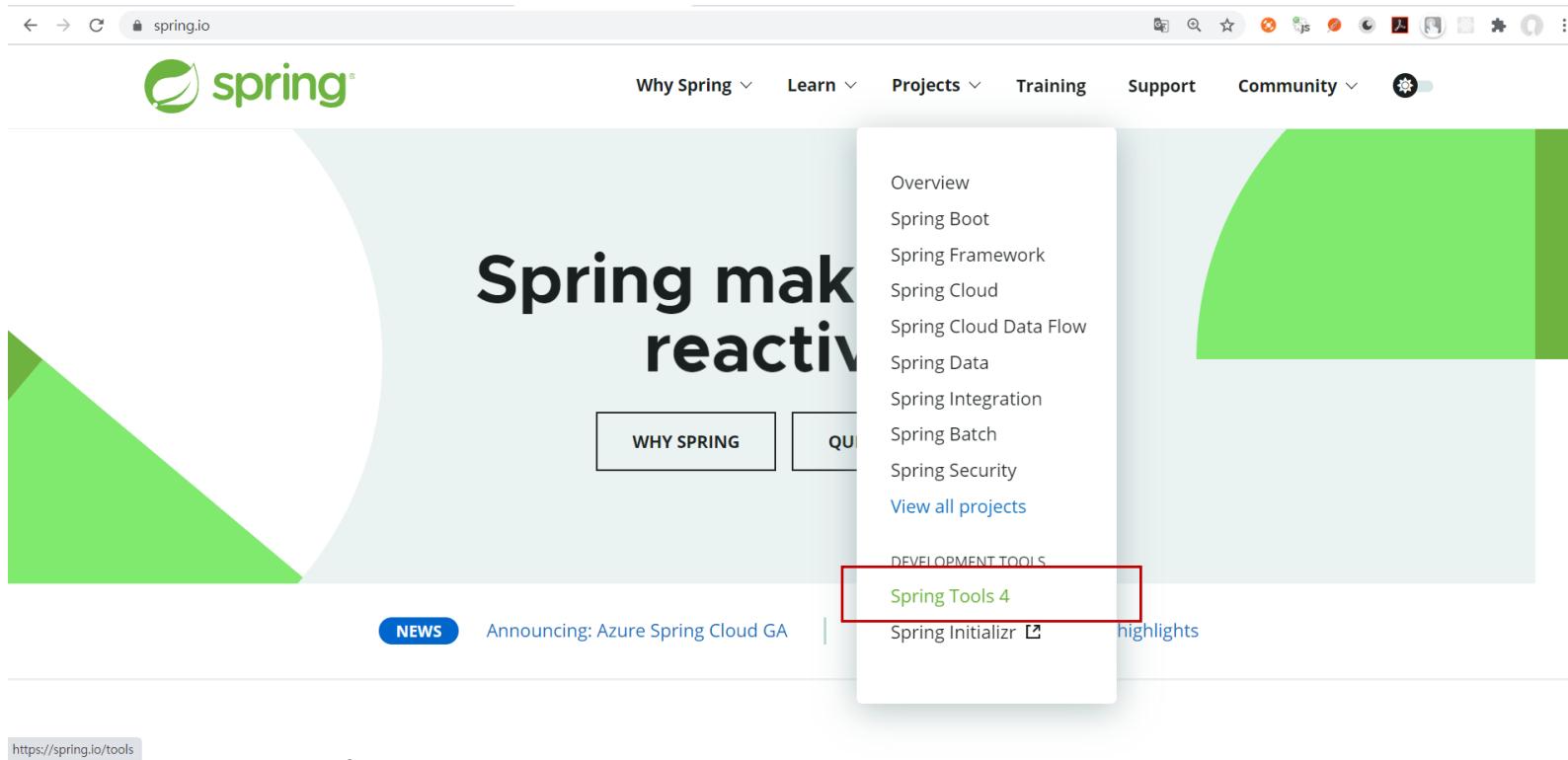
출처 : <https://velog.io/@hono2030/Spring-%EA%B8%B0%EB%B3%B8-%EC%A0%95%EB%A6%AC>

스프링부트 개발 도구

- 이클립스
 - spring.io 사이트에서 다운받을 수 있음
- 인텔리제이
 - 무료버전과 ultimate 버전이 있음
 - jetbrains 사이트에서 다운받을 수 있음
 - 최근에 사용자가 급증하고 있음
 - 무료버전의 경우 직접 프로젝트 생성이 불가능, Spring Initializr를 이용해서 작성해야 함

스프링 부트 툴

- <http://spring.io>



스프링 부트

spring.io/tools

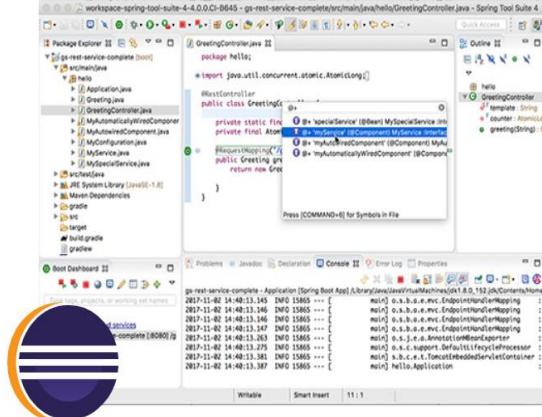
Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.
Free. Open source.

4.8.0 - LINUX 64-BIT

4.8.0 - MACOS 64-BIT

4.8.0 - WINDOWS 64-BIT



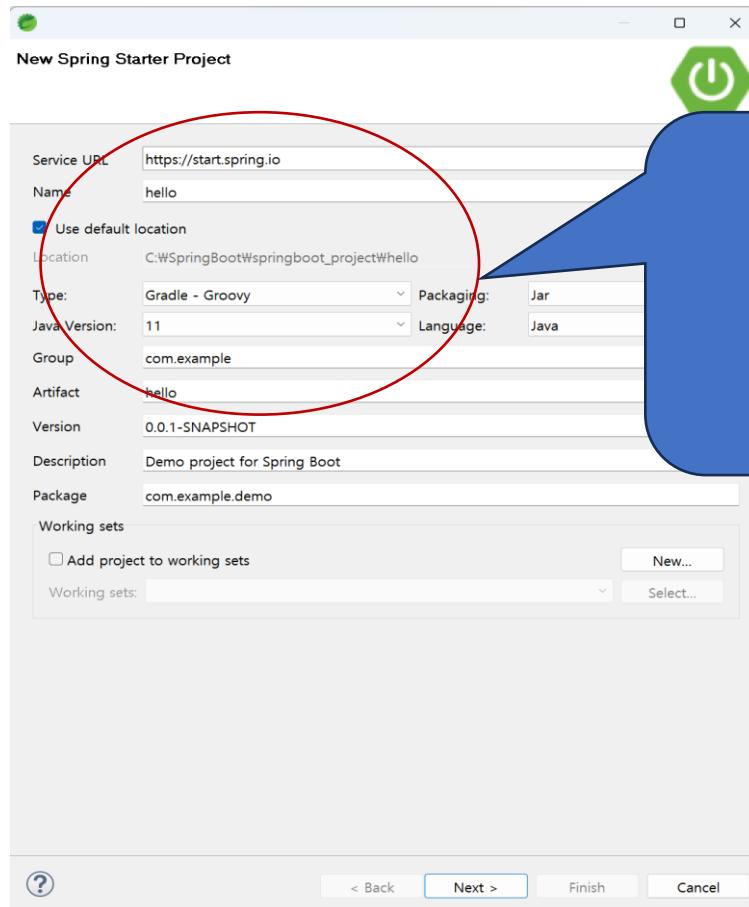
A screenshot of the Spring Tool Suite 4 interface. The interface includes a top navigation bar with various icons. Below it is a toolbar with buttons for file operations like New, Open, Save, and Print. The main workspace consists of several views: the Package Explorer view on the left showing project structure; the Java Editor view in the center displaying code for a GreetingController.java file; and the Console view at the bottom showing application logs. A central status bar at the bottom indicates the current file is 'SiteApplication.java'.

스프링 설치(이클립스기반)

- 파일을 더블클릭하면 jar파일의 압축이 풀립니다.
- 만일 알집이 설치되어 있으면 jar파일이 실행되는 것이 아니고 압축이 풀려서 제대로 동작하지 않으므로 알집을 삭제 후 다시 실행하거나 직접 java명령어로 압축을 풀어야 합니다.
- 자바가 설치되어 있어야 합니다.
- `jar -xvf yourfile.jar`
- 스프링 3으로 개발할 목적이 아니면 별도의 톰캣을 설치 하지 않습니다.
- 스프링부트에서는 내장 was를 사용하고 배포버전은 war형식이 아니라 jar 형식입니다.

Hello, Web Application

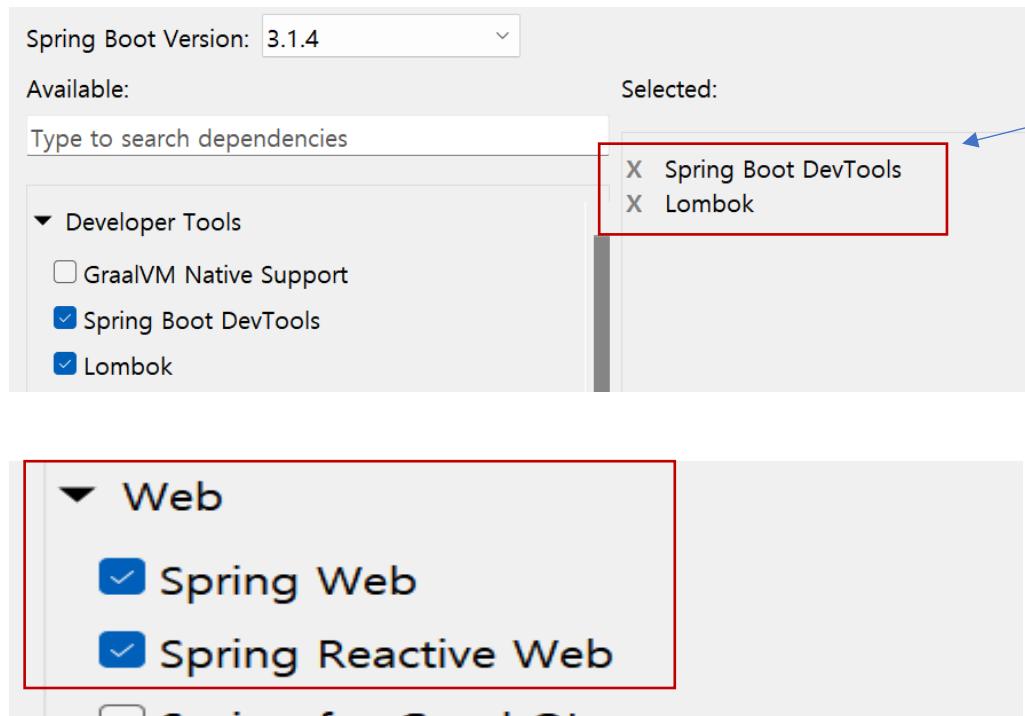
- file – new – spring starter project



name : demo
Type : Gradle – Groovy
java Version : 17
Packaging : jar
Group:com.example
Artifact: hello

Hello, Web Application

- 필요한 라이브러리 지정하기



최소한의 라이브러리
Spring Boot DevTools –
코드변경서 서버재시작을
돕는다.

Lombok-수많은 어노테이션을
제공해서 개발자들이
Voiler plate 코딩을 하지 않도록
도와준다.

Spring Web
Spring Reactive Web

Hello, Web Application

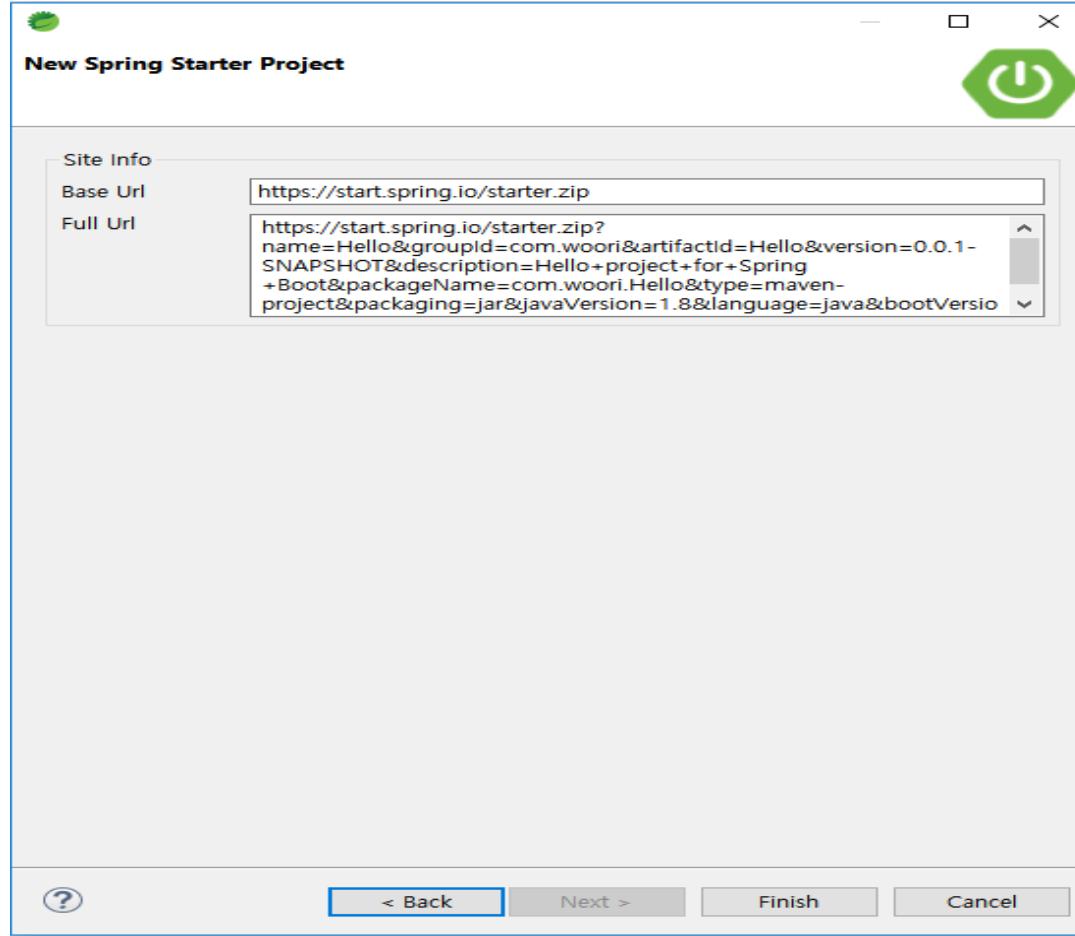
- 프로젝트명 : Hello
- Group에 패키지명을 입력하세요, 최소 2depth입니다.
- 패키지가 있을경우에 패키지명은 회사 domain을 거꾸로 해야 합니다.
- 예시(com.mycompany) 보다 아래에 클래스를 두어야 합니다.
- 3depth 이하에 코드를 두어야 합니다.
- 프레임워크는 직접 설정파일을 통해 작성했으나 스프링부트는 자동설정입니다.

부트의 모든 프로젝트는 main으로 시작합니다.

@SpringBootApplication 어노테이션이 여기로부터 시작함을 의미합니다

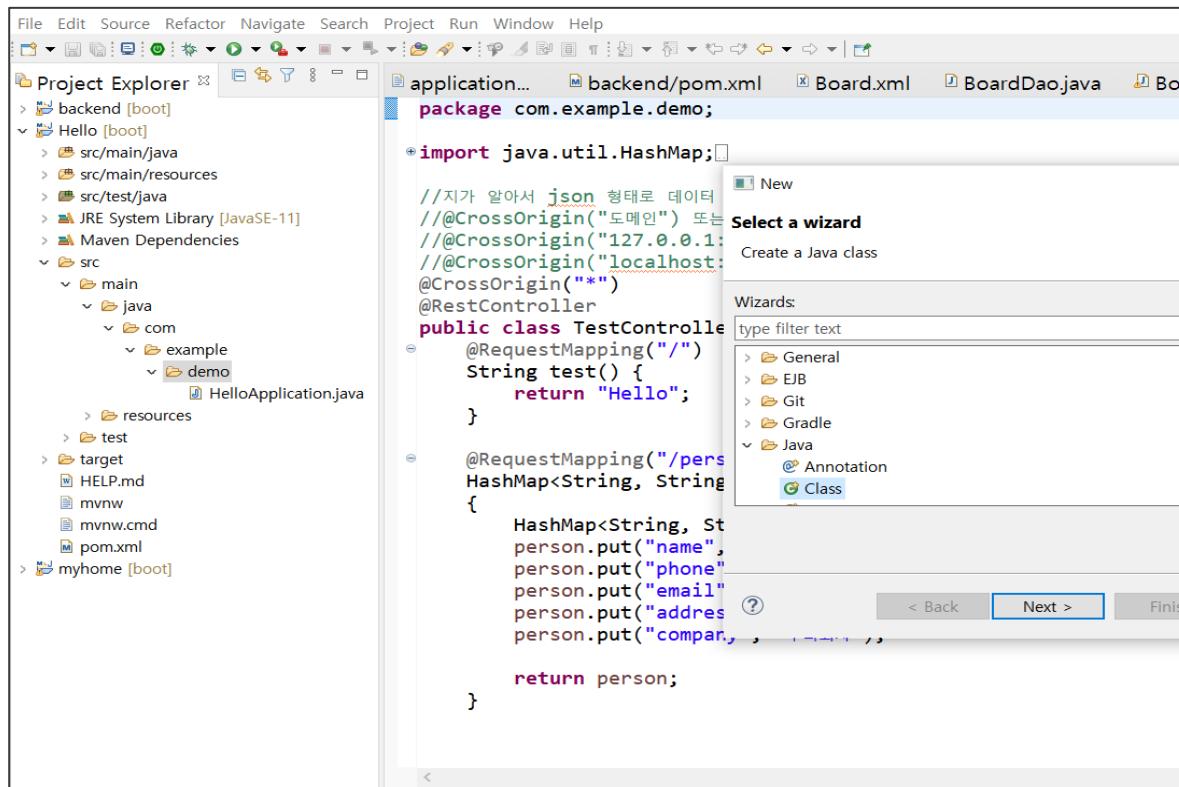
```
@SpringBootApplication  
public class MyappFileApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(MyappFileApplication.class, args);  
    }  
}
```

Hello, Web Application

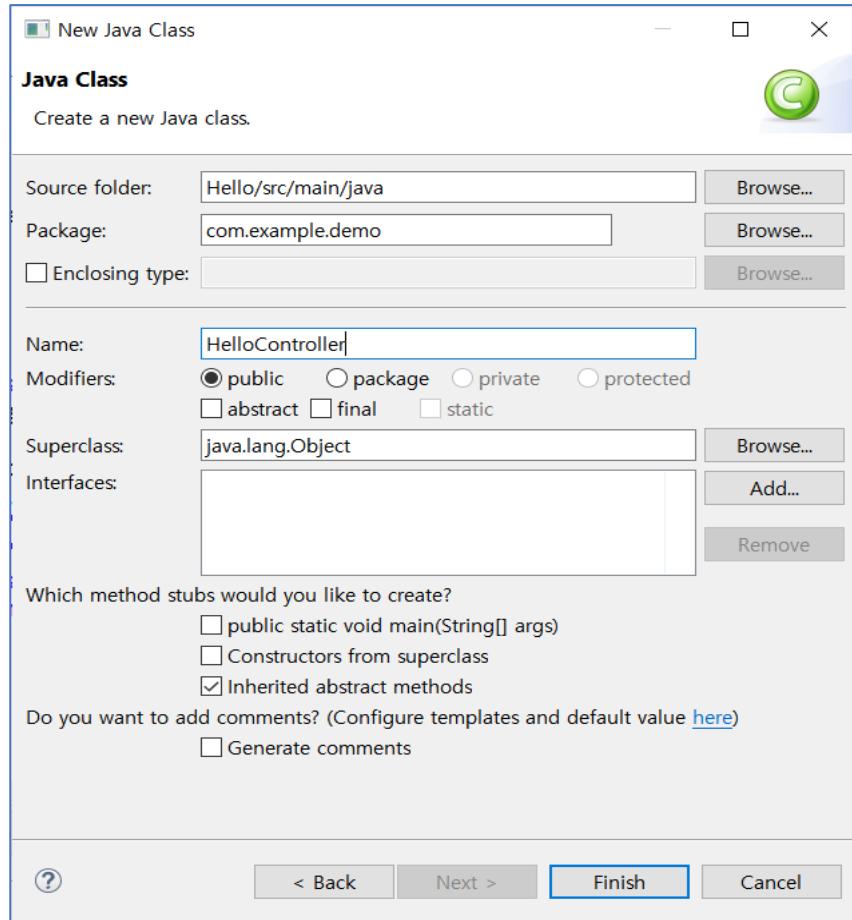


HelloController 추가

- src/main/java/com/example/hello 최소 이 아래에 추가해야 동작합니다
- 마우스오른쪽 other - class



HelloController 추가



HelloController 작성

```
package com.example.demo;

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

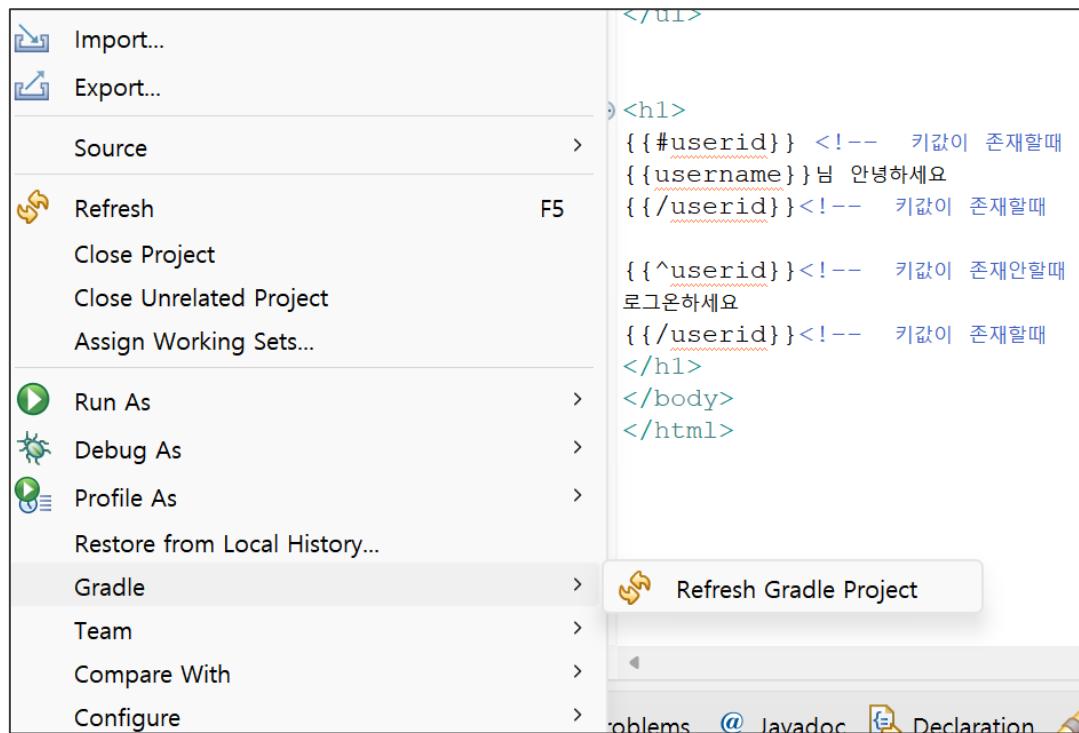
@RestController
public class HelloController {

    @RequestMapping("/")
    String hello() {
        return "Hello Spring boot";
    }
}
```

@RestController : 반환값을 json 형태로 변환해서 클라이언트로 보내준다. 부트는 기본적으로 jsp 템플릿을 지원하지 않는다

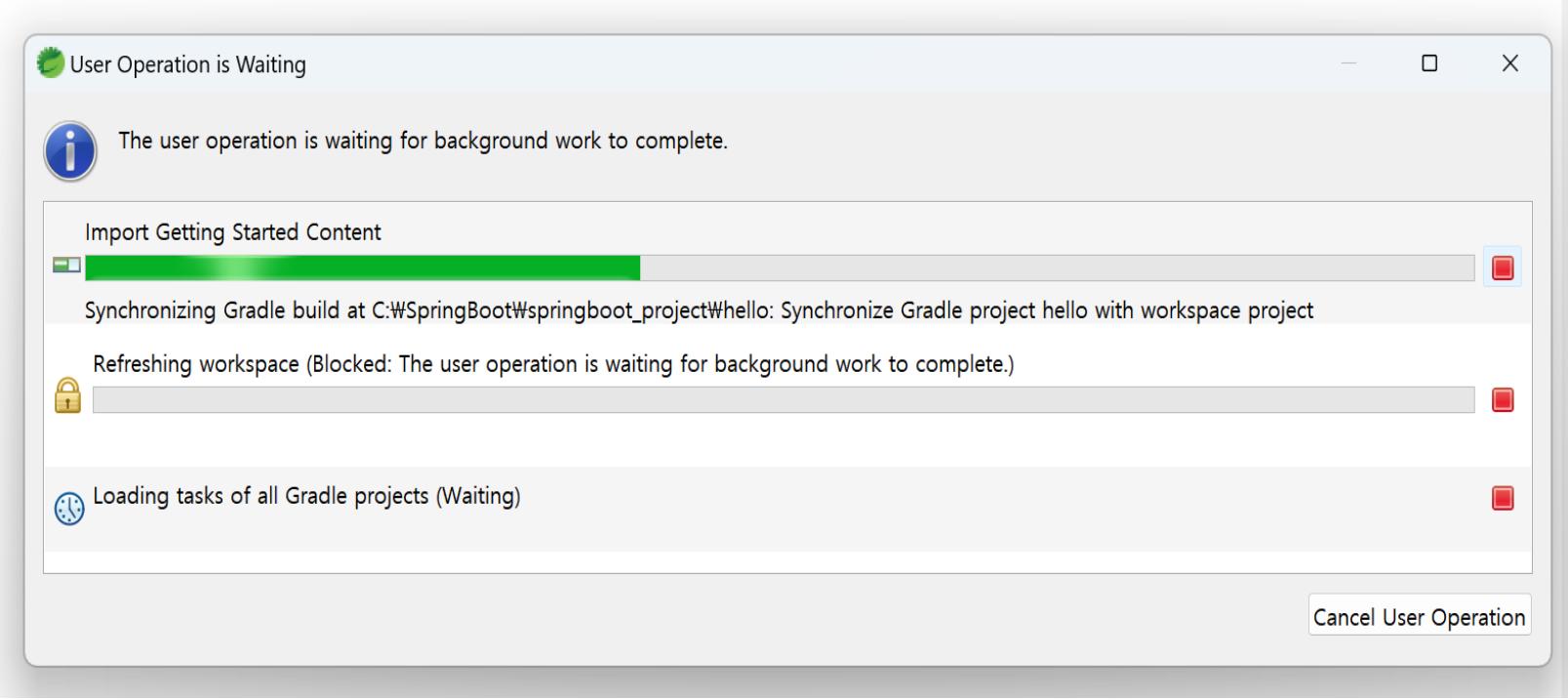
Gradle install

- project – 마우스 오른쪽 - gradle – Refresh Gradle Project
- Gradle 이 필요한 라이브러리를 가져와서 빌딩을 해준다



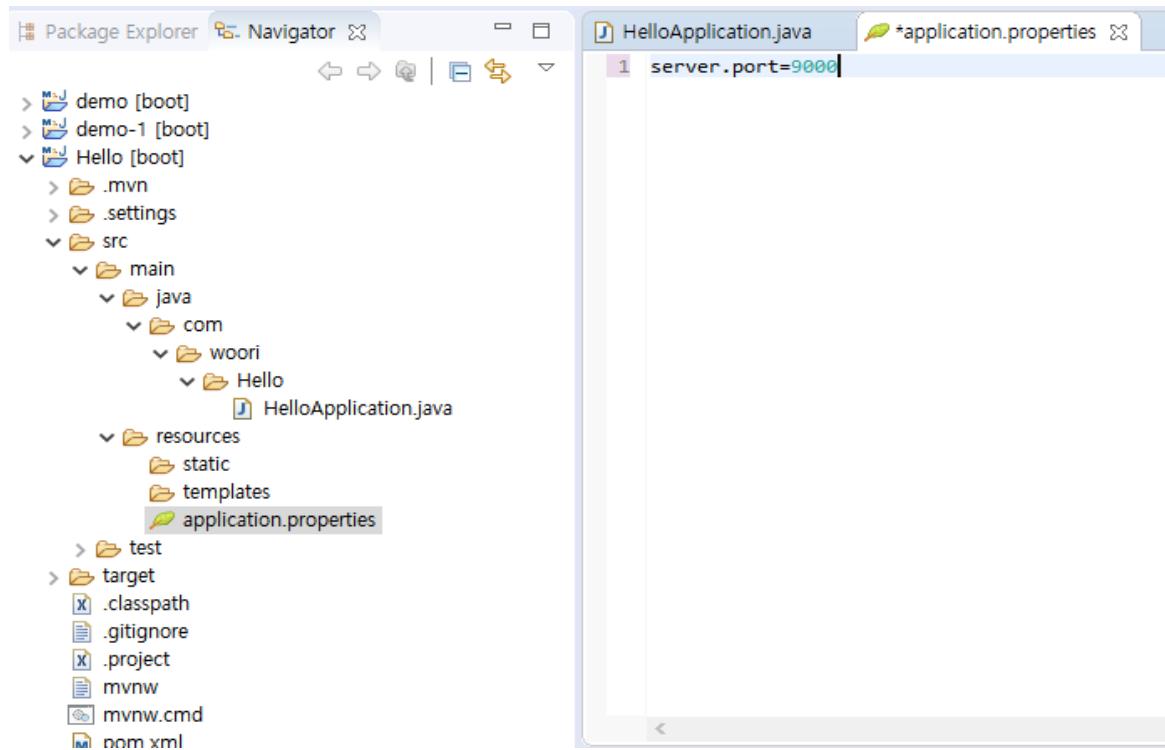
Gradle

- 빌딩 중



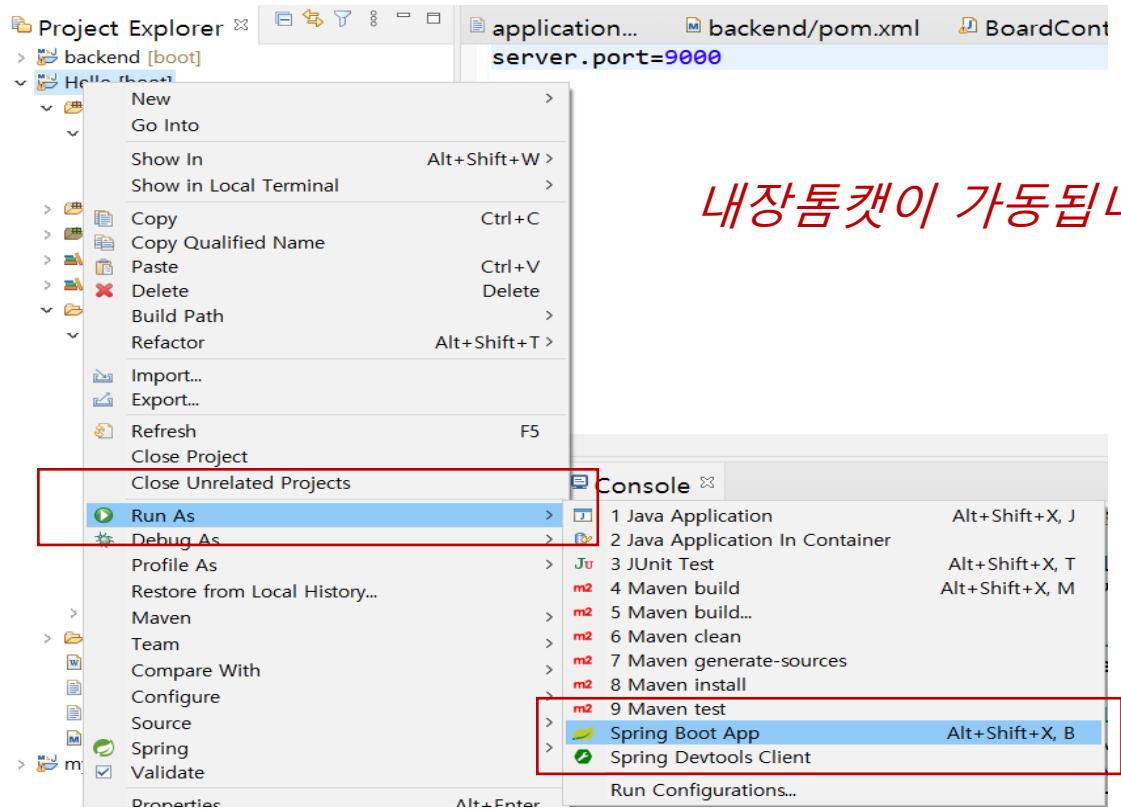
Hello, Web Application

- application.properties 는 환경설정을 담당합니다.(yml파일도 가능)
- 이 파일에 포트번호를 지정합니다.



서버 실행

- 프로젝트명 - 마우스 오른쪽 - run as - spring boot app



Hello, Web Application

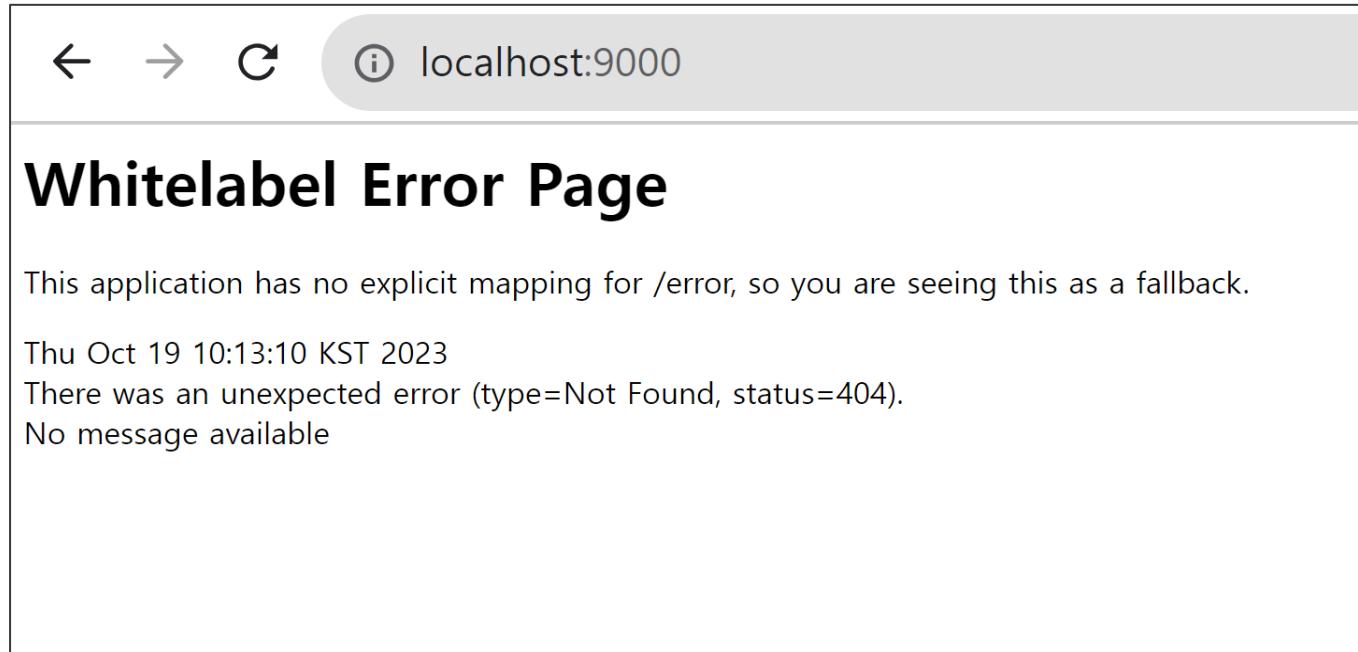
서버시작이 정상적일때의 상황입니다.

```
.\\_/\_\_((_)_-_-\\_\_\\_\_
((_)\\_\_((_)_/_\_\_((_)_)))
`|__|_/_|_/_|_/_\_,_|_//_/
=====|=====/=_/_/_/
:: Spring Boot ::      (v2.1.0.RELEASE)

2018-11-06 16:21:11.625 INFO 16200 --- [      main] com.woori.Hello.HelloApplication      : Starting HelloApplication on LAPTOP-8IK90LE8 with PID 16200 (C:\SpringBoot)
2018-11-06 16:21:11.629 INFO 16200 --- [      main] com.woori.Hello.HelloApplication      : No active profile set, falling back to default profiles: default
2018-11-06 16:21:12.611 INFO 16200 --- [      main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat initialized with port(s): 9000 (http)
2018-11-06 16:21:12.628 INFO 16200 --- [      main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-11-06 16:21:12.628 INFO 16200 --- [      main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/9.0.12
2018-11-06 16:21:12.634 INFO 16200 --- [      main] o.a.catalina.core.AprLifecycleListener    : The APR based Apache Tomcat Native library which allows optimal performance
2018-11-06 16:21:12.763 INFO 16200 --- [      main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-11-06 16:21:12.763 INFO 16200 --- [      main] o.s.web.context.ContextLoader          : Root WebApplicationContext: initialization completed in 1083 ms
2018-11-06 16:21:12.806 INFO 16200 --- [      main] o.s.b.w.servlet.ServletRegistrationBean   : Servlet dispatcherServlet mapped to [/]
2018-11-06 16:21:12.810 INFO 16200 --- [      main] o.s.b.w.servlet.FilterRegistrationBean     : Mapping filter: 'characterEncodingFilter' to: [*]
2018-11-06 16:21:12.810 INFO 16200 --- [      main] o.s.b.w.servlet.FilterRegistrationBean     : Mapping filter: 'hiddenHttpMethodFilter' to: [*]
2018-11-06 16:21:12.810 INFO 16200 --- [      main] o.s.b.w.servlet.FilterRegistrationBean     : Mapping filter: 'formContentFilter' to: [*]
2018-11-06 16:21:12.810 INFO 16200 --- [      main] o.s.b.w.servlet.FilterRegistrationBean     : Mapping filter: 'requestContextFilter' to: [*]
2018-11-06 16:21:13.006 INFO 16200 --- [      main] o.s.s.concurrent.ThreadPoolTaskExecutor    : Initializing ExecutorService 'applicationTaskExecutor'
2018-11-06 16:21:13.224 INFO 16200 --- [      main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port(s): 9000 (http) with context path ''
2018-11-06 16:21:13.227 INFO 16200 --- [      main] com.woori.Hello.HelloApplication      : Started HelloApplication in 1.944 seconds (JVM running for 2.921)
```

Hello, Web Application

- 아무런 코드가 없기 때문에 아래와 같은 화면이 나타납니다.



HelloController 추가하기

- class 추가 : 3depth 이하에 클래스를 설치해야 인식됩니다. (자동설정)
- com.example.demo.HelloController.java

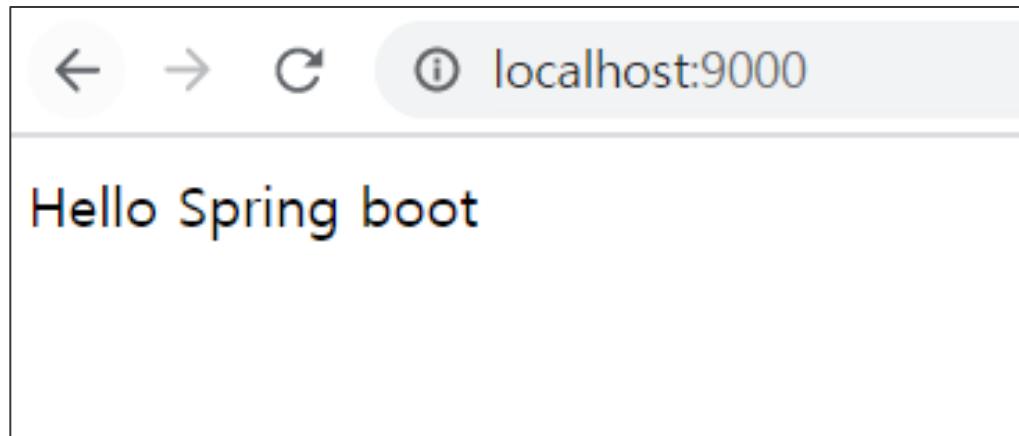
```
package com.example.demo;

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

@RestController
public class HelloController {
    @GetMapping("/")
    public String hello() {
        return "Hello Spring Boot";
    }
}
```

Hello, Web Application

실행화면



@RestController 는 응답을 JSON객체로 합니다.
별도의 template 엔진이 필요하지 않습니다.

Hello, Web Application

- `@RestController` : 별도의 html 페이지가 필요 없음, JSON형태로 전달합니다.
스프링부트 내부에 JSON 엔진이 장착되어 있습니다.
- `@Controller` : 별도의 html 페이지가 필요합니다. 뷰엔진이 기본 탑재가 아니므로
별도의 엔진을 설치해야 합니다.(thymeleaf 나 mustache)

```
1 package com.woori.Hello;
2
3 import org.springframework.web.bind.annotation.RestController;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.stereotype.Controller;
6
7 @Controller
8 public class TestController {
9
10    @RequestMapping("/test")
11    public String test()
12    {
13        return "test";
14    }
15}
16
```

인텔리제이

- JetBrains 사이트
- <https://www.jetbrains.com/ko-kr/idea/download/?section=windows>



인텔리제이

- <https://spring.io> 사이트로 이동 후 [project] – [Spring Initialize]를 선택합니다.

Project

Gradle - Groovy
 Gradle - Kotlin Maven Groovy

Spring Boot

3.3.0 (SNAPSHOT) 3.3.0 (M2) 3.2.4 (SNAPSHOT)
 3.2.3 3.1.10 (SNAPSHOT) 3.1.9

Project Metadata

Group com.example

Artifact demo

Name hello

Description Demo project for Spring Boot

Package name com.example.demo

Packaging Jar War

Java 21 17

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

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.

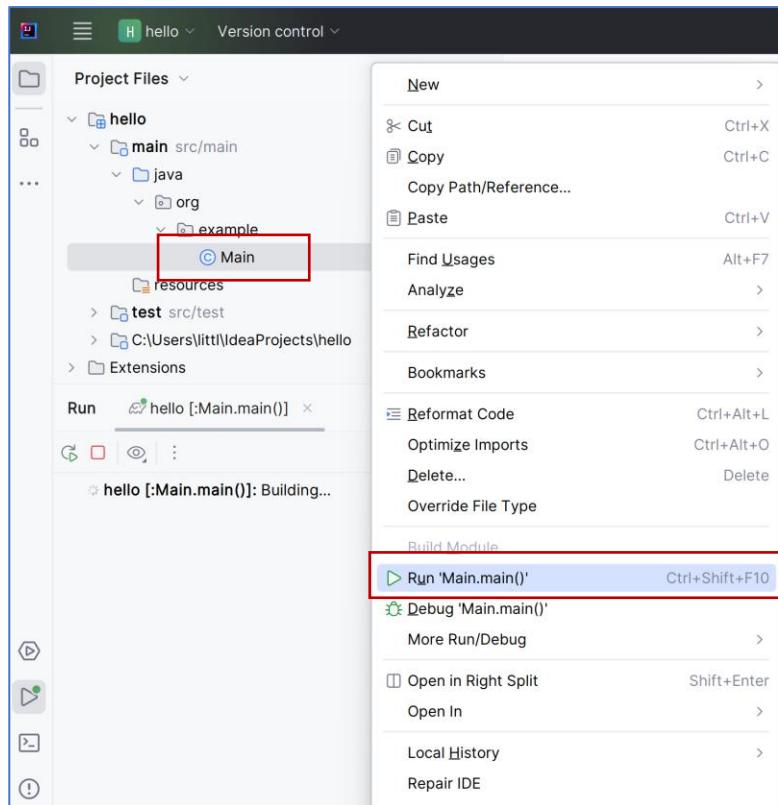
인텔리제이 jdk 설정

- 파일 – 설정 – 빌드, 실행, 배포 – 빌드 도구 – Gradle - corretto-17



인텔리제이

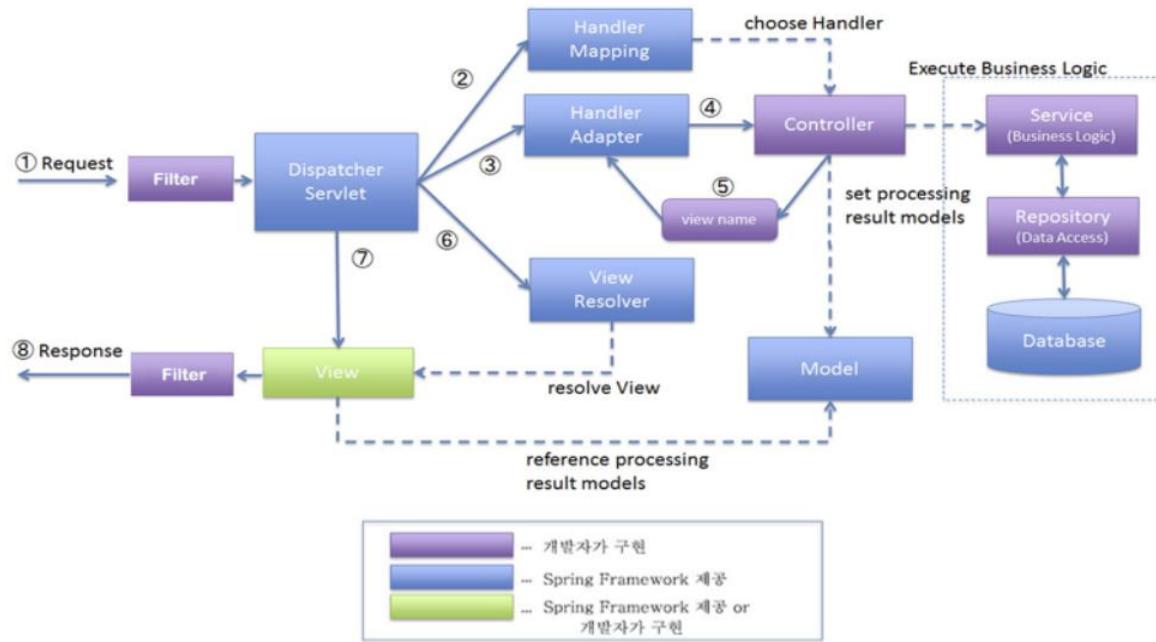
- 모든 프로젝트에 main 함수를 포함하는 클래스가 있습니다. 프로젝트에 있는 main 함수를 가지고 있는 클래스에서 마우스 오른쪽을 누르고 run 을 눌러주세요.



Spring boot MVC

Spring Boot MVC 의 구조

spring MVC 패턴



<https://velog.io/@h220101/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81-%EB%B6%80%ED%8A%B8-spring-MVC-%ED%8C%A8%ED%84%B4-%EB%8F%99%EC%9E%91>

MVC2 모델

- FrontController 패턴

Front Controller 패턴은 웹 애플리케이션에서 모든 요청을 단일 진입점(컨트롤러)으로 처리하는 디자인 패턴입니다. 이 패턴을 사용하면 클라이언트의 모든 요청을 중앙에서 처리하고, 필요한 로직을 통해 적절한 핸들러나 뷰로 분기할 수 있습니다. 이를 통해 코드의 중복을 줄이고, 요청 처리 로직을 중앙 집중화 할 수 있습니다.

MVC2모델 처리절차

1. 클라이언트요청

브라우저 특정 URL에 요청을 DispatcherServlet이 받는다. (내부설정)

2. DispatcherServlet이 URI를 보고 controller를 식별하기위해 핸들러 매핑과 통신한다.

핸들러 매핑은 요청을 처리하는 특정 핸들러메서드를 반환(return)한다.

3. 컨트롤러에 처리요청

DispatcherServlet은 특정 핸들러 메서드를 호출한다. (public String 특정메서드 (Model model)호출)

핸들러 메서드는 모델과 뷰를 반환(return)한다.

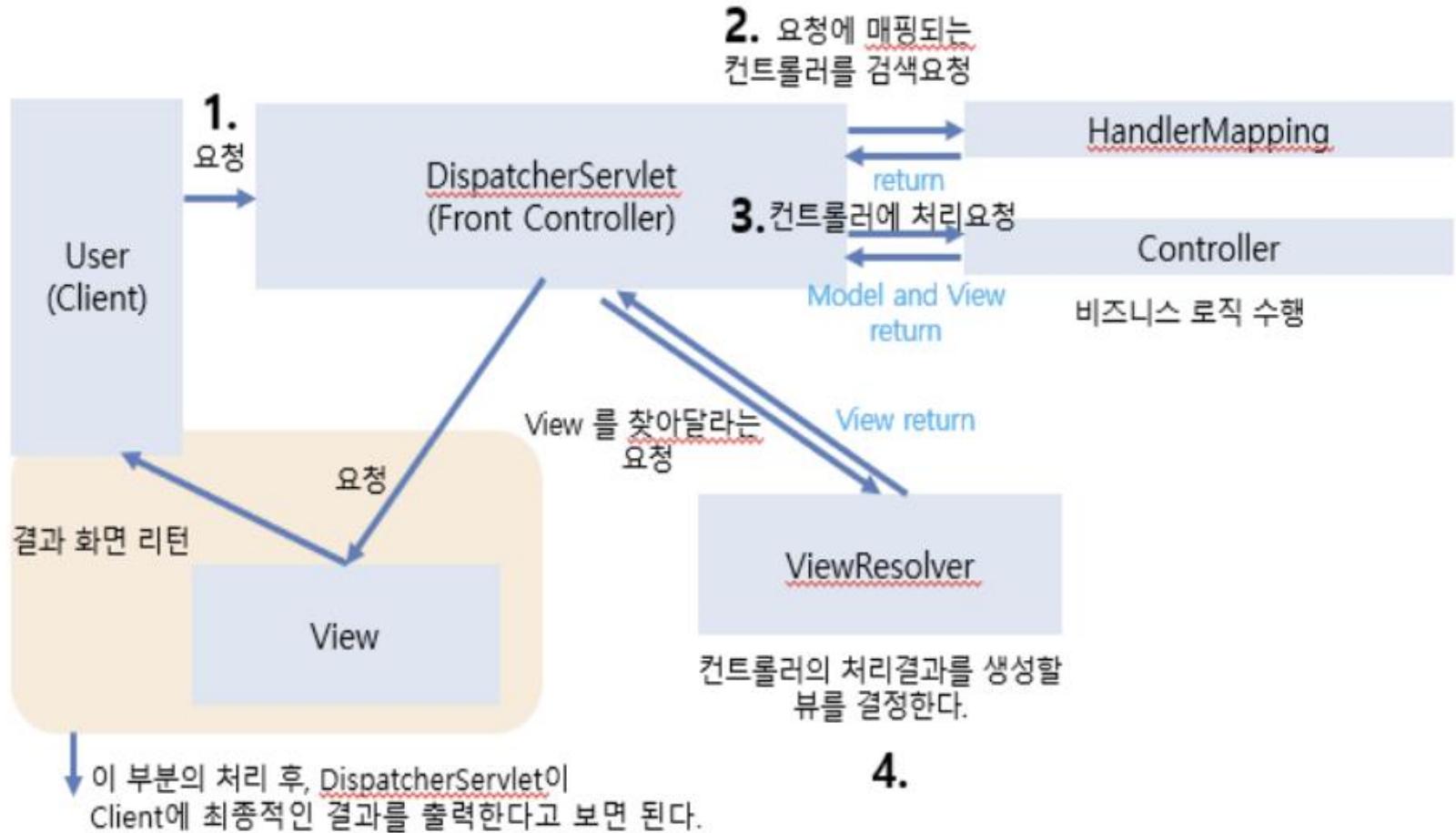
4. 컨트롤러의 처리결과를 생성할 뷰를 결정한다.

DispatcherServlet은 논리적 뷰를 결정하는 뷰 리졸버를 찾아 호출한다.

뷰 리졸버는 논리적 뷰 이름 -> 물리적 뷰 이름에 매핑하는 로직을 실행한다.

(/template/view.html 으로 변환)

Spring Boot MVC 의 구조 2



Spring boot MVC

- Controller
 - HTTP 요청을 처리하고 모델 데이터를 반환하는 클래스입니다.
 - @Controller나 @RestController 어노테이션을 붙여야 스프링 부트가 객체를 생성합니다.
- Service
 - 비즈니스 로직을 구현하는 클래스입니다.
 - @Service 어노테이션을 붙여야 스프링 부트가 객체를 생성합니다.
 - 트랜잭션 처리를 담당합니다.
- Repository
 - 데이터 액세스 로직을 처리하는 클래스입니다.
 - @Repository 어노테이션을 붙여야 스프링 부트가 객체를 생성합니다.
 - 디비와 데이터를 읽고 쓰는 일을 전담합니다.

Spring boot MVC

- Model
 - 애플리케이션에서 사용되는 데이터 객체입니다.
 - @Entity 어노테이션을 사용합니다.
- View
 - 클라이언트에게 응답을 표시하는 템플릿입니다.
 - JSP-스프링 부트에서는 권장하지 않습니다.
 - thymeleaf나, mustache 등이 있습니다만 현재 주류는 thymeleaf 입니다.

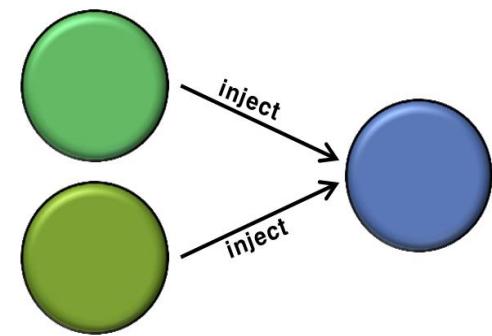
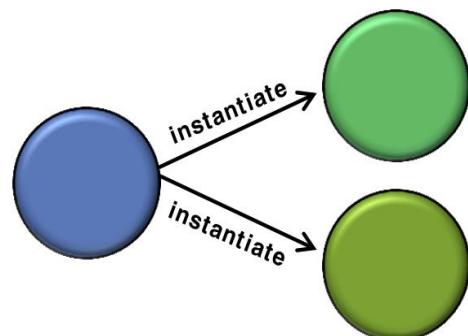
Dependency Injection

- 객체 지향 프로그래밍에서 객체 간의 의존성을 외부에서 주입하는 디자인 패턴입니다. 이 패턴은 객체가 직접 필요한 의존성을 생성하는 대신, 외부에서 주입받음으로써 객체 간의 결합도를 낮추고, 코드의 재사용성과 테스트 용이성을 높여줍니다.
- 객체가 다른 객체를 사용할 때, 그 사용되는 객체를 의존성이라고 합니다. 예를 들어, UserService 클래스가 UserRepository 객체를 사용한다면, UserService는 UserRepository에 의존하고 있다고 말합니다.
- 의존성 주입(Dependency Injection)이란 객체가 사용할 의존성을 외부에서 주입받는 과정입니다. 의존성을 직접 생성하는 것이 아니라, 필요한 의존성을 외부에서 전달받아 사용합니다. 이를 통해 클래스는 자신이 필요로 하는 객체를 주입받게 됩니다.
- 장점
 - **결합도 감소** - 객체가 필요한 의존성을 스스로 생성하는 것이 아니라 외부에서 주입받기 때문에, 객체 간의 결합도가 낮아집니다. 이는 코드를 더 유연하고 확장 가능하게 만듭니다.
 - **유연성 및 재사용성 향상** - 의존성을 외부에서 주입받기 때문에, 동일한 클래스라도 다른 상황에 맞게 다양한 의존성을 주입하여 사용할 수 있습니다.
 - 테스트 용이성- DI를 사용하면 객체가 사용하는 의존성을 Mock 객체로 교체하여 테스트할 수 있습니다. 이는 유닛 테스트 작성에 매우 유리합니다.
 - 유지보수 용이- 코드의 변경이 발생하더라도, 의존성 주입을 사용하면 변화의 영향을 최소화할 수 있습니다. 필요한 의존성만 교체하면 되기 때문입니다.

Dependency Injection

전통적인 객체 생성방식

외부로부터의 주입



출처 : 전자정부프레임워크 교재

Dependency Injection(의존성 주입)

- 필드주입

1. @Autowired

여러개의 클래스가 동일한 인터페이스를 상속받은 경우 문제발생

2. @Resource

```
@Repository("boardDao")
```

```
public class BoardDaolmpl implements BoardDao{
```

```
    @Resource(name="boardDao") //부여된 이름을 기준으로 객체를 찾습니다.
```

```
    BoardDao boardDao;
```

Dependency Injection(의존성 주입)

- 생성자주입

- 생성자 주입은 생성자에 @Autowired 어노테이션을 사용하여 의존성을 주입하는 방식입니다.
- Spring 4.3 이후부터는 생성자가 하나만 있을 경우 @Autowired 어노테이션을 생략할 수 있습니다.
- 변수에는 **final** 키워드를 붙여줍니다.
- 생성자 주입은 불변성을 보장하고 의존성을 명시적으로 선언하여 코드의 가독성과 유지보수성을 높입니다

예시

```
@Service("boardService")  
  
public class BoardServiceImpl implements BoardService{  
  
    // @Autowired 어노테이션을 사용하지 않아도 Spring이 자동으로 주입된다.  
  
    public BoardServiceImpl(BoardDao boardDao) {  
  
        this.boardDao = boardDao;  
  
    }  
  
    .....
```

Dependency Injection(의존성 주입)

- 생성자주입
- lombok을 이용한 방식도 가능함.

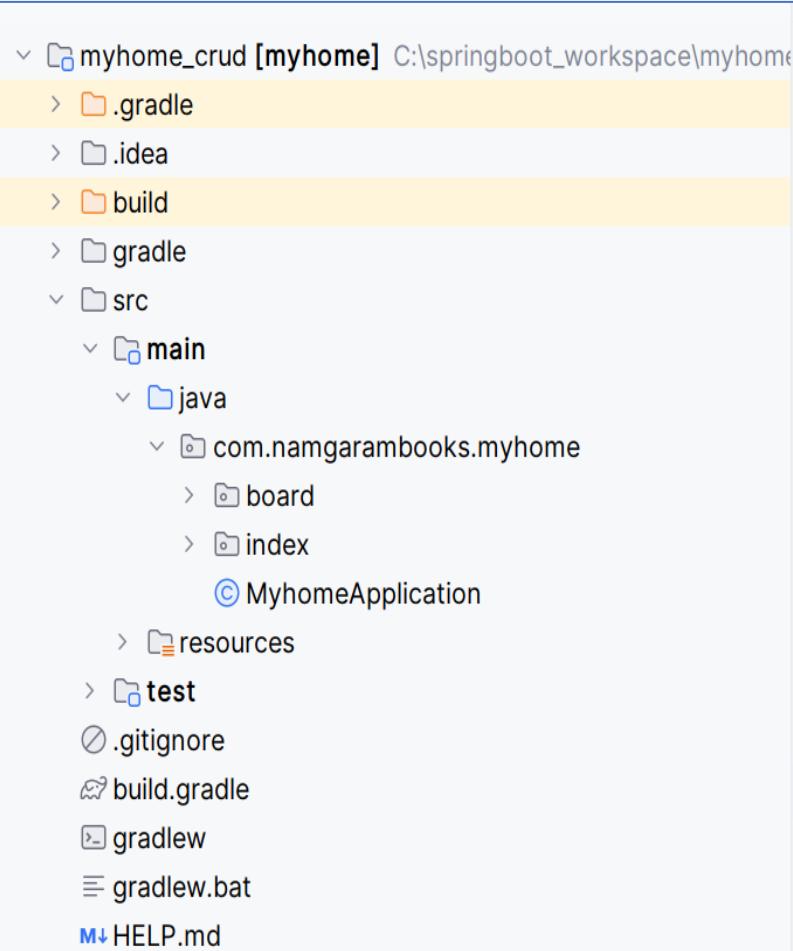
@ RequiredArgsConstructor // 클래스의 final이나 @NonNull로 선언된 필드들만 포함한
생성자를 자동으로 생성합니다.

```
@Service("boardService")
```

```
public class BoardServiceImpl implements BoardService{
```

```
    private final BoardDao boardDao; // 반드시 private final이어야 한다.
```

폴더구조



src/main/java 아래에 모든 java코드를 두어야 한다.

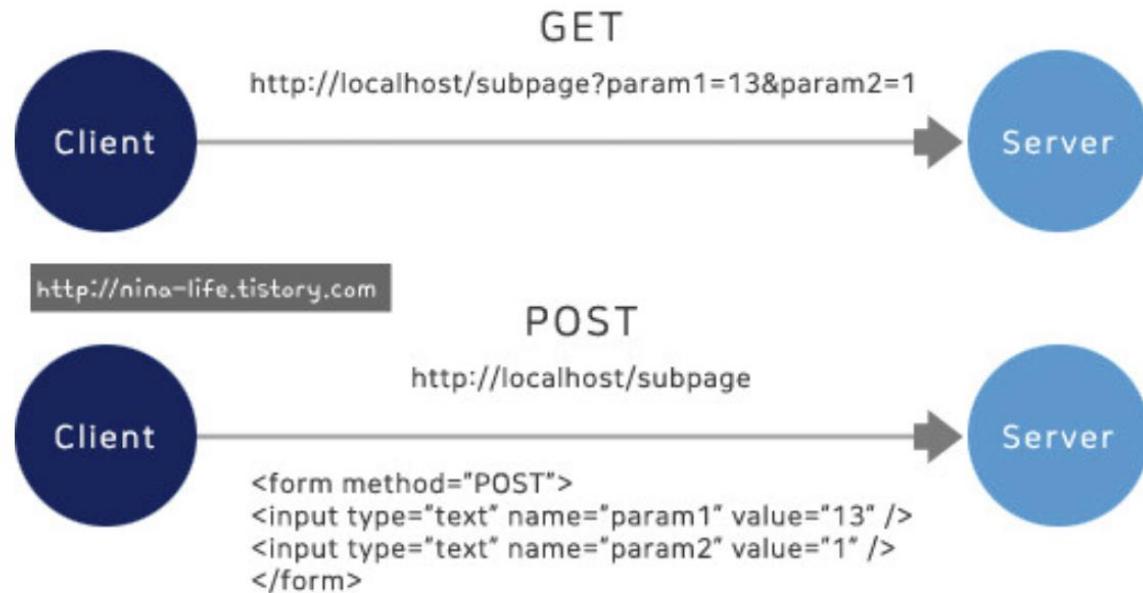
spring boot 는 무조건

프로젝트명Application.java 파일을 만들고 여기서부터 모든 시작이 이뤄진다.

resources 폴더에는 css, js, image, html, xml파일등을 두어야 배포시 문제가 발생하지 않는다.

데이터전송방식

데이터전송방식



출처 : <https://nina-life.tistory.com/20>

Get방식

- GET 요청은 주로 데이터를 서버에서 조회할 때 사용됩니다. URL을 통해 서버에 요청을 보내고, 서버는 요청에 맞는 데이터를 응답으로 반환합니다.
- 특징:
 - URL에 데이터 포함: GET 요청은 전송할 데이터를 쿼리 스트링으로 URL에 포함 시켜 보냅니다.
 - 예를 들어, <https://example.com/search?query=spring&sort=asc> 처럼 요청합니다.
 - 길이 제한: GET 요청에는 URL 길이 제한이 있습니다. 브라우저와 서버마다 다르지만 일반적으로 2,048자 정도로 제한됩니다.

@PathVariable을 이용한 데이터전송

- 경로 변수 기반의 RESTful 데이터 전송 방식입니다.
- PathVariable은 URL 경로의 일부로 사용되고 있고 이는 리소스를 직접적으로 식별하는 데 유리합니다. Get방식은 추가적인 필터링, 페이징, 검색 등을 할 때 주로 사용됩니다.
- POST, PUT, DELETE에서도 사용 가능합니다.

예시) get방식 <http://localhost:9090/getList?pg=1>

예시) 경로방식 <http://localhost:9090/getList/1>

POST 방식

- POST 요청은 서버로 데이터를 전송하거나 서버의 상태를 변경할 때 사용됩니다. 주로 폼 데이터나 파일 업로드 시에 사용됩니다.
- 특징
 - 데이터가 body에 포함됨: POST 요청은 전송할 데이터를 요청 body에 담아 보냅니다. 이로 인해 전송할 데이터의 양에 제한이 없고, GET 방식처럼 URL에 데이터가 노출되지 않습니다.
 - 길이 제한 없음: POST 요청은 body에 데이터를 담기 때문에, GET 요청과 달리 전송 데이터의 크기에 대한 제한이 없습니다.
 - 보안: 데이터가 URL이 아닌 본문에 담기므로 상대적으로 GET 방식보다 안전합니다. 그러나 여전히 HTTPS와 같은 암호화 방법을 사용해야 안전하게 전송됩니다.
 - 서버 상태 변경: POST 요청은 서버의 데이터를 변경하는 작업에 주로 사용됩니다. 예를 들어, 새로운 사용자 등록, 게시물 작성, 데이터베이스에 레코드 추가 등이 이에 해당됩니다.

Get방식 전송 예

- http://localhost:9000/add?x=5&y=8

```
//add?x=5&y=8
@GetMapping("/add")
public HashMap<String, String> add1( int x, int y)
{
    int result = x+y;
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("x", String.valueOf(x));
    map.put("y", String.valueOf(y));
    map.put("result", String.valueOf(result));

    return map;
}
```

@PathVariable을 이용한 데이터전송 예

- http://localhost:9000/add/5/8

```
//add?x=5&y=8 --> add/5/8
@GetMapping("/add/{x}/{y}")
public HashMap<String, String>add2(
    @PathVariable("x") int x,
    @PathVariable("y") int y)
{
    int result = x+y;
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("x", String.valueOf(x));
    map.put("y", String.valueOf(y));
    map.put("result", String.valueOf(result));

    return map;
}
```

POSTMAN 다운로드 및 설치

- <https://www.postman.com/downloads/>
- 구글 계정 필요, 회원가입 필요

The screenshot shows the Postman website with a red box highlighting the 'Download Postman' section. Inside this box, the text '눌러서 다운로드 받는다' (Press to download) is overlaid. A green arrow points from this text to the 'Download the App' button, which is also circled in red. Below this, there's a note about the app being updated every two weeks. The right side of the screenshot shows the Postman application interface with a 'Twitter API v2 / Tweet Lookup / Single Tweet' request in the main pane, and a sidebar showing collections like 'Twitter API v2' and 'Twitter's Public Workspace'.

눌러서 다운로드 받는다

The Postman app

The ever-improving Postman app (a new release every two weeks!) gives you a full-featured Postman experience.

Download the App

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

Version 8.12.4 · [Release Notes](#) · [Product Roadmap](#)

Not your OS? Download for Mac ([macOS](#)) or Linux ([x64](#))

Postman on the web

You can now access Postman through your web browser. Simply create a free Postman account and you're in.

Postman-win64-8.12.4.exe

Sign In Sign Up for Free

Home Workspaces Reports Explore

Twitter API v2 / Tweet Lookup / Single Tweet

GET https://api.twitter.com/2/tweets/:id

Params Authorization Headers Body Pre-request Scripts Tests Settings Cookies

Query params

Key	Value	Description	Actions
tweet.fields		Comma-separated list of fields to expand. Expansions...	Bulk edit Presets
expansions		Comma-separated list of fields for the poll object. Expl...	
media.fields		Comma-separated list of fields for the media object. Expl...	
poll.fields		Comma-separated list of fields for the poll object. Expl...	
place.fields		Comma-separated list of fields for the place object. Expl...	
user.fields		Comma-separated list of fields for the user object. Expl...	

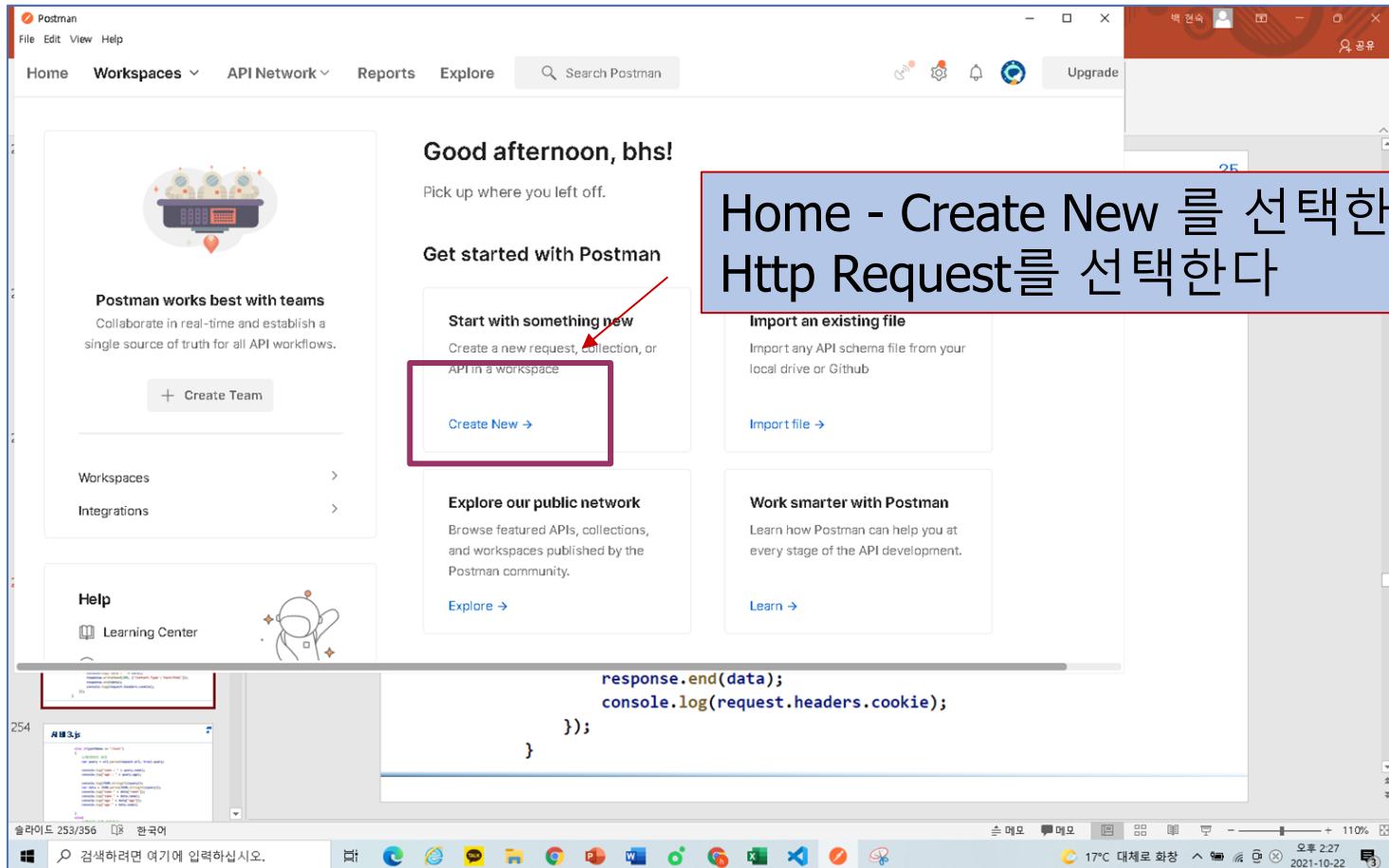
Path Variables

Key	Value	Description	Actions
id	1403216129661628420	Required. Enter a single Tweet ID.	Bulk edit Presets

Body Cookies Headers Test Results 200 OK 468 ms 734 B Save Response

모두 표시

설치 후



POST 방식

- GET방식은 데이터 전송시 header만 보낸다. 비교적 간단한 정보만 보낸다.
- POST방식은 데이터를 전송할 때 우선 header를 보내고 body를 보낸다.

1. form-data : 파일 업로드시 form태그에 enctype="multipart/form-data" 속성을 추가해서 보내야 한다.

2. x-www-form-urlencoded : 일반적인 POST 방식을 말한다.

3. raw : JSON방식으로 전송할때 사용한다.

(axios라이브러리, 자바스크립트 fetch 함수)

add_post(X-www-form-urlencoded)

```
@PostMapping("/add_post")
public HashMap<String, String>add_post(int x, int y)
{
    int result = x+y;
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("x", String.valueOf(x));
    map.put("y", String.valueOf(y));
    map.put("result", String.valueOf(result));

    return map;
}
```

POST 전송 1

The screenshot shows the Postman application interface. At the top, it displays the URL `POST http://localhost:9000`. Below the header, the request method is set to `POST` and the endpoint is `http://localhost:9000/add_post`. The `Body` tab is selected, showing the following parameters:

Key	Value
x	5
y	8

Below the body parameters, the response is displayed in `Pretty` format:

```
1   "result": "13",
2   "x": "5",
3   "y": "8"
```

add_form 함수(파일업로드시)

```
//postman body - x-www-form-urlencoded <form 태그로 전송
@PostMapping("/add_form")
public HashMap<String, String> add_form(int x, int y)
{
    int result = x+y;
    HashMap<String, String> map = new HashMap<String, String>();
    //form 태그에 enctype="multipart/form-data" 로 전송하면
    //request 객체로 값을 못받고 MultipartResolver 의해 값을 처리
    //해야 한다. 스프링부트가 사고 안나게 내부적인 처리가 된다.
    //map.put("req_x", req.getParameter("x")); //서블릿에서 작업함
    //map.put("req_y", req.getParameter("y"));

    //String.valueOf(기본타입변수) -> 문자열로 전환해준다
    //x(int) => int 는 객체가 아니다. toString() 함수 없음
    //java int => 객체로 전환해야 할때가 있는데 Integer, Float, Double
    //wrapper class 들로 기본값을 감싸서 객체로 전환할때는 toString()
    //함수가 있다. 그냥 기본값 자체로 개체로 전환되지 않는다
    //c#은 int와 Integer 간에 기본타입과 객체타입간에 전환이 자유자재임
    // s = new Integer(x); s.toString();
    // x + "" : 앞에 스트링이 아닌 타입을 스트링으로 전환시켜서 더한다
    map.put("x", String.valueOf(x));
    map.put("y", String.valueOf(y));

    map.put("result", String.valueOf(result));

    return map;
}
```

POST 전송 2(form-data)

- 파일 전송시 사용한다.

The screenshot shows the Postman application interface. At the top, it displays the URL `http://localhost:9000/add_post`. Below the URL, the method is set to `POST`. The `Body` tab is selected, showing the following form-data:

Key	Value
x	5
y	8

Below the body, the `Body` tab is selected again, showing the JSON response:

```
1 {  
2   "result": "13",  
3   "x": "5",  
4   "y": "8"  
5 }
```

add_json

```
@PostMapping("/add_json")
public HashMap<String, String> add_json(@RequestBody HashMap<String, String> paramMap)
{
    //json으로 받을때는 @RequestBody로 받아야 한다.
    //기본타입변수로는 못 받는다
    //HashMap 이나 또는 Dto로만 받을 수 있다
    int result = Integer.parseInt(paramMap.get("x"))
        +Integer.parseInt(paramMap.get("y"));

    HashMap<String, String> map = new HashMap<String, String>();
    map.put("x", paramMap.get("x"));
    map.put("y", paramMap.get("x"));
    map.put("result", String.valueOf(result));

    return map;
}
```

json형식으로
받으려면 반드시
@RequestBody
어노테이션을 주어야
한다.
HashMap, Dto
형태로 전환되어
받는다.

POST 3(json형식)

- body – raw – json : 키값은 반드시 ""로 감싸야 한다.

The screenshot shows the Postman application interface for a POST request to `http://localhost:9000/add_json`.

Request Headers:

- Method: POST
- URL: http://localhost:9000/add_json
- Headers (9): (This section is visible but empty in the screenshot)
- Body (green dot): Selected
- Pre-request Script: (None selected)
- Tests: (None selected)
- Settings: (None selected)

Request Body (raw JSON):

```
1 {  
2   "x": 5,  
3   "y": 8  
4 }
```

Response Body (Pretty):

```
1 {  
2   "result": "13",  
3   "x": "5",  
4   "y": "5"  
5 }
```

문제 1

1. 다음과 같은 결과를 가져오기 위한 함수를 작성하시오.

url : score_json

전송데이터

```
{  
    "name": "홍길동",  
    "kor": 90,  
    "eng": 80,  
    "mat": 70  
}
```

결과

{

```
    "name": "홍길동",  
    "kor": 90,  
    "eng": 80,  
    "mat": 70,  
    "total": 240,  
    "avg": 80  
}
```

문제 1

The screenshot shows the Postman application interface. At the top, it displays a POST request to the URL `http://localhost:9000/score_json`. Below the URL, there are tabs for Params, Authorization, Headers (9), Body (selected), Pre-request Script, Tests, and Settings. Under the Body tab, there are options for none, form-data, x-www-form-urlencoded, raw (selected), binary, and GraphQL, with a dropdown menu set to JSON.

The raw body content is:

```
1 {
2   "name": "홍길동",
3   "kor": 90,
4   "eng": 80,
5   "mat": 70
6 }
7
```

Below the raw body, there are tabs for Body (selected), Cookies, Headers (5), and Test Results. Under the Body tab, there are buttons for Pretty (selected), Raw, Preview, Visualize, and a dropdown set to JSON. The Pretty tab shows the JSON response:

```
1 {
2   "mat": "홍길동",
3   "total": 240,
4   "avg": 80,
5   "name": "홍길동",
6   "kor": 90,
7   "eng": 80
8 }
```

문제 2

2. 다음과 같은 결과를 가져오기 위한 함수를 작성하시오.

url : trade_json

data :

```
{  
  "trade_id": "100",  
  "trade": [  
    {"payment": 100},  
    {"payment": 200},  
    {"payment": 150},  
    {"payment": 2700},  
    {"payment": 190}  
  ]  
}
```

```
{  
  "trade_id": "100",  
  "trade": [  
    {"payment": 100},  
    {"payment": 200},  
    {"payment": 150},  
    {"payment": 2700},  
    {"payment": 190}  
  ],  
  "sum": 3340  
}
```

머스티치

html 뷰(머스티치)

- html 뷰를 지원하라면 반드시 아래 코드를 build.gradle에 넣어야 한다.
- spring boot 에서는 jsp 엔진을 지원하지 않는다.
- 패키징을 할 때 war형식에서 jar형식으로 변환되면서 jsp의 사용을 차단했음.
- 대신 mustache 나 thymeleaf를 사용한다.
- 두 엔진 중 뭐가 더 나은지에 대해서는 특별한 권장사항이 없고 보통 스타트업 회사들이 많이 사용한다 현재는 thymeleaf가 압독적임.
- 둘다 알아두는 것이 좋다.

html 뷰(머스티치)

- 머스티치 지원 라이브러리를 build.gradle에 추가해야 한다.

<https://mvnrepository.com/>에서 spring mustache를 검색한다.

[**Spring Boot Starter Mustache**](#)를 검색한다.

html 뷰(머스티치)

 **Spring Boot Starter Mustache**

Starter for building web applications using Mustache views

License	Apache 2.0
Tags	mustache spring framework starter
Ranking	#19254 in MvnRepository (See Top Artifacts)
Used By	19 artifacts

Central (183) Spring Plugins (22) Spring Lib M (5) Spring Milestones (41)
Evolveum (1) Kyligence Public (2)

Version	Vulnerabilities	Repository	Usages	Date
3.1.4		Central	0	Sep 21, 2023
3.1.3		Central	1	Aug 24, 2023
3.1.x		Central	4	Jul 20, 2023
3.1.2				

적당한 버전을 선택한다.
버전이 맞지 않을 경우에는 다른 버전으로 바꿔본다.

html 뷰(머스티치)

- gradle 이나 gradle (short)탭을 선택한다

Used By 19 artifacts

Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen

Buildr

```
// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-mustache  
implementation 'org.springframework.boot:spring-boot-starter-mustache:3.1.4'
```

html 뷰(머스티치)

- build.gradle 파일을 열어

implementation 'org.springframework.boot:spring-boot-starter-mustache:3.1.4' 를
복사 붙여넣기 한다.

```
dependencies {  
    // https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-mustache  
    implementation 'org.springframework.boot:spring-boot-starter-mustache'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-webflux'  
    compileOnly 'org.projectlombok:lombok'  
    developmentOnly 'org.springframework.boot:spring-boot-devtools'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    testImplementation 'io.projectreactor:reactor-test'  
}
```

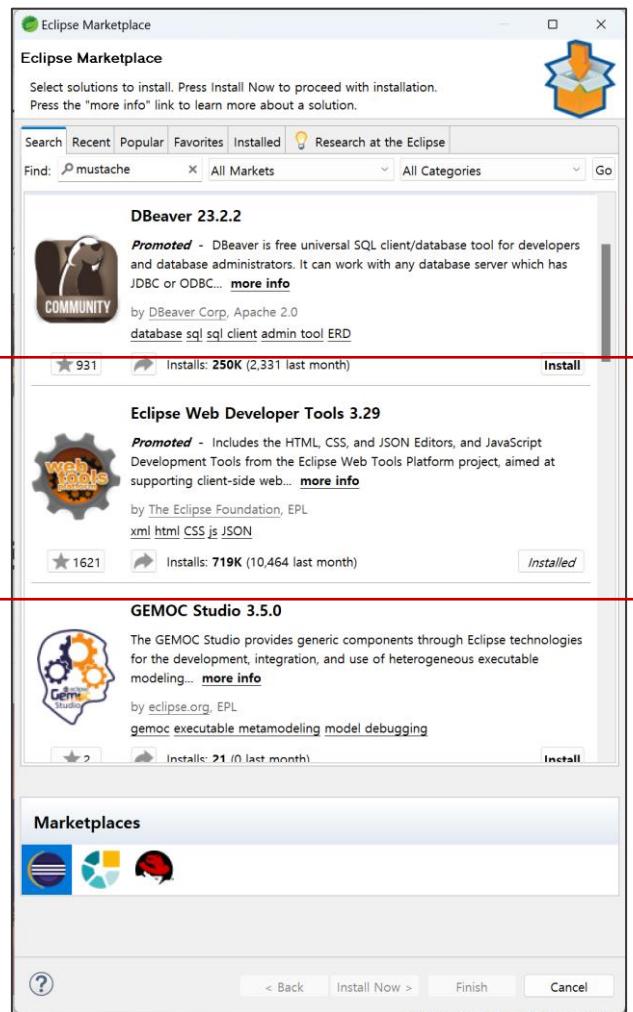
html 뷰(머스티치)

- src/main/resources/application.properties 파일에 아래처럼 기술해야 한다. 이부분에 대한 설정을 안하면 html 문서를 인식하지 못한다. 파일의 확장자를 모두 mustache로 바꿔야 한다.
- spring.mustache.suffix: .html
- server.servlet.encoding.force=true #한글 안깨지게

```
server.port=9090
#파일 확장자가 .mustache 로 해야 엔진이 이해를 한다.
#test.html을 쓰면 이걸 mustache 로 해석해라
#키와 값 사이에 공백 안된다.
spring.mustache.suffix=.html
#spring frame work <filter ~~ 대신에 사용
#한글 안깨지게 설정한다
server.servlet.encoding.force=true

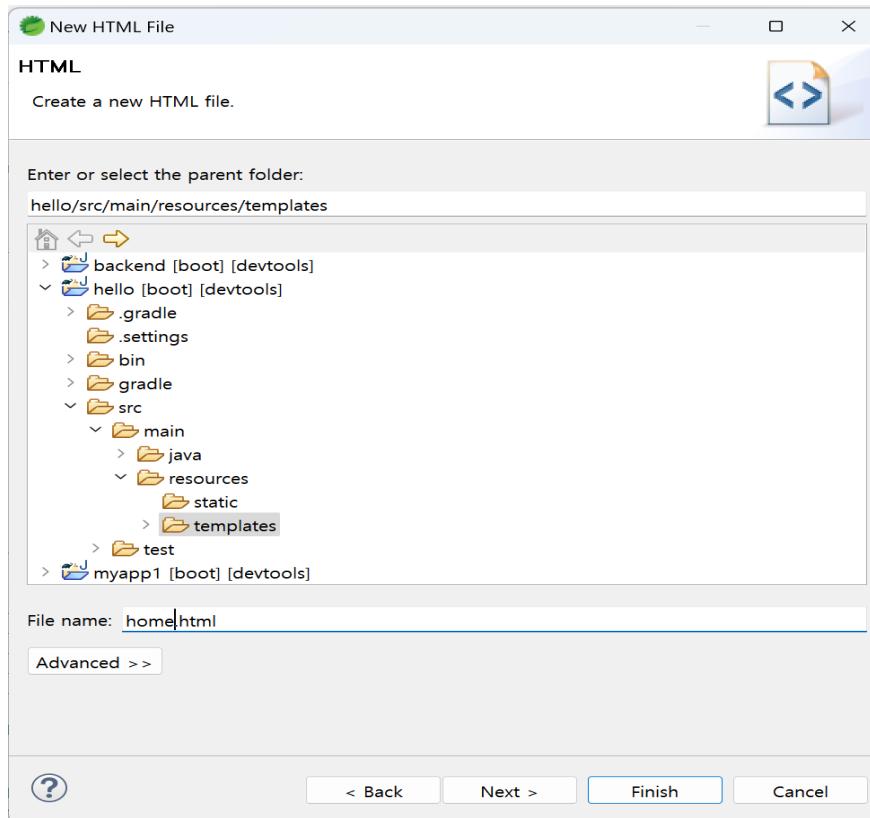
#세션값을 mustache 엔진에서 표출되도록 설정해야 한다
spring.mustache.servlet.expose-session-attributes=true
```

help – marketplace – web developer 설치



home.html

- src/main/resources/templates 마우스오른쪽 – new – other – web – html – home.html 추가



템플릿1

- home.html

```
<!DOCTYPE html>
› <html>
› <head>
<meta charset="utf-8">
<title>Insert title here</title>
</head>
› <body>
    <h1>템플릿 테스트 입니다</h1>
</body>
</html>
```

템플릿1

- MustacheController.java 추가한다.
- @RestController 가 아니라 @Controller 이어야 한다.

```
// 템플릿과 연결한다. Controller 를 사용해야 한다
@Controller
public class MustacheController {

    @GetMapping("/home")
    public String home(Model model, HttpServletRequest req)
    {
        //src/main/home 폴더로 찾아간다
        return "home"; //home.html로 찾아간다
    }
}
```

템플릿2

- template1.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>{ ${msg} }</h1>
<h3>{ ${name} }</h3>
<h3>{ ${age} }</h3>
</body>
</html>
```

템플릿2

- 반복문

```
<ul>  
{{#sList}}  
  <li>{{.}}</li>  
{{/sList}}  
</ul>
```

템플릿3

```
<h1>반복문2</h1>

<ul>

{ {{#boardList}} }

<li>{{id}} {{title}} {{writer}} {{contents}}</li>

{ {{/boardList}} }

</ul>
```

템플릿4

- 세션값 처리 (application.properties)

```
spring.mustache.servlet.expose-session-attributes=true
```

```
<h1>if문</h1>
<h1>
{{#userid}} <!-- 키값이 존재할때 -->
{{username}} 님 안녕하세요
{{/userid}} <!-- 키값이 존재할때 -->

{{^userid}} <!-- 키값이 존재안할때 -->
로그온하세요
{{/userid}} <!-- 키값이 존재안할때 -->
</h1>
```

파일 포함

- include

```
{ {>layout/header} }
```

```
{ {>layout/nav} }
```

<h1>게시판</h1>

```
{ {>/layout/footer} }
```

layout

- templates/layout/header.html
- templates/layout/nav.html
- templates/layout/footer.html

thymeleaf

thymeleaf란

- 타임리프(Thymeleaf)는 View Template Engine으로 JSP, Freemarker와 같이 서버에서 클라이언트에게 응답할 브라우저 화면을 만들어주는 역할을 합니다.
- html뷰어에서도 확인이 가능하고 보안도 지원하고 있어서 최근에 많이 사용중입니다.

thymeleaf 의 목적

- 타임리프의 주 목표는 템플릿을 만들 때 유지관리가 쉽도록 하는 것입니다.
- 이를 위해 디자인 프로토로타입으로 사용되는 템플릿에 영향을 미치지 않는 방식인 Natural Templates을 기반으로 합니다.
- Natural Templates은 기존 HTML 코드와 구조를 변경하지 않고 덧붙이는 방식입니다.

장점

- 코드를 변경하지 않기 때문에 디자인 팀과 개발 팀간의 협업이 편해집니다.
- JSP와 달리 Servlet Code로 변환되지 않기 때문에 비즈니스 로직과 분리되어 오로지 View에 집중할 수 있습니다.
- 서버상에서 동작하지 않아도 되기 때문에 서버 동작 없이 화면을 확인할 수 있습니다. 때문에 더미 데이터를 넣고 화면 디자인 및 테스트에 용이합니다.

필요라이브러리

build.gradle 의 dependencies 에 다음 의존성을 추가합니다.

```
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
```

application.properties

```
#spring framework <filter ~~ 대신에 사용
```

```
#한글 안깨지게 설정한다
```

```
server.servlet.encoding.force=true
```

```
#thymeleaf의 파일이 있는 위치를 지정합니다.
```

```
spring.thymeleaf.prefix=classpath:/templates/
```

```
spring.thymeleaf.suffix=.html
```

머시티치나 타임리프 둘중 하나를 사용해야 합니다.

html 파일

- 문서의 맨위쪽에 네임스페이스를 추가해야 합니다.

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

를 먼저 기술해야 합니다.

사용법

- **th:text**

태그 안의 텍스트를 서버에서 전달 받은 값에 따라 표현하고자 할 때 사용된다. 태그를 텍스트로 표시한다.

```
<span th:text="${msg}">message</span>
```

```
<p style='color:red;'>스프링부트 템플릿 엔진</p>
```

← 전달된 문자열이 그대로 보인다.

- **th:utext**

변수에서 받은 값에서 html태그가 있다면 태그값을 적용해서 표시해준다

```
<span th:utext="${msg}">message</span>
```

```
스프링부트 템플릿 엔진
```

← 태그가 적용되어서 보인다.

사용법

- **th:value**

value속성을 갖는 태그들에 값을 부여할때 사용한다. input 태그등에 사용한다.

```
<input type="text" name="name" th:value="${name}" />
```

- **th:with**

변수에 값을 지정해서 사용할 수 있다.

temp라는 변수를 만들고 변수에 값을 넣은 후 그 값을 다시 출력한다.

```
<h3 th:with="temp=${name}" th:text="${temp}">temp
```

사용법

- **th:switch**

Switch-case문이 필요할 때 사용한다.

th:case에서 case문을 다루고 *로 case문에서 다루지 않은 모든 경우가 처리된다.

- **th:if**

조건문이 필요할 때 사용한다. else문이 필요한 경우에는 th:unless를 사용한다.

- **th:each**

반복문이 필요한 경우에 사용한다.

리스트와 같은 collection 자료형을 서버에서 넘겨주면 그에 맞춰 반복적인 작업이 이루어질 때 사용한다.

사용법

- 세션의 경우에는 \${session.키값} 형태로 사용해야 한다

```
<h1>if문</h1>
└<h1 th:if="${userid} != ''">
    <span th:text="${session.username}">○○</span>님 안녕하세요
</h1>
└<h1 th:unless="${true}">
    로그온하세요
</h1>
```

레이아웃이란

- 웹 애플리케이션의 UI 구조를 재사용하고 일관성 있는 디자인을 유지하기 위해 템플릿을 분리하고 병합하는 기능입니다. 이를 통해 여러 페이지에서 공통적으로 사용되는 부분(예: 헤더, 푸터, 네비게이션 바 등)을 하나의 레이아웃 템플릿으로 정의한 후, 각 페이지에서 이를 참조하여 확장할 수 있습니다.

레이아웃

- **th:block**를 이용한다. **th:block** 은 주로 HTML 문서에서 특정 블록을 감싸거나 여러 요소를 묶어 처리할 때 사용됩니다. if문등으로 블록을 정해서 처리하고자 할때 많이 사용합니다.
- header라는 이름의 fragment를 사용한다.
- /layout2/header.html

```
<th:block th:fragment="header">  
<title>Bootstrap 5 Example</title>  
<meta charset="utf-8">  
<meta name="viewport" content="width=device-width, initial-scale=1">  
<link  
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"  
    rel="stylesheet">  
<script  
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js">  
</script>  
</th:block>
```

레이아웃

layout2/nav.html

```
<th:block th:fragment=nav>

<nav class="navbar navbar-expand-sm bg-light">
    <div class="container-fluid">
        <!-- Links -->
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" href="#">Link 1</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link 2</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link 3</a>
            </li>
        </ul>
    </div>
</nav>
</th:block>
```

레이아웃

```
<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
<meta charset="UTF-8">
<title>layout</title>
    <th:bloc th:replace="layout2/header::header"></th:bloc>
</head>
<body>
    <th:bloc th:replace="layout2/nav::nav"></th:bloc>
    <th:bloc layout:fragment="content">
        <h1>게시판</h1>
    </th:bloc>
    <th:bloc th:replace="layout2/footer::footer"></th:bloc>
</body>
</html>
```

룸복

롬복이란

- 스프링 룸복(Spring Lombok)은 자바 프로그래밍을 편리하게 만들어 주는 오픈소스 라이브러리입니다. 주로 스프링 프레임워크와 함께 사용되며, 반복적이고 장황한 코드를 줄여주는 기능을 제공합니다. Lombok을 사용하면 불필요한 보일러플레이트 코드 등을 제거할 수 있습니다.
- "보일러플레이트 코드(boilerplate code)"는 프로그래밍에서 반복적으로 발생하지만, 핵심 로직과는 직접적인 관련이 없는 코드를 말합니다. 이 코드들은 주로 언어나 프레임워크의 요구 사항을 충족하기 위해 작성되며, 개발자가 반복해서 작성해야 하는 일상적이고 지루한 작업을 포함합니다. 예를 들어, 클래스의 필드를 정의하고, 게터(getter)와 세터(setter)를 작성하고, 생성자를 만드는 등의 작업이 보일러플레이트 코드에 해당합니다.

롬복이 제공하는 어노테이션

- **@Getter/@Setter**

각 필드에 대한 getter와 setter 메서드를 자동으로 생성합니다.

- **@ToString**

객체의 `toString()` 메서드를 자동으로 생성합니다.

- **@NoArgsConstructor/@AllArgsConstructor/@RequiredArgsConstructor**

기본 생성자, 모든 필드에 대한 생성자, 또는 필수(`final` 또는 `@NonNull`) 필드에 대한 생성자를 자동으로 생성합니다.

- **@Data**

`@Getter`, `@Setter`, `@ToString`, `@EqualsAndHashCode`, `@RequiredArgsConstructor`를 포함한 종합적인 어노테이션입니다.

- **@Builder**

빌더 패턴을 쉽게 사용할 수 있도록 지원하는 어노테이션입니다. 보통 `AllArgsConstructor`과 많이 사용됩니다.

빌더 패턴

- 빌더 패턴(Builder Pattern)은 객체 생성의 복잡성을 줄이고, 가독성을 높이기 위해 사용합니다. 또한 객체를 생성하는 방법을 다양하고 유연하게 할 수 있게 하는 디자인 패턴입니다. 이 패턴은 매개변수가 많은데 모든 값을 입력할 필요가 없을 때나 다양한 조합의 매개변수를 필요로 하는 객체를 생성할 때 유용합니다.
- 주요 개념
 1. 객체 생성의 분리 - 객체의 생성과 표현을 분리하여 동일한 객체 생성 과정에서도 서로 다른 표현을 만들 수 있습니다.
 2. 메서드 체인 - 객체의 속성을 설정하는 메서드를 체인 형식으로 호출할 수 있어, 코드의 가독성을 높입니다.
 3. 불변성 유지 - 빌더 패턴을 통해 생성된 객체는 보통 불변 객체(immutable object)로 설계되어, 객체의 상태가 한 번 설정된 이후 변경되지 않도록 합니다.

빌더패턴 예시

- 생성자를 이용한 객체 생성

```
BoardDto dto = new BoardDto(1, "제목1", "내용1", "작성자1",
```

매개변수가 많을 경우에 객체 생성시 정확한 데이터 타입과 생성자에서의
매개변수들의 순서를 알고 있어야 한다.

- 빌드패턴을 이용한 객체생성**

```
BoardDto dto = BoardDto.builder()  
    .title("제목1")  
    .contents("내용1")  
    .build()
```

많은 매개변수를 가지는 객체를 생성할 때 매우 유용하며, 코드dml 가독성 및 유지
보수성을 높여준다 .

롬복설치하기

- **롬복 다운로드**
- **<https://projectlombok.org/download>**
- 이클립스(Eclipse)가 설치된 경로에 **lombok.jar** 파일을 추가하고, jar를 실행해 주세요.
- cmd 창을 관리자 권한으로 켜 후
- **java -jar lombok.jar** 로 실행하기

롬복설치하기

Project Lombok v1.18.24 - Installer



Javac (and tools that invoke javac such as *ant* and *maven*)
Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.
Example: `javac -cp lombok.jar MyCode.java`

IDEs
Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.

[Show me what this installer will do to my IDE installation.](#)
[Uninstall lombok from selected IDE installations.](#)

<https://projectlombok.org> v1.18.24 [View full changelog](#)

롬복설치하기

Project Lombok v1.18.24 - Installer



Javac (and tools that invoke javac such as *ant* and *maven*)
Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.
Example: `javac -cp lombok.jar MyCode.java`

IDEs
Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.
 C:\SpringBoot\sts-4.15.1.RELEASE\SpringToolSuite4.exe

Specify location...

[Show me what this installer will do to my IDE installation.](#)

<https://projectlombok.org> v1.18.24 [View full changelog](#)

롬복설치하기

- 이클립스를 다시 켠다.
- 기존 프로젝트일 경우에 파일들을 다시 컴파일 한다.
- 인텔리제이의 경우에는 Lombok 플러그인을 자동으로 인식합니다.

CRUD

CRUD

- Controller
 - url을 받아서 처리한다, 여러개의 서비스를 소유한다
 - @Controller 또는 @RestController
- Service
 - Service 클래스, 여러개의 Dao를 소유할 수 있고, 트랜잭션 처리를 담당한다.
 - @Service가 지정되어야 한다.
 - 인터페이스를 만들어야 한다.
- Dao
 - Data Access Object (데이터베이스와 직접 접촉하여 데이터 액세스를 한다.
 - @Repository(객체명)
 - 인터페이스를 만들어야 한다. mapper를 사용하거나 jpa 를 사용할 수 있다.
- Dto
 - Data Transfer Object(보통 테이블에 하나씩 Dto를 만든다. 각 필드의 값을 저장할 클래스 임, join해서 가져올 경우에는 두개 이상의 테이블 필드를 가질 수 도 있다)

전체 파일

- html 파일
 - board_list.html
 - board_write.html
 - board_view.html
- 클래스파일
 - BoardDto.java
 - BoardDao.java -interface
 - BoardDaolmpl - BoardDao 인터페이스 구현
 - BoardService.java - interface
 - BoardServiceImpl.java - BoardService 인터페이스 구현
 - BoardController.java

Dto와 Vo

- *DTO* (Data Transfer Object)
 - 데이터를 전송하기 위한 객체
 - 데이터를 주고받을 때 사용 (주로 계층 간)
 - **가변 객체** (*setter*로 값 변경 가능)
 - 비즈니스 로직이 없음
- *VO* (Value Object)
 - 특정 값을 표현하기 위한 객체
 - 불변 객체생성시 필요 (한 번 생성된 후 값이 변경되지 않음)
 - 내부 값으로 동등성 비교 (값이 같으면 동일한 객체로 취급)
 - 간단히, DTO는 데이터를 주고받는 데 사용되며, VO는 값을 표현하고 동등성 비교에 초점을 둡니다.
 - DTO와 반대로 로직을 포함할 수 있으며, VO의 경우 특정 값 자체를 표현하기 때문에 불변성의 보장을 위해 생성자를 사용하여야 한다.

BoardDto.java

- 룸복을 이용해서 클래스를 만들어보자

```
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class BoardDto extends BaseDto{  
    private String seq="";  
    private String title="";  
    private String writer="";  
    private String contents="";  
    private String filename="";  
    private String image_url="";  
    private String wdate="";  
    private String rnum="";  
    private String hit="0";  
}
```

Dao(Data Access Object)

- DAO(Data Access Object)는 데이터베이스와 상호작용하는 코드를 캡슐화하여 데이터베이스 접근을 보다 구조화하고 체계적으로 관리하기 위한 디자인 패턴입니다. DAO는 데이터베이스의 데이터에 접근하고, 이를 가져오거나 수정하는 등의 작업을 전담하는 역할을 수행합니다.
- DAO 패턴을 사용하면, 비즈니스 로직과 데이터 접근 로직을 분리하여 코드의 유지보수성과 재사용성을 높일 수 있습니다. 즉, 데이터베이스와의 상호작용을 쉽게 관리하고 비즈니스 로직과 데이터 접근 코드를 독립적으로 관리할 수 있게 해줍니다.

BoardDao.java

- 객체간의 결합력 약화와 응집력 강화를 위해서 인터페이스를 작성한다.

```
package com.mycompany.myhome1.board;

import java.util.List;

public interface BoardDao {
    List<BoardDto> getList();
    BoardDto getView(String id);
    void insert(BoardDto dto);
    void update(BoardDto dto);
    void delete(String id);
}
```

BoardDaoImpl.java

```
@Repository("boardDao")
public class BoardDaoImpl implements BoardDao{

    List<BoardDto> list = new ArrayList<BoardDto>();

    public BoardDaoImpl()
    {
        for(int i=1; i<=10; i++)
        {
            list.add( new BoardDto(""+i,"제목"+i, "내용"+i, "작성자"+i, "작성일"+i));
        }
    }

    @Override
    public List<BoardDto> getList() {
        return list;
    }

    @Override
    public void insert(BoardDto dto) {
        list.add(dto);
    }

    @Override
    public BoardDto getView(String id) {

        int nID= Integer.parseInt(id);
        return list.get(nID);
    }

    @Override
    public void update(BoardDto dto) { }

    @Override
    public void delete(String id) { }

}
```

BoardService.java

- Contoller 와 Dao를 연결한다. 여러개의 Dao를 소유할 수 있다 .
- 비지니스로직과 트랜잭션 처리를 진행해야 한다

```
package com.mycompany.myhome1.board;

import java.util.List;

public interface BoardService {
    List<BoardDto> getList();
    void insert(BoardDto dto);
    BoardDto getView(String id);
    void update(BoardDto dto);
    void delete(String id);
}
```

BoardServiceImpl

```
@Service("boardService")
public class BoardServiceImpl implements BoardService{

    @Resource(name="boardDao") //boardDao라는 아이디를 갖는 객체를 찾아서 참조를 전달한다
    BoardDao boardDao;

    @Override
    public List<BoardDto> getList() {
        // TODO Auto-generated method stub
        return boardDao.getList();
    }

    @Override
    public void insert(BoardDto dto) {
        boardDao.insert(dto);
    }

    @Override
    public BoardDto getView(String id) {

        return boardDao.getView(id);
    }

    @Override
    public void update(BoardDto dto) {
        boardDao.update(dto);
    }

    @Override
    public void delete(String id) {
        boardDao.delete(id);
    }
}
```

BoardController.java

```
@Controller
public class BoardController {

    // @Autowired - 특정객체를 고를 수 없어서 그냥 타입이 맞으면 시스템에 알아서 객체를 주입시켰음
    //           동일한 인터페이스를 상속받았을때 ......

    @Resource(name="boardService")
    BoardService boardService;

    @RequestMapping(value="/board/list")
    public String board_list(Model model)
    {
        List<BoardDto> list = boardService.getList();

        model.addAttribute("boardList", list); // 데이터 읽어와서 request 객체에 저장 - Model 클래스

        return "/board/list";
    }

    @RequestMapping(value="/board/view")
    public String board_view(Model model, String id, HttpServletRequest request, BoardDto dto )
    {
        //파라미터의 이름이 중요하지 순서는 전혀 상관없다.
        String iid = request.getParameter("id");

        System.out.println("id(spring) : " + id);
        System.out.println("iid(old) : " + iid);

        System.out.println("id(dto) : " + dto.getId());

        model.addAttribute("boardDto", boardService.getView(id));
        return "/board/view";
    }
}
```

BoardController.java

```
//write.jsp 로 이동하기
@RequestMapping(value="/board/write")
public String board_write(Model model, String id, HttpServletRequest request, BoardDto dto )
{
    model.addAttribute("boardDto", new BoardDto());

    return "/board/write";
}

@RequestMapping(value="/board/save")
public String board_save(Model model, String id, HttpServletRequest request, BoardDto dto )
{
    boardService.insert(dto);
    return "redirect:/board/list";
}

@RequestMapping(value="/board/modify")
public String board_modify(Model model, String id, BoardDto dto )
{
    model.addAttribute("boardDto", boardService.getView(id));

    return "/board/write";
}

@RequestMapping(value="/board/update")
public String board_update(Model model, BoardDto dto )
{
    boardService.update(dto);
    return "redirect:/board/list";
}

@RequestMapping(value="/board/delete")
public String board_delete(Model model, BoardDto dto, String id )
{
    boardService.delete(dto.getId());
    return "redirect:/board/list";
}
```

JPA

ORM 이란

- ORM(Object-Relational Mapping)은 객체 지향 프로그래밍 언어(예: Java, Python)에서 사용하는 객체와 관계형 데이터베이스(RDBMS)의 테이블 간의 데이터를 자동으로 매핑해주는 기술입니다.
- ORM을 통해 개발자는 SQL을 직접 작성하지 않고도 데이터베이스와 상호작용할 수 있으며, 객체 지향 방식으로 데이터를 처리할 수 있습니다.

ORM 장점

- SQL문이 아닌 Method를 통해 DB를 조작할 수 있어, 개발자는 객체 모델을 이용하여 비즈니스 로직을 구성하는 데만 집중할 수 있음.
(내부적으로는 쿼리를 생성하여 DB를 조작함. 하지만 개발자가 이를 신경 쓰지 않아도 됨)
- Query와 같이 필요한 선언문, 할당 등의 부수적인 코드가 줄어들어, 각종 객체에 대한 코드를 별도로 작성하여 코드의 가독성을 높임.
- 객체지향적인 코드 작성이 가능하다. 오직 객체지향적 접근만 고려하면 되기 때문에 개발 생산성 증가.
- 매팅하는 정보가 Class로 명시 되었기 때문에 ERD를 보는 의존도를 낮출 수 있고 유지보수 및 리팩토링에 유리하다.
- 예를들어 기존 방식에서 MySQL 데이터베이스를 사용하다가 PostgreSQL로 변환한다고 가정해보면, 새로 쿼리를 짜야하는 경우가 생김. 이런 경우에 ORM을 사용한다면 쿼리를 수정할 필요가 없다.

ORM 단점

- 프로젝트의 규모가 크고 복잡하여 설계가 잘못된 경우, 속도 저하 및 일관성을 무너뜨리는 문제점이 생길 수 있음.
- 복잡하고 무거운 Query는 속도를 위해 별도의 튜닝이 필요하기 때문에 결국 SQL 문을 써야할 수도 있음.
- 학습비용이 비쌈.

JPA란

- JPA가 제공하는 API를 사용하면 객체를 DB에 저장하고 관리할 때, 개발자가 직접 SQL을 작성하지 않아도 된다.
- JPA가 개발자 대신 적절한 SQL을 생성해서 DB에 전달하고, 객체를 자동으로 Mapping 해준다.
- JPA는 내부적으로 JDBC API를 활용하는데, 개발자가 직접 JDBC API를 활용하면 패러다임 불일치, SQL의 존성 등으로 인해 효율성이 떨어진다.
- 이 때, JPA를 활용한다면 모든 SQL에 대해 개발자 대신 JPA가 자동으로 해결해 준다는 점에서 생산성을 크게 높인다.

JPA 탄생배경

- 반복적인 SQL 사용
 - 테이블이 100개 존재한다면 100개의 CRUD를 작성해야 한다.
 - 동적 쿼리 작성시 반드시 그런 것은 아니다.
- SQL 의존적 개발
 - 테이블에 하나의 Column 추가 시 모든 SQL 변경이 필요하다.
- DAO와 테이블의 강한 의존성이 존재한다.
 - 객체지향의 장점을 사용하지 않고 객체를 단순히 데이터 전달 목적(VO, DTO)에만 사용한다.
 - 객체지향적이지 못하다

장점

- 생산성
 - Hibernate는 SQL을 직접 사용하지 않고 메서드 호출만으로 쿼리를 수행하기 때문에 SQL 반복 작업을 하지 않으므로 생산성 향상된다.
- 유지보수가 용이 하다
 - 테이블 칼럼 변경 시 이전에는 SQL을 모두 확인 후 수정이 필요했다.
 - JPA는 JPA가 대신 작업을 수행하므로 유지보수 측면에서 장점이 있다.
- 특정 벤더에 종속적이지 않음
 - 여러 DB 벤더 (MySQL, Oracle, mssql 등) 마다 다른 SQL을 사용하기 때문에 DB 변경이 어려웠다.
 - JPA는 추상화된 데이터 접근 계층을 제공하여 특정 벤더에 종속적이지 않다. 어떤 DB를 사용하고 있는지만 설정하면 얼마든지 DB 변경이 가능하다.

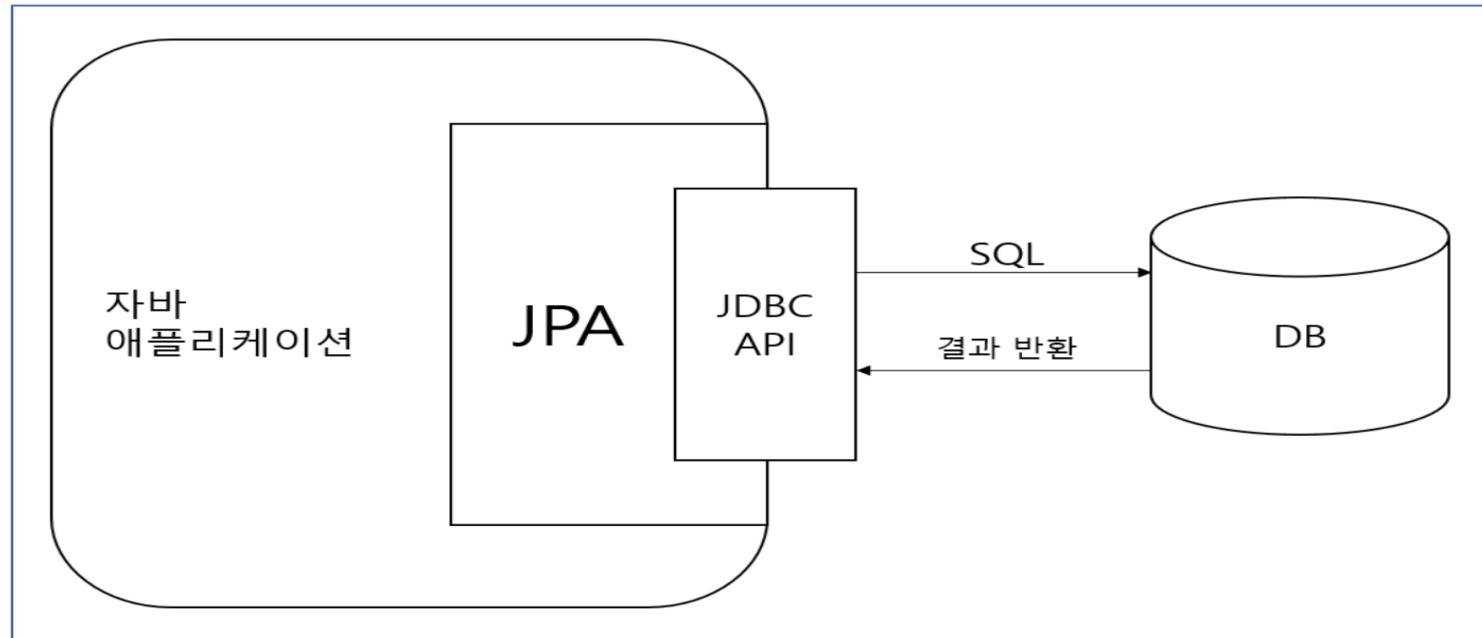
단점

- 성능
 - 메서드 호출로 쿼리를 실행하는 건 내부적인 동작이 많다는 것을 의미한다.
그래서 직접 SQL을 호출하는 것보다 **성능이 낮을 수** 있다.
- 러닝커브
 - JPA를 사용하기 위해서는 학습해야 할 것들이 많다.
 - JPA를 사용해도 SQL을 알아야 Hibernate가 수행한 쿼리 확인 및 성능 모니터링이 가능하다.
- 여러 테이블을 조인하거나 서브쿼리가 필요할 경우 사용이 어렵다.

JPA란

- JPA(Java Persistence API)는 자바 애플리케이션에서 데이터베이스와 상호작용하는 방식을 정의한 자바 표준 인터페이스입니다.
- JPA는 객체 지향 프로그래밍의 객체와 관계형 데이터베이스의 테이블 간의 매핑(ORM, Object Relational Mapping)을 제공하여, 개발자가 데이터베이스와 상호작용하는 코드를 단순화할 수 있습니다.

JPA 구조



출처 : <https://noobnim.tistory.com/39?pidx=0>

JPA 의 주요개념

1. ORM(Object-Relational Mapping):

객체 모델(클래스)과 관계형 데이터베이스 모델(테이블)을 자동으로 매핑하는 기술입니다. 이를 통해 자바 객체를 데이터베이스 테이블의 행(row)으로 쉽게 저장하거나 조회할 수 있습니다.

2. Entity

JPA에서 **Entity**는 데이터베이스의 테이블과 매핑되는 자바 클래스입니다. 객체 지향 프로그래밍의 객체와 관계형 데이터베이스의 테이블을 연결하는 역할을 합니다.

3. Persistence Context

JPA는 엔티티 객체를 영속성 컨텍스트(Persistence Context)에서 관리합니다. 이는 엔티티 객체가 데이터베이스와 연결된 상태를 유지하는 메커니즘으로, 데이터의 변경 사항이 자동으로 데이터베이스에 반영됩니다.

4. JPQL (Java Persistence Query Language)

JPA는 데이터베이스의 SQL을 직접 사용하지 않고, 객체 기반의 쿼리 언어인 JPQL을 사용합니다. JPQL은 SQL과 비슷하지만, 데이터베이스의 테이블이 아닌 엔티티 객체를 대상으로 쿼리를 실행합니다.

5. Entity Manager

EntityManager는 JPA의 핵심 인터페이스로, 엔티티의 저장, 삭제, 조회, 업데이트 등의 작업을 수행합니다.

Entity 란

- JPA에서 Entity는 데이터베이스 테이블에 대응하는 자바 객체를 의미합니다.
- Entity 클래스는 데이터베이스 테이블과 매핑됩니다. 클래스의 각 필드는 테이블의 열(column)에 대응됩니다.
- Entity는 반드시 기본 키를 가져야 합니다. 기본 키는 엔티티의 각 인스턴스를 고유하게 식별하는 값으로, 데이터베이스 테이블의 기본 키에 매핑됩니다.
- 기본 키는 보통 `@Id` 어노테이션으로 설정되며, 자동으로 생성되거나 수동으로 설정될 수 있습니다.
- Entity는 영속성 컨텍스트에 의해 관리됩니다. 이는 데이터베이스와 메모리 간의 객체 상태를 동기화하는 역할을 합니다.
- 영속성 컨텍스트에 있는 Entity는 변경되면 트랜잭션이 끝날 때 자동으로 데이터베이스에 반영됩니다.

Entity

- 엔티티는 Getter만 두고 Setter를 두지 않습니다. 이유는 외부에서 엔티티의 상태를 직접 변경할 수 없기 때문에 엔티티의 상태가 예측 가능한 방식으로만 변경되게 하기 위해서입니다.
- 불변 객체는 특히 다중 스레드 환경에서 안전합니다. 여러 스레드가 동시에 엔티티를 조회할 때, 그 엔티티의 상태가 변경될 가능성이 없으므로 일관된 상태를 보장할 수 있습니다.
- 엔티티에 Getter만 두면, 필드 값을 읽을 수는 있지만 필드를 변경할 수 없기 때문에, 상태를 외부에 노출하지 않고 보호할 수 있습니다. 상태를 변경하는 작업은 엔티티 내부의 특정 로직을 통해서만 이루어질 수 있어, 데이터의 무결성을 보장할 수 있습니다.
- 엔티티에서 Getter만 제공하는 것은 데이터의 불변성, 캡슐화, 일관성을 유지하고, 비즈니스 로직을 명확히 표현하기 위한 중요한 설계 원칙입니다. Setter를 남용하면 객체의 상태가 외부에서 무분별하게 변경될 수 있어 데이터 무결성이 깨질 수 있기 때문에, 엔티티의 상태 변경은 반드시 비즈니스 메서드를 통해서만 이루어지도록 하는 것이 좋습니다.

Persistence Context

- JPA(Java Persistence API)에서 엔티티 객체의 생명 주기와 상태를 관리합니다. 이는 엔티티 객체와 데이터베이스 간의 중간 저장소 역할을 하며, 엔티티의 상태 변화를 추적하고 데이터베이스와 동기화합니다.
- Persistence Context는 1차 캐시로 작동하여 엔티티 객체를 메모리에 저장합니다. 엔티티 매니저(EntityManager)를 통해 조회한 엔티티는 Persistence Context에 저장되고, 이후 동일한 엔티티를 조회하면 데이터베이스를 다시 조회하지 않고 Persistence Context에서 해당 엔티티를 반환합니다
- Persistence Context는 엔티티의 생명주기와 상태를 관리합니다. 엔티티 객체는 다음과 같은 네 가지 상태 중 하나에 속할 수 있습니다:

Transient(비영속), Persistent(영속), Detached(준영속), Removed(삭제) 등
- Persistence Context는 변경 감지 기능을 통해 엔티티의 상태 변화를 감지하고, 트랜잭션이 커밋될 때 자동으로 데이터베이스에 반영합니다. 엔티티의 필드를 직접 변경하면 JPA는 이를 추적하고, 해당 변경 사항을 데이터베이스 업데이트 쿼리로 변환해 실행합니다.
- 동일한 Persistence Context 안에서는 같은 엔티티를 조회할 때 동일한 객체가 반환됩니다. 예를 들어, 같은 ID를 가진 엔티티를 여러 번 조회하더라도, 동일한 Persistence Context에서는 항상 같은 객체를 반환합니다. 이는 JPA가 객체의 동일성(identity)을 보장해주는 중요한 기능 중 하나입니다.

Persistence Context 장단점

- **성능 최적화** 1차 캐시를 통해 동일한 엔티티에 대한 불필요한 데이터베이스 쿼리를 방지합니다.
- 변경 감지 기능을 통해 효율적으로 데이터베이스의 상태를 변경할 수 있습니다.
- 엔티티의 변경 사항을 메모리에서 관리하고, 필요한 시점에만 데이터베이스에 반영합니다.
- **트랜잭션과 일관성 보장** 트랜잭션 내에서 엔티티의 상태를 관리하고, 커밋 시 데이터베이스에 일괄적으로 반영되므로, 트랜잭션 일관성을 쉽게 유지할 수 있습니다.
- **객체 동일성 보장** 동일한 엔티티 매니저(Persistence Context) 내에서는 같은 엔티티를 조회할 때 항상 동일한 객체를 반환하여 객체의 동일성을 보장합니다. 이는 비즈니스 로직에서 객체 비교를 용이하게 만듭니다.
- **메모리 사용** Persistence Context는 영속성 컨텍스트에 엔티티를 캐시하기 때문에, 많은 엔티티가 메모리에 저장될 경우 메모리 사용량이 증가할 수 있습니다. 특히 트랜잭션이 오래 지속되거나, 많은 엔티티를 한 번에 로딩하는 경우 주의해야 합니다.
- **성능 저하** 변경 감지(Dirty Checking)를 위해 영속성 컨텍스트 내의 모든 엔티티에 대해 변경 사항을 추적하므로, 영속성 컨텍스트에 너무 많은 엔티티가 있을 경우 성능이 저하

JPQL이란

- JPQL(Java Persistence Query Language)은 JPA(Java Persistence API)에서 사용하는 객체 지향 쿼리 언어입니다.
- JPQL은 SQL과 유사하지만, SQL이 데이터베이스의 테이블과 컬럼을 대상으로 하는 반면, JPQL은 **엔티티 객체와 그 필드를 대상으로** 쿼리를 실행합니다.
- JPQL은 SQL과 매우 유사한 문법을 사용합니다. SELECT, FROM, WHERE, GROUP BY, ORDER BY 같은 구문이 SQL과 동일하게 사용됩니다.
- JPQL은 특정 데이터베이스에 종속되지 않고 플랫폼 독립적입니다.
JPQL 쿼리는 특정 데이터베이스에 맞게 변환되므로, 데이터베이스 종류에 관계없이 사용할 수 있습니다.
- JPQL은 동적 쿼리를 지원합니다. 런타임 시 조건이나 파라미터를 동적으로 생성하여 쿼리를 실행할 수 있습니다.
- JPQL은 엔티티 객체의 연관된 엔티티까지 쿼리할 수 있으며, 엔티티의 관계 (1, N:1 등)에 따라 조인 쿼리도 작성할 수 있습니다.

JPQL

- SELECT: 데이터베이스에서 가져올 데이터를 지정합니다.
- FROM: 쿼리할 엔티티 클래스를 지정합니다.
- WHERE: 조건을 지정하여 필터링합니다.
- JOIN: 관계를 맺고 있는 엔티티 간의 조인을 수행합니다.
- GROUP BY: 특정 필드를 기준으로 그룹화합니다.
- ORDER BY: 결과를 정렬합니다.

JPQL 예시

- // 모든 사용자 조회

```
String jpql = "SELECT m FROM Member m";  
  
List<Member> members = entityManager.createQuery(jpql,  
Member.class).getResultList();
```

- // 특정 사용자 이름으로 조회

```
String jpql = "SELECT m FROM Member m WHERE m.name = :name";  
  
List<Member> members = entityManager.createQuery(jpql, User.class)  
    .setParameter("name", "John")  
    .  
    .getResultSet();
```

의존성

JPA 를 위한 라이브러리를 추가합니다.

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

디비설정

```
@Configuration  
public class DBConfig {  
    @Autowired  
    ApplicationContext applicationContext;  
    //hikari 설정하기 - java가 제공하는 경량의 디비커넥션풀, spring boot에 기본장착  
    @Bean  
    @ConfigurationProperties(prefix = "spring.datasource.hikari")  
    public HikariConfig hikariConfig() {  
        return new HikariConfig();  
    }  
  
    @Bean  
    public DataSource dataSource() {  
        return new HikariDataSource(hikariConfig());  
    }  
}
```

hikari와 JPA 설정하기

```
#Log4JDBC라는 라이브러리에서 제공하는 JDBC 드라이버입니다.  
#SQL 쿼리와 그에 관련된 로그를 쉽게 출력할 수 있도록 도와줍니다.  
spring.datasource.hikari.driver-class-name=net.sf.log4jdbc.sql.jdbcapi.DriverSpy  
spring.datasource.hikari.jdbc-url=jdbc:log4jdbc:oracle:thin:@localhost:1521:XE  
spring.datasource.hikari.username=user02  
spring.datasource.hikari.password=1234  
#spring.datasource.hikari.connection-test-query=SELECT SYSDATE FROM DUAL  
  
# JPA 관련 설정하기  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=none  
spring.jpa.properties.hibernate.format_sql=true  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
#spring.jpa.defer-datasource-initialization: true  
spring.jackson.serialization.fail-on-empty-beans=false  
.hql.bulk_id_strategy
```

Entity클래스 생성하기 1

```
@Getter  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor(access = AccessLevel.PROTECTED) // 직접 객체 생성을 막으려고, Builder 패턴형태로 만들고자 한다  
@Entity  
@Table(name = "TB_MEMBER")  
@SequenceGenerator(  
    name = "SEQ_GENERATOR",    // 식별자 생성기 이름  
    sequenceName = "seq_tb_member", // 데이터베이스에 등록되어 있는 시퀀스 이름  
    initialValue = 1,           // 처음 시작하는 수를 지정  
    allocationSize = 1          // 시퀀스 한 번 호출에 증가하는 수  
)  
  
public class UserDto {  
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator="SEQ_GENERATOR")  
    @Column(name = "member_id")  
    private Long member_id; // PK  
    @Id  
    @Column(name = "user_id", length=80, unique=true)  
    private String user_id; //  
  
    private String password; //  
    private String name; //  
    private String email; //  
    private String phone; //  
}
```

Entity클래스 생성하기 2

```
@Getter  
@NoArgsConstructor(access = AccessLevel.PROTECTED)  
@Entity  
@Table(name = "TB_BOARD2")  
@SequenceGenerator(  
    name = "SEQ_GENERATOR",    // 식별자 생성기 이름  
    sequenceName = "seq_tb_board2", // 데이터베이스에 등록되어 있는 시퀀스 이름  
    initialValue = 1,           // 처음 시작하는 수를 지정  
    allocationSize = 1          // 시퀀스 한 번 호출에 증가하는 수  
)  
//@BatchSize(size = 200)  
public class BoardDto extends BaseDto{  
    @Id  
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator="SEQ_GENERATOR")  
    private Long id; // PK  
    private String title; // 제목  
  
    @Column()  
    @Lob  
    private String contents; // 내용  
    @Column(name = "user_id")  
    private String user_id; // 작성자  
    private Integer hits=0; // 조회 수  
    private String deleteYn="N"; // 삭제 여부  
    private LocalDateTime createdDate = LocalDateTime.now(); // 생성일  
    private LocalDateTime modifiedDate= LocalDateTime.now(); // 수정일  
  
    @Builder  
    public BoardDto(Long id, String title, String contents, String user_id) {  
        this.id = id;  
        this.title = title;  
        this.contents = contents;  
        this.user_id = user_id;  
    }  
    //tb_member 의 user_id 필드를 외부키로 한다 테이블을 참조한다  
    //지연로딩을 하게 되면 member 정보는 필요할 때까지 로드되지 않습니다  
    //JoinColumn - 외래키를 연결한다  
    @ManyToOne(fetch = FetchType.LAZY)  
    @JoinColumn(name = "user_id", insertable=false, updatable=false)  
    private UserDto member;  
}
```

Join하기

- JPA에서 **JOIN**을 사용해 두 개 이상의 테이블을 조인하여 데이터를 조회하는 방법은 주로 **JPQL**(Java Persistence Query Language) 또는 **Criteria API**를 통해 이루어집니다. JPA는 객체 간의 연관 관계를 맺는 방식으로, 객체 지향적으로 조인 작업을 수행할 수 있습니다.
- Member ←----- Board 가 참조일때 Board쪽 클래스에 추가

```
@ManyToOne(fetch = FetchType.LAZY)  
@JoinColumn(name = "user_id", insertable=false, updatable=false)  
private UserDto member;
```

Join하기

//JPQL을 사용하여 한다

```
String jpql = "SELECT m FROM Member m JOIN m.team t WHERE t.name = :teamName";  
TypedQuery<Member> query =  
    em.createQuery(jpql, Member.class); query.setParameter("teamName", "Development");  
List<Member> result = query.getResultList();
```

CriteriaBuilder를 통해 동적 쿼리 생성

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Order> cq = cb.createQuery(Order.class);
Root<Order> order = cq.from(Order.class);

// User와 조인
Join<Order, User> user = order.join("user", JoinType.INNER);

// where 절 추가 (user의 이름이 "John"인 경우)
cq.select(order).where(cb.equal(user.get("name"), "John"));

List<Order> orders = entityManager.createQuery(cq).getResultList();
```

데이터생성쿼리

```
CREATE OR REPLACE PROCEDURE insert_sample_data_loop AS
BEGIN
    -- FOR 루프: 1부터 10까지 반복하여 데이터를 삽입
    FOR i IN 1..10 LOOP
        INSERT INTO tb_board2 (
            ID,
            CONTENTS,
            CREATED_DATE,
            DELETE_YN,
            HITS,
            MODIFIED_DATE,
            TITLE,
            USER_ID
        ) VALUES (
            i, -- ID는 루프 변수로 설정
            'This is sample content ' || i, -- CONTENTS는 루프 변수와 함께 동적 값 생성
            SYSDATE, -- 현재 타임스탬프 사용
            CASE WHEN MOD(i, 2) = 0 THEN 'Y' ELSE 'N' END, -- 짝수 ID는 'Y', 홀수는 'N'
            i * 10, -- 조회수는 10의 배수로 설정
            SYSDATE, -- 수정 날짜
            'Sample Title ' || i, -- TITLE도 루프 변수와 함께 동적 값 생성
            CASE WHEN MOD(i, 2) = 0 THEN 'user' ELSE 'test' END
        );
    END LOOP;

    -- 커밋하여 변경 사항 저장
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('10 rows of sample data inserted successfully.');

EXCEPTION
    -- 예외 처리: 오류가 발생할 경우 롤백
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/

exec insert_sample_data_loop;
select count(*) from tb_board2;
```

Repository 만들기

- 스프링부트에서 JPA Repository 는 간단히 만들수 있습니다.

1. JPA 엔티티 클래스 작성

2. 인터페이스 작성

```
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BoardRepository extends JpaRepository<Board, Long> {  
    // 추가적인 쿼리 메서드를 여기에 정의할 수 있습니다.  
    // JpaRepository 는 Generic으로 Entity 클래스와 Entity클래스의 키값 타입을 필요로 합니다.  
    //아무것도 없더라도 기본 crud를 지원합니다.  
}
```

기본 CRUD 메서드 1

- `save(S entity)`: 주어진 엔티티를 저장하거나, 엔티티가 이미 존재하는 경우 업데이트합니다.
- `findById(ID id)`: 기본 키 값으로 엔티티를 조회합니다. `Optional<T>` 타입으로 반환되며, 값이 없으면 비어있는 `Optional`이 반환됩니다.
- `findById(ID id)`: 기본 키 값으로 엔티티를 조회합니다. `Optional<T>` 타입으로 반환되며, 값이 없으면 비어있는 `Optional`이 반환됩니다.
- `existsById(ID id)`: 특정 ID의 엔티티가 존재하는지 여부를 확인합니다.

 true 또는 false를 반환합니다.
- `count()`: 엔티티의 총 개수를 반환합니다.
- `deleteById(ID id)`: 기본 키를 이용해 해당 엔티티를 삭제합니다.
- `delete(T entity)`: 주어진 엔티티를 삭제합니다.
- `deleteAll()`: 모든 엔티티를 삭제합니다.

페이지 및 정렬

- JpaRepository는 페이지과 정렬을 위한 메서드도 기본적으로 제공합니다.
- findAll(Pageable pageable): 페이지 처리된 결과를 반환합니다.

예시)

```
Page<Board> boardPage = boardRepository.findAll(PageRequest.of(0, 10));
```

findAll(Sort sort): 주어진 정렬 기준에 따라 모든 엔티티를 조회합니다.

예시)

```
List<Board> sortedBoards = boardRepository.findAll(Sort.by(Sort.Direction.ASC, "title"));
```

예시)

```
Sort sort = Sort.by(Sort.Direction.fromString("desc"), "id");
```

```
Pageable pageable = PageRequest.of(page, size, sort); // 정렬 정보 포함
```

```
List<Board> list = repository.findAll(pageable).getContent();
```

조회

- **findBy:** 특정 필드 값에 따라 데이터를 조회합니다.
 - 예시) List<Board> findByTitle(String title);
- **findFirstBy:** 첫 번째 결과만 조회합니다.
 - 예시) Board findFirstByOrderByCreatedDateDesc();
- **findTop3By:** 상위 n개의 결과를 조회합니다.
 - 예시) List<Board> findTop3ByOrderByViewsDesc();
- **조건부여 :** Spring Data JPA에서 사용할 수 있는 여러 조건 키워드를 이용하여 복합적인 조건 쿼리를 생성할 수 있습니다.
- **And:** 두 조건을 모두 만족하는 데이터를 조회합니다.
 - 예시) List<Board> findByTitleAndContent(String title, String content);
- **Or:** 두 조건 중 하나라도 만족하는 데이터를 조회합니다.
 - 예시) List<Board> findByTitleOrContent(String title, String content);
- **Between:** 두 값 사이에 있는 데이터를 조회합니다.
 - 예시) List<Board> findByCreatedDateBetween(LocalDate start, LocalDate end);

조회

- LessThan, GreaterThan: 특정 값보다 작은 또는 큰 데이터를 조회합니다.
- 예시) List<Board> findByViewsLessThan(int views);
- Like: 특정 패턴을 만족하는 데이터를 조회합니다.
- 예시) List<Board> findByTitleLike(String titlePattern);
- In: 주어진 값들 중 하나라도 일치하는 데이터를 조회합니다.
- 예시) List<Board> findByIdIn(List<Long> ids);
- IsNull, IsNotNull: 필드 값이 null 또는 null이 아닌 데이터를 조회합니다.
- 예시) List<Board> findByContentIsNull();

order by

- OrderBy 키워드를 사용하여 특정 필드를 기준으로 정렬된 데이터를 조회할 수 있습니다.
- 예시) List<Board> findByTitleOrderByCreatedDateDesc(String title);

서비스

```
2개 사용 위치
@Service
@Transactional
@RequiredArgsConstructor // 클래스에 final이 있을 경우에 객체를 주입하는 생성자를 만든다
public class BoardServiceImpl implements BoardService{

    private final BoardRepository repository;

1개 사용 위치
    public List<BoardDto> getList(BaseDto dto){
        int page = dto.getPg();
        int size = dto.getPgSize();

        Sort sort = Sort.by(Sort.Direction.fromString( value: "desc"), ...properties: "id");
        Pageable pageable = PageRequest.of(page, size, sort); // 정렬 정보 포함

        List<Board> list = repository.findAll(pageable).getContent();
        for(int i=0; i<list.size(); i++)
        {
            System.out.println(list.get(i).getTitle());
            System.out.println(list.get(i).getMember().getName());
        }
        //return list
        return list.stream() Stream<Board>
            .map(board -> BoardDto.builder()
                .id(board.getId())
                .title(board.getTitle())
                .contents( board.getContents())
                .user_id( board.getUser_id())
                .name(board.getMember().getName())
                .email(board.getMember().getEmail())
                .build()) Stream<BoardDto>
            .collect(Collectors.toList());
    }

    //검색을 해야 할 경우에

    //return repository.findByTitleLike("%"+board.getTitle()+"%", pageable);
    //return repository.searchByTitleLike(board.getTitle(), board.getContents());
    //return repository.searchByTitleWriterContentsLike(board.getTitle(), board.getWriter(), board.getContents(), pageable);
}
```

서비스

```
1개 사용 위치
@Transactional
public void write(Board dto) {
    repository.save(dto);
}

1개 사용 위치
@Transactional
public void update(Board dto) {
    System.out.println(dto.getId());
    Optional<Board> resultDto = repository.findById(dto.getId());
    resultDto.orElseThrow(() -> new RuntimeException(dto.getId() + "가 존재하지 않습니다"));
    repository.save(dto); // 게시글 수정
}

1개 사용 위치
public void delete(Board dto) {
    //BoardDto dto = repository.findById(id).get();
    Optional<Board> resultDto = repository.findById(dto.getId());
    resultDto.orElseThrow(() -> new RuntimeException(dto.getId() + "가 존재하지 않습니다"));
    repository.delete(dto);
}

1개 사용 위치
public BoardDto getView(Long id){

    Optional<Board> resultDto = repository.findById(id);

    resultDto.orElseThrow(() -> new RuntimeException(id + "가 존재하지 않습니다"));
    Board board = resultDto.get();
    BoardDto dto = BoardDto.builder()
        .id(board.getId())
        .title(board.getTitle())
        .contents( board.getContents())
        .user_id( board.getUser_id())
        .name(board.getMember().getName())
        .email(board.getMember().getEmail())
        .build();

    return dto;
}
}
```

컨트롤러 예시

```
@Controller
@RequestMapping(value="/board")
public class BoardController {

    8개 사용 위치
    private final BoardService boardService;

    0개의 사용 위치
    public BoardController(BoardService boardService) { this.boardService = boardService; }

    0개의 사용 위치
    @GetMapping("/list")
    public String getList(Model model, BoardDto dto){

        model.addAttribute( attributeName: "boardList", boardService.getList(dto));
        return "board/board_list";
    }

    0개의 사용 위치
    @GetMapping("/view/{id}")
    public String getView(Model model, @PathVariable("id")String id){

        model.addAttribute( attributeName: "boardView", boardService.getView(id).get());
        System.out.println(boardService.getView(id).get());
        return "board/board_view";
    }

    0개의 사용 위치
    @GetMapping("/write")
    public String write(Model model ){
        model.addAttribute( attributeName: "mode", attributeValue: "insert");
        return "board/board_write";
    }

    0개의 사용 위치
    @PostMapping("/save")
    public String save(BoardDto dto){
        boardService.insert(dto);
        return "redirect:/board/list";
    }
}
```

MyBatis 연동

Mybatis 라이브러리(build.gradle)

```
implementation group: 'com.oracle.database.jdbc', name: 'ojdbc8', version:  
'18.3.0.0'  
  
implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'  
  
implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.0'
```

application.properties

```
#oracle 접속 -- 디비 커넥션 풀이 아니다.  
#spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:XE  
#spring.datasource.driver-class-  
name=oracle.jdbc.driver.OracleDriver  
#spring.datasource.username=user01  
#spring.datasource.password=1234
```

JDBC가 지원하는 디비커넥션풀 프레임워크 : hikari

```
spring.datasource.hikari.jdbc-  
url=jdbc:oracle:thin:@127.0.0.1:1521:XE  
spring.datasource.hikari.driver-class-  
name=oracle.jdbc.driver.OracleDriver  
spring.datasource.hikari.username=user01  
spring.datasource.hikari.password=1234
```

설정파일작성하기

//@Configuration 이 있으면 설정클래스로 인식을 한다 파일명과 클래스명은 마음대로 작성할 수 있다.

```
@Configuration
```

```
public class MyBatisConfig {
```

```
    @Autowired
```

```
        ApplicationContext applicationContext;
```

```
//hikari 설정하기
```

```
    @Bean //객체 생성 – 설정파일 항목을 읽어와서 초기화를 한다 .
```

```
    @ConfigurationProperties(prefix = "spring.datasource.hikari")
```

```
    public HikariConfig hikariConfig() {
```

```
        return new HikariConfig();
```

```
}
```

```
    @Bean //DataSource 객체가 생성되어야 한다 datasource 함수호출시 객체가 생성된다.
```

```
    public DataSource dataSource() {
```

```
        return new HikariDataSource(hikariConfig());
```

```
}
```

설정파일작성하기

```
@Bean  
public SqlSessionFactory makeSqlSessionFactory(DataSource dataSource) throws Exception  
{  
    //SqlSessionFactory - Factory 공장객체를 먼저 만든다.  
    final SqlSessionFactoryBean factory = new SqlSessionFactoryBean();  
    // factory 객체와 application.properties 파일에 있는 datasource와 연결  
    factory.setDataSource(dataSource);  
    PathMatchingResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();  
    //설정파일과 연동하기(mybatis-config.xml)파일과 연동, classpath - src/main/resource  
    Resource configLocation = resolver.getResource("classpath:mybatis-config.xml");  
    factory.setConfigLocation(configLocation);  
    return factory.getObject();  
}  
  
@Bean  
public SqlSessionTemplate makeSqlSession(SqlSessionFactory factory)  
{  
    return new SqlSessionTemplate(factory);  
}  
}
```

mybatis-config.xml

- mybatis 전체 맵퍼파일을 연결하거나 클래스경로를 단축한 alias 등을 만들기 위한 기본파일이다.
- src/main/resources 아래에 mybatis-config.xml 을 둔다.
- application.properties에서 한꺼번에 처리하는 방법도 있다 둘중 어느방법을 쓰던 상관없다.

mybatis-config.xml

```
-//mybatis.org//DTD Config 3.0//EN (doctype)
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 원래 클래스명은 패키지까지 포함하기 때문에 너무 길어서 짧은
     별명을 만들어서 접근하기 위해 기술한다 -->
<typeAliases>
    <typeAlias alias="GuestbookDto"
        type="com.kosa.myapp.guestbook.domain.GuestbookDto"/>

    <typeAlias alias="BoardDto"
        type="com.kosa.myapp.board.domain.BoardDto"/>

</typeAliases>

<mappers>
    <!-- sql 쿼리를 두는 곳 Guestbook.xml 파일을 만들어야 한다 -->
    <!-- src/main/resources 폴더에 두자 -->
    <mapper resource= "mapper/Guestbook.xml" />
    <mapper resource= "mapper/Board.xml" />

</mappers>

</configuration>
```

Board.xml

쿼리파일은 src/main/resources 폴더아래에 mapper라는 폴더를 만들고 그곳에 둔다 .

```
<?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="Board">
    <select id="Board_getTotalCnt" parameterType="BoardDto" resultType="Integer">
        select count(*)
        from tb_board2
    </select>
</mapper>
```

mybatis를 통한 쿼리 접근 예시

```
package com.kosa.myapp.board.repository;

import java.util.List;

@Repository("boardDao")
public class BoardDaoRepository implements BoardDao{
    @Autowired
    SqlSessionTemplate sm;

    @Override
    public List<BoardDto> getList(BoardDto dto) {
        return sm.selectList("Board_getList", dto);
    }

    @Override
    public int getTotalCnt(BoardDto dto) {
        return sm.selectOne("Board_getTotalCnt", dto);
    }

    @Override
    public BoardDto getView(BoardDto dto) {
        return sm.selectOne("Board_getView", dto);
    }

    @Override
    public void insert(BoardDto dto) {
        sm.insert("Board_insert", dto);
    }
}
```

맵퍼(BoardMapper)

- 몇년전부터 Dao 대신 Mapper를 많이 사용합니다. Mapper는 인터페이스 구축만으로 별도의 구현 클래스가 필요 없습니다.
- 자바 코드와 SQL문(*.xml 형식)을 분리하여 편리하게 관리하도록 합니다.
- DAO는 주로 데이터베이스와의 CRUD 작업을 처리하고, 비즈니스 로직에서 데이터 접근을 담당하는 패턴입니다. DAO는 SQL 쿼리 작성과 데이터베이스 트랜잭션 처리에 더 초점을 맞추고 있습니다.
- Mapper는 객체와 객체, 또는 객체와 SQL 쿼리 결과 간의 데이터 매핑을 담당하며, 특히 MyBatis 같은 SQL Mapper나 DTO-Entity 간 변환에 사용됩니다. Mapper는 SQL 쿼리와 자바 객체 간의 매핑을 자동으로 처리하는 역할을 합니다.

맵퍼 사용 예

```
package com.example.mapper.UserMapper  
//namespace와 패키지가 반드시 일치해야 한다  
  
@Mapper  
  
public interface UserMapper {  
  
    User getUserById(int id);  
  
    List<User> getAllUsers();  
  
    void insertUser(User user);  
}
```

Mapper

- xml 을 작성하지 않고 Mapper 클래스만 작성해도 된다. 그러나 복잡한 쿼리의 경우 따로 작성하는것이 좋다. Mapper 에 직접 쿼리를 부여하는 방법도 있다

```
<mapper namespace="com.example.mapper.UserMapper">

    <select id="getUserById" parameterType="int" resultType="User">
        SELECT * FROM users WHERE id = #{id}
    </select>

    <insert id="insertUser" parameterType="User">
        INSERT INTO users (name, email) VALUES (#{name}, #{email})
    </insert>

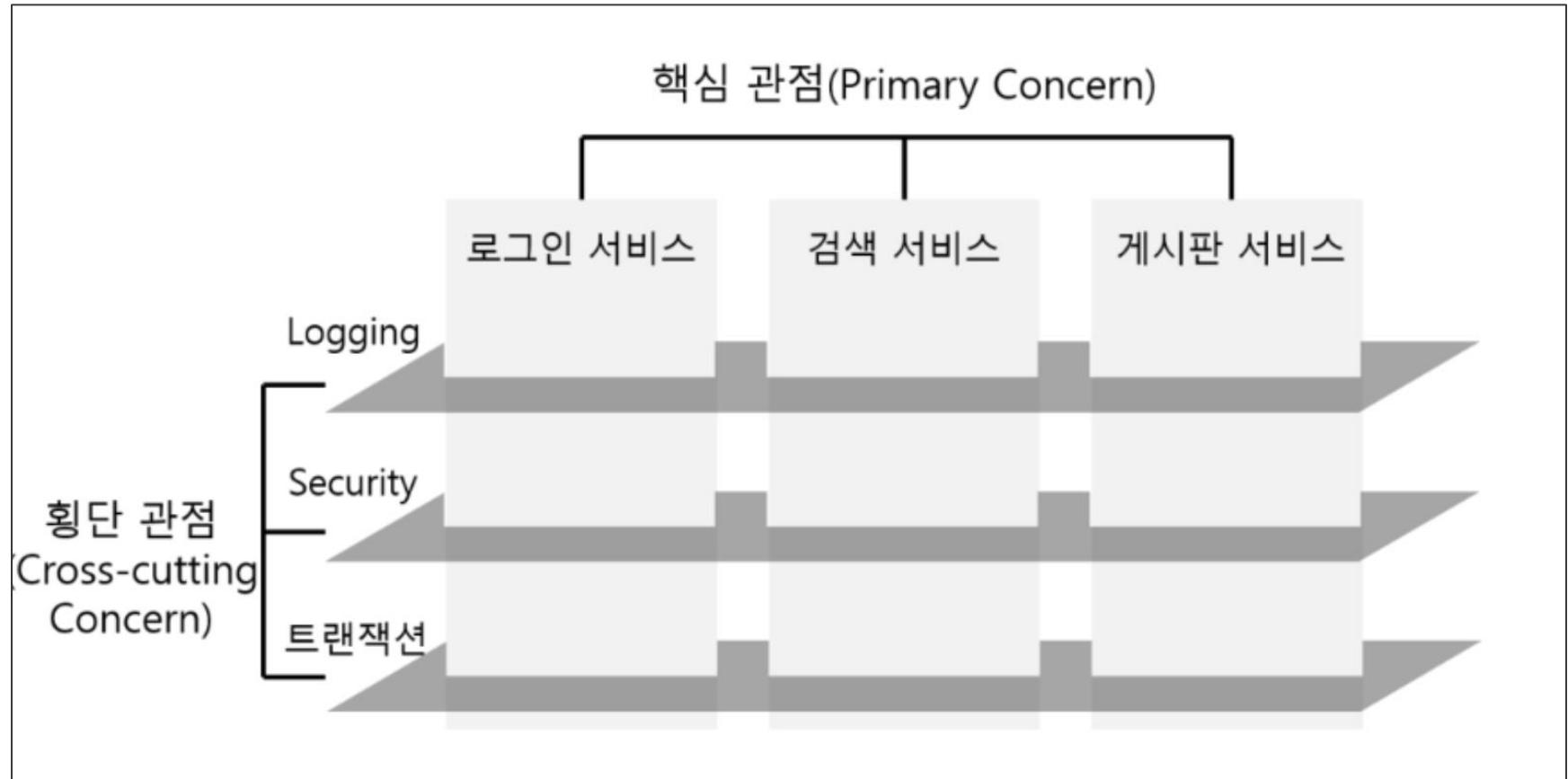
</mapper>
```

AOP

AOP란

- AOP란 Aspect-Oriented Programming의 약자로 관점 지향 프로그램이라고 합니다. AOP는 비지니스 로직부분과 공통의 관심사부분을 분리하여 개발자로 하여금 핵심적인 비지니스 로직에 집중해서 개발하게 하는 개념입니다. 주로 로깅, 트랜잭션 관리, 보안과 같은 횡단 관심사를 비즈니스 로직에서 분리하는 강력한 기술입니다. 이를 통해 실제 코드 수정 없이 여러 지점에서 재사용 가능한 기능을 정의하고 적용할 수 있습니다.
- 그러나 최근에는 로깅은 보안과 함께 Spring Security, 로그는 logback 라이브러리, 트랜잭션처리도 기존의 AOP보다 훨씬 쉽게 별도 처리가 가능하여 큰 의미가 사라지고 있습니다. 내부에서는 여전히 사용중 있지만 개발자가 모든 의미를 알고 모든 것을 설정하는 것으로 부터 벗어났다는 의미입니다.

AOP란



출처 : <https://sharonprogress.tistory.com/195>

AOP 주요개념

- Aspect(관점): 여러 객체에서 공통으로 사용되는 기능을 모듈화한 클래스입니다. Advice(충고) 와 Pointcut(시점)을 포함합니다.
- Advice(충고): 특정 Join Point(결합점)에서 Aspect가 수행하는 동작입니다. 종류는 다음과 같습니다.
 - @Before: 메서드 실행 전에 동작합니다.
 - @After: 메서드 실행 후에 동작합니다.
 - @AfterReturning: 메서드가 정상적으로 반환된 후에 동작합니다.
 - @AfterThrowing: 메서드 실행 중 예외가 발생했을 때 동작합니다.
 - @Around: 메서드 실행 전후 또는 실행을 감싸는 형태로 동작합니다.
- Join Point(결합점): 프로그램 실행 중 Aspect가 적용될 수 있는 지점(예: 메서드 호출)입니다.
- Pointcut(시점): 어떤 Join Point에서 Advice를 적용할지를 결정하는 표현식입니다.
- Weaving(위빙): Aspect를 타겟 객체에 적용하는 과정입니다.

aop 예제

- implementation 'org.springframework.boot:spring-boot-starter-aop'
- // AOP 의존성을 추가한다.
- @EnableAspectJAutoProxy 를 main이 있는 클래스에 추가한다.

```
@SpringBootApplication  
@EnableAspectJAutoProxy  
public class MyappAopApplication {  
  
    public static void main(String[] args) { SpringApplication.run(MyappAopApplication.class, args); }  
}
```

aop 예제

```
@Aspect
@Component
public class LoggingAspect {
    0개의 사용위치
    @Before("execution(* com.namgrambooks...*.service.*.*(..))") // 서비스 클래스의 모든 메서드 실행 전에 동작
    public void logBeforeMethodExecution() {
        System.out.println("메서드 실행 전 로그 출력");
    }

    0개의 사용위치
    @After("execution(* com.namgrambooks...*.service.*.*(..))") // 서비스 클래스의 모든 메서드 실행 후에 동작
    public void logAfterMethodExecution() {
        System.out.println("메서드 실행 후 로그 출력");
    }

    0개의 사용위치
    @Around("execution(* com.namgrambooks...*.service.*.*(..))") // 서비스 클래스의 모든 메서드 실행 전후에 동작
    public Object logAroundMethodExecution(ProceedingJoinPoint joinPoint) throws Throwable {
        long startTime = System.currentTimeMillis(); // 메서드 실행 전 시간 측정
        // 메서드 실행
        Object result = joinPoint.proceed();

        long timeTaken = System.currentTimeMillis() - startTime; // 메서드 실행 후 시간 측정
        System.out.println(joinPoint.getSignature() + " 실행 시간: " + timeTaken + "ms");

        return result; // 메서드 결과 반환
    }
}
```

Service and Controller

```
import org.springframework.stereotype.Service;

2개 사용 위치
@Service
public class TestService {
    1개 사용 위치
    public void insert(){
        for(int i=1; i<=100; i++)
        {
            System.out.printf("i=%d\n", i);
        }
    }

    0개의 사용 위치
    public void update(){
        for(int i=1; i<=100; i+=2)
        {
            System.out.printf("i=%d\n", i);
        }
    }
}
```

```
0개의 사용 위치
@RestController
@RequiredArgsConstructor
public class TestController{

    private final TestService testService;

    0개의 사용 위치
    @RequestMapping("/")
    public String index(){
        testService.insert();
        return "aop test";
    }
}
```

트랜잭션

트랜잭션(Transaction)이란 데이터베이스 또는 시스템에서 **하나의 논리적 작업 단위**로 처리되는 일련의 작업을 의미합니다. 트랜잭션은 여러 작업을 하나로 묶어서 모두 성공하거나, 하나라도 실패하면 전체를 실패로 처리하여 데이터의 일관성을 보장하는 것이 목표입니다.

트랜잭션의 특성 (ACID)

1. Atomicity (원자성) - 트랜잭션의 모든 작업은 하나의 단위로 처리됩니다. 모든 작업이 성공해야 트랜잭션이 완료되며, 하나라도 실패하면 전체가 실패로 처리됩니다.
2. Consistency (일관성) - 트랜잭션이 시작되기 전과 후의 데이터 상태는 일관성을 유지해야 합니다. 트랜잭션 전과 후의 데이터는 항상 데이터베이스의 제약 조건을 만족해야 합니다.
3. Isolation (고립성) - 동시에 실행되는 트랜잭션들이 서로의 작업에 영향을 주지 않아야 합니다. 각 트랜잭션은 독립적으로 실행되며, 중간 상태의 데이터를 다른 트랜잭션이 볼 수 없도록 보장합니다.
4. Durability (영속성) - 트랜잭션이 성공적으로 완료된 후에는 데이터베이스에 영구적으로 저장되어야 하며, 시스템이 다운되더라도 그 결과가 유지됩니다.

트랜잭션 예

- 트랜잭션은 데이터베이스에서 중요한 역할을 합니다. 예를 들어, 은행의 계좌 이체 작업을 생각해 볼 수 있습니다. 다음과 같은 두 가지 작업이 있습니다.

1. A 계좌에서 100달러를 인출
2. B 계좌에 100달러를 입금

이 두 작업은 하나의 트랜잭션으로 처리되어야 합니다.

만약 첫 번째 작업은 성공하고 두 번째 작업에서 오류가 발생한다면, 트랜잭션을 롤백하여 A 계좌에서 인출된 금액도 원래대로 복구해야 합니다. 그렇지 않으면 데이터 불일치가 발생할 수 있습니다.

Spring boot에서의 트랜잭션

- Spring에서는 @Transactional 어노테이션을 사용하여 트랜잭션을 관리할 수 있습니다. 데이터베이스 작업을 수행하는 메서드에 이 어노테이션을 적용하면, 해당 메서드는 트랜잭션 내에서 실행되며 작업 도중 예외가 발생하면 자동으로 롤백됩니다.

```
@Service
public class BankService {

    @Autowired
    private AccountRepository accountRepository;

    @Transactional
    public void transferMoney(Long fromAccountId, Long toAccountId, BigDecimal amount) {
        Account fromAccount = accountRepository.findById(fromAccountId);
        Account toAccount = accountRepository.findById(toAccountId);

        fromAccount.withdraw(amount);
        toAccount.deposit(amount);

        // 예외 발생 시 전체 작업이 롤백됨
        if (fromAccount.getBalance().compareTo(BigDecimal.ZERO) < 0) {
            throw new RuntimeException("잔고가 부족합니다");
        }
    }
}
```

logback

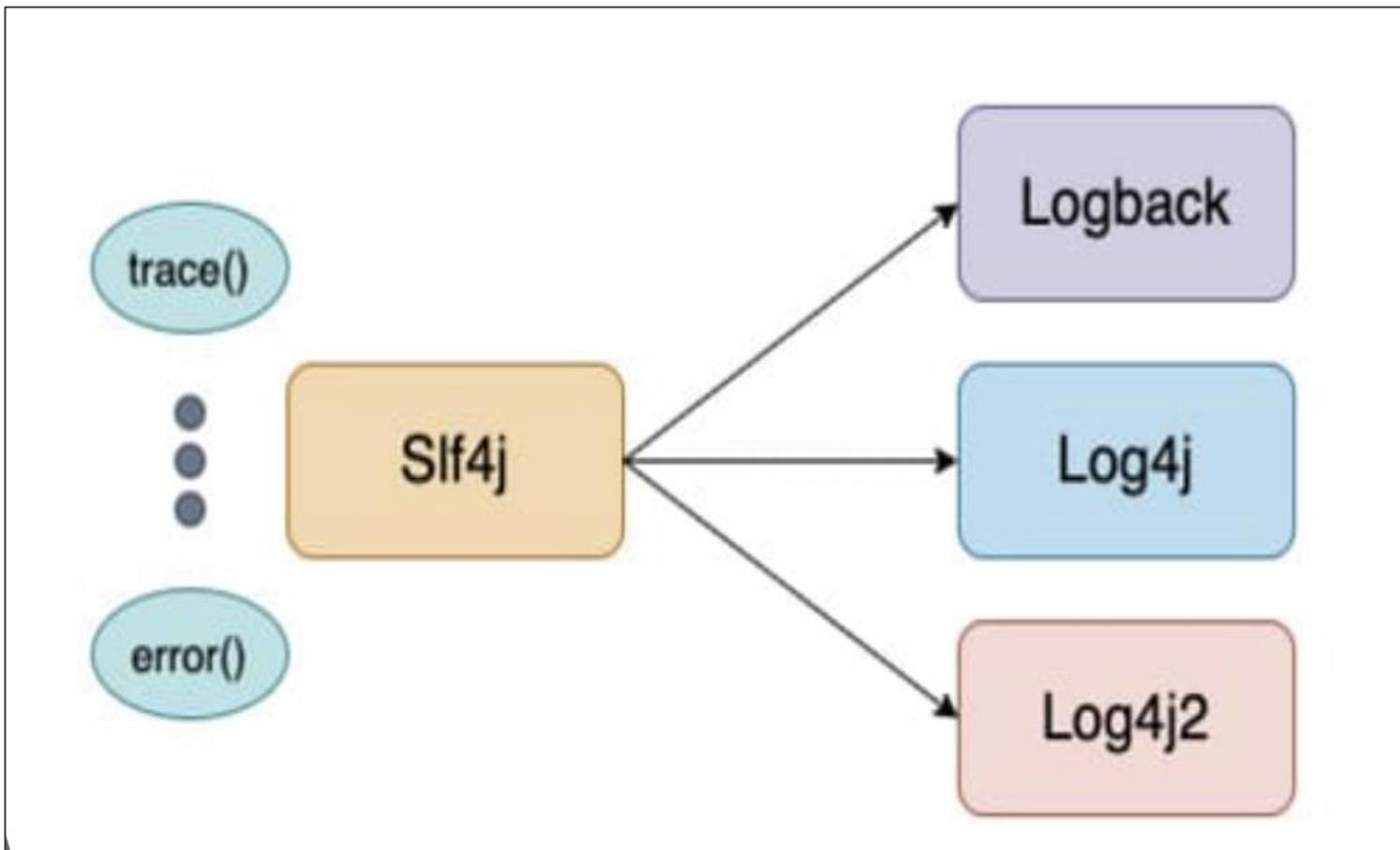
SLF4J(Simple Logging Facade for Java)

- SLF4J (Simple Logging Facade for Java)는 Java 애플리케이션에서 다양한 로깅 프레임워크에 대한 공통 인터페이스를 제공하는 로깅 추상화 라이브러리입니다.
- SLF4J는 사용자가 원하는 로깅 프레임워크를 선택하고 그에 맞는 구현체를 쉽게 바꿀 수 있게 해줍니다. 예를 들어, Logback, Log4j, java.util.logging 등과 함께 사용할 수 있습니다.
- SLF4J의 주요 특징
 - 추상화: SLF4J는 다양한 로깅 프레임워크에 대해 단일 API를 제공합니다. 애플리케이션에서 SLF4J를 사용하면 실제 로깅 구현체를 바꾸더라도 코드를 변경할 필요가 없습니다.
 - 정적 바인딩: SLF4J는 컴파일 시점에 로깅 구현체를 결정합니다. 따라서 특정 로깅 구현체가 필요할 때 관련 의존성을 추가하는 것으로 쉽게 사용할 수 있습니다.
 - 파라미터화된 로그 메시지: SLF4J는 로그 메시지를 파라미터화하여 성능을 향상시킬 수 있는 기능을 제공합니다. 이로 인해 불필요한 문자열 결합을 피할 수 있습니다.

SLF4J

- SLF4J의 장점
 - 로깅 프레임워크에 대한 종속성을 줄이고, 애플리케이션의 유연성을 높일 수 있습니다.
 - 다양한 로깅 구현체와 함께 사용할 수 있어 필요에 따라 쉽게 변경할 수 있습니다.
 - 성능 개선 및 코드 가독성을 높일 수 있습니다.
- spring boot 에서는 기본적으로 slf4j와 logback 이 연동되어 있습니다.

slf4f



출처: <https://deeplify.dev/back-end/spring/logging>

Spring Boot LogBack 적용하기

- Logback은 로그를 관리하는 기본 로깅 프레임워크로, SLF4J(Simple Logging Facade for Java)를 통해 로그 메시지를 출력합니다. Logback은 고성능과 유연한 설정을 제공하며, XML 또는 Groovy 형식으로 구성할 수 있습니다.
- 의존성
- implementation group: 'org.springframework.boot', name: 'spring-boot-starter-logging', version: '3.3.4'

로그백 설정파일

```
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>
        </encoder>
    </appender>
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>logs/myapp.log</file>
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>
        </encoder>
    </appender>
    <logger name="com.example" level="DEBUG">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
    </logger>
    <appender name="ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>logs/myapp.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>logs/myapp-%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>30</maxHistory> <!-- 최근 30일의 로그 파일 유지 -->
        </rollingPolicy>
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>
        </encoder>
    </appender>
    <root level="INFO">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
    </root>
</configuration>
```

logback-spring.xml 파일을 resources 아래에 둔다
파일이 없다고 로그가 출력이 안되는것은 아니다 다만
원하는 상태로 조율할 수 있다.

로그백

- Appender: 로그를 출력할 대상을 정의합니다. 콘솔 또는 파일에 출력할 수 있습니다. ConsoleAppender, FileAppender, RollingFileAppender 등이 있습니다.
- ConsoleAppender: 콘솔에 로그를 출력합니다.
- RollingFileAppender: 파일에 로그를 기록하고, 일정한 크기나 날짜가 지나면 새로운 파일로 롤링합니다.
- Pattern: 로그 출력 형식을 지정합니다. %d, %level, %logger, %msg 등의 패턴을 사용하여 로그의 형식을 지정할 수 있습니다.
 - %d{yyyy-MM-dd HH:mm:ss}: 로그 출력 시 날짜와 시간을 표시합니다.
 - %level: 로그 레벨 (예: INFO, DEBUG, ERROR 등)을 출력합니다.
 - %logger{36}: 로그를 기록한 클래스의 이름을 표시합니다.
 - %msg: 실제 로그 메시지를 출력합니다.

로그백

- Level: 로그 레벨을 지정합니다. 로그 레벨은 다음과 같은 순서로 설정됩니다.
 - TRACE: 가장 상세한 로그. 거의 모든 동작을 기록합니다.
 - DEBUG: 개발할 때 주로 사용하는 디버그 레벨의 로그입니다.
 - INFO: 일반적인 운영 환경에서 사용할 수 있는 정보성 로그입니다.
 - WARN: 주의가 필요한 상황을 나타냅니다.
 - ERROR: 오류가 발생했을 때 기록합니다.
 - OFF: 모든 로그를 끕니다.
- RollingPolicy: 로그 파일의 롤링 정책을 설정합니다. 시간 기반 또는 크기 기반으로 설정할 수 있습니다. 위 예시에서는 날짜 기반으로 롤링되며, maxHistory 속성을 사용해 30일 동안의 로그 파일을 유지합니다.

사용 예

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

3개 사용 위치
@Service
public class TestService {

    4개 사용 위치
    private static final Logger logger = LoggerFactory.getLogger(TestService.class);

    1개 사용 위치
    public void performTask() {
        logger.debug("디버깅 메시지");
        logger.info("정보 메시지");
        logger.warn("경고 메시지");
        logger.error("오류 메시지");
    }
}
```

Spring Security

Spring Security

- **Spring Security**는 스프링 프레임워크 기반 애플리케이션에서 인증(authentication)과 권한 부여(authorization)를 제공하는 강력한 보안 프레임워크입니다. 이를 통해 애플리케이션의 보안 요구 사항을 쉽게 설정하고 구현할 수 있습니다.
- Spring Security는 주로 웹 애플리케이션이나 RESTful API의 보안을 강화하기 위해 사용됩니다.

기능

- **인증(Authentication)**: 사용자가 누구인지를 확인하는 과정입니다. 일반적으로 로그인 과정에서 사용자 자격 증명(예: 사용자 이름, 비밀번호)을 확인하고, 이를 통해 사용자를 인증합니다.
- **권한 부여(Authorization)**: 인증된 사용자가 애플리케이션에서 어떤 자원을 사용할 수 있는지를 결정하는 과정입니다. 예를 들어, 관리자만 특정 페이지에 접근할 수 있게 할 수 있습니다.
- **보안 필터(Security Filters)**: Spring Security는 여러 개의 보안 필터를 제공하며, 이 필터들이 HTTP 요청을 가로채고 인증 및 권한 부여를 처리합니다.
- **세션 관리(Session Management)**: 인증된 사용자를 유지하기 위한 세션 관리 기능을 제공합니다. 세션 고정 공격(Session Fixation) 방지 기능도 기본적으로 제공됩니다.
- **OAuth 2.0 / OpenID Connect 지원**: Spring Security는 OAuth 2.0과 OpenID Connect 표준을 지원하여 소셜 로그인(Google, Facebook 등)이나 외부 인증 서버를 통한 인증을 쉽게 설정할 수 있습니다.
- **CSRF 보호**: Cross-Site Request Forgery(CSRF) 공격을 방지하는 기능을 제공합니다. CSRF는 사용자의 권한을 이용해 의도치 않은 작업을 수행하는 공격 방식입니다.
- **암호화 및 비밀번호 관리**: Spring Security는 비밀번호를 안전하게 저장하기 위해 암호화 알고리즘(예: BCrypt, SCrypt)을 제공합니다.

Spring Security의 동작 원리

- Spring Security는 주로 필터 기반 아키텍처를 사용합니다. 이 필터들이 HTTP 요청을 가로채서, 요청이 적합한지 판단하고 적절한 보안 작업을 수행합니다.
- SecurityContext: Spring Security는 모든 인증 정보를 SecurityContext라는 객체에 저장합니다. 애플리케이션 전역에서 인증 정보를 참조하려면 이 컨텍스트를 사용합니다. 이 객체는 세션에 저장됩니다.
- AuthenticationManager: 인증 과정을 처리하는 핵심 구성 요소입니다. 사용자의 자격 증명(예: 사용자 이름과 비밀번호)이 유효한지 확인하고 인증을 완료합니다.
- UserDetailsService: 사용자 정보를 가져오는 인터페이스입니다. 주로 데이터베이스에서 사용자의 자격 증명을 확인하기 위해 사용됩니다.

Spring Security 설정

- implementation group: 'org.springframework.boot', name: 'spring-boot-starter-security', version: '3.3.4'
- 이전 버전에서는 WebSecurityConfigurerAdapter 을 상속받아 처리했으나 Spring Security 6 이상에서는 deprecated 되었습니다. 대신, 보안 설정을 SecurityFilterChain과 SecurityConfigurer 클래스를 통해 구성합니다.

SecurityConfig

```
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/public/**").permitAll() // "/public" 경로는 인증 없이 접근 가능, 이전버전에서는 anyRequest 를 사용함
                .anyRequest().authenticated()           // 그 외의 경로는 인증이 필요
            )
            .formLogin(form -> form.permitAll())
            )
            .logout(logout -> logout.permitAll()); // 로그아웃도 인증 없이 접근 가능
        return http.build();
    }
}
```

SecurityConfig

```
@Bean
public UserDetailsService userDetailsService() {
    // In-memory 사용자 추가
    UserDetails user = User.builder()
        .username("user")
        .password(passwordEncoder().encode("1234"))
        .roles("USER")
        .build();

    UserDetails admin = User.builder()
        .username("admin")
        .password(passwordEncoder().encode("1234"))
        .roles("ADMIN")
        .build();

    return new InMemoryUserDetailsManager(user, admin); //memory 디비 , 메모리에 넣어놓는다
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(); // BCryptPasswordEncoder 사용
}
```

SecurityConfig

- SecurityFilterChain

SecurityFilterChain을 통해 보안 규칙을 설정합니다. HttpSecurity 객체를 사용하여 다양한 보안 정책을 설정할 수 있으며, 마지막에 http.build()로 필터 체인을 반환합니다.

- authorizeHttpRequests

authorizeHttpRequests 메서드는 HTTP 요청에 대한 접근 규칙을 정의합니다. 특정 경로를 누구나 접근할 수 있도록 허용하거나(permitAll), 인증된 사용자만 접근할 수 있도록 제한할 수 있습니다(authenticated).

- formLogin

로그인 페이지와 관련된 설정입니다. 로그인 폼을 커스터마이징하거나 기본 폼을 사용할 수 있습니다.

- UserDetailsService

UserDetailsService는 사용자 정보를 로드하는 역할을 합니다. 여기서는 InMemoryUserDetailsManager를 사용해 메모리 내 사용자 계정을 정의했지만, 실제로는 데이터베이스나 외부 서비스에서 사용자 정보를 로드할 수 있습니다.

- PasswordEncoder

비밀번호를 안전하게 저장하기 위해 BCryptPasswordEncoder를 사용하여 비밀번호를 암호화합니다.

폼인증

- Spring Security 는 폼인증을 기본으로 제공합니다.
- Spring Security의 폼 인증(Form-Based Authentication)은 사용자가 웹 애플리케이션에 로그인할 수 있도록 지원하는 주요 메커니즘 중 하나입니다. Spring Security는 기본적으로 여러 가지 메서드와 엔드포인트를 제공하여 인증 과정을 처리하며, 이를 통해 개발자는 복잡한 보안 로직을 직접 구현하지 않고도 손쉽게 보안을 강화할 수 있습니다.
- Spring Security에서 폼 인증은 HttpSecurity를 통해 설정되며, 다양한 기본 메서드들을 제공합니다.

로그인 페이지 요청 (GET 요청)

- 기본적으로 Spring Security는 내장된 로그인 페이지를 제공합니다.
사용자 정의 로그인 페이지를 사용하려면 보안 설정에서 `loginPage` 메서드를 사용하여 커스터마이징할 수 있습니다.

예시)

```
http
    .authorizeHttpRequests(auth -> auth
        .requestMatchers("/public/**").permitAll()
        //로그온을 안하고 접근 가능한 페이지를 지정한다
        .anyRequest().authenticated()
    )
    .formLogin(form -> form
        .loginPage("/custom-login") // 사용자 정의 로그인 페이지
        .permitAll()
    );
);
```

로그인하기(user/1234)

Please sign in

Spring Security 의 품인증절차

1. 사용자가 로그인 폼에 아이디와 비밀번호 입력 후 제출한다.
2. UsernamePasswordAuthenticationFilter가 요청을 가로채고, 자격 증명을 추출한다.
3. 추출된 자격 증명은 AuthenticationManager에게 전달된다.
4. AuthenticationManager는 UserDetailsService를 사용하여 사용자 정보를 가져온다.
5. 사용자 비밀번호를 암호화 방식으로 비교하여 검증한다.
6. 인증 성공 시, SecurityContextHolder에 인증된 사용자 정보 저장한다.
7. 인증 성공 시 원래 요청했던 페이지로 리다이렉트 한다.
8. 세션이나 Remember Me로 로그인 상태를 유지한다.
9. 각 요청에 대해 권한 검사를 통해 접근 허용 또는 거부를 진행한다.

주요클래스

1. UsernamePasswordAuthenticationFilter

폼 인증 요청을 가로채고, 자격 증명을 처리하는 필터이다.

2. AuthenticationManager

인증 요청을 처리하는 주체, 일반적으로 ProviderManager가 사용된다.

3. AuthenticationProvider

인증 처리 로직을 제공하는 컴포넌트로, 기본적으로 'DaoAuthenticationProvider'가 사용된다.

4. UserDetailsService

사용자 정보를 로드하는 서비스이다. 커스터마이징하여 필요한 내용을 입력하면된다.

5. PasswordEncoder

비밀번호 암호화 및 비교를 담당하는 인터페이스이다.

6. SecurityContextHolder

인증된 사용자 정보를 보관하는 컨테이너 이다. 세션에 저장한다.

폼 인증 절차

1. 로그인 폼 접근

사용자가 보호된 리소스(예: /user/profile 같은 URL)에 접근하려 하면, Spring Security는 해당 요청이 인증되지 않았다는 사실을 인지합니다.

Spring Security는 기본적으로 제공하는 로그인 페이지(또는 커스텀 로그인 페이지)로 리다이렉트합니다. 이 로그인 페이지는 사용자에게 아이디와 비밀번호를 입력하도록 요구합니다.

URL 예시: /login

2. 로그인 요청 처리

사용자가 로그인 폼에 아이디와 비밀번호를 입력하고 로그인 버튼을 클릭하면, 이 정보는 POST 요청으로 서버에 전송됩니다. 기본적으로 Spring Security는 /login URL로 폼 데이터를 처리합니다.

이 경로는 UsernamePasswordAuthenticationFilter에 의해 가로채어 집니다.

3. UsernamePasswordAuthenticationFilter 동작

이 필터는 로그인 폼에서 제출된 username과 password를 추출하여 Authentication 객체를 생성합니다. Authentication 객체는 사용자 자격 증명을 포함한 토큰으로, 이 토큰은 인증 프로세스에서 사용됩니다.

폼 인증 절차

4. AuthenticationManager에게 인증 위임:

Authentication 객체는 AuthenticationManager에게 전달되며, Spring Security에서 기본으로 사용하는 구현체는 ProviderManager입니다. ProviderManager는 여러 개의 AuthenticationProvider를 통해 인증을 시도합니다.

5. UserDetailsService 호출:

DaoAuthenticationProvider가 기본적으로 동작하며, 이 클래스는 UserDetailsService를 사용하여 사용자 정보를 조회합니다. UserDetailsService는 사용자 이름을 기반으로 데이터베이스나 인메모리에서 사용자 정보를 불러오고, 해당 정보를 UserDetails 객체로 반환합니다.

@Bean

```
public UserDetailsService userDetailsService() {  
    return new InMemoryUserDetailsManager(  
        User.withUsername("user")  
            .password(passwordEncoder().encode("password"))  
            .roles("USER")  
            .build()  
    );  
}
```

디비로부터 사용자 정보를 가져오려면 UserDetailsService 클래스를 커스터마이징하고 loadByUserName 메서드를 오버라이딩해야 합니다.

폼 인증 절차

6. 비밀번호 검증

사용자 정보를 불러온 후, DaoAuthenticationProvider는 사용자가 입력한 비밀번호와 데이터베이스에 저장된 비밀번호(암호화된 형태)를 비교합니다. Spring Security는 PasswordEncoder 인터페이스를 통해 비밀번호를 암호화하여 비교합니다. 기본적으로 BCryptPasswordEncoder가 자주 사용됩니다.

```
@Bean  
  
public PasswordEncoder passwordEncoder() {  
  
    return new BCryptPasswordEncoder();  
  
}
```

7. 인증 성공/실패 처리

1) 성공 시: 인증에 성공하면 Authentication 객체가 생성되고, 이 객체는 Spring Security의 SecurityContext에 저장됩니다. SecurityContextHolder는 현재 인증된 사용자의 정보를 보관하는 역할을 합니다.
사용자는 원래 접근하려고 했던 리소스로 리다이렉트됩니다.

2) 실패 시: 자격 증명이 실패하면 AuthenticationFailureHandler가 호출되며, 기본적으로 로그인 페이지로 리다이렉트되고 에러 메시지가 전달됩니다.

폼인증절차

8. 세션 및 Remember Me 처리:

Spring Security는 기본적으로 세션을 사용하여 인증 정보를 저장합니다. 사용자가 로그인한 상태를 유지하려면 이 세션을 통해 관리됩니다.

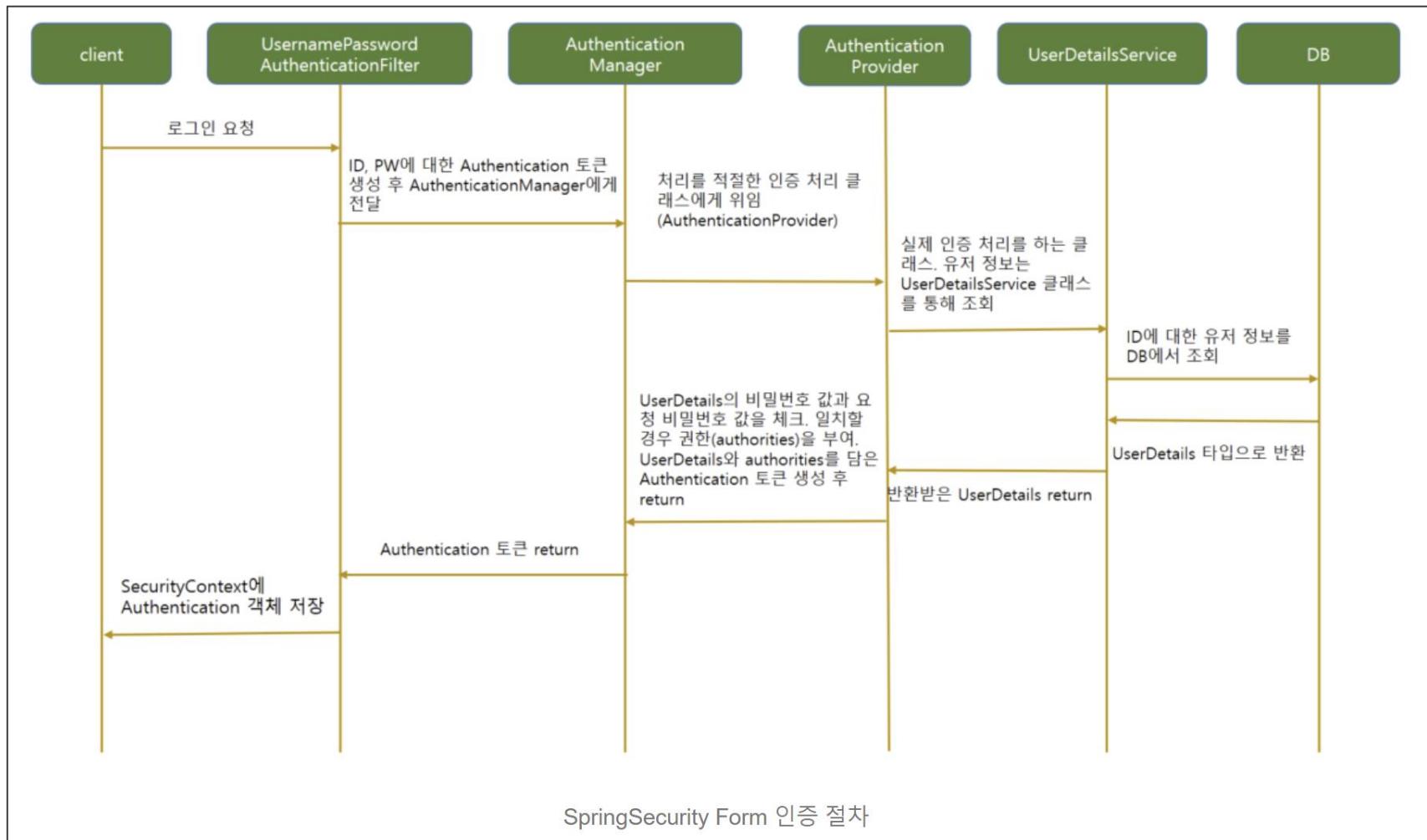
Remember Me 기능을 사용하면, 세션이 만료된 후에도 사용자의 인증 정보를 유지할 수 있습니다. 이는 쿠키 기반으로 작동합니다.

9. 인가 처리:

인증이 성공하면 사용자는 시스템에 로그인한 상태가 되고, 이제 인가(Authorization) 단계로 넘어갑니다.

각 요청에 대해 SecurityContext에서 인증된 사용자 정보를 기반으로 리소스에 대한 접근 권한을 확인합니다. 권한이 충분하지 않으면 AccessDeniedException이 발생하고, 권한이 있으면 리소스에 접근할 수 있습니다.

폼인증절차



SecurityContext



출처: <https://wildeveloperetrain.tistory.com/163>

SecurityContextHolder

- SecurityContextHolder는 SecurityContext를 세션에 저장하고 사용자의 요청마다 이를 참조합니다.
- 세션 기반 저장: SecurityContextHolder는 SecurityContext를 세션에 저장합니다. 이렇게 하면 사용자가 여러 요청을 보낼 때마다 인증 정보를 유지할 수 있습니다. 세션이 만료되거나 사라지면, SecurityContext도 더 이상 유지되지 않습니다.
- 커스텀 전략: SecurityContextHolder는 기본적으로 MODE_THREADLOCAL을 사용하여 각 스레드에 대한 인증 정보를 유지하지만, 필요하다면 다른 전략으로 설정할 수도 있습니다. 이를 통해 세션에 저장하지 않도록 하거나, 다른 방식으로 저장을 처리할 수 있습니다.
- 즉, 기본적으로 SecurityContextHolder는 세션에 인증 정보를 저장하여 각 요청마다 사용자의 인증 상태를 유지합니다.

SecurityContext

- SecurityContext는 현재 사용자의 인증 정보를 담고 있는 Authentication 객체를 보유합니다.
- 이 Authentication 객체는 사용자가 로그인했을 때 생성되며, 사용자 이름, 권한, 자격 증명 등을 포함합니다.
- 애플리케이션의 요청이 들어오면, Spring Security는 SecurityContext를 통해 해당 사용자가 인증되었는지 확인합니다. 인증이 되었다면, 이 SecurityContext는 사용자에 대한 정보를 반환해 요청을 처리할 수 있도록 돋습니다.
- Spring Security는 SecurityContextHolder를 통해 SecurityContext에 접근할 수 있도록 합니다. 이를 통해 애플리케이션 전반에서 인증 정보를 쉽게 조회할 수 있습니다.

사용자정의로그온페이지작성하기

- 필요한클래스
 - UserEntity : 사용자 정보가 저장되는 클래스 jpa 사용일때 가정
 - CustomUserDetails : UserDetails 클래스 상속, SecurityContext가 UserDetails 객체를 저장 함. 반드시 커스터마이징을 해야 하는것은 아니지만 UserDetails 클래스는 username, 암호화된 password, 권한만 저장됨, 별도의 추가 항목을 있을 경우에 커스터마이징 하는것이 좋음
 - UserRepository : 디비에 통신을 담당하기 위한 클래스
findByName 메서드 하나만 생성
 - CustomUserDetailsService : UserDetailsService를 상속
loadByName 함수를 오버라이딩 , UserDetails 객체를 반환하면
SecurityContextHolder 클래스를 통해 SecurityContext 에 저장된다.
 - LoginController : 사용자 페이지를 호출하기 위한 컨트롤러

사용자정의로그온페이지작성하기 2

- html 파일들
 - index.html – 메인페이지
 - login.html – 로긴 페이지
 - hello.html
 - home.html

login.html

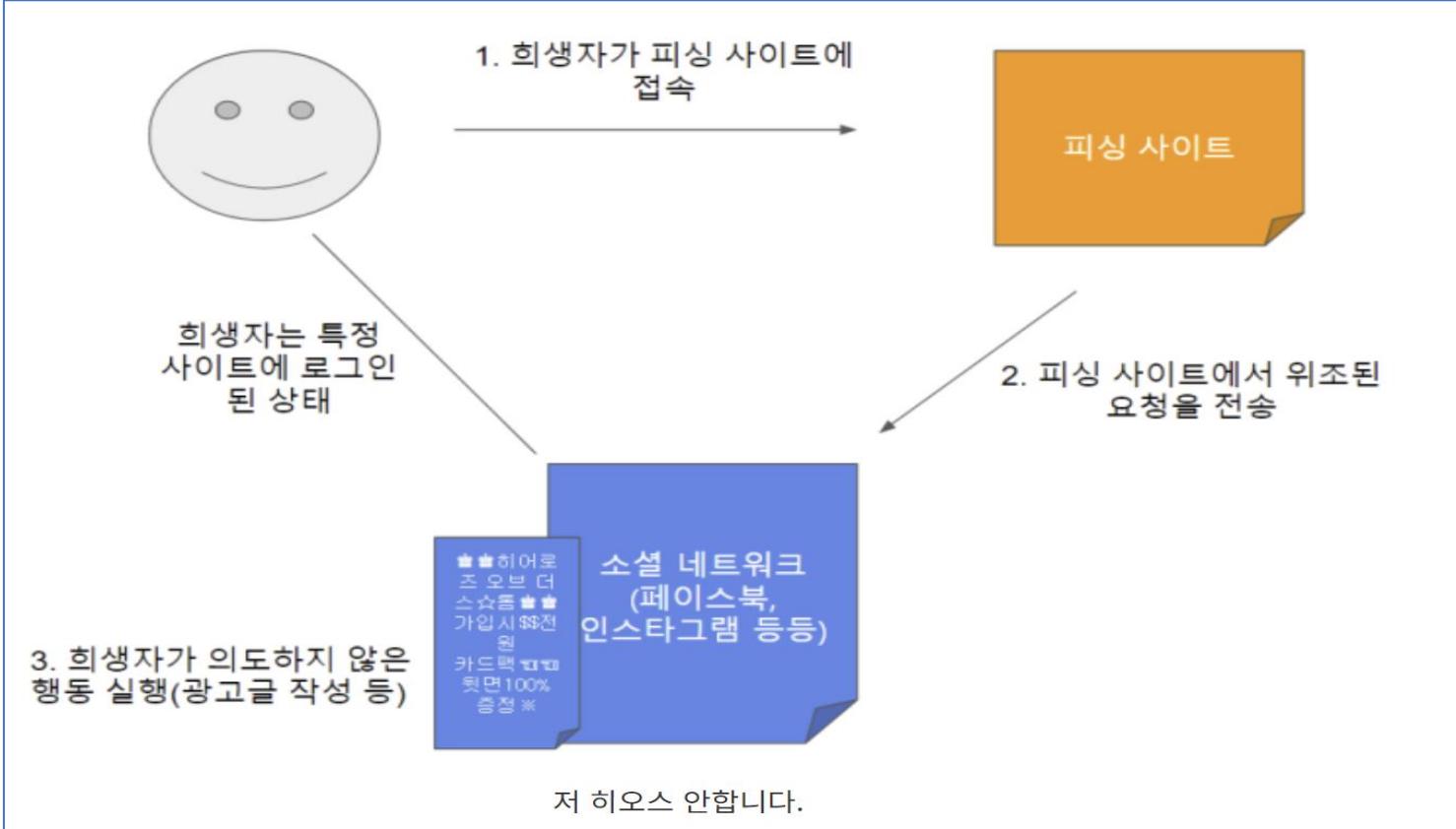
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
<div class="container">
    <h2>Login</h2>
    <form th:action="@{/login}" method="post">
        <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/> <!-- CSRF 토큰 -->
        <div class="form-group">
            <label for="username">Username</label>
            <input type="text" id="username" name="username" class="form-control" required>
        </div>
        <div class="form-group">
            <label for="password">Password</label>
            <input type="password" id="password" name="password" class="form-control" required>
        </div>
        <button type="submit" class="btn btn-primary">Login</button>
    </form>
</div>
</body>
</html>
```

CSRF 토큰 필요

csrf란

- CSRF는 Cross Site Request Forgery(사이트 간 요청 위조)의 줄임말로 웹 취약점 중 하나입니다.
- 공격자가 희생자의 권한을 도용하여 특정 웹 사이트의 기능을 실행하게 할 수 있으며 이는 희생자의 의도와는 무관하게 이루어집니다.
- CSRF 취약점을 이용하면 공격자가 희생자의 계정으로 네이버 카페나 인스타그램, 페이스북 등 다수의 방문자가 있는 사이트에 광고성 혹은 유해한 게시글을 업로드하는 것도 가능해집니다.

csrf 절차



출처: <https://tibetsandfox.tistory.com/11>

csrf란

- 희생자가 위조 요청을 보낼 사이트에 로그인 되어있는 상태로 피싱 사이트에 접속합니다.
- 공격자는 피싱 사이트 접속 유도를 위해 피싱 메일, 팝업 광고를 띄우는 등의 행동을 합니다.
- 희생자가 피싱 사이트에 접속하면 피싱 사이트에서 희생자로 가장하여 요청을 위조해 전송합니다.
- 위조 요청을 받은 사이트는 해당 요청에 대한 응답을 하게 되고 이로 인해 희생자가 의도하지 않은 행동이 실행됩니다.

JWT 처리

토큰기반인증

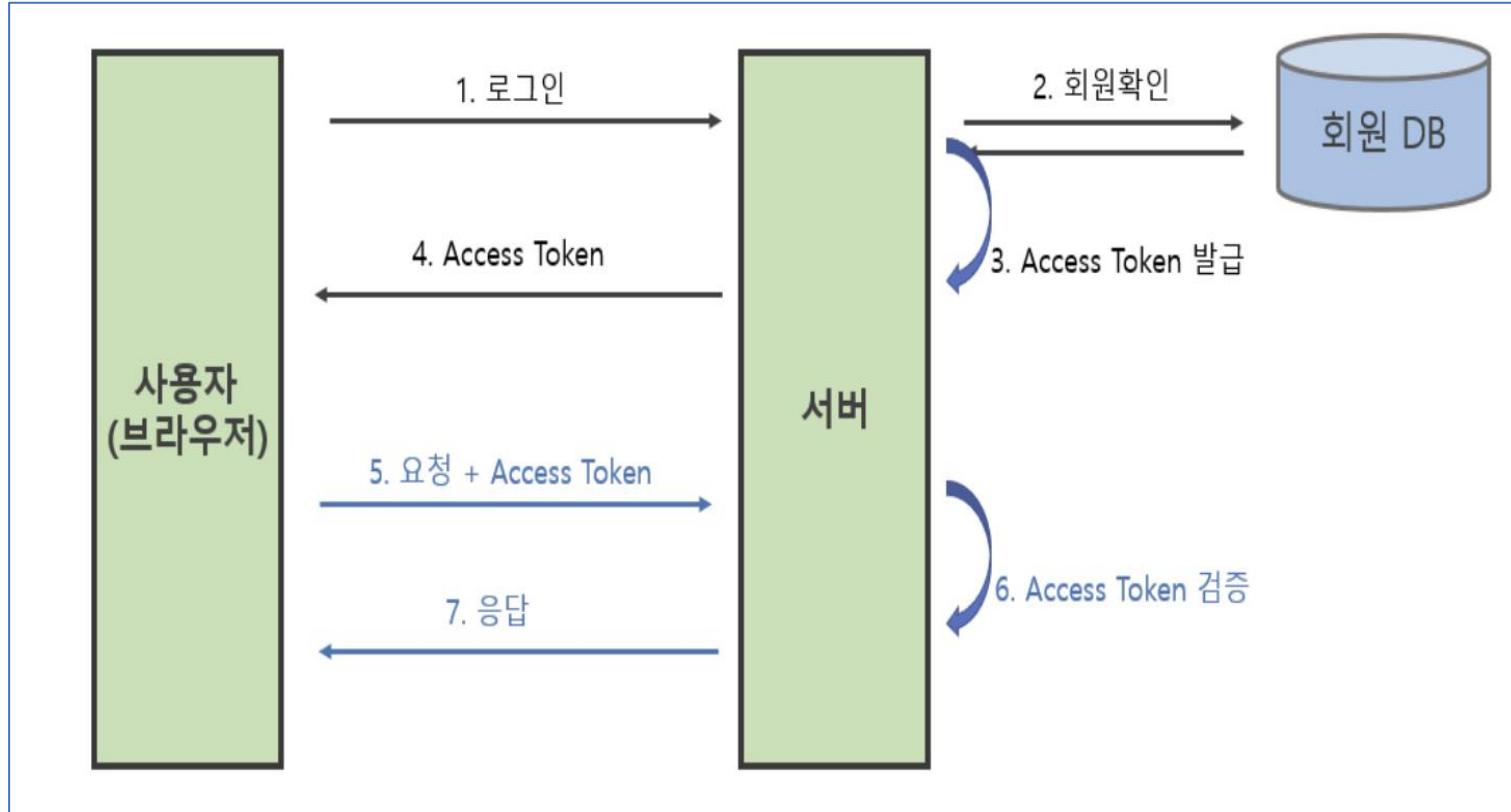
- 토큰 인증(Token-based Authentication)은 사용자 인증을 위해 세션 대신 토큰을 사용하는 방식입니다. 이 방식은 특히 RESTful API, 모바일 애플리케이션, 클라우드 서비스 같은 Stateless(상태저장을 하지 않는) 환경에서 많이 사용됩니다. 토큰은 사용자가 인증에 성공하면 서버에서 생성하여 클라이언트에게 제공하며, 클라이언트는 이후의 요청에서 이 토큰을 서버에 전달해 인증된 사용자임을 증명합니다.
- 자바풀스택 개발에서 프론트앤드와 백엔드가 분리되서 개발하기 때문에 백엔드와 프론트앤드는 서로 다른 사이트이므로 세션을 이용한 로그인 기능을 구현할 수 없습니다. 그래서 품인증 대신에 토큰기반 인증방식을 사용해야 합니다

토큰기반 인증절차

1. 사용자 로그인 요청 - 사용자가 사용자 이름과 비밀번호를 이용해 로그인 요청을 보냅니다.
서버는 해당 정보가 유효한지 확인합니다.
2. 토큰 발급 - 서버가 사용자의 인증을 확인한 후, 특정 형식의 토큰을 발급합니다.
일반적으로 이 토큰은 JWT (JSON Web Token) 형식이 많이 사용됩니다. 토큰에는 사용자 정보 및 만료 시간 등의 정보가 포함되어 있으며, 서버에서 서명하여 변조되지 않도록 보호됩니다.
3. 클라이언트에서 토큰 저장 - 클라이언트는 발급받은 토큰을 LocalStorage, SessionStorage, 쿠키등에 저장합니다.
이후 요청 시마다 이 토큰을 포함해 서버에 요청을 보냅니다.
4. 요청 시 토큰 전송 - 클라이언트는 보호된 API 또는 리소스에 접근할 때 Authorization 헤더에 토큰을 포함해 요청을 보냅니다.

```
http    Authorization: Bearer <JWT 토큰>
```
5. 서버에서 토큰 검증 - 서버는 요청을 받을 때마다 토큰의 유효성을 검증합니다.
서명이 올바른지, 토큰이 만료되지 않았는지, 해당 토큰이 변조되지 않았는지 확인합니다.
6. 인증된 요청 처리 - 토큰이 유효하다면 서버는 사용자의 요청을 처리하고, 유효하지 않으면 적절한 오류 메시지를 반환합니다.
7. 토큰 만료 및 갱신 - 토큰은 일반적으로 만료 시간이 설정되어 있습니다. 만료된 토큰으로는 요청을 할 수 없으며, 클라이언트는 필요 시 서버에 새로운 토큰을 요청해야 합니다.

토큰기반 인증절차



출처 : <https://doing7.tistory.com/88>

токен기반인증방식의 장점

1. Stateless

서버가 사용자 세션을 저장하지 않기 때문에 확장성이 좋고, 여러 서버에서 쉽게 부하 분산이 가능합니다. 각 요청은 독립적이며 서버는 상태를 기억할 필요가 없습니다.

2. 보안성

токен에는 서명이 포함되어 있어, 그 내용이 클라이언트에서 변조될 수 없습니다. HTTPS와 결합하여 사용하면 안전하게 사용할 수 있습니다.

3. 다양한 클라이언트 지원

웹 애플리케이션, 모바일 애플리케이션, IoT 등 여러 플랫폼에서 동일한 방식으로 인증을 구현할 수 있습니다.

4. API와의 호환성

RESTful API와 같은 비상태(Stateless) 방식의 통신에 적합합니다. 요청에 토큰을 추가해 인증을 처리하기 때문에, API 서버는 인증 상태를 따로 관리할 필요가 없습니다.

5. CSRF 공격 방지

일반적으로 API 호출 시에는 토큰을 사용하기 때문에, 세션 쿠키를 사용하지 않으며 CSRF(Cross-Site Request Forgery) 공격에 대한 방어가 자연스럽게 이루어집니다.

6. 유연성

토큰은 사용자 정보를 담고 있으므로, 권한 및 추가 정보를 쉽게 포함할 수 있습니다. 이로 인해 인증 외에도 권한 확인, 사용자 상태 등을 관리할 수 있습니다.

토큰기반인증방식의 단점

1. 토큰 탈취 위험

토큰이 탈취될 경우, 만료 전까지는 그 토큰으로 서버에 요청을 보낼 수 있어 보안 문제가 발생할 수 있습니다. 이 문제는 HTTPS로 해결할 수 있으며, 짧은 만료 시간을 설정하고 토큰 갱신(refresh token)을 통해 어느 정도 방지할 수 있습니다.

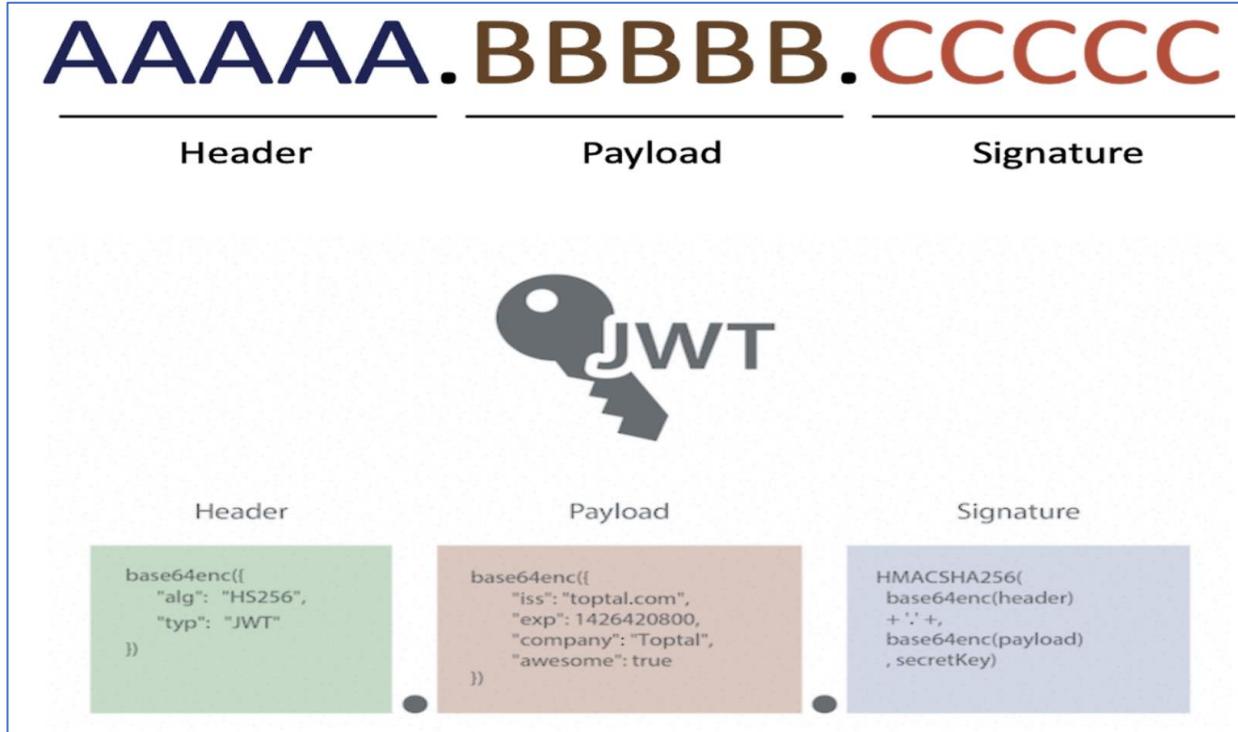
2. 토큰 무효화의 어려움

토큰 자체는 서버에 저장되지 않으므로, 사용자가 로그아웃하거나 토큰을 강제로 무효화하는 등의 작업이 어렵습니다. 이를 해결하기 위해 토큰 블랙리스트나 리프레시 토큰을 사용하는 방법이 있습니다.

JWT란

- Json Web Token 의 약자
- 토큰 기반 인증 중 가장 많이 사용되는 것이 JWT(JSON Web Token)입니다.
- JWT는 (인증)정보를 담고있는 JSON 객체를 암호화시키고 HTTP 헤더에 추가하여 (인증)정보를 안전하게 전송합니다.
- 이때 암호화는 비밀키(HMAC)또는 공개키/개인키쌍(RSA)을 이용합니다.
- JWT는 개방형 표준(RFC 7519)입니다. 따라서 Java, Ruby, PHP, Python 등 여러 환경에서 지원이 가능합니다. 구글, 마이크로소프트와 같은 수많은 회사에서 이를 사용하고 있습니다.

JWT의 구조



출처 : <https://tistory.slowtuttle.co.kr/entry/JSON-Web-Token-%EC%9D%B4%EB%9E%80>

Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side.

Algorithm HS256 ▾

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG9lIiwiawF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

⌚ Signature Verified

SHARE JWT

jwt의 구조

- JWT는 Header, Payload, Signature로 이루어져있습니다.
- Header에는 토큰의 타입이 JWT라는 것과 무슨 해싱알고리즘(HMAC/RSA)을 사용하였는지 정보가 담겨있습니다.
- Payload에는 사용자의 인증정보가 담겨져있습니다.
- Signature에는 전자서명이 들어가게됩니다. 이 서명은 헤더에서 명시한 해싱을 통해 생성됩니다.

Spring boot 에서 jwt 절차

1. 의존성추가
2. TokenProvider 클래스

JWT 생성 및 검증 로직을 처리합니다.

3. JWT 필터 생성

JWT를 검증하는 필터 클래스를 생성하여, 모든 요청에서 JWT를 확인하고 유효성을 검사합니다.

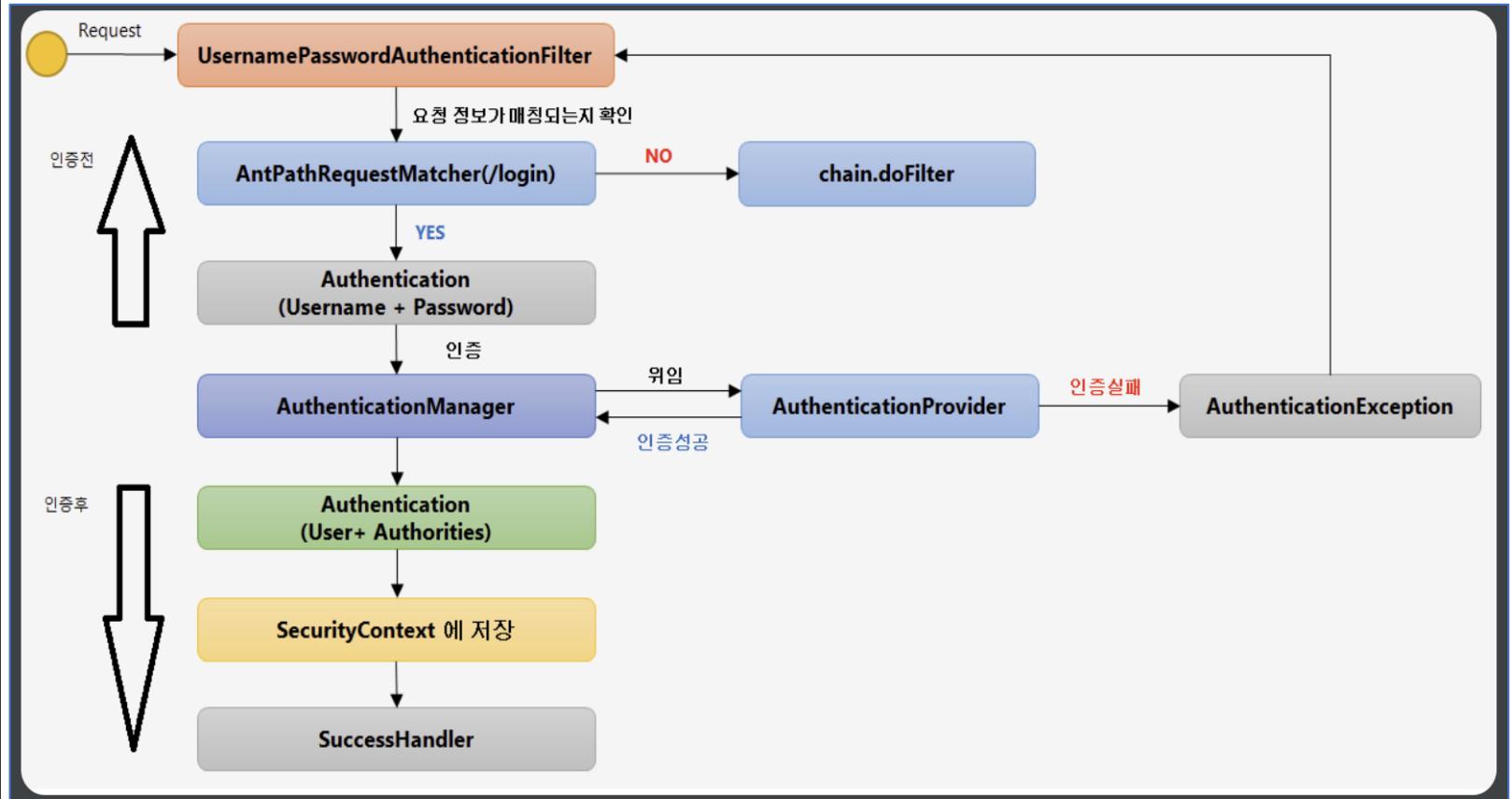
4. Spring Security 설정

SecurityConfig에서 JWT 필터를 적용하도록 설정합니다

5. JWT 토큰 발급 API

JWT 토큰을 발급하는 엔드포인트를 추가합니다.

Spring boot 에서 jwt절차



출처:<https://tscofet.oopy.io/a183712d-b21c-48a3-b4ab-8b4ce27067b3>

Spring boot filter

- **Filter**는 HTTP 요청 및 응답을 가로채서 처리하거나 수정할 수 있는 강력한 메커니즘입니다. 필터는 요청이 컨트롤러에 도달하기 전 또는 응답이 클라이언트에 도달하기 전에 추가적인 로직을 실행할 수 있도록 도와줍니다.
- 주로 인증 및 권한 검사, 로깅, CORS 처리 등에 사용됩니다.

사용자 필터 작성하기

```
package com.namgarambooks.myhome.filter;

import jakarta.servlet.*;
import org.springframework.stereotype.Component;

import java.io.IOException;

0개의 사용위치
@Component //필터로 등록됨
public class MyCustomFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    0개의 사용위치
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        System.out.println("Request intercepted by custom filter");
        chain.doFilter(request, response);
    }

    @Override
    public void destroy() {
    }
}
```

filter 등록

- @Component
- @FilterRegistrationBean을 사용한 등록

```
@Configuration
public class FilterConfig {

    @Bean
    public FilterRegistrationBean<MyCustomFilter> loggingFilter() {
        FilterRegistrationBean<MyCustomFilter> registrationBean = new FilterRegistrationBean<>();

        registrationBean.setFilter(new MyCustomFilter());
        registrationBean.addUrlPatterns("/api/*"); // 필터를 적용할 URL 패턴
        registrationBean.setOrder(1); // 필터의 우선순위 (낮을수록 먼저 실행)

        return registrationBean;
    }
}
```

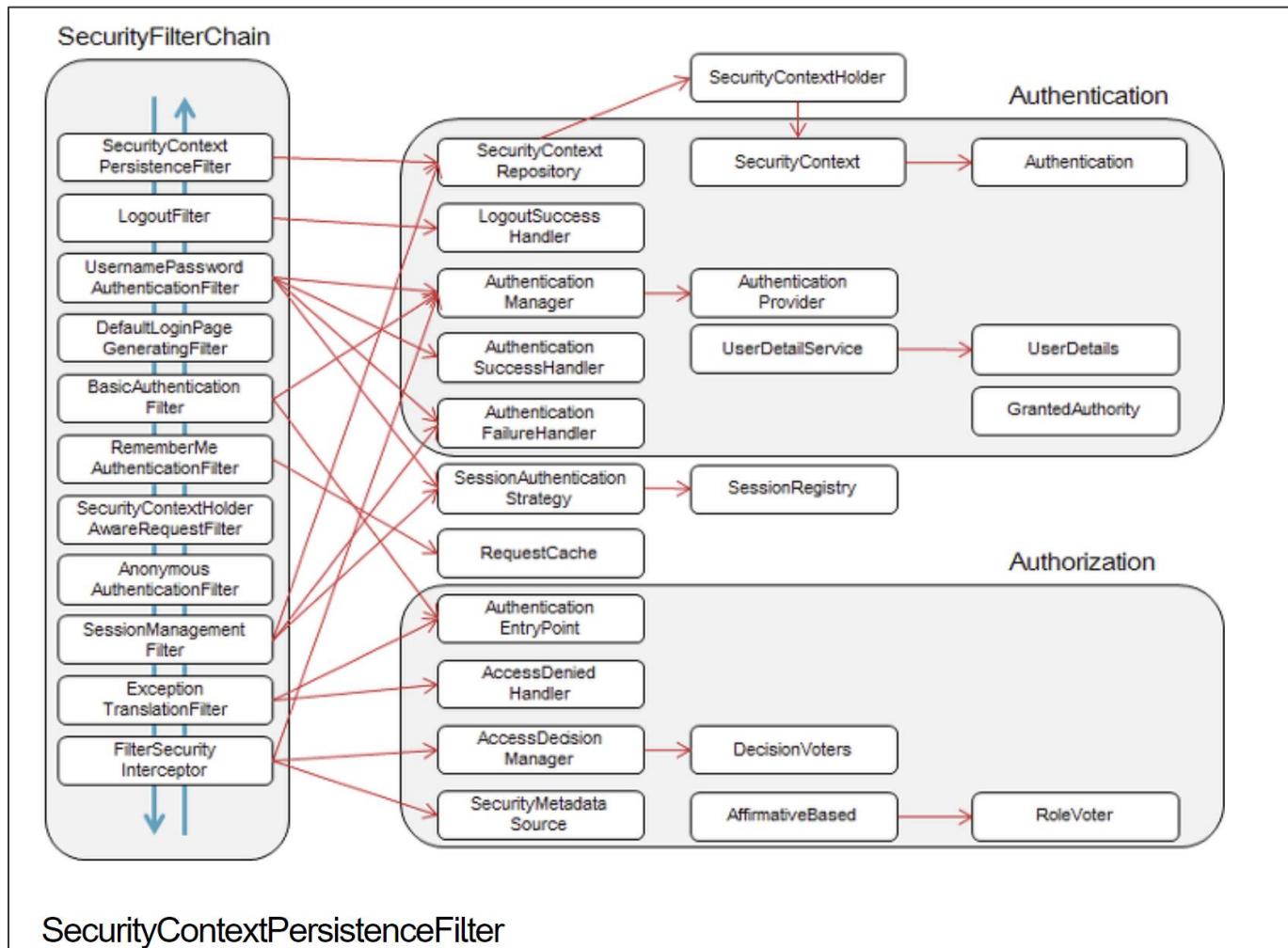
Spring Security 필터 체인과 통합

- Spring Security를 사용하는 경우, 보통 필터는 Spring Security 필터 체인 내에서 동작합니다. JWT 인증 필터나 로그인 필터 등을 Spring Security 필터 체인에 포함 시켜 적용할 수 있습니다.

필터실행순서제어

- @Order 또는 FilterRegistrationBean#setOrder()를 사용해 필터의 실행 순서를 제어 할 수 있다.

SecurityFilterChain



Spring boot 에서 jwt 절차

1. 의존성추가

```
implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
```

```
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2'
```

```
runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.2'
```

```
// JSON 처리용
```

폴더구조

```
java
  com.example.demo
    config
      CorsConfig
      SecurityConfig
    controller
      UserController
    dto
      AuthenticationRequest
      AuthenticationResponse
    entity
      MemberDto
    exception
      CustomNotFoundException
      ErrorResponseDTO
      GlobalExceptionHandler
      JwtAuthenticationEntryPoint
    jwt
      JwtFilter
      TokenProvider
    repository
      UserDao
    service
      UserService
    util
      SecurityUtil
  MyappJwtApplication
```

클래스는 별도로

TokenProvider

- 토큰생성과 추출

```
public String createToken(String username) {  
  
    long now = (new Date()).getTime();  
    Date validity = new Date(now + this.tokenValidityInMilliseconds);  
  
    String access_token = Jwts.builder()  
        .setSubject(username)  
        .claim("email", "test@ddd.com") //추가적인 정보, 권한등을 가지  
        .signWith(key, SignatureAlgorithm.HS512)  
        .setExpiration(validity)  
        .compact();  
    return access_token;  
}  
  
1개 사용 위치  
public String getUsername(String token) {  
    Claims claims = Jwts  
        .parserBuilder()  
        .setSigningKey(key)  
        .build()  
        .parseClaimsJws(token)  
        .getBody();  
  
    return claims.getSubject().toString();  
}
```

JwtFilter

- OncePerRequestFilter 클래스를 상속받아야 한다.
 - OncePerRequestFilter 클래스는 Spring에서 제공하는 추상 클래스로, 모든 요청마다 단 한 번만 실행되는 필터를 구현할 때 사용됩니다. Spring Security에서 자주 사용되며, 주로 JWT 토큰 검증이나 사용자 인증과 같은 작업을 수행하기 위해 사용됩니다.
- 1) 각 HTTP 요청에 대해 한 번만 필터가 실행됩니다. 즉, 동일한 요청 내에서 여러 번 필터가 호출되지 않도록 보장됩니다.
- 2) doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) 메서드를 구현하여 필터 로직을 정의할 수 있습니다.
- 3) 필터 체인의 다음 필터로 요청을 전달하려면
`filterChain.doFilter(request, response)`를 호출해야 합니다.
- 4) 이 필터는 요청이 컨트롤러에 도달하기 전에 실행되므로, 요청 처리 전후에 필요한 로직을 삽입할 수 있습니다. 보통 인증 또는 인가 로직을 처리하기 위해 사용됩니다.

JwtFilter

```
@RequiredArgsConstructor
public class JwtFilter extends OncePerRequestFilter {

    0개의 사용위치
    private static final Logger logger = LoggerFactory.getLogger(JwtFilter.class);
    0개의 사용위치
    public static final String AUTHORIZATION_HEADER = "Authorization";
    private final TokenProvider tokenProvider;

    0개의 사용위치
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

        String authorizationHeader = request.getHeader("Authorization"); // 2. 토큰 추출
        String token = null;
        String username = null;

        // JWT 토큰은 'Bearer'로 시작하는 경우가 많으므로 이를 검증한다. 토큰이 존재하는지 확인한다.
        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer")) {
            token = authorizationHeader.substring(7); // 'Bearer'를 제외한 토큰만 추출
            // JWT 토큰에서 사용자 정보 추출
            username = tokenProvider.getUsername(token);

        }

        // JWT 토큰이 유효하다면, SecurityContext에 인증 정보 설정 - 다른곳에서 접근 가능하도록 하기 위해서
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            // 토큰에서 사용자 이름을 추출해 SecurityContext에 설정
            UsernamePasswordAuthenticationToken authentication
                = new UsernamePasswordAuthenticationToken(username, null, null); // 권한은 생략 가능, 필요시 추가
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            // SecurityContextHolder에 인증 정보 설정
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        // 다음 필터 또는 서블릿으로 전달
        filterChain.doFilter(request, response);
    }
}
```

CorsConfig

```
@Configuration  
public class CorsConfig {  
    0개의 사용위치  
    @Bean  
    public CorsFilter corsFilter() {  
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
        CorsConfiguration config = new CorsConfiguration();  
        config.setAllowCredentials(true);  
        config.addAllowedOriginPattern("*");  
        config.addAllowedHeader("*");  
        config.addAllowedMethod("*");  
  
        source.registerCorsConfiguration( pattern: "/api/**", config);  
  
        return new CorsFilter(source);  
    }  
}
```

SecurityConfig

```
@EnableWebSecurity
@EnableMethodSecurity
@Configuration
@RequiredArgsConstructor
public class SecurityConfig {
    private final TokenProvider tokenProvider;
    private final CorsFilter corsFilter;
    private final JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    0개의 사용위치
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    // 생성자를 이용해서 객체를 전달한다
    0개의 사용위치
    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

SecurityConfig

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        // token을 사용하는 방식이기 때문에 csrf를 disable합니다.
        .csrf(csrf -> csrf.disable())

        .addFilterBefore(corsFilter, UsernamePasswordAuthenticationFilter.class)
        .exceptionHandling(exceptionHandling -> exceptionHandling
            // .accessDeniedHandler(jwtAccessDeniedHandler)
            .authenticationEntryPoint(jwtAuthenticationEntryPoint)
        )
        .authorizeHttpRequests(authorizeHttpRequests -> authorizeHttpRequests
            .requestMatchers(
                new AntPathRequestMatcher( pattern: "/api/hello"),
                new AntPathRequestMatcher( pattern: "/api/login"),
                new AntPathRequestMatcher( pattern: "/api/signup")).permitAll()
            .requestMatchers(PathRequest.toH2Console()).permitAll()
            .anyRequest().authenticated()
        )

        // 세션을 사용하지 않기 때문에 STATELESS로 설정
        .sessionManagement(sessionManagement ->
            sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        )

        // enable h2-console
        .headers(headers ->
            headers.frameOptions(options ->
                options.sameOrigin()
            )
        )

        .addFilterBefore(
            new JwtFilter(tokenProvider), UsernamePasswordAuthenticationFilter.class
        );
        // 필터추가하기
    return http.build();
}
```

UserController

0개의 사용위치

UserDetailsService 구현 클래스를 생성해야 한다

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody AuthenticationRequest request) throws Exception {

    System.out.println("username ===> " + request.getUsername());
    System.out.println("password ===> " + request.getPassword());

    //MemberDto dto = userService.findByUserName(request.getUsername());
    //
    // 1. UsernamePasswordAuthenticationToken 객체 생성: 사용자가 입력한 사용자명과 비밀번호를 토대로 인증 토큰을 생성합니다.
    // 2. authenticationManager.authenticate(authenticationToken): AuthenticationManager는 이 토큰을 사용해 인증을 시도합니다.
    // Spring Security는 설정된 UserDetailsService, PasswordEncoder 등을 이용해 DB에 저장된 정보와 사용자가 입력한 정보를 비교합니다.
    //
    // 3.
    // 인증 성공 시 Authentication 객체 반환: 인증이 성공하면 인증된 Authentication 객체를 반환하고, 실패 시 예외를 던집니다 (AuthenticationException).
    // 반환값을 별도로 처리할 필요가 없다. 만일 비밀번호가 안맞으면 여기서 예외를 던지게 되어 있다

    authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword())
    );
    // JWT 토큰 생성
    String jwt = tokenProvider.createToken(request.getUsername());
    return ResponseEntity.ok(new AuthenticationResponse(jwt));
}
```

loadByUserName 메서드가 호출된다.

UserService

```
@RequiredArgsConstructor
@Service
public class UserService implements UserDetailsService {

    private final UserDao userDao;

    0개의 사용위치

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // DB에서 사용자 조회
        MemberDto user = userDao.findByUserName(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found with username: " + username));

        System.out.println("called loadUserByUsername");
        // UserDetails 객체로 변환하여 반환 (Spring Security가 요구)
        return new org.springframework.security.core.userdetails.User(
            user.getUsername(),
            user.getPassword(),
            user.getAuthority() // 반드시!
        );
    }
}
```

postman(token 가져오기)

The screenshot shows a Postman request to `http://localhost:9000/api/login`. The request method is `POST`, and the body is a JSON object:

```
1 {  
2   "username": "user01",  
3   "password": "1234"  
4 }
```

The response status is `200 OK`, with a response time of `260 ms` and a response size of `622 B`. The response body is a JSON object containing a `jwt` token:

```
1 {  
2   "jwt": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VyMDEiLCJlbWFpbCI6InRlc3RAZGRkLmNvbSIsImV4cCI6MTcyODM0OTY2M30.B2LR6ezb07TxAg4wl27TRbh0D2p5BroCzxk0Fsbd1acp3PF8gAJ5SRTgdf-WWgUf0_mwXJePM5QKqoaKxiJSQ"  
3 }
```

postman(api호출)

- 헤더에 Authorization 키 추가
- Bearer 뒤에 토큰 추가

The screenshot shows the Postman interface with the following details:

- Request Method:** GET
- Request URL:** http://localhost:9000/api/getList
- Headers Tab:** The "Headers (10)" tab is selected.
- Authorization Header:** An "Authorization" header is present with the value: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJ1c2VyMDEiLCJib...
- Params Tab:** Params (1)
- Body Tab:** Body (1)
- Scripts Tab:** Scripts (1)
- Settings Tab:** Settings (1)
- Hidden Headers:** 9 hidden

Ajax

Ajax란

- Ajax(Asynchronous JavaScript and XML)는 웹 페이지가 전체를 다시 로드하지 않고도 서버와 비동기적으로 데이터를 주고받을 수 있게 해주는 웹 기술입니다. Ajax를 사용하면 웹 애플리케이션이 화면을 더 매끄럽게 제공할 수 있으며, 페이지의 일부만 업데이트할 수 있습니다.
- Ajax의 주요 특징
 - 비동기 통신: Ajax는 서버와의 통신이 비동기적으로 이루어집니다. 즉, 서버로부터 응답을 기다리는 동안에도 사용자는 웹 페이지를 계속 사용할 수 있습니다. 페이지 전체를 다시 로드하지 않으므로 더 빠른 반응성을 제공합니다.
 - 데이터 교환 포맷: Ajax는 데이터를 주고받을 때 다양한 형식으로 처리할 수 있습니다. 전통적으로는 XML을 사용했지만, 현재는 JSON(JavaScript Object Notation) 형식이 더 널리 사용됩니다. JSON은 가볍고, 자바스크립트에서 쉽게 파싱할 수 있기 때문에 웹 애플리케이션에서 선호됩니다.
 - 페이지 리로드 없이 업데이트: Ajax를 사용하면 페이지의 특정 부분만 업데이트할 수 있습니다. 사용자가 요청을 보낼 때 페이지가 깜빡이지 않고 변경된 부분만 새로고침되기 때문에 화면이 더 부드러워집니다.

초창기 코드

```
<body>
    <div id="demo"></div>
    <button id="btnCall" onclick="loadDoc()">ajax call</button>
</body>
</html>
<script>

function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
</script>
```

jquery ajax

jquery 를 이용한 ajax

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>
  <div id="demo"></div>
  <button id="btnCall" type="button">ajax call</button>
</body>
</html>
<script>
$(()=>{
  $("#btnCall").click( ()=>{
    $.ajax({
      url: "ajax_info.txt",
    }).done(function(data) {
      $("#demo" ).html( data );
    }).fail(function(error){
      console.log(error);
    });
  });
}

</script>
```

fetch 메서드

- fetch 메서드는 비동기적으로 HTTP 요청을 보내고 서버로부터 응답을 받기 위한 최신 자바스크립트 API입니다.
- 이전의 XMLHttpRequest 객체에 비해 문법이 간결하고 사용하기 편리해, 웹 애플리케이션에서 많이 사용됩니다.
- fetch 메서드는 Promise를 반환하므로, 비동기 코드를 쉽게 작성할 수 있으며, `async/await`과 함께 쓰면 더욱 직관적인 코드를 작성할 수 있습니다.

fetch 사용법

```
fetch(url, options)

.then(response => {

    return response.json(); // 응답을 JSON으로 파싱

})

.then(data => {

    console.log(data); // JSON 데이터 처리

})

.catch(error => {

    console.error(error);

});
```

fetch 동기식

```
async function logJSONData() {  
  
    const response = await fetch("http://example.com/movies.json");  
  
    const jsonData = await response.json();  
  
    console.log(jsonData);  
  
}
```

fetch 예제

```
<script>
function loadDoc() {
    fetch('ajax_resp2')
        .then(response => response.json()) // 응답을 JSON으로 변환
        .then(data => {
            console.log(data); // 데이터를 처리
            document.querySelector("#demo").innerHTML = `${data.name} , ${data.age}`;
        })
        .catch(error => {
            console.error('Error:', error); // 에러 처리
        });
}
</script>
```

axios 사용법

- Axios는 node.js와 브라우저를 위한 Promise 기반 HTTP 클라이언트입니다. 동일한 코드베이스로 브라우저와 node.js에서 실행할 수 있습니다. 서버 사이드에서는 네이티브 node.js의 http 모듈을 사용하고, 클라이언트(브라우저)에서는 XMLHttpRequest를 사용합니다.
- 최근에 fetch나 axios를 모두 사용하거나 하나만 사용하거나 합니다.

axios 설치

- jsDelivr CDN 사용하기:

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```
- unpkg CDN 사용하기:

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

axios get방식

```
<script>
$( ()=>{

    $("#btnCall").click( ()=>{
        let url = "/ajax_resp5";
        axios.get(url)
        .then(function (response) {
            console.log(response.data);
            response.data.forEach( item=>{
                const newItem = `<li>${item}</li>`;
                $('#demo').append(newItem); // <ul> 태그에 <li> 추가
            });
        })
        .catch(function (error) {
            console.log(error);
        })
        .finally(function () {

        });
    });
};

</script>
```

axios post방식

```
<script>
$( ()=>{
    $("#btnCall").click( async ()=>{
        let url = "/ajax_resp7";
        let params = $("#myform").serialize(); //name속성필요
        console.log( params );

        try {
            // Axios를 사용하여 POST 요청 보내기
            let result = await axios.post(url, params, {
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded' // URL 인코딩 방식
                }
            });

            console.log(result.data); // 서버 응답 결과 출력
        } catch (error) {
            console.error("Error:", error); // 오류 처리
        }
    });
}

</script>
```

kakao 호출하기

```
<script>
$( ()=>{
    $("#btnSearch").click(async ()=>{

        var key="4bbb8bd4057a75d806247ec39ba37e18";
        /*
        curl -X GET "https://dapi.kakao.com/v2/search/image?sort=accuracy&page=1&size=80&query=%EC%A7%84%EB%8F%84%EA%B0%9C" \
        -H "Authorization: KakaoAK {REST_API_KEY}"
        get:모든 정보가 url에 노출된다. 한글- 별도의 인코딩 작업이 없으면 받는쪽에서 문자가 깨진다.
        post:
        */
        let result = await axios.post( "https://dapi.kakao.com/v2/search/image",
            {"sort":"accuracy", "page":"1", "size":"80", "query":$("#keyword").val()},
            {
                headers:{
                    "Authorization": "KakaoAK 4bbb8bd4057a75d806247ec39ba37e18",
                    'Content-Type': 'application/x-www-form-urlencoded'
                },
            }
        );

        let data = result.data;
        console.log(result);

        var doc = data.documents;
        console.log(doc);
        doc.forEach((e)=>{
            //console.log(e);
            console.log(e.image_url);
            $("#imageList").append("<li><a target='_blank' href='"+e.image_url+"'" +e.image_url+"</a></li>");
        });
    });
}
</script>
```

파일업로드

설정

- 파일 업로드 모듈은 이미 있으므로 설정만 합니다.
- application.properties 파일
- spring.servlet.multipart.max-file-size=5MB
- spring.servlet.multipart.max-request-size=10MB

파일업로드

0개의 사용위치

@RestController

```
public class FileUploadController {  
    // 프로젝트 루트 경로를 기준으로 상대 경로 설정  
    2개 사용 위치  
    private final String projectRootPath = System.getProperty("user.dir");  
  
    0개의 사용위치  
    @PostMapping("/api/upload")  
    public ResponseEntity<String> fileUpload(@RequestParam("file") MultipartFile file) {  
        if (file.isEmpty()) {  
            return ResponseEntity.badRequest().body("파일을 선택하세요");  
        }  
        try {  
  
            String fileName = file.getOriginalFilename(); //파일이름가져오기  
            long fileSize = file.getSize(); //파일크기  
  
            //절대 경로를 사용하는것이 나음  
            //file.transferTo(new File("c:/uploads/image/"+fileName));  
  
            //상대경로- 이클립스는 괜찮은데 인텔리제이에서는 상대경로를 톰캣 경로로 잡는다. 그래서  
            //직접 지정을 해줘야 한다  
            Path uploadDirectory = Paths.get(projectRootPath, ...more: "uploads/image");  
            Files.createDirectories(uploadDirectory); // 디렉토리가 없으면 생성  
  
            // 파일 저장 경로  
            Path filePath = uploadDirectory.resolve(file.getOriginalFilename());  
  
            // 파일 저장  
            file.transferTo(filePath.toFile());  
            return ResponseEntity.ok( body: "File uploaded successfully: " + fileName + " (" + fileSize + " bytes)");  
        } catch (Exception e) {  
            return ResponseEntity.status(500).body("Failed to upload file: " + e.getMessage());  
        }  
    }  
}
```

파일업로드테스트

HTTP <http://localhost:8080/api/upload>

POST <http://localhost:8080/api/upload>

Params Authorization Headers (9) **Body** Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value	Description
<input checked="" type="checkbox"/>	file	File <input type="button" value="File"/>	OHWGkP.jpg <input type="button" value="Upload"/>
	Key	Text <input type="button" value="Text"/>	Value <input type="button" value="Value"/>

Body Cookies Headers (5) Test Results **200 OK** • 4

Pretty Raw Preview Visualize Text

```
1 File uploaded successfully: OHWGkP.jpg (344759 bytes)
```

파일다운로드

```
// 파일 다운로드 메소드
0개의 사용위치
@GetMapping("/download/{fileName}")
public ResponseEntity<Resource> downloadFile(@PathVariable String fileName) {
    try {
        // 실제 파일 경로 (이 부분을 노출하지 않음)
        Path filePath = Paths.get(projectRootPath, ...more: "/uploads/image").resolve(fileName).normalize();
        Resource resource = new UrlResource(filePath.toUri());

        if (resource.exists()) {
            return ResponseEntity.ok()
                .contentType(MediaType.APPLICATION_OCTET_STREAM)
                .header(HttpHeaders.CONTENT_DISPOSITION, ...headerValues: "attachment; filename=\"" + resource.getFilename() + "\"")
                .body(resource);
        } else {
            return ResponseEntity.notFound().build();
        }
    } catch (Exception ex) {
        return ResponseEntity.status(500).build();
    }
}
```

이미지 노출시키기

- 이미지 경로 만들기
- <http://localhost:8080/files/image/OHWGkP.jpg>

0개의 사용위치

```
@Configuration  
public class WebConfig implements WebMvcConfigurer{  
    0개의 사용위치  
    public WebConfig(){  
        System.out.println("#####");  
    }  
    0개의 사용위치  
    @Override  
    public void addResourceHandlers(ResourceHandlerRegistry registry) {  
  
        //http://localhost:8080/image/image/OHWGkP.jpg  
        // "/images/**"로 시작하는 요청을 특정 디렉토리 경로로 매팅  
        registry.addResourceHandler( ...pathPatterns: "/files/**")  
            .addResourceLocations("file:uploads/image");  
    }  
}
```

Vue 연동

Vue 프로젝트 생성

- npm create vite@latest ↵vue, customize 선택

```
C:\springboot_workspace\vue>npm create vite@latest

> npx
> create-vite

✓ Project name: ... frontend_vue
✓ Select a framework: » Vue
✓ Select a variant: » Customize with create-vue ✘

> npx
> create-vue frontend_vue

Vue.js - The Progressive JavaScript Framework

✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? » No
✓ Add ESLint for code quality? ... No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes

Scaffolding project in C:\springboot_workspace\vue\frontend_vue...

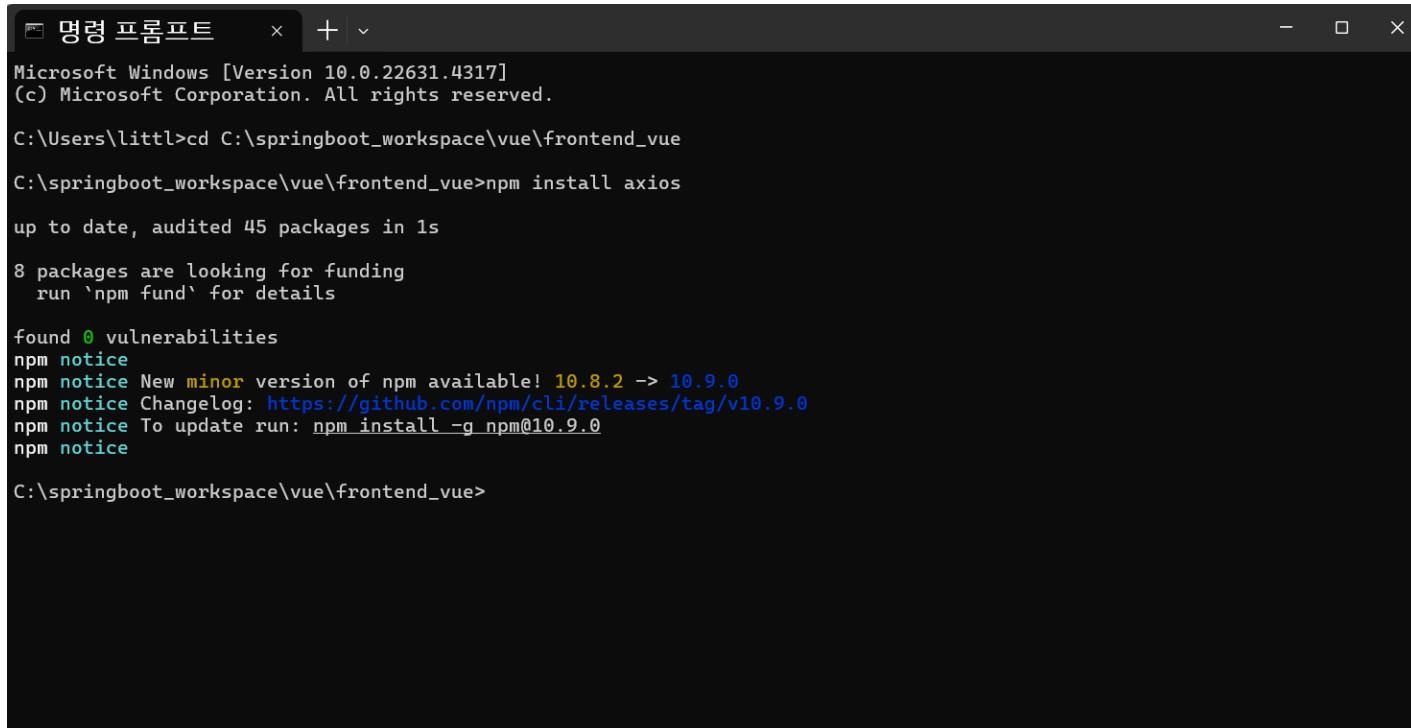
Done. Now run:

  cd frontend_vue
  npm install
  npm run dev

C:\springboot_workspace\vue>
```

추가모듈설치

- npm install axios



```
명령 프롬프트 + ▾ Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\littl>cd C:\springboot_workspace\vue\frontend_vue
C:\springboot_workspace\vue\frontend_vue>npm install axios
up to date, audited 45 packages in 1s

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 10.8.2 -> 10.9.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.9.0
npm notice To update run: npm install -g npm@10.9.0
npm notice

C:\springboot_workspace\vue\frontend_vue>
```

login – Login.vue

```
<script setup>
import { ref } from 'vue';
import axios from 'axios';
import { useRouter } from 'vue-router';

// 사용자 입력 데이터
const username = ref('');
const password = ref('');
const errorMessage = ref(null);
const router = useRouter();

// 로그인 함수
const login = async () => {
  try {

    console.log( username.value );
    console.log( password.value );

    const response = await axios.post('http://localhost:9000/api/login', {
      username: username.value,
      password: password.value,
    });

    console.log(response.data);
    // 로그인 성공 시 JWT 토큰 저장 (로컬 스토리지 등)
    localStorage.setItem('token', response.data.jwt);

    // 로그인 후 페이지 이동 (대시보드 등)
    router.push('/about');
  } catch (error) {
    // 에러 처리 (401, 403 등)
    errorMessage.value = '로그인에 실패했습니다. 다시 시도하세요.';
  }
};
</script>
```

```
<template>
  <div>
    <h2>로그인</h2>
    <form @submit.prevent="login">
      <div>
        <label for="username">사용자명:</label>
        <input v-model="username" id="username" type="text" required />
      </div>
      <div>
        <label for="password">비밀번호:</label>
        <input v-model="password" id="password" type="password" required />
      </div>
      <button type="submit">로그인</button>
    </form>

    <!-- 에러 메시지 출력 -->
    <div v-if="errorMessage" style="color: red;">{{ errorMessage }}</div>
  </div>
</template>

<style scoped>
  form {
    max-width: 300px;
    margin: 0 auto;
  }

  div {
    margin-bottom: 15px;
  }

  button {
    padding: 10px;
    background-color: #42b983;
    color: white;
    border: none;
    cursor: pointer;
  }

  button:hover {
    background-color: #358a5e;
  }
</style>
```

데이터 가져오기 - About.vue

```
<script setup>
import { ref, onMounted } from 'vue';
import axios from 'axios';

// 데이터를 담을 변수
const items = ref([]);
const isLoading = ref(true);
const errorMessage = ref(null);

// 데이터 불러오기 함수
const fetchData = async () => {
  try {

    console.log( localStorage.getItem('token'));

    // API 호출
    const response = await axios.get('http://localhost:9000/api/getList', {
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`,
        'Content-Type': 'application/json',
      },
    });
    items.value = response.data.data;
    console.log(items.value);
  } catch (error) {
    // 에러 처리
    errorMessage.value = '데이터를 불러오는 중 오류가 발생했습니다.';
  } finally {
    // 로딩 상태 변경
    isLoading.value = false;
  }
};

// 컴포넌트가 마운트될 때 데이터를 불러옴
onMounted(() => {
  fetchData();
});
</script>

<template>
  <div>
    <h2>데이터 불러오기 페이지</h2>

    <!-- 로딩 상태 -->
    <div v-if="isLoading">로딩 중...</div>

    <!-- 에러 메시지 -->
    <div v-if="errorMessage" style="color: red;">{{ errorMessage }}</div>

    <!-- 데이터 목록 -->
    <ul v-else>
      <li v-for="item in items" :key="item.id">
        {{ item.name }}
      </li>
    </ul>
  </div>
</template>

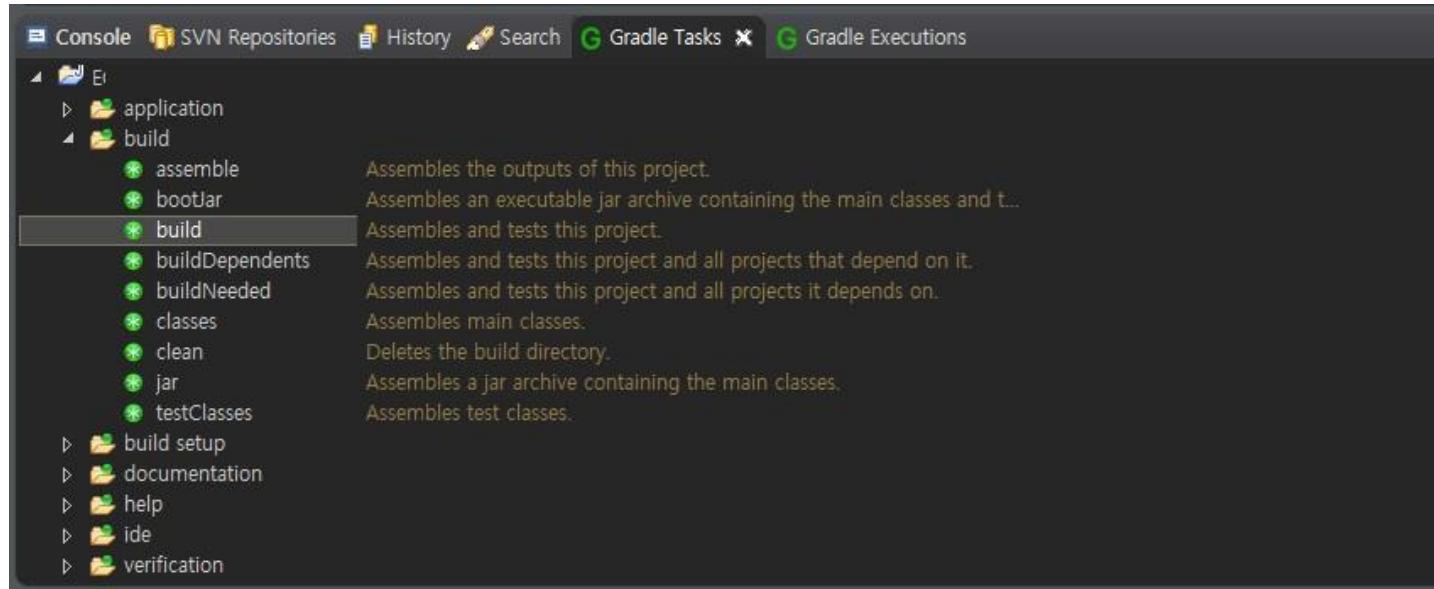
<style scoped>
/* 스타일링 */
ul {
  list-style: none;
  padding: 0;
}

li {
  margin: 5px 0;
}
</style>
```

스프링 부트 배포

gradle 사용시 1

- Eclipse 상단 메뉴의 [Window] - [Show View] - [Other]에서 Gradle Tasks를 검색해서 연다
- Build를 더블클릭한다



gradle 사용시 2

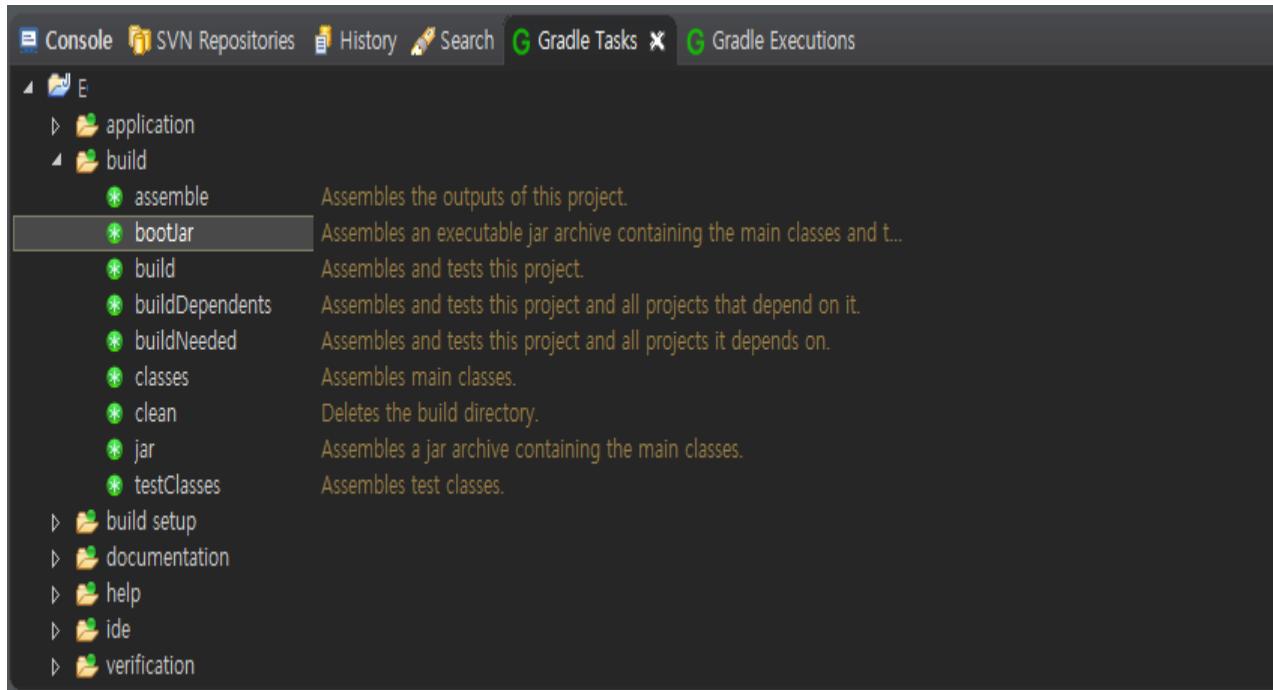
Run build	4.567 s
Load build	
Evaluate settings	0.015 s
Apply script settings.gradle to settings 'ECALL_Application_WSM'	0.010 s
Finalize build cache configuration	0.002 s
Configure build	0.000 s
Load projects	0.157 s
Notify projectsLoaded listeners	0.002 s
Execute 'rootProject()' action	0.001 s
Cross-configure project :	0.000 s
Configure project :	0.000 s
Notify beforeEvaluate listeners of :	0.152 s
Register task :init	0.000 s
Register task :wrapper	0.000 s
Apply plugin org.gradle.help-tasks to root project 'ECALL_WSM'	0.004 s
Register task :help	0.000 s
Register task :projects	0.001 s

gradle 사용 시 3

- xml파일은 build할 때 생성이 안되서 따로 복사 붙여넣기를 해주어야 한다.
- Xml이 별도로 존재할때 workspace의 프로젝트의 [build] - [classes] - [java] - [main] - ... xml 파일의 경로로 들어가 붙여넣기 하면 된다.
- 보통은 프로젝트 작성시 resources 폴더에 둔다. 이 폴더에 있는 파일만 자동으로 빌드해준다.

gradle 사용시 4

- jar 파일을 만든다.
- Gradle Tasks 탭에서 해당 프로젝트의 [build] bootjar를 더블 클릭한다.



gradle 사용시 5

- 프로젝트의 [build] - [libs] 로 들어가보면 jar파일이 생긴 것을 확인할 수 있을 것이다.

보기				
PC > 로컬 디스크 (C:) > spring_boot > springboot_workspace > backend2 > build > libs				
	이름	수정한 날짜	유형	크기
...	backend2-0.0.1-SNAPSHOT.jar	2022-03-24 오후 5:05	Executable Jar File	24,198KB
...	backend2-0.0.1-SNAPSHOT-plain.jar	2022-03-24 오후 5:05	Executable Jar File	28KB

gradle 사용시 6

- Java -jar 파일명

```
C:\#project>java -jar backend2-0.0.1-SNAPSHOT.jar

           .   ---.  ,--( )--. --.----.----.----.----.
          ( ( )--. | | | | | | | | | | | | | | | | | | | |
            \---| | | | | | | | | | | | | | | | | | | |
              | | | | | | | | | | | | | | | | | | | |
              =====| |=====| |=====| |=====| |=====| |=====|
              :: Spring Boot ::          (v2.6.3)

2022-03-24 17:10:43.797  INFO 20256 --- [           main] com.example.demo.Backend2Application : Starting Backend2-0.0.1-SNAPSHOT.jar started by student in C:\#project)
2022-03-24 17:10:43.801  INFO 20256 --- [           main] com.example.demo.Backend2Application : No active p
2022-03-24 17:10:44.245  INFO 20256 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping
2022-03-24 17:10:44.254  INFO 20256 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Sp
2022-03-24 17:10:44.315  WARN 20256 --- [           main] o.m.s.mapper.ClassPathMapperScanner      : No MyBatis
```

참고자료

- <https://yeonyeon.tistory.com/153>
- <https://chung-develop.tistory.com/5>
- <https://velog.io/@imsooyeon/Spring-JPA-%EA%B0%9C%EB%85%90%EA%B3%BC-%EC%9E%A5%EB%8B%A8%EC%A0%90>
- <https://velog.io/@h220101/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81-%EB%B6%80%ED%8A%B8-spring-MVC-%ED%8C%A8%ED%84%B4-%EB%8F%99%EC%9E%91>
- <https://dbjh.tistory.com/77>
- <https://tibetsandfox.tistory.com/11>
- <https://technology-share.tistory.com/20>
- <https://tlatmsrud.tistory.com/74>
- chatgpt