

Spring Boot 입문과 활용

2023.2.7 ~ 2.8

백명숙



과정개요

■ Spring Boot의 개념과 Architecture를 이해하고,기본 원리와 자동 설정에 대한 내용들을 이해한다.
Spring Boot를 활용할 수 있는 여러가지 기술(Web MVC, JPA, Security 등)들을 이해하는 과정입니다.



LO	커리큘럼
Spring Boot 소개	<ul style="list-style-type: none"> - Spring Boot 소개 - 개발환경 구축
Spring Boot 원리	<ul style="list-style-type: none"> - 의존성 관리 - Spring Boot Auto Configuration
Spring Boot 활용 I	<ul style="list-style-type: none"> - Spring Application - 외부 설정 (application.properties) - Profile - Logging - Spring-Boot-Devtools - Spring Boot Data Access (JPA, Java Persistence API)
Spring Boot 활용 II	<ul style="list-style-type: none"> - Spring Boot Web MVC (Thymeleaf, Restful Web Service) - Spring Boot Actuator (모니터링) - Spring Boot Security (Form 인증)

스프링 부트



Spring Boot 시작하기

- 스프링 부트는 2014년부터 개발된 스프링의 하위 프로젝트 중 하나입니다.
- 단독으로 실행이 가능하고(stand-alone), 제품 수준의(production-grade) 스프링 기반 어플리케이션을 제작하는 것을 목표로 진행된 프로젝트입니다.
- 스프링 부트의 주요 기능
 - 단독 실행이 가능한 수준의 스프링 어플리케이션 제작이 가능합니다.
 - 내장된 Tomcat, Jetty, Undertow 등의 서버를 이용해서 별도의 서버를 설치하지 않고 실행 가능합니다.
 - 최대한 내부적으로 자동화된 설정을 제공합니다.
 - XML 설정 없이 단순한 설정 방식을 제공합니다.

Spring Boot 시작하기

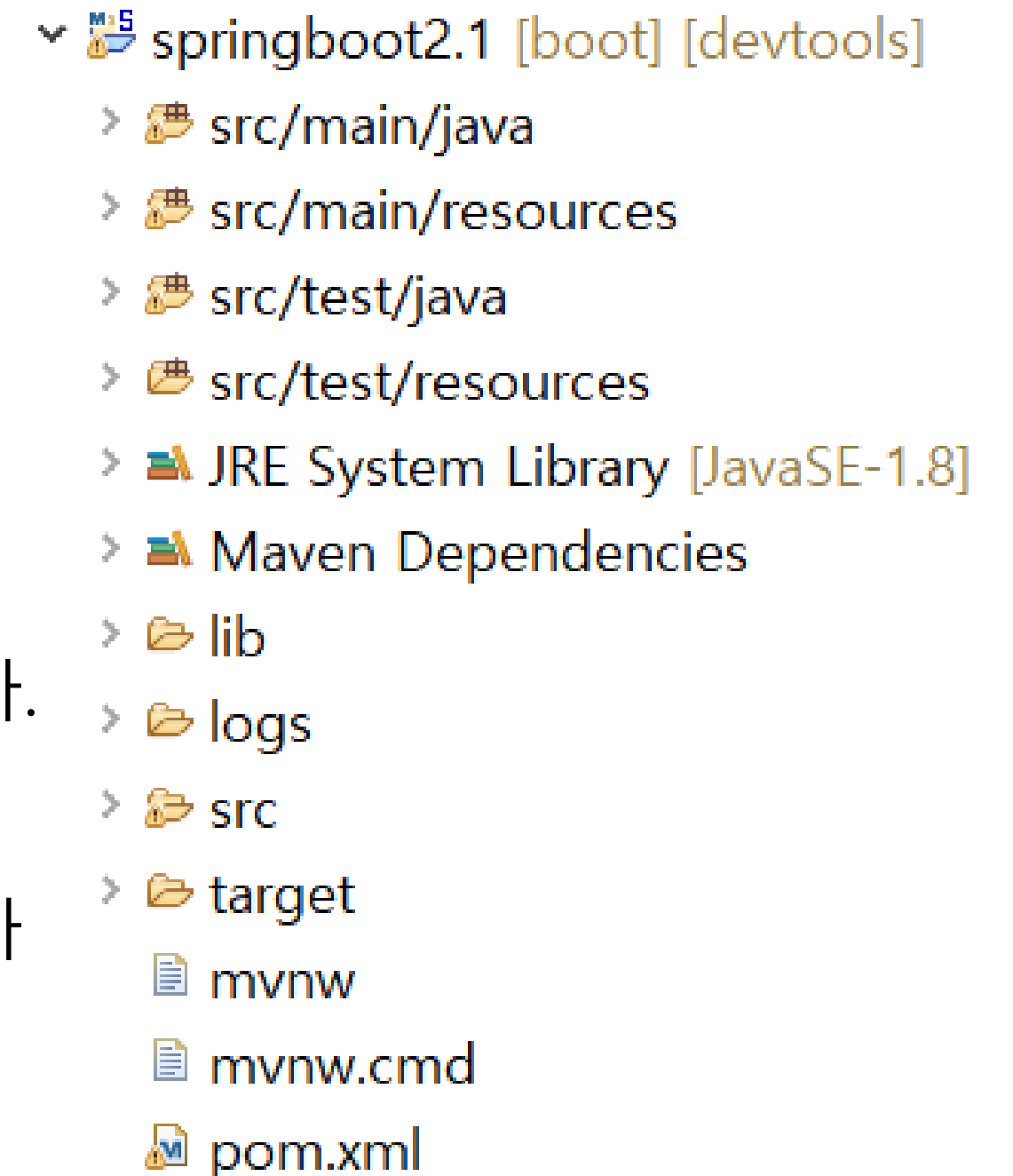
- 스프링 프레임워크를 사용할 때 했던 많은 XML 설정들이 어노테이션으로 간편 해졌고, Java Config를 사용하며 더욱 단순해 졌는데 이것보다 더 단순화 된 프로젝트가 스프링 부트 (Spring Boot)입니다.
- 스프링 부트는 스프링의 여러 기술들(Data, Batch, Integration, Web, JDBC, Security)을 사용자가 쉽게 사용할 수 있게 해줍니다.
- Spring Boot (<https://spring.io/projects/spring-boot>)
- 스프링 부트 프로젝트 생성 (<https://start.spring.io/>)

Spring Boot 시작하기

- 스프링 부트 프로젝트 구조

src/main/java	자바 Source 파일들
src/main/resources/application.properties	스프링 부트 프로퍼티 값들을 모아 놓은 파일
src/main/resources/static	html, css 같은 정적 파일들
src/main/resources/templates	jsp, thymeleaf 같은 동적 파일들
src/test/java	자바 테스트 파일들

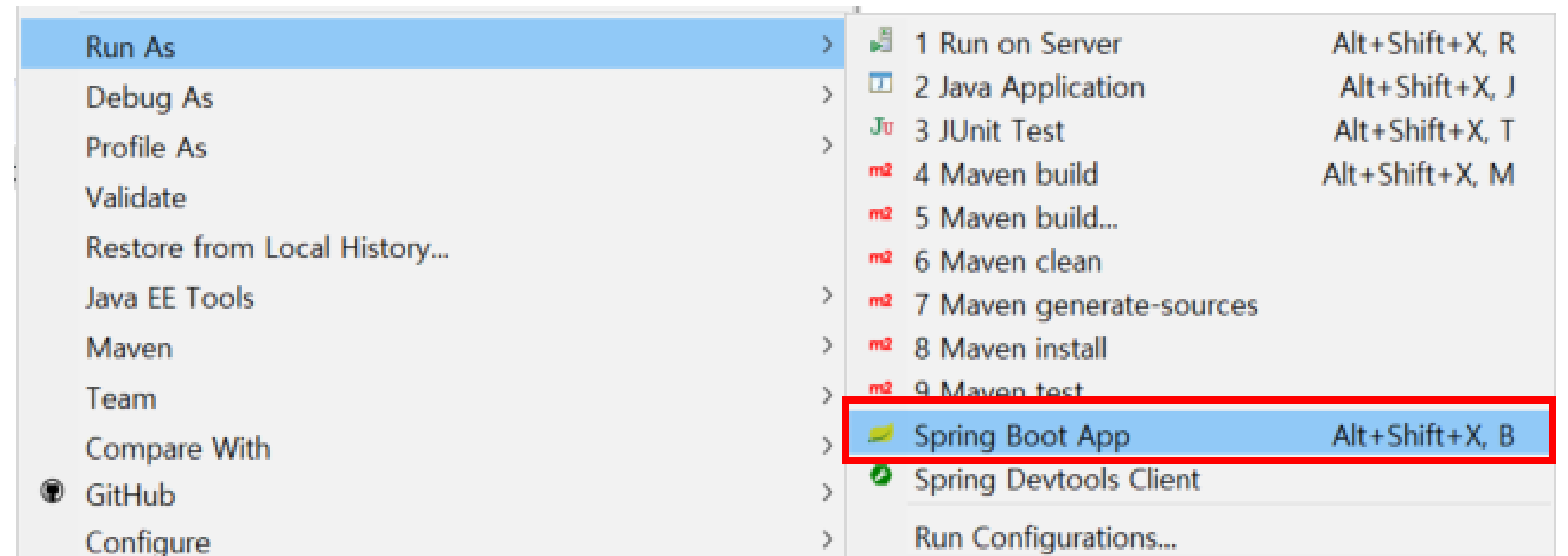
@SpringBootApplication 클래스의 적정 위치는 디폴트 패키지에 위치해야 합니다.
왜냐하면 @SpringBootApplication 어노테이션의 내부에서 선언된
@ComponentScan 어노테이션은 해당 클래스 기준으로 현 위치와 그 아래 위치까
지 스프링 Bean을 나타내는 어노테이션이 선언된 클래스를 찾기 때문입니다.



Spring Boot 시작하기

- 스프링 부트 프로젝트 실행

Run As -> Spring Boot App



- Springboot Maven 플러그인으로 실행하기

```
$ mvnw spring-boot:run
```

- Port 번호 변경 (port 번호 충돌이 발생할 경우에 변경함)

server.port 속성 변경

```
src/main/resources/application.properties
```

```
server.port=8080
```


Spring Boot 원리

- 의존성 관리 이해

Spring Boot는 의존성 관리를 내부적으로 해주기 때문에 Spring Framework에 비해 개발자가 따로 의존관계를 설정 부분이 많이 줄어 들었다.

- Dependency 계층구조

`<artifactId>spring-boot-starter-parent</artifactId> : pom.xml`

`<artifactId>spring-boot-dependencies</artifactId>`

`: spring-boot-starter-parent.pom`

`<artifactId>spring-boot-dependencies</artifactId>`

`: spring-boot-dependencies.pom`

spring-boot-starter-* 라이브러리는 parent에서 버전을 명시하므로 따로 버전을 명시할 필요가 없다.

Spring Boot 원리

- 기존의 의존성 버전 변경하기

: Spring Framework 버전 변경 (Version 이 변경될 수 있습니다.)

pom.xml

```
<properties>  
    <spring-framework.version>5.3.23</spring-framework.version>  
</properties>
```

Spring Boot 원리

- 자동 설정 이해

@SpringBootApplication은 아래의 3가지 어노테이션을 합친 것이다.

=> **@SpringBootConfiguration + @ComponentScan + @EnableAutoConfiguration**

@SpringBootApplication 설정은 Bean을 두 단계로 나눠서 등록한다.

1단계 : @ComponentScan : project 생성시 정해진 default 패키지 부터 scanning을 한다.

2단계 : @EnableAutoConfiguration

- 1 단계 : @ComponentScan

: @ComponentScan은 스프링 프레임워크에서 @Repository, @Configuration, @Service 등 스프링 빈을 나타내는 어노테이션을 @ComponentScan이 붙은 클래스가 위치해 있는 현재 패키지를 기준으로 그 아래 패키지까지 찾아서 스프링 빈으로 등록하는 기능을 가진 어노테이션입니다.

Spring Boot 원리

- 자동 설정 이해
- 2단계 : @EnableAutoConfiguration

: @EnableAutoConfiguration은 스프링 부트에서 스프링 프레임워크에서 많이 쓰이는 스프링 bean 들을 자동적으로 컨테이너에 등록하는 역할을 하는 어노테이션입니다. @EnableAutoConfiguration이 등록하는 bean들의 목록은 spring-boot-autoconfigure-2.X.X.RELEASE.jar 파일에 포함되어 있습니다.

spring-boot-autoconfigure-2.x.x.RELEASE.jar\META-INF\spring.factories

: org.springframework.boot.autoconfigure.EnableAutoConfiguration

: spring.factories내에 선언된 @Configuration 설정 클래스들을 모두 Load한다.

: @Configuration, @ConditionalOnXxxYyyZzz

: org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration

위 목록 중에는 @EnableAutoConfiguration를 선언했을 때 스프링 부트 프로젝트를 기본적으로 웹 프로젝트로 만들 수 있는 기본값이 설정되어 있습니다

Spring Boot 원리

- 자동 설정 이해
- 스프링 부트를 웹 어플리케이션 프로젝트로 만들지 않고, 일반 프로젝트 용도로 사용하고자 한다면 Application 클래스를 아래와 같은 코드로 작성해야 합니다.

src/main/java/Application.java

```
SpringApplication application = new SpringApplication(Application.class);  
application.setWebApplicationType(WebApplicationType.NONE);  
application.run(args);
```

Spring Boot 활용

1. Spring Application #1

- Application 의 Log Level 변경

- 기본 로그 레벨은 INFO,
- 로그레벨 DEBUG로 변경 (VM argument에 환경변수 추가)

: Application -> Run Configuration -> VM arguments -> -Ddebug 를 추가한다.

- Spring Boot Banner 변경하기

: resources/banner.txt | gif | jpg | png

: \${spring-boot.version} , \${application.version} 등의 변수를 사용할 수 있음.

src/main/resources/banner.txt

My Spring Boot \${spring-boot.version} / \${application.version}

<https://devops.datenkollektiv.de/banner.txt/index.html>

Spring Boot 활용

1. Spring Application #1

- Banner 파일의 위치 변경

: spring.banner.location 속성 변경

```
src/main/resources/application.properties
```

```
spring.banner.location=classpath:mybanner.txt
```

Spring Boot 활용

1. Spring Application #1

- Spring Boot 프로젝트를 jar 파일로 생성하기
: Run As -> Maven Build -> **Goals : package** -> Run
- pom.xml에 **<packaging>jar</packaging>** 추가

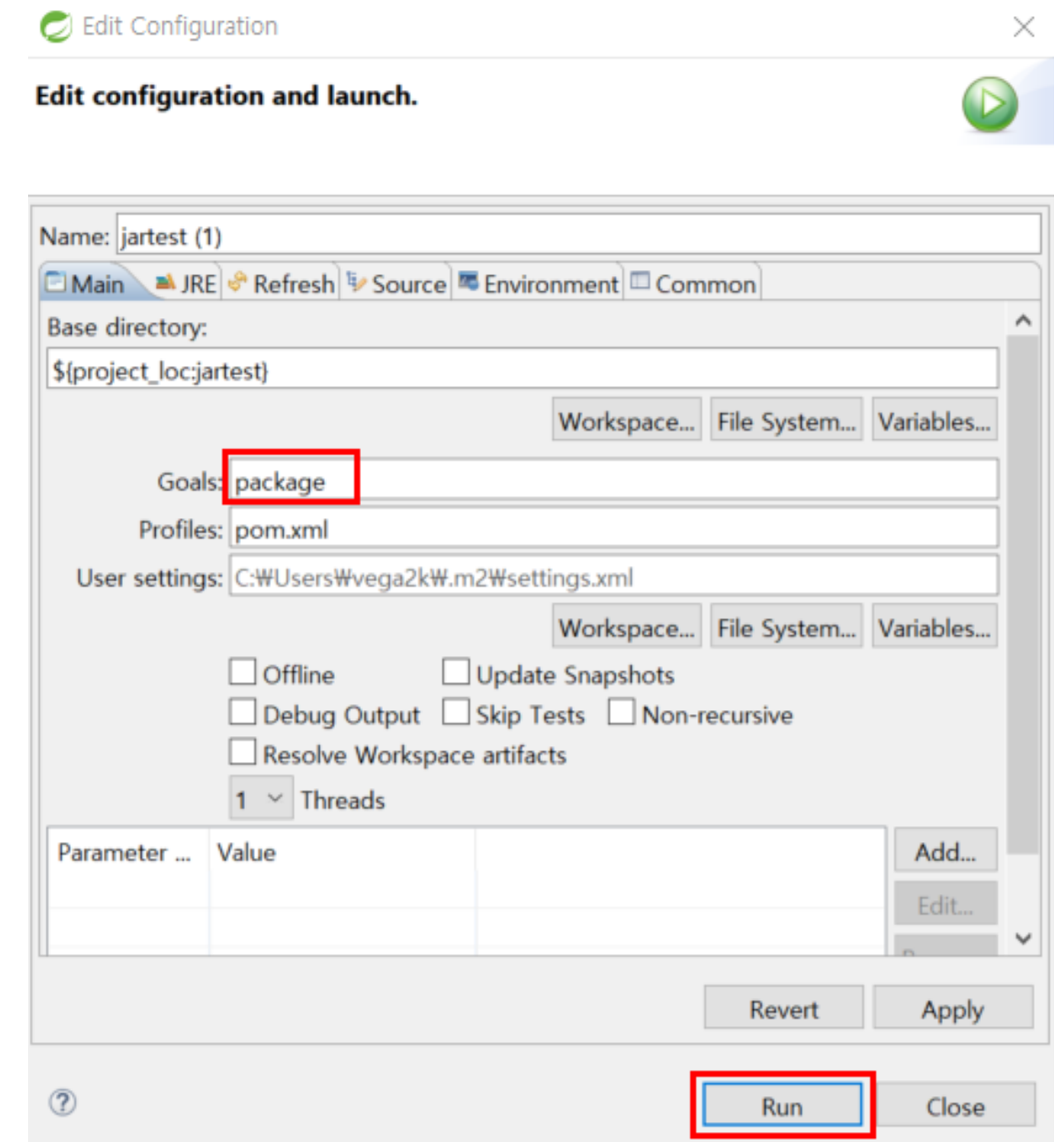
```
<groupId>com.vega2k</groupId>
<artifactId>jartest</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>jartest</name>
<description>test project for Spring Boot</description>
```

- Maven Command로 jar 생성하기

```
$ mvnw package
```

- 생성된 jar 실행 (jar 파일명이 다를 수 있음)

```
target>java -jar jartest-0.0.1-SNAPSHOT.jar
```



Spring Boot 활용

1. Spring Application #1

- Spring Boot Banner 변경하기

: 배너 끄는 방법 `application.setBannerMode(Mode.OFF);`

: Banner 클래스 구현하고 `SpringApplication.setBanner()`로 설정 가능.

`src/main/java/Application.java`

```
SpringApplication application = new SpringApplication(Application.class);
//Banner 출력 안되게 함
//application.setBannerMode(Mode.OFF);
application.setBanner(new Banner() {
    @Override
    public void printBanner(Environment environment, Class<?> sourceClass, PrintStream out) {
        out.println("=====");
        out.println("Wt My 스프링 부트 ");
        out.println("=====");
    }
});
application.run(args);
```

Spring Boot 활용

2. Spring Application #2

- 이벤트 리스너 (Event Listener)

: 스프링 부트를 실행할 시 구동되는 단계마다 여러 이벤트들이 발생하게 됩니다.

스프링부트에서는 개발자가 이 이벤트들을 나타내는 객체를 인자로 받아 각 단계마다 원하는 처리를 할 수 있습니다.

: 스프링 부트가 구동될 때 여러 이벤트들이 있지만 그 중에서 **ApplicationStartingEvent** 와 **ApplicationStartedEvent**를 예로 들어 알아보겠습니다.

: 주의 할 것은 ApplicationStartingEvent는 스프링 컨테이너가 생성되기 전에 생성되는 이벤트이기 때문에 이 이벤트를 처리하려면 SpringApplication 객체에 해당 리스너를 추가해야 합니다.

: ApplicationStartedEvent를 포함한 스프링 컨테이너가 만들어진 이후에 생성되는 이벤트들은 스프링 Bean 등록을 통해 이벤트를 처리할 수 있습니다.

Spring Boot 활용

2. Spring Application #2

- ApplicationStartingEvent 등록

:ApplicationContext를 만들기 전에 호출되는 ApplicationListener<ApplicationStartingEvent> 리스너는 @Bean으로 등록할 수 없으므로 SpringApplication.addListeners() 해주어야 한다.

src/main/java/com/basic/boot/SampleListener.java

```
public class SampleListener implements ApplicationListener<ApplicationStartingEvent>{
    public void onApplicationEvent(ApplicationStartingEvent arg0) {
        System.out.println("=====");
        System.out.println("Application is Starting");
        System.out.println("=====");
    }
}
```

src/main/java/com/basic/boot/Application.java

```
public class Application {
    public static void main(String[] args) {
        SpringApplication application = new SpringApplication(Application.class);
        application.addListeners(new SampleListener());
        application.run(args);
    }
}
```

Spring Boot 활용

2. Spring Application #2

- ApplicationStartedEvent 등록

: ApplicationContext를 만든 후에 호출되는 ApplicationListener<ApplicationStartedEvent> 리스너는 Bean으로 등록할 수 있다.

src/main/java/com/basic/boot/SampleListener.java

```
@Component
public class SampleListener implements ApplicationListener<ApplicationStartedEvent>{
    public void onApplicationEvent(ApplicationStartedEvent arg0) {
        System.out.println("=====");
        System.out.println("Application is Started");
        System.out.println("=====");
    }
}
```

src/main/java/com/basic/boot/Application.java

```
public class Application {
    public static void main(String[] args) {
        SpringApplication application = new SpringApplication(Application.class);
        application.run(args);
    }
}
```


Spring Boot 활용

2. Spring Application #2

- 웹 어플리케이션 타입 지정

- : 스프링 부트는 `SpringApplication` 객체를 통해 어플리케이션 타입을 지정할 수 있습니다.

- `SpringApplication` 객체는 스프링 컨테이너의 인터페이스인 `ApplicationContext`를 개발자 대신 만들어줍니다. 개발자는 `ApplicationContext`의 구현체를 다음과 같은 방법으로 지정할 수 있습니다.

- : 만약 `Spring MVC`와 `Spring webFlux` 둘 다 설정되어 있으면 `Spring MVC`가 우선적으로 적용됩니다.

- : `WebApplicationType.SERVLET` - `AnnotationConfigServletWebServerApplicationContext`

- : `WebApplicationType.REACTIVE` - `AnnotationConfigReactiveWebServerApplicationContext`

- : `WebApplicationType.NONE` - `AnnotationConfigApplicationContext` 로 설정됨

```
src/main/java/com/basic/boot/Application.java
```

```
@SpringBootApplication
public class Application {
    SpringApplication application = new SpringApplication(Application.class);
    application.setWebApplicationType(WebApplicationType.SERVLET);    //웹어플리케이션 타입 지정
    application.run(args);
}
```

Spring Boot 활용

2. Spring Application #2

- ApplicationRunner 작성 : 커맨드 아규먼트 처리
 - : SpringApplication 실행된 후에 arguments 값을 받거나, 무엇인가를 실행하고 싶을 때 ApplicationRunner 인터페이스를 구현하는 Runner 클래스를 작성합니다.
 - : 순서 지정 가능하다 @Order(1) - 숫자가 낮을 수록 우선 순위가 높다.
 - : 아규먼트 값은 아래와 같이 설정한다.

Application -> Run Configuration -> Program arguments -> --bar 를 추가한다.

-> VM arguments -> -Dfoo 를 추가한다.

-Dfoo VM 아규먼트는 무시하고, --bar 아규먼트는 처리한다.

src/main/java/com/basic/boot/runner/MyRunner.java

```
@Component
public class MyRunner implements ApplicationRunner {
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("foo : " + args.containsOption("foo"));
        System.out.println("bar : " + args.containsOption("bar"));
    }
}
```

```
foo: false
bar: true
```

Spring Boot 활용

3. 외부 설정 #1

스프링 부트는 외부 설정을 통해 스프링 부트 어플리케이션의 환경설정 혹은 설정값을 정할 수 있습니다. 스프링 부트에서 사용할 수 있는 외부 설정은 크게 properties, YAML, 환경변수, 커맨드 라인 인수 등이 있습니다.

- Properties 파일을 통한 설정 : properties의 값은 @value 어노테이션을 통해 읽어올 수 있다.

src/main/java/com/basic/boot/runner/MyRunner.java

```
@Component
public class MyRunner implements ApplicationRunner {
    @Value("${myboot.name}")
    private String name;
    @Value("${myboot.age}")
    private int age;
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("*****");
        System.out.println(name);
        System.out.println(age);
    }
}
```

src/main/resources/application.properties

```
myboot.name=Spring
myboot.age=${random.int(1,100)}
myboot.fullName=${myboot.name} Boot
```

Spring Boot 활용

3. 외부 설정 #1

- 프로퍼티 우선 순위

1. 유저 홈 디렉토리에 있는 `spring-boot-dev-tools.properties`
2. 테스트에 있는 `@TestPropertySource`
3. `@SpringBootTest` 애노테이션의 `properties` 애트리뷰트
4. 커맨드 라인 아규먼트
5. `SPRING_APPLICATION_JSON` (환경 변수 또는 시스템 프로퍼티)에 들어있는 프로퍼티
6. `ServletConfig` 파라미터
7. `ServletContext` 파라미터
8. `java:comp/env JNDI` 애트리뷰트
9. `System.getProperties()` 자바 시스템 프로퍼티
10. OS 환경 변수

Spring Boot 활용

3. 외부 설정 #1

- 프로퍼티 우선 순위

11. RandomValuePropertySource

12. JAR 밖에 있는 특정 프로파일용 application properties

13. JAR 안에 있는 특정 프로파일용 application properties

14. JAR 밖에 있는 application properties

15. JAR 안에 있는 application properties

16. @PropertySource

17. 기본 프로퍼티 (SpringApplication.setDefaultProperties)

Externalized Configuration

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-external-config>

Spring Boot 활용

3. 외부 설정 #1

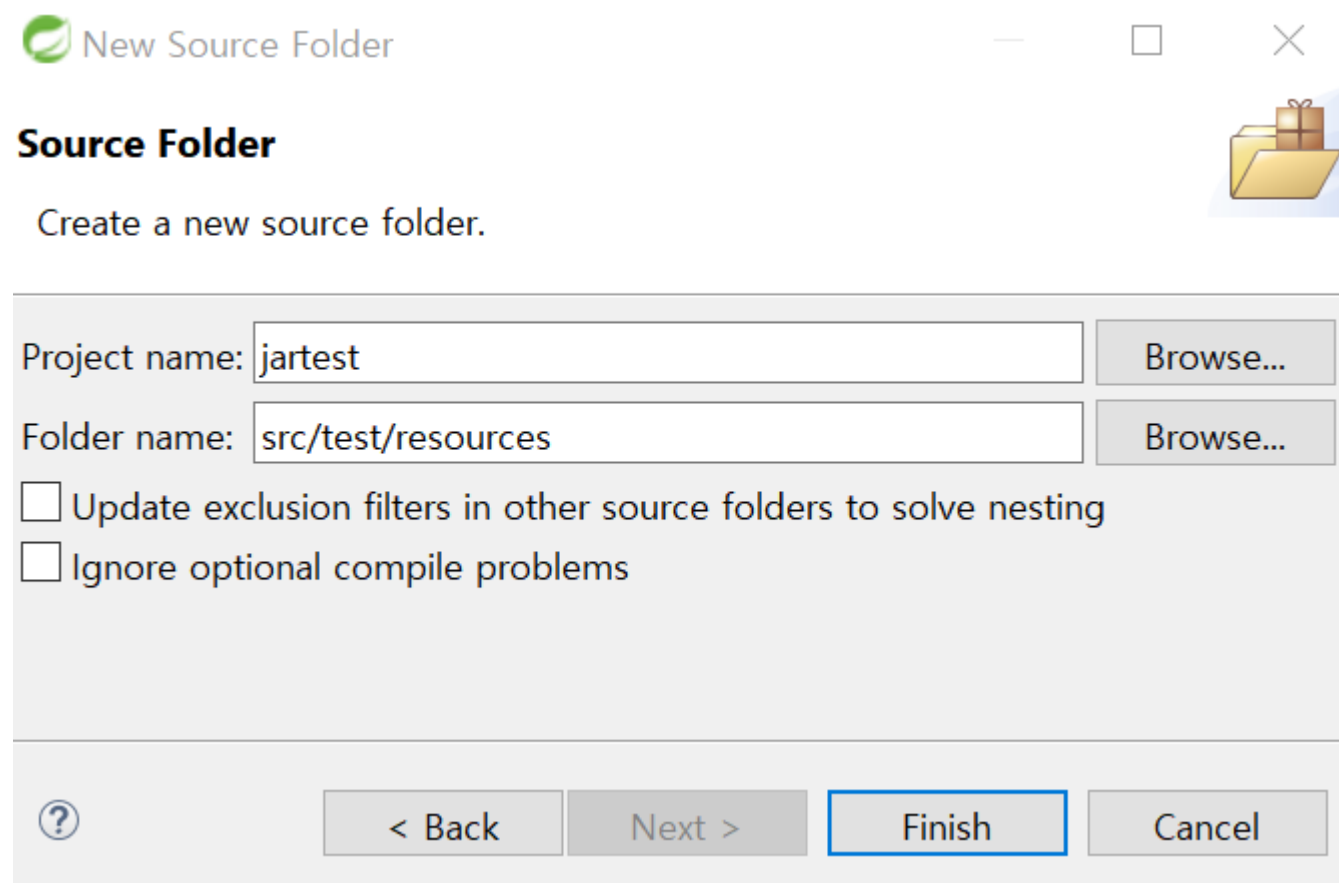
- 4번째 우선 순위인 커맨드 라인 아규먼트로 설정

: jar 실행시 아규먼트 설정

```
target>java -jar jar-test-0.0.1-SNAPSHOT.jar --myboot.name=springboot
```

- 2번째 우선 순위인 테스트에 있는 @TestPropertySource

: src/test/resources 디렉토리를 생성하고, test.properties 파일을 작성한다.



src/test/resources/test.properties

myboot.name=test

Spring Boot 활용

3. 외부 설정 #1

- 2번째 우선 순위인 테스트에 있는 @TestPropertySource
: ApplicationTests 클래스 작성

src/test/java/com/basic/boot/ApplicationTests.java

```
@RunWith(SpringRunner.class)
@TestPropertySource(locations="classpath:/test.properties")
@SpringBootTest
public class ApplicationTests {
    @Autowired
    Environment environment;
    @Test
    public void contextLoads() {
        assertThat(environment.getProperty("basic.name")).isEqualTo("test");
    }
}
```

Spring Boot 활용

4. 외부 설정 #2

- @ConfigurationProperties 어노테이션을 통한 외부 설정값 주입
 - : @ConfigurationProperties 프로퍼티 파일의 값을 받은 클래스를 하나 생성하여 그 클래스를 @Autowired 같은 어노테이션을 통해 자동 주입하는 방법이 type-safe, 유지보수 측면에서 더 좋은 장점을 가집니다.
 - : 프로퍼티 클래스를 작성하면 여러 프로퍼티를 묶어서 읽어올 수 있다.
 - : 프로퍼티 클래스를 Bean으로 등록해서 다른 Bean에 주입할 수 있다.
 - : application.properties에 똑같은 key값을 가진 property가 많은 경우에 프로퍼티 클래스를 작성 할 수 있다.

Spring Boot 활용

4. 외부 설정 #2

- @ConfigurationProperties 사용한 타입-세이프 프로퍼티 클래스를 작성

src/test/java/com/basic/boot/MybootProperties.java

```
@Component
@ConfigurationProperties("myboot")
public class MybootProperties {
    private String name;
    private int age;
    private String fullName;

    getter();
    setter();
}
```

src/main/java/com/basic/boot/runner/MyRunner.java

```
@Component
public class MyRunner implements ApplicationRunner {
    @Autowired
    MybootProperties mybootProperties;

    public void run(ApplicationArguments args) throws Exception {
        System.out.println(mybootProperties.getName());
        System.out.println(mybootProperties.getFullName());
        System.out.println(mybootProperties.getAge());
    }
}
```

Spring Boot 활용

4. 외부 설정 #2

- @ConfigurationProperties 사용

: @ConfigurationProperties 어노테이션을 사용하려면 META 정보를 생성 해주는
spring-boot-configuration-processor 의존성 설치

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>
```

Spring Boot 활용

5. 스프링 부트 프로파일

- 스프링 부트에서는 프로파일(Profile)을 통해 스프링 부트 애플리케이션의 런타임 환경을 관리할 수 있습니다. 애플리케이션 작동 시 테스트 환경에서 실행할 지, 프로덕션 환경에서 실행할 지를 프로파일을 통해 관리할 수 있습니다.
- 프로덕션과 테스트 환경을 각각 외부 설정 파일을 통해서 관리합니다. `spring.profiles.active` 키값을 통해 어떤 프로파일을 활성화 할 것인지를 결정합니다.
- `@Profile` 어노테이션을 통해 프로파일 기능을 구현할 수 있습니다. `@Profile`에 인자로 들어가는 값은 프로파일이 현재 인자값과 일치할 때 명시한 스프링 bean을 등록하라는 뜻입니다.

Spring Boot 활용

5. 스프링 부트 프로파일

- @Profile 어노테이션을 사용하여 어떤 profile을 활성화 할 것인가?

src/main/java/com/basic/boot/BaseConfiguration.java

```
@Profile("prod")
@Configuration
public class BaseConfiguration {
    @Bean
    public String hello() {
        return "hello prod";
    }
}
```

src/main/java/com/basic/boot/TestConfiguration.java

```
@Profile("test")
@Configuration
public class TestConfiguration {
    @Bean
    public String hello() {
        return "hello test";
    }
}
```

src/main/java/com/basic/boot/runner/MyRunner.java

```
@Component
public class MyRunner implements ApplicationRunner {
    @Autowired
    private String hello;
    public void run(ApplicationArguments args) throws Exception {
        System.out.println(hello);
    }
}
```

src/test/resources/application.properties

spring.profiles.active=prod

Spring Boot 활용

5. 스프링 부트 프로파일

- jar 실행시 아규먼트 설정 (properties file 보다 우선 순위가 더 높다)

```
target>java -jar jartest-0.0.1-SNAPSHOT.jar --spring.profiles.active=test
```

- 프로파일용 properties file

: application-{profile}.properties

: application-prod.properties / application-test.properties

src/main/resources/application-prod.properties

```
myboot.name=SpringBoot prod
```

src/main/resources/application-test.properties

```
myboot.name=SpringBoot test
```

src/main/java/com/basic/boot/runner/MyRunner.java

```
@Component
public class MyRunner implements ApplicationRunner {
    @Autowired
    MybootProperties mybootProperties;
    public void run(ApplicationArguments args) throws Exception {
        System.out.println(mybootProperties.getName());
    }
}
```

Spring Boot 활용

6. Logging #1

- 로깅 퍼사드 vs 로거

: Commons Logging , **SLF4j**(simple logging façade)

- Logger API를 추상해 놓은 로깅 퍼사드 인터페이스들
- 로깅 퍼사드를 통해서 Logger를 사용했을 때 장점은 로깅 구현체들을 교체하기 쉽도록 해준다.

: JUL(java.util.logging), Log4j2, **Logback**

- 로깅 퍼사드 구현체들

- 스프링 부트 로깅

--debug (일부 핵심 라이브러리만 디버깅 모드로)

--trace (전부 다 디버깅 모드로)

로그 파일 출력: logging.file.path

로그 레벨 조정: logging.level.패키지명 = 로그 레벨

Logging

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-logging.html>

Spring Boot 활용

6. Logging #1

- 스프링 부트 기본 로거 설정

파일 출력: logging.path를 지정하면 logs/spring.log 파일이 생성된다.

로그 레벨 조정: logging.level.패키지명 = 로그 레벨

src/main/resources/application-test.properties

```
logging.file.path=logs
logging.level.com.basic.boot=debug
```

src/main/java/com/basic/boot/runner/MyRunner.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component
public class MyRunner implements ApplicationRunner {
    @Autowired
    MybootProperties mybootProperties;
    private Logger logger = LoggerFactory.getLogger(MyRunner.class)

    public void run(ApplicationArguments args) throws Exception {
        logger.debug("-----");
        logger.debug(mybootProperties.getName());
    }
}
```

Spring Boot 활용

6. Logging #2

- Default 로거인 logback 설정 커스터마이징

<https://docs.spring.io/spring-boot/docs/current/reference/html/howto-logging.html>

- 스프링 부트에서는 기본적으로 logback 모듈을 제공합니다. 따라서 logback 모듈을 아래와 같이 xml 파일로 따로 설정 정보를 관리하면서 개발할 수 있습니다.

src/main/resources/logback-spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
    <logger name="com.basic.boot" level="DEBUG"/>
</configuration>
```

Spring Boot 활용

6. Logging #2

- 로거를 Log4j2로 변경하기
- logback을 사용하지 않고 다른 로깅 모듈(log4j2)로 바꾸고 싶을 때는 pom.xml에 아래와 같이 logback 모듈에 대한 의존성을 제거해야 합니다.

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<!-- log4j2 -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

Spring Boot 활용

7. Spring-Boot-Devtools

- 클래스 패스에 있는 파일이 변경 될 때마다 자동으로 재 시작 해준다.
: 캐쉬 설정을 개발 환경에 맞게 off 해준다. 직접 껐다 켜는 (cold start) 보다 빠르다.

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

: devtools dependency 추가

pom.xml

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

: restart 기능 끄려면? `spring.devtools.restart.enabled=false`

스프링 부트 데이터

Spring Boot 데이터터

1. Spring Data : In-Memory 데이터베이스

- 지원하는 In-Memory 데이터베이스
 - : H2(추천, 콘솔기능 제공), HSQL, Derby
- H2 데이터베이스 의존성 추가와 설정

pom.xml

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.3.176</version>
  <scope>runtime</scope>
</dependency>
```

application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
```


Spring Boot 데이터

1. Spring Data : In-Memory 데이터베이스

- H2 데이터베이스 기본 연결 정보 확인

: URL => jdbc:h2:mem:testdb (DB URL이 콘솔에 보여짐)

: Username => SA

src/main/java/com/basic/boot/runner/DatabaseRunner.java

```
@Component
public class DatabaseRunner implements ApplicationRunner {
    @Autowired
    DataSource dataSource;

    @Override
    public void run(ApplicationArguments args) throws Exception {
        try(Connection connection = dataSource.getConnection()){
            System.out.println(connection.getMetaData().getURL());
            System.out.println(connection.getMetaData().getUserName());
        }
    }
}
```

Spring Boot 데이터터

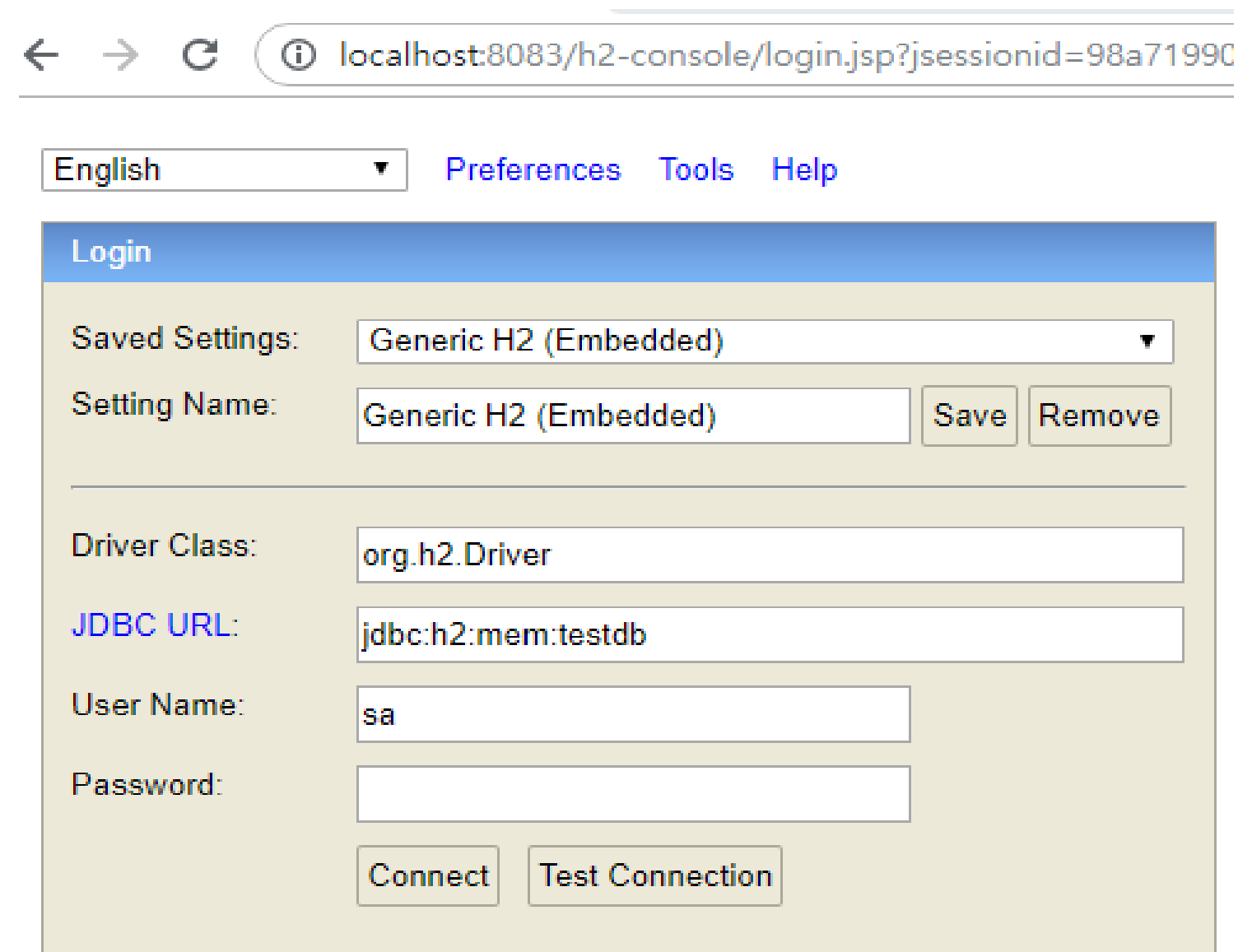
1. Spring Data : In-Memory 데이터베이스

- H2 Console 사용하는 방법

`spring.h2.console.enabled=true`

`http://localhost:8080/h2-console`

JDBC URL를 `jdbc:h2:mem:testdb` 로 설정한다.



Spring Boot 데이터

2. Spring Data : Maria 데이터베이스

- 스프링 부트가 지원하는 DBCP(Database Connection Pooling)

: HikariCP (기본) - `spring.datasource.hikari.*`

<https://github.com/brettwooldridge/HikariCP#frequently-used>

: Tomcat CP - `spring.datasource.tomcat.*`

: Commons DBCP2 - `spring.datasource.dbcp2.*`

- MariaDB Client 의존성 추가

pom.xml

```
<dependency>
  <groupId>org.mariadb.jdbc</groupId>
  <artifactId>mariadb-java-client</artifactId>
  <version>2.7.3</version>
</dependency>
```

Spring Boot 데이터터 – 사용자 계정과 DB 생성

root 계정으로 접속하여 사용자 계정 생성

```
mysql -u root -p
```

```
MariaDB [(none)]> show databases;
```

```
MariaDB [(none)]> use mysql;
```

```
MariaDB [mysql]> create user 'boot'@'%' identified by 'boot';
```

```
MariaDB [mysql]> grant all on *.* to 'boot'@'%';
```

```
MariaDB [mysql]> select user, host from user;
```

```
MariaDB [mysql]> flush privileges;
```

```
MariaDB [mysql]> exit;
```

boot 사용자 계정으로 접속하여 DB생성

```
mysql -u boot -p
```

```
MariaDB [(none)]> show grants for current_user;
```

```
MariaDB [(none)]> create database boot_db;
```

```
MariaDB [(none)]> show databases;
```

```
MariaDB [(none)]> use boot_db;
```

Spring Boot 데이터터

- Spring Data : MariaDB DataSource 설정
- MariaDB DataSource 설정

src/main/resources/application.properties

```
spring.datasource.url=jdbc:mariadb://127.0.0.1:3306/boot_db  
spring.datasource.username=boot  
spring.datasource.password=boot  
spring.datasource.driverClassName=org.mariadb.jdbc.Driver
```

Spring Boot 데이터

3. Spring Data : Oracle 데이터베이스

- Oracle JDBC Driver 의존성 추가 (local에 ojdbc6.jar가 준비되어 있어야 한다.)

pom.xml

```
<!-- Oracle JDBC driver -->
<dependency>
  <groupId>ojdbc</groupId>
  <artifactId>ojdbc</artifactId>
  <version>6</version>
  <scope>system</scope>
  <systemPath>${basedir}/lib/ojdbc6.jar</systemPath>
</dependency>
```

- Oracle DataSource 설정

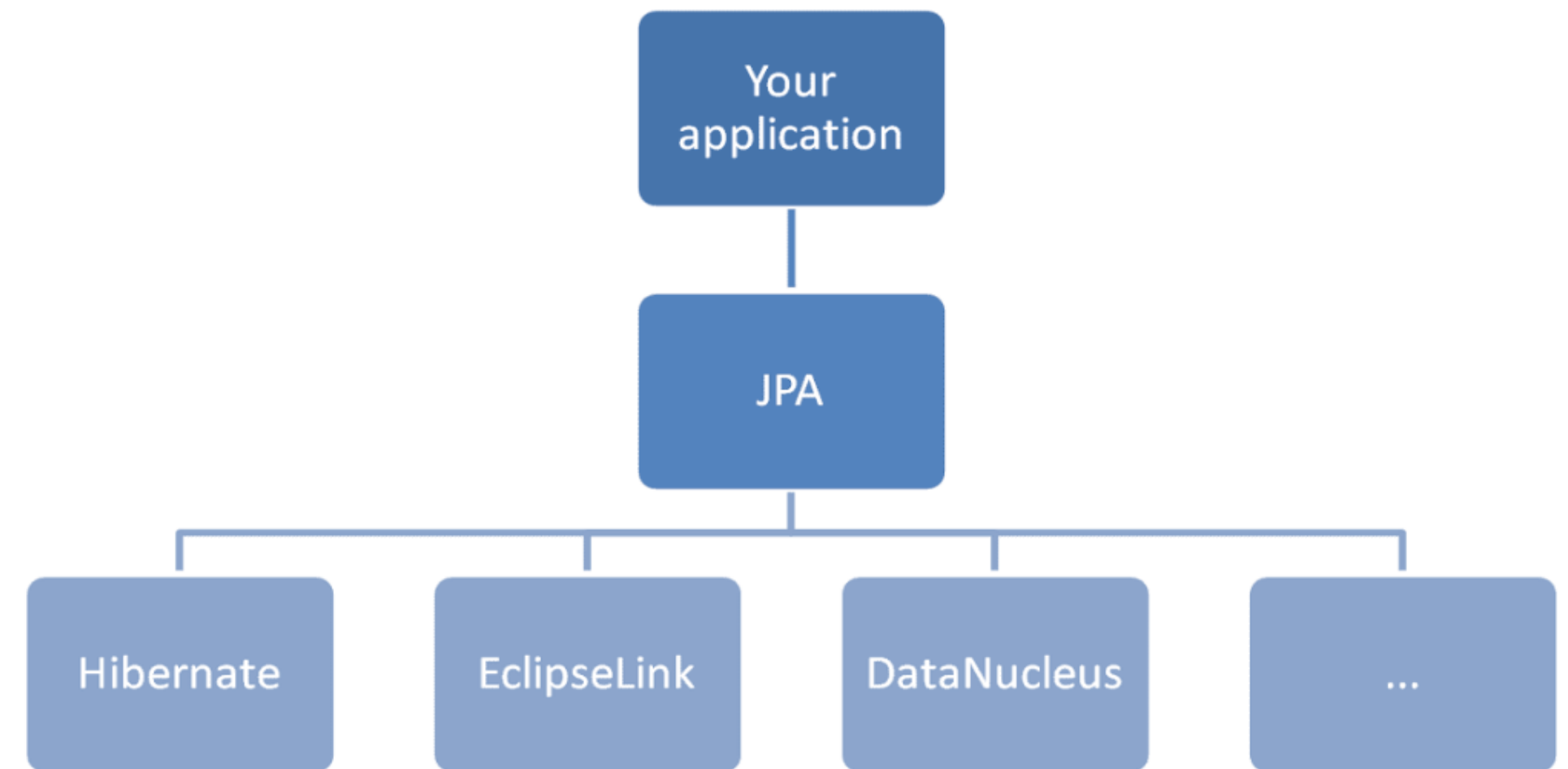
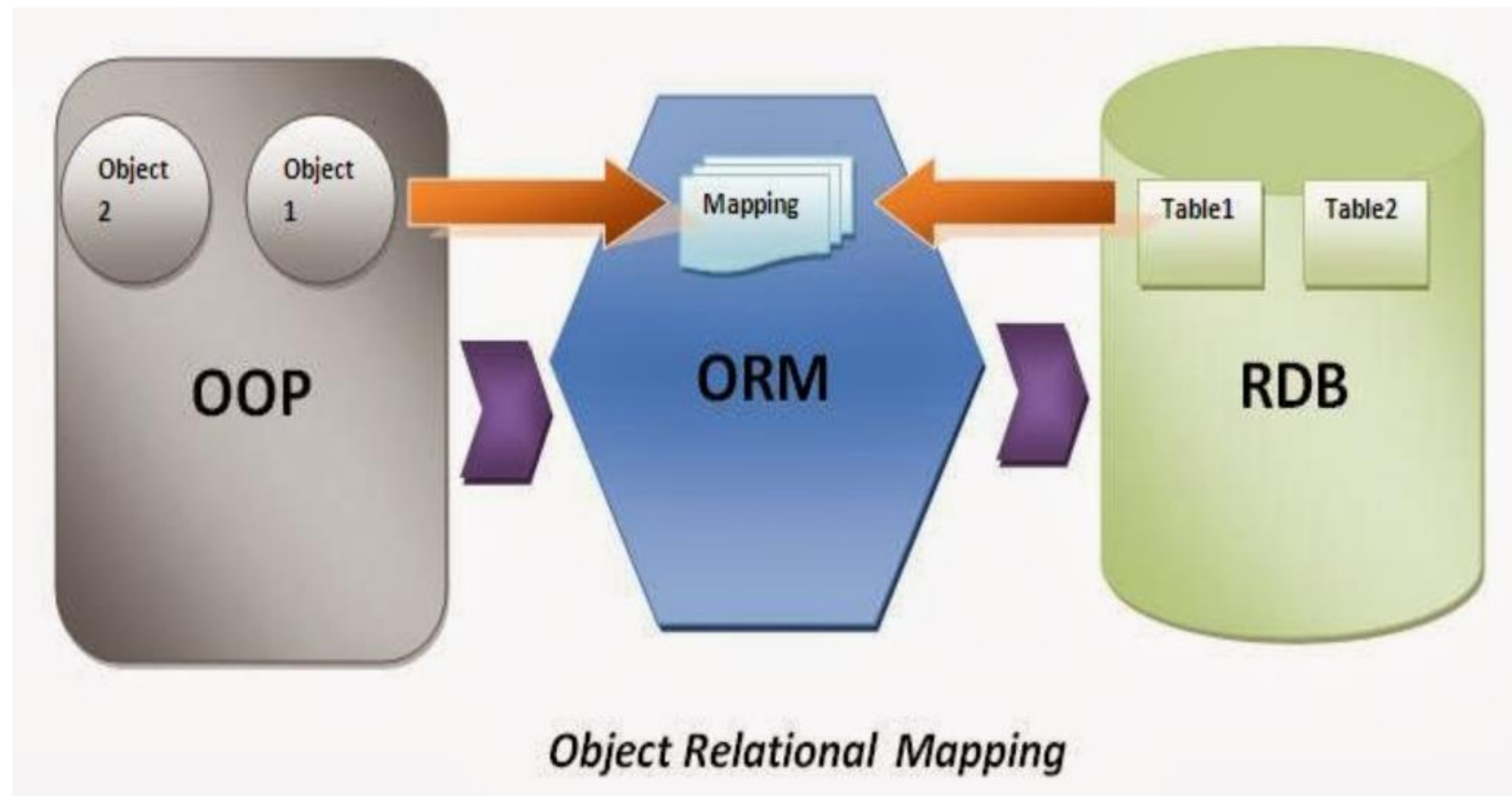
src/main/resources/application.properties

```
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
spring.datasource.username=scott
spring.datasource.password=tiger
spring.datasource.driverClassName=oracle.jdbc.OracleDriver
```

Spring Boot 데이터

4. Spring Data : ORM과 JPA

- ORM(Object-Relational Mapping)과 JPA (Java Persistence API)
: 객체와 릴레이션을 맵핑 할 때 발생하는 개념적 불일치를 해결하는 프레임워크
- JPA (Java Persistence API)
: ORM을 위한 자바 (EE) 표준이다.



Spring Boot 데이터

4. Spring Data : Spring-Data-JPA

- Spring-Data-JPA 란?
 - : JPA를 쉽게 사용하기 위해 스프링에서 제공하는 프레임워크 입니다.
 - : Repository Bean을 자동 생성해 준다.
 - : 쿼리 메소드 자동 구현
 - : `@EnableJpaRepositories` (스프링 부트가 자동으로 설정 해줌.)
- 스프링 데이터 JPA 의존성 추가

pom.xml

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


Spring Boot 데이터터

4-1. Spring Data : Entity 클래스 작성

- `@Entity` : 엔티티 클래스임을 지정하며 DB 테이블과 매핑하는 객체를 나타내는 어노테이션입니다. 엔티티(Entity)란 데이터베이스에서 표현하려고 하는 유형, 무형의 객체로서 서로 구별되는 것을 뜻합니다. 이 객체들은 DB 상에서는 보통 `table`로서 나타내어 집니다.
- `@Id` : 엔티티의 기본키를 나타내는 어노테이션입니다.
- `@GeneratedValue` : 주 키의 값을 자동 생성하기 위해 명시하는 데 사용되는 어노테이션입니다. 자동 생성 전략은 (**AUTO**, IDENTITY, SEQUENCE, TABLE) 이 있습니다.
- Entity 클래스 작성

src/main/java/com/basic/boot/entity/Account.java

```
@Entity
public class Account {
    @Id @GeneratedValue
    private Long id;
    @Column(unique=true)
    private String username;
    @Column
    private String password;

    getter() , setter(), equals(), hashCode()
}
```

Spring Boot 데이터터

4-1. Spring Data : Repository 인터페이스 작성

- Repository 인터페이스 작성

: AccountRepository의 구현체를 따로 작성하지 않아도 Spring-Data-JPA가 자동적으로 해당 문자열 username에 대한 인수를 받아 자동적으로 DB Table과 매핑합니다.

```
src/main/java/com/basic/boot/repository/AccountRepository.java
```

```
public interface AccountRepository extends JpaRepository<Account, Long>{  
    Account findByUsername(String username);  
}
```

Query method

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

Spring Boot 데이터

4-2. Spring Data : JPA를 사용한 데이터베이스 초기화

- application.properties 파일에 JPA에 의한 데이터베이스 자동 초기화 설정

src/main/resources/application.properties

```
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.show-sql=true  
spring.jpa.database-platform=org.hibernate.dialect.MariaDB103Dialect
```

Spring Boot 데이터

4-2. Spring Data : JPA를 사용한 데이터베이스 초기화

- `spring.jpa.hibernate.ddl-auto=create|create-drop|update|validate`

- ✓ create

JPA가 DB와 상호작용할 때 기존에 있던 스키마(테이블)을 삭제하고 새로 만드는 것을 뜻한다.

- ✓ create-drop

JPA 종료 시점에 기존에 있었던 테이블을 삭제합니다.

- ✓ update

기존 스키마는 유지하고, 새로운 것만 추가하고, 기존의 데이터도 유지한다. 변경된 부분만 반영함
주로 개발 할 때 적합하다.

- ✓ validate

엔티티와 테이블이 정상 매핑 되어 있는지를 검증합니다.

- `spring.jpa.show-sql=true`

: JPA가 생성한 SQL문을 보여줄 지에 대한 여부를 알려주는 프로퍼티입니다.

Spring Boot 데이터

▪ Spring Data : Dialect (방언)이란?

- ANSI SQL은 모든 DBMS에서 공통적으로 사용이 가능한 표준 SQL이지만 DBMS에서 만든 SQL은 DB벤더들만의 독자적인 기능을 추가 하기 위해 만든 것으로 사용하는 특정 벤더의 DBMS에서만 사용이 가능합니다.
- JPA는 기본적으로 어플리케이션에서 직접 JDBC 레벨의 SQL을 작성하지 않고 JPA가 직접 SQL을 생성해줍니다. JPA에 어떤 DBMS를 사용하는지를 알려주는 방법이 방언(Dialect)을 설정하는 방법입니다.
- JPA에 Dialect를 설정할 수 있는 추상화 방언 클래스를 제공하고 설정된 방언으로 각 DBMS에 맞는 구현체를 제공합니다.
- `hibernate.dialect=org.hibernate.dialect.MariaDB103Dialect`

MariaDB 10.3 버전 사용시

```
database-platform: org.hibernate.dialect.MariaDB103Dialect
```

Oracle 11g 버전 사용시

```
database-platform: org.hibernate.dialect.Oracle10gDialect
```

Spring Boot 데이터

4-2. Spring Data : JPA를 사용한 데이터베이스 초기화

- Update 옵션 설정 시 아래와 같이 SQL문이 자동적으로 실행되는 것을 볼 수 있다.

```
Hibernate: alter table if exists public.account add column password varchar(255)
```

```
Hibernate: alter table if exists public.account add column username varchar(255)
```

```
Hibernate: create sequence hibernate_sequence start 1 increment 1
```

- Create 옵션 설정 시 테이블을 drop하고 다시 생성하는 SQL문이 실행되는 것을 볼 수 있다.

```
Hibernate: drop table if exists account cascade
```

```
Hibernate: drop sequence if exists hibernate_sequence
```

```
Hibernate: create sequence hibernate_sequence start 1 increment 1
```

```
Hibernate: create table account (id int8 not null, password varchar(255), username varchar(255), primary key (id))
```

- validate 옵션은 엔티티와 해당 매핑하려는 테이블이 제대로 매핑 되었는지 지 체크합니다.
- 운영 모드에서 적합한 설정

```
spring.jpa.hibernate.ddl-auto=validate
```

Spring Boot 데이터

4-3. Spring Data : JPA 테스트 #1

src/test/java/com/basic/boot/repository/AccountRepositoryTest.java

```
@SpringBootTest
public class AccountRepositoryTest {
    @Autowired
    AccountRepository accountRepository;
    @Test
    public void account() throws Exception {
        Account account = new Account();
        account.setUsername("basic");
        account.setPassword("pass");
        Account newAcct = accountRepository.save(account);
        System.out.println(newAcct.getId() + " " + newAcct.getUsername());
        assertNotNull(newAcct);

        Account existAcct = accountRepository.findByUsername(newAcct.getUsername());
        assertNotNull(existAcct);
        Account notExistAcct = accountRepository.findByUsername("test");
        assertNull(notExistAcct);
    }
}
```

Spring Boot 데이터

4-4. Spring Data : Repository 인터페이스 수정

- Optional 객체 반환
 - : Java8은 함수형 언어의 접근 방식에서 영감을 받아 `java.util.Optional<T>`라는 새로운 클래스를 도입하였습니다. (<https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>)
 - : Optional는 "존재할 수도 있지만 없을 수도 있는 객체", 즉, "null이 될 수도 있는 객체" 을 감싸고 있는 일종의 래퍼 클래스입니다.
 - : 명시적으로 해당 변수가 null일 수도 있다는 가능성을 표현할 수 있습니다.
(따라서 불필요한 NullPointerException 방어 로직을 줄일 수 있습니다.)

src/main/java/com/basic/boot/repository/AccountRepository.java

```
import java.util.Optional;

public interface AccountRepository extends JpaRepository<Account, Long>{
    Optional<Account> findByUsername(String username);
}
```


Spring Boot 데이터

4-4. Spring Data : JPA 테스트 #2

- 스프링 데이터 Repository 테스트 클래스 작성 : 레코드 저장 및 조회

src/test/java/com/basic/boot/repository/AccountRepositoryTest.java

```
@SpringBootTest
public class AccountRepositoryTest {
    @Test
    public void account() throws Exception {
        Account account = new Account();
        account.setUsername("basic");
        account.setPassword("pass");
        Account newAcct = accountRepository.save(account);
        System.out.println(newAcct.getId() + " " + newAcct.getUsername());
        assertThat(newAcct).isNotNull();

        Optional<Account> existAcct = accountRepository.findByUsername(newAcct.getUsername());
        assertThat(existAcct).isNotEmpty();
        Optional<Account> notExistAcct = accountRepository.findByUsername("test");
        assertThat(notExistAcct).isEmpty();
    }
}
```

Spring Boot 데이터

4-5. Spring Data : JPA를 사용한 데이터베이스 초기화

- validate 속성 테스트 : Account 클래스에 email 속성을 추가(컬럼 추가)한다.
- 테이블과 매핑하려는 엔티티가 이 테이블과 맞지 않을 경우 에러를 발생하게 됩니다.

src/main/java/com/basic/boot/entity/Account.java

```
@Entity
public class Account {
    @Id @GeneratedValue
    private Long id;
    @Column(unique=true)
    private String username;
    @Column
    private String password;
    @Column
    private String email;

    public String getEmail() { return email; }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

src/main/resources/application.properties

```
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
```

org.hibernate.tool.schema.spi.SchemaManagementException: Schema-validation: missing column [email] in table [account]

Spring Boot 데이터

4-5. Spring Data : JPA를 사용한 데이터베이스 초기화

- update 속성 테스트 : update로 설정값을 변경하고 다시 어플리케이션을 실행하게 되면 엔티티 소스 코드 상에서 추가 되었던 email 변수가 그대로 테이블에 반영되는 것을 볼 수 있습니다.

```
src/main/resources/application.properties
```

```
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true
```

```
Hibernate: alter table if exists public.account add column email varchar(255)
```

```
springboot=# select * from account;  
id | password | username | email  
----+-----+-----+-----
```

스프링 부트 웹 MVC

Spring Boot Web MVC

1. Spring Boot Web MVC

- web on Servlet Stack

: <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html>

- 자동 설정으로 제공하는 여러 기본 기능

Spring MVC 설정을 개발자가 하지 않아도 내부에 `spring-boot-autoconfigure.jar` 파일에 포함된 `META-INF` 디렉토리 내에 `spring.factories`의

`org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration`에서 `WebMvc`와 관련된 자동 설정 클래스가 적용되기 때문이다.

- 스프링 MVC 확장 (추가적인 설정)

: `@Configuration + WebMvcConfigurer` 인터페이스 구현

Spring Boot Web MVC

1. Spring Boot Web MVC : RestController (JSON)

- @RequestBody 어노테이션을 통한 HTTP 메시지와 객체 매핑

: JsonMessageConverter

HTTP 요청 본문을 json객체로 변경하거나, json객체를 HTTP 응답 본문으로 변경할 때는 JsonMessageConverter가 사용 된다.

`{"username":"basic", "password":"123"}` <-> User

: Controller에서 json 타입에 대한 정보를 명시하지 않아도 ContentNegotiationViewResolver를 통해 자동적으로 json 형식으로 데이터를 반환하도록 스프링 부트에서 제공함. 이 ViewResolver는 Converter와 연관되어 있어 Content-type을 기준으로 어떤 Converter를 사용할지 결정한다.

: @PostMapping

@GetMapping과 비슷하게 @RequestMapping(method=RequestMethod.POST)의 축약형

Spring Boot Web MVC

1. Spring Boot Web MVC :RestController (JSON) – Entity와 Repository

src/main/java/com/basic/boot/entity/User.java

```
@Entity
public class User {
    @Id
    @GeneratedValue
    private Long id;
    @Column
    private String name;
    @Column(unique=true)
    private String email;
}
```

src/main/java/com/basic/boot/repository/UserRepository.java

```
public interface UserRepository extends JpaRepository<User, Long>{
    Optional<User> findByName(String name);
}
```

Spring Boot Web MVC

1-1. Spring Boot Web MVC :RestController (JSON) – Controller #1

src/main/java/com/basic/boot/controller/UserRestController.java

```
@RestController
public class UserRestController {
    @Autowired
    private UserRepository userRepository;

    @PostMapping("/users")
    public User create(@RequestBody User user) {
        return userRepository.save(user);
    }

    @RequestMapping(value = "/users/{id}")
    public User getUser(@PathVariable Long id) {
        return userRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("User", "id", id));
    }

    @RequestMapping(value="/users", produces = { "application/json" })
    public List<User> getUsers() {
        return userRepository.findAll();
    }
}
```


Spring Boot Web MVC

1-1. Spring Boot Web MVC :RestController (JSON) – Controller #2

src/main/java/com/basic/boot/controller/UserRestController.java

```
@DeleteMapping("/users/{id}")
public ResponseEntity<?> deleteUser(@PathVariable Long id) {
    User user = userRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("User", "id", id));
    userRepository.delete(user);
    //return ResponseEntity.ok(user);
    return ResponseEntity.ok().build();
}

@PutMapping("/users/{id}")
public User updateUser(@PathVariable Long id, @RequestBody User userDetail) {
    User user = userRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("User", "id", id));
    user.setName(userDetail.getName());
    user.setEmail(userDetail.getEmail());
    User updatedUser = userRepository.save(user);
    return updatedUser;
}
}
```

Spring Boot Web MVC

1-2. Spring Boot Web MVC :RestController (JSON) – 사용자 정의 Exception 클래스

src/main/java/com/basic/boot/exception/ResourceNotFoundException.java

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    private String resourceName;
    private String fieldName;
    private Object fieldValue;

    public ResourceNotFoundException( String resourceName, String fieldName, Object fieldValue) {
        super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));
        this.resourceName = resourceName;
        this.fieldName = fieldName;
        this.fieldValue = fieldValue;
    }

    public String getResourceName() { return resourceName; }
    public String getFieldName() { return fieldName; }
    public Object getFieldValue() { return fieldValue; }
}
```

Spring Boot Web MVC

2. Spring Boot Web MVC : RestController (XML)

- ViewResolver
: Controller에서 반환한 값(ModelAndView 혹은 Model)을 통해 뷰를 만드는 역할
- ContentNegotiationViewResolver
: 동일한 URI에서 HTTP Request에 있는 Content-type 및 Accept 헤더를 기준으로 다양한 Content-type으로 응답할 수 있게 하는 ViewResolver
- XML 메시지 컨버터 의존성 추가하기

pom.xml

```
<dependency>  
  <groupId>com.fasterxml.jackson.dataformat</groupId>  
  <artifactId>jackson-dataformat-xml</artifactId>  
  <version>2.12.1</version>  
</dependency>
```

Spring Boot Web MVC

2. Spring Boot Web MVC : RestController (XML)

src/main/java/com/basic/boot/controller/UserRestController.java

```
@RestController
public class UserRestController {

    @RequestMapping(value="/users2", produces = {"application/xml"})
    public List<User> getUsers2() {
        return userRepository.findAll();
    }

    @RequestMapping(value="/usersxml", produces = { "application/xml"})
    public Users getUsersXml() {
        Users users = new Users();
        users.setUsers(userRepository.findAll());
        return users;
    }
}
```

```
<List>
  <Item>
    <id>1</id>
    <name>홍길동</name>
    <email>test@aa.com</email>
  </Item>
</List>

<Users>
  <User id="1">
    <name>홍길동</name>
    <email>test@aa.com</email>
  </User>
</Users>
```

Spring Boot Web MVC

2. Spring Boot Web MVC : RestController (XML)

src/main/java/com/basic/boot/entity/Users.java

```
@JacksonXmlRootElement
public class Users implements Serializable{

    private static final long serialVersionUID = 22L;

    @JacksonXmlProperty(localName="User")
    @JacksonXmlElementWrapper(useWrapping=false)
    private List<User> users = new ArrayList<>();

    public void setUsers(List<User> users) {
        this.users = users;
    }

    public List<User> getUsers() {
        return users;
    }

}
```

src/main/java/com/basic/boot/entity/User.java

```
@Entity
public class User implements Serializable {
    private static final long serialVersionUID = 21L;

    @Id
    @GeneratedValue
    @JacksonXmlProperty(isAttribute = true)
    private Long id;

    @JacksonXmlProperty
    private String name;

    @JacksonXmlProperty
    private String email;
}
```

Spring Boot Web MVC

3. Spring Boot Web MVC : 정적 리소스 위치

- 정적 리소스 지원
- 기본 리소스 위치는 아래와 같다.

`classpath:/static`

`classpath:/public`

`classpath:/resources/`

`classpath:/META-INF/resources`

예) `http://localhost:8080/hello.html`

src/main/resources/static/hello.html

```
<html>
<body>
    <h1>Hello Static Resources</h1>
</body>
</html>
```

Spring Boot Web MVC

3. Spring Boot Web MVC : 정적 리소스 맵핑 설정 변경

- `spring.mvc.static-path-pattern`: 맵핑 설정 변경 가능하다.
- “/hello.html” => /static/hello.html

```
src/main/resources/application.properties
```

```
spring.mvc.static-path-pattern=/static/**
```

Spring Boot Web MVC

3. Spring Boot Web MVC : 정적 리소스 맵핑 커스터 마이징

- webMvcConfigurer를 통한 정적 리소스 매핑

: webMvcConfigurer의 addResourceHandlers 메서드를 재정의 하여 Spring MVC가 제공하는 기본 리소스 설정을 그대로 유지하면서 리소스 맵핑 설정을 추가할 수 있다.

: addResourceHandlers는 리소스 등록 및 핸들러를 관리하는 객체인 ResourceHandlerRegistry를 통해 리소스 위치와 이 리소스와 매칭될 url을 등록합니다.

: setCachePeriod()는 캐시를 얼마나 지속할 지 정하는 메서드입니다. 20초로 설정됨.

src/main/java/com/basic/boot/config/WebConfig.java

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/m/**")
            //반드시 m 다음에 / 을 주어야 한다.
            .addResourceLocations("classpath:/m/")
            .setCachePeriod(20); //20초
    }
}
```

src/main/resources/m/hello.html

```
<html>
<body>
    <h1>Mobile Hello Static
Resources</h1>
</body>
</html>
```

http://localhost:8080/m/hello.html

Spring Boot Web MVC

4. Spring Boot Web MVC : webjar

- Front End에서 사용되는 Javascript library도 maven에서 다운받을 수 있다.

<https://www.webjars.org/>

- jQuery library 추가

pom.xml

```
<dependency>
  <groupId>org.webjars.bower</groupId>
  <artifactId>jquery</artifactId>
  <version>3.3.1</version>
</dependency>
```

src/main/resources/hello.html

```
<body>
  <h1>Hello Static Resources</h1>
  <script src="/webjars/jquery/3.3.1/dist/jquery.min.js"> </script>
  <script>
    $(function() {
      alert("ready!");
    });
  </script>
</body>
```

- library 버전을 생략하고 사용하려면 webjars-locator-core 의존성 추가해야 한다.

pom.xml

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>webjars-locator-core</artifactId>
  <version>0.36</version>
</dependency>
```

```
<script src= "/webjars/jquery/dist/jquery.min.js"> </script>
```

Spring Boot Web MVC

5. Spring Boot Web MVC : index페이지와 favicon

- welcome 페이지

index.html을 찾고 있으면 제공하고, 없으면 에러페이지를 보여준다.

- 파비콘 favicon

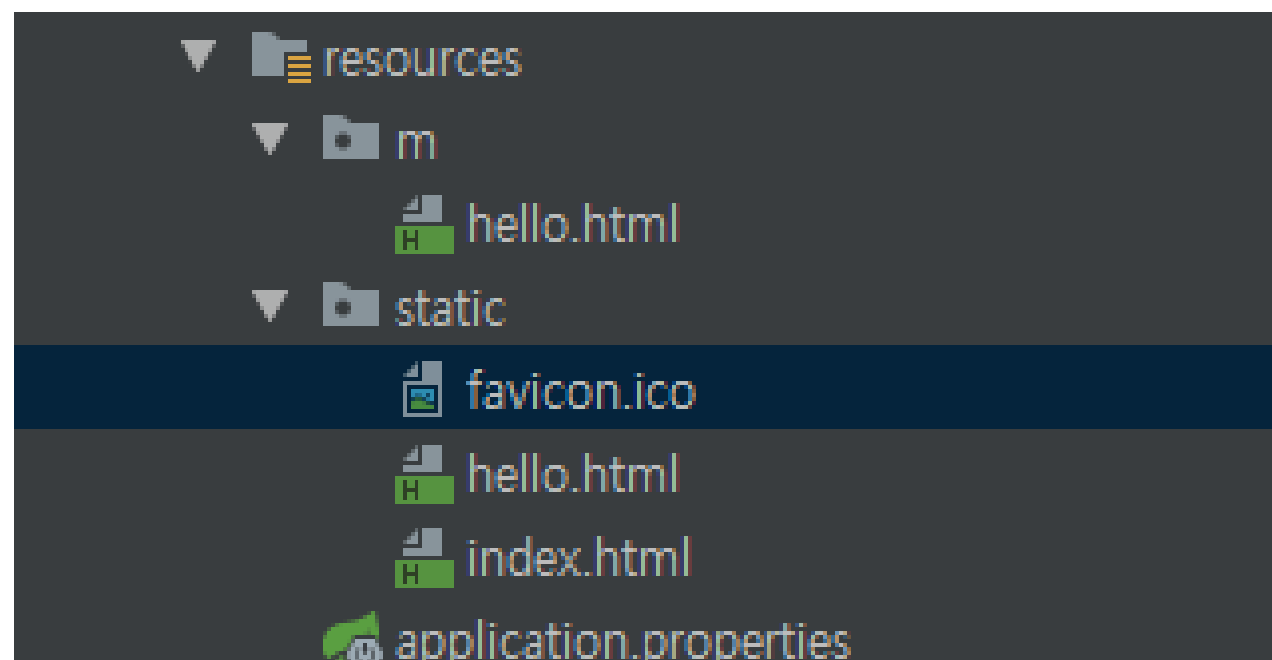
favicon.ico 만들기 <https://favicon.io/favicon-generator/>

favicon이 안 바뀔 때?

<https://stackoverflow.com/questions/2208933/how-do-i-force-a-favicon-refresh>

- 다음 아래의 경로에 파비콘을 추가합니다.

단, 파일명은 favicon.ico으로 해야합니다.



Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- 타임리프(Thymeleaf)

: Thymeleaf는 스프링 부트가 자동 설정을 지원하는 웹 템플릿 엔진입니다. HTML문서에 HTML5 문법으로 서버쪽 로직을 수행하고 적용시킬 수 있습니다.

: HTML 디자인에 전혀 영향을 미치지 않고 웹 템플릿 엔진을 통해 HTML을 생성할 수 있습니다.

: 템플릿 엔진, th:xx 형식으로 속성을 html 태그에 추가하여 값을 처리할 수 있습니다.

: JSP, Groovy등 다른 템플릿도 스프링 부트에서 사용 가능하지만 thymeleaf가 가장 많이 사용된다.

: 타임리프 페이지는 <html xmlns:th="http://www.thymeleaf.org"> 로 시작해야 합니다.

- JSP를 권장하지 않는 이유

- ✓ JAR 패키징 할 때는 동작하지 않고, WAR 패키징 해야 함.

- ✓ Undertow(Servlet Engine)는 JSP를 지원하지 않음

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#boot-features-jsp-limitations>

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

<https://www.thymeleaf.org/>

<https://www.thymeleaf.org/doc/articles/standarddialect5minutes.html>

- Thymeleaf 의존성 추가

pom.xml

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

- Thymeleaf 템플릿 페이지 위치

`/src/main/resources/templates/`

- Thymeleaf 튜토리얼 : <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html>
- Thymeleaf+Spring 튜토리얼 : <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- Thymeleaf 표현식

- ✓ 1. Variable Expressions - `${ }`

: 해당 Context의 포함된 변수들을 사용할 수 있습니다.

```
<p>Today is: <span th:text="${today}">13 february 2011</span>.</p>
```

- ✓ 2. Selection Variable Expressions - `*{ }`

: 가까운 DOM에 th:object로 정의된 변수가 있다면 그 변수에 포함된 값을 나타낼 수 있습니다.

```
<div th:object="${session.user}">
  <p>Name: <span th:text="*{firstName}">Sebastian</span>.</p>
  <p>Surname: <span th:text="*{lastName}">Pepper</span>.</p>
  <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>
```

- ✓ 3. Message Expressions - `#{ }`

: 미리 정의된 message.properties 파일이 있다면 #표현식으로 나타낼 수 있습니다.

- ✓ 4. Link URL Expressions - `@{ }`

: @표현식을 이용하여 다양하게 URL을 표현할 수 있습니다.

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- Thymeleaf 예제
 - : Model 객체에 포함된 값을 통해 Thymeleaf 템플릿 엔진이 해당 템플릿에서 명시한 값을 변환합니다.
 - : `xmlns:th="http://www.thymeleaf.org"` 를 명시해야 템플릿이 제대로 렌더링 됩니다.
 - : `th:text="${name}"`에서 Model에서 넘어온 값을 변환합니다.

src/main/java/com/basic/boot/controller/TemplateController.java

```
@Controller
public class TemplateController {
    @GetMapping("/leaf")
    public String leaf(Model model) {
        model.addAttribute("name","basic");
        return "leaf";
    }
}
```

src/main/resources/templates/leaf.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Spring Boot Thymeleaf</title>
</head>
<body>
<h1 th:text="${name}">Name</h1>
<h1>Hello, <span th:text="${name}"></span> </h1>
<h1>Hello, [[${name}]]</h1>
</body>
</html>
```

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- User 리스트 Controller와 Page

src/main/java/com/basic/boot/controller/UserController.java

```
@Controller
public class UserController {
    @Autowired
    UserRepository userRepository;

    @GetMapping("/index")
    public String index(Model model) {
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }
}
```

src/main/resources/static/index.html

```
<html>
<body>
    <h2> <a href="/index">사용자관리</a> </h2>
</body>
</html>
```

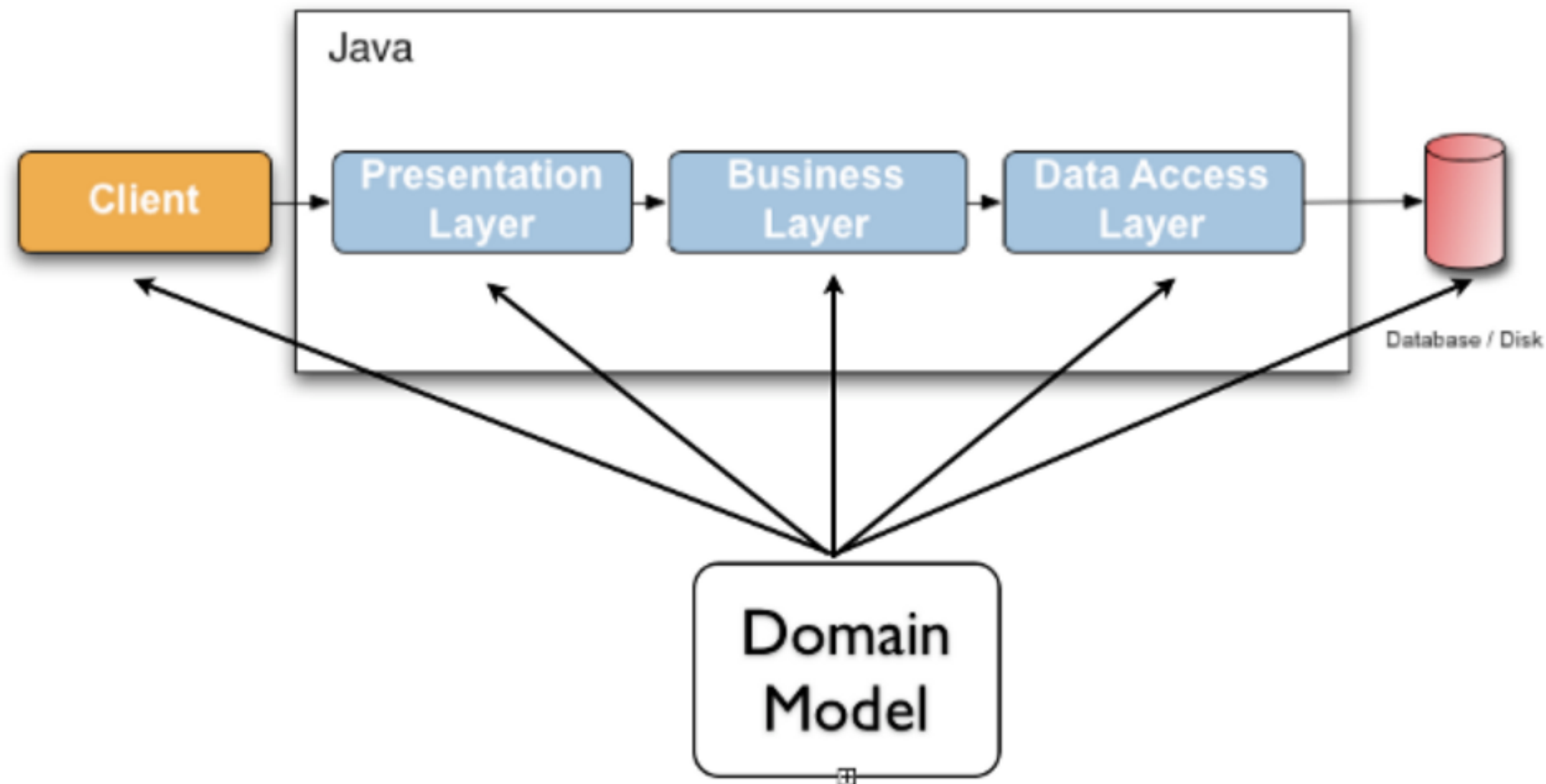
src/main/resources/templates/index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<meta charset="UTF-8">
<body>
    <table>
        <tr>
            <th>Name</th>
            <th>Email</th>
        </tr>
        <tr th:each="user : ${users}">
            <td th:text="${user.name}"> </td>
            <td th:text="${user.email}"> </td>
        </tr>
    </table>
</body>
</html>
```


Spring Boot Web MVC : @Valid

■ Java Bean Validation

- 데이터 검증을 위한 로직을 도메인 모델 자체에 묶어서 표현하는 방법이 있습니다.
- Java Bean Validation 에서 데이터 검증을 위한 어노테이션 (Annotation) 을 제공하고 있습니다.



pom.xml

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


Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- User 등록 Controller 와 Page

src/main/java/com/basic/boot/controller/UserController.java

```
@Controller
public class UserController {
    @GetMapping("/signup")
    public String showSignUpForm(User user) {
        return "add-user";
    }

    @PostMapping("/adduser")
    public String addUser(@Valid User user, BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "add-user";
        }
        userRepository.save(user);
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }
}
```

src/main/resources/templates/index.html

```
<a href="/signup">insert</a>
```

User.java

```
@Entity
public class User {
    @NotBlank(message = "Name is mandatory")
    @JacksonXmlProperty
    private String name;

    @NotBlank(message = "Email is mandatory")
    @JacksonXmlProperty
    private String email;
}
```

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- `@Valid` (J2EE API doc <https://javaee.github.io/javaee-spec/javadocs/>)
 - `Dispatcherservlet`이 메소드 안에 선언된 객체를 만들어주고 값을 넣어준다.
 - `@Valid` 선언된 객체에 설정을 검사한 후에 검증 에러가 있다면 `BindingResult`에 담아준다.
 - 검증에러가 발생하면 `FieldError`객체를 만들어서 `BindingResult`에 넣어준다.
 - Thymeleaf 태그에서 object에 담긴 객체의 프로퍼티를 `field *{필드이름}` 형식으로 사용한다
 - Thymeleaf 태그 `errors`는 `BindingResult`에 있는 에러값을 출력해준다.
 - `BindingResult.hasErrors()` : 에러가 있는지 검사한다.

- `BindingResult`

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/validation/BindingResult.html>

- Thymeleaf 의 validation 과 Error Messages

<https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html#validation-and-error-messages>

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- User 등록 페이지

src/main/resources/templates/add-user.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<meta charset="UTF-8">
<body>
    <form action="#" th:action="@{/adduser}" th:object="${user}" method="post">
        <label for="name">Name</label>
        <input type="text" th:field="*{name}" id="name">
        <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}"> </span>
        <br/>
        <label for="email">Email</label>
        <input type="text" th:field="*{email}" id="email">
        <span th:if="${#fields.hasErrors('email')}" th:errors="*{email}"> </span>
        <br/>
        <input type="submit" value="Add User">
    </form>
</body>
</html>
```

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- 한글 인코딩 설정

src/main/resources/application.properties

Charset of HTTP requests and responses. Added to the "Content-Type" header if not set explicitly.

spring.http.encoding.charset=UTF-8

Enable http encoding support.

spring.http.encoding.enabled=true

Force the encoding to the configured charset on HTTP requests and responses.

spring.http.encoding.force=true

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- User 수정 Controller

src/main/java/com/basic/boot/controller/UserController.java

```
@Controller
public class UserController {
    @GetMapping("/edit/{id}")
    public String showUpdateForm(@PathVariable("id") long id, Model model) {
        User user = userRepository.findById(id).orElseThrow(() -> new IllegalArgumentException("Invalid user Id:" + id));
        model.addAttribute("user", user);
        return "update-user";
    }
    @PostMapping("/update/{id}")
    public String updateUser(@PathVariable("id") long id, @Valid User user, BindingResult result, Model model) {
        if (result.hasErrors()) {
            user.setId(id);
            return "update-user";
        }
        userRepository.save(user);
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }
}
```

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- User 수정 페이지 열기

src/main/resources/templates/index.html

```
<tr> <th>Name</th> <th>Email</th> <th>Edit</th> </tr>
<tr th:each="user : ${users}">
  <td th:text="${user.name}"> </td>
  <td th:text="${user.email}"> </td>
  <td> <a th:href="@{/edit/{id}(id=${user.id})}">update</a> </td>
</tr>
```

src/main/resources/templates/update-user.html

```
<form action="#" th:action="@{/update/{id}(id=${user.id})}" th:object="${user}" method="post">
  <label for="name">Name</label>
  <input type="text" th:field="*{name}" id="name">
  <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}"> </span> <br />
  <label for="email">Email</label>
  <input type="text" th:field="*{email}" id="email">
  <span th:if="${#fields.hasErrors('email')}" th:errors="*{email}"> </span> <br />
  <input type="submit" value="Update User">
</form>
```

Spring Boot Web MVC

6. Spring Boot Web MVC : Thymeleaf

- User 삭제 Controller

src/main/java/com/basic/boot/controller/UserController.java

```
public class UserController {
    @GetMapping("/delete/{id}")
    public String deleteUser(@PathVariable("id") long id, Model model) {
        User user = userRepository.findById(id).orElseThrow(() -> new IllegalArgumentException("Invalid user Id:" + id));
        userRepository.delete(user);
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }
}
```

src/main/resources/templates/index.html

```
<tr> <th>Name</th> <th>Email</th> <th>Edit</th> <th>Delete</th> </tr>
<tr th:each="user : ${users}">
    <td th:text="${user.name}"> </td>
    <td th:text="${user.email}"> </td>
    <td> <a th:href="@{/edit/{id}(id=${user.id})}">update</a> </td>
    <td> <a th:href="@{/delete/{id}(id=${user.id})}">delete</a> </td>
</tr>
```

Spring Boot Web MVC

7. Spring Boot Web MVC : HtmlUnit

- Html 템플릿 뷰 테스트를 좀 더 전문적으로 하고 싶을 때 사용한다.

<http://htmlunit.sourceforge.net/>

<http://htmlunit.sourceforge.net/gettingStarted.html>

- HtmlUnit은 프로그래밍적으로 HTML 사이트와 상호작용할 수 있게 하는 자바 오픈소스입니다.
- 테스트 프레임워크로서 생각할 수 있지만 브라우저와 프로그래밍적으로 상호작용 할 수 있게 하는 확장된 개념입니다.
- 스프링 4 이후로 스프링에 통합되어 MVC 테스트(특히 템플릿 뷰 테스트) 때 유용하게 쓰일 수 있습니다.

Spring Boot Web MVC

7. Spring Boot Web MVC : HtmlUnit

- HtmlUnit 의존성 추가

pom.xml

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>htmlunit-driver</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>net.sourceforge.htmlunit</groupId>
  <artifactId>htmlunit</artifactId>
  <scope>test</scope>
</dependency>
```

Spring Boot Web MVC

7. Spring Boot Web MVC : HtmlUnit

- HtmlUnit 예제 테스트

- : HtmlPage 객체를 통해 html 페이지와 자바 언어를 통해 상호작용 할 수 있습니다.
- : xpath를 통해 h1 태그의 정보를 얻어서 그것을 테스트 하는 코드입니다.
- : 웹 페이지의 타이틀이 무엇인지 테스트할 수 있는 등 여러가지 웹 클라이언트와 상호 할 수 있는 API를 제공합니다.

src/test/java/com/basic/boot/controller/TemplateControllerTest.java

```
@RunWith(SpringRunner.class)
@WebMvcTest(TemplateController.class)
public class TemplateControllerTest {
    @Autowired
    WebClient webClient;

    @Test
    public void html() throws Exception {
        HtmlPage page = webClient.getPage("/leaf");
        HtmlHeading1 h1 = page.getFirstByXPath("//h1");
        assertThat(h1.getTextContent()).isEqualToIgnoringCase("basic");
    }
}
```

Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- 스프링 부트에서는 `ExceptionHandler`를 기본적으로 등록하여 `Exception`을 처리하고 있습니다.
- 기본 예외 처리기는 스프링에서 자동적으로 등록하는 `BasicErrorController`에서 관리합니다.
(에러 발생 시 JSON 형식으로 리턴)
- 커스텀 `Exception` 핸들러, 커스텀 `Exception` 클래스를 만들어서 예외를 처리할 수 있습니다.
- 스프링 @MVC 예외 처리 방법
- `@ExceptionHandler`
: 스프링은 컨트롤러 안에 `@ExceptionHandler`를 선언한 메서드가 존재하면 그 메서드가 컨트롤러 내부의 예외를 처리하도록 해준다.
- `@ControllerAdvice`
: `@ControllerAdvice` 애노테이션을 통해서 이 클래스의 객체가 컨트롤러에서 발생하는 `Exception`을 전문적으로 처리하는 클래스라는 것을 명시한다.

Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- BasicExceptionHandler는 스프링 부트가 제공하는 기본 예외 처리기 역할을 담당한다.
: `org.springframework.boot.autoconfigure.web.servlet.error.BasicExceptionHandler`
: HTML과 JSON 응답 지원
- <https://www.mkyong.com/spring-mvc/spring-mvc-exceptionhandler-example/>

Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- @ExceptionHandler를 사용한 예제 작성 : TemplateController 클래스

src/main/java/com/basic/boot/controller/TemplateController.java #1

```
@Controller
public class TemplateController {
    @GetMapping("/custom")
    public String custom(Model model) {
        throw new CustomException("E888", "This is custom message");
    }
    @GetMapping("/generic")
    public String generic(Model model) throws Exception{
        throw new Exception();
    }
    @ExceptionHandler(CustomException.class)
    public ModelAndView handleCustomException(CustomException ex) {
        ModelAndView model = new ModelAndView("error/generic_error");
        model.addObject("errCode", ex.getErrCode());
        model.addObject("errMsg", ex.getErrMsg());
        return model;
    }
}
```

Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- @ExceptionHandler를 사용한 예제 작성 : TemplateController 클래스

src/main/java/com/basic/boot/controller/TemplateController.java #2

```
@ExceptionHandler(Exception.class)
public ModelAndView handleAllException(Exception ex) {
    ModelAndView model = new ModelAndView("error/generic_error");
    model.addObject("errMsg", "this is Exception.class");
    return model;
}
}
```

Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- @ExceptionHandler를 사용한 예제 작성 : CustomException 클래스

src/main/java/com/basic/boot/exception/CustomException.java

```
public class CustomException extends RuntimeException {  
    private String errCode;  
    private String errMsg;  
  
    public CustomException(String errCode, String errMsg) {  
        this.errCode = errCode;  
        this.errMsg = errMsg;  
    }  
    getter()  
    setter()  
}
```

Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- @ExceptionHandler를 사용한 예제 작성 : generic_error.html

src/main/resources/templates/error/generic_error.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Spring Boot Thymeleaf</title>
</head>
<body>
    <h2>Spring Boot MVC @ExceptionHandler Example</h2>
    <div th:if="!${#strings.isEmpty(errCode)}">
        <h2 th:text="${errCode}">: System Errors</h2>
    </div>
    <div th:if="${#strings.isEmpty(errCode)}">
        <h2> System Errors</h2>
    </div>
    <div th:if="!${#strings.isEmpty(errMsg)}">
        <h2 th:text="${errMsg}"> </h2>
    </div>
</body>
</html>
```


Spring Boot Web MVC

8. Spring Boot Web MVC : 예외처리

- HTML 문서를 작성할 시 HTTP Status(에러) 코드에 맞게 에러 페이지를 작성해야 합니다.
- HTML 문서의 파일명이 상태코드와 같거나 아니면 5xx 와 같이 패턴을 맞추어서 만들어야 합니다.
- Http 에러 코드 값에 따른 에러 페이지 작성 디렉토리
 - : `src/main/resources/static/error/`
 - `404.html`
 - `5xx.html`

Spring Boot Web MVC

9. Spring Boot Web MVC : CORS

- Ajax 통신은 보안 상의 이슈 때문에 동일 출처(Single Origin Policy)를 준수합니다.
- SOP(Single Origin Policy)
 - : 같은 origin에만 요청을 보낼 수 있다.
- CORS(Cross-Origin Resource Sharing)
 - : Single Origin Policy를 우회하기 위한 기법
 - : 서로 다른 origin 간에 resource를 share 하기 위한 방법
- Origin 이란?
 - : URI 스키마 (http, https) + hostname (localhost) + 포트 (8080, 18080)
- @CrossOrigin 어노테이션
 - : @Controller나 @RequestMapping 어노테이션과 같이 사용할 수 있다.
 - : @CrossOrigin(origins="http://localhost:18080")
- webMvcConfigurer 사용해서 글로벌 설정

Spring Boot Web MVC

9. Spring Boot Web MVC : CORS

- 프로젝트를 두개 생성하여 하나는 클라이언트 쪽에서 직접 리소스를 요청하는 웹 어플리케이션, 다른 하나는 ajax를 통해 다른 출처에서 자원을 요청할 때 그 자원을 제공하는 웹 어플리케이션을 만듭니다.
- 자원을 제공하는 웹어플리케이션의 `CrossOrigin`을 허용해 주는 `Config` 클래스
: `WebMvcConfigurer` 사용해서 글로벌 설정

src/main/java/com/basic/boot/config/WebConfig.java

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:18080")
            .allowedMethods("*");
    }
}
```

Spring Boot Web MVC

9. Spring Boot Web MVC : CORS

- 자원을 요청하는 client 웹어플리케이션의 페이지

src/main/resources/application.properties

```
server.port=18080
```

src/main/resources/static/index.html

```
<script src="/webjars/jquery/3.3.1/dist/jquery.min.js"> </script>
<script>
    $(function() {
        $.ajax("http://localhost:8080/hello")
        .done(function(msg){ alert(msg);})
        .fail(function(){ alert("fail"); });
    });
</script>
```

스프링 부트 Actuator

Spring Boot Actuator

1. Spring Boot Actuator 소개

스프링 부트는 애플리케이션 운영 환경에 필요한 유용한 기능들을 제공합니다. 엔드포인트와 메트릭스 그 데이터를 활용하는 모니터링 기능들이 있습니다.

- 스프링 부트 Actuator 소개

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#production-ready-endpoints>

- 애플리케이션의 각종 정보를 확인할 수 있는 Endpoints
 - ✓ 다양한 Endpoints 제공.
 - ✓ JMX 또는 HTTP를 통해 접근 가능 함.
 - ✓ shutdown을 제외한 모든 Endpoint는 기본적으로 활성화 상태
- 스프링 부트 Actuator 의존성 추가

pom.xml

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

Spring Boot Actuator

2. Spring Boot Actuator 사용하기

1) HTTP 사용하기

<http://localhost:8080/actuator>

- health와 info를 제외한 대부분의 Endpoint가 기본적으로 비공개 상태이다.
- 공개 옵션 조정

```
src/main/resources/application.properties
```

```
management.endpoints.web.exposure.include=*  
management.endpoints.web.exposure.exclude=env,beans
```

2) Jconsole 사용하기

- JDK/bin 에 있는 jconsole.exe를 실행한다.

<https://docs.oracle.com/javase/tutorial/jmx/mbeans/>

<https://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html>

3) visualvm 사용하기

<https://visualvm.github.io/download.html>

Spring Boot Actuator

2. Spring Boot Actuator 사용하기

4) Spring-Boot Admin

<https://github.com/codecentric/spring-boot-admin>

스프링 부트 Actuator UI 를 제공한다.

4.1) Admin Server 역할을 하는 새로운 스프링 부트 프로젝트를 생성한다. (<https://start.spring.io/>)

: Web Dependency를 추가한다.

: server.port=8090 으로 설정한다.

: @EnableAdminServer 어노테이션 선언

: spring-boot-admin-starter-server dependency 추가한다.

src/main/java/com.basic.admin/AdminApplication

```
@SpringBootApplication
@EnableAdminServer
public class SpringbootAdminApplication {
}
```

pom.xml

```
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-starter-server</artifactId>
</dependency>
```


Spring Boot Actuator

2. Spring Boot Actuator 사용하기

4) Spring-Boot Admin

4.2) 기존에 작성했던 프로젝트에 admin-client dependency를 추가한다.

: spring.boot.admin.client.url=http://localhost:8090 을 설정한다.

src/main/resources/application.properties

```
spring.boot.admin.client.url=http://localhost:8090
```

pom.xml

```
<dependency>  
  <groupId>de.codecentric</groupId>  
  <artifactId>spring-boot-admin-starter-client</artifactId>  
</dependency>
```

Spring Boot Actuator

2. Spring Boot Actuator 사용하기

- Spring-Boot Admin Sever 요청

The screenshot shows the Spring Boot Admin web interface. The browser address bar displays 'localhost:8090/#/applications'. The header includes the 'Spring Boot Admin' logo and navigation links for 'Wallboard', 'Applications', 'Journal', and 'About'. The main content area features three summary cards: 'APPLICATIONS' with a count of '1', 'INSTANCES' with a count of '1', and 'STATUS' with the text 'all up'. Below these cards, a table lists the registered applications. The first entry is 'spring-boot-application', which is marked as 'UP' with a green checkmark and a '2m' uptime indicator. The application's URL, 'http://192.168.0.121:8083/', is highlighted with a red rectangular box.

APPLICATIONS	INSTANCES	STATUS
1	1	all up

UP	Application	URL
✓ 2m	spring-boot-application	http://192.168.0.121:8083/

스프링 부트 Security

Spring Boot 시큐리티

1. Spring Security 소개

- 웹 시큐리티, 메서드 시큐리티
- 다양한 인증 방법 지원
 - : Basic 인증, Form 인증, OAuth, LDAP
- 스프링 부트 시큐리티 자동 설정
 - : SecurityAutoConfiguration
 - : UserDetailsServiceImplAutoConfiguration
 - : 모든 요청에 인증이 필요함.
 - : 기본 사용자를 자동으로 생성해준다.

Username : user

Password : 애플리케이션을 실행할 때 마다 랜덤 값 생성 (콘솔에 출력됨)

Spring Boot 시큐리티

2. Spring Security : Basic 인증 구현

- spring boot security 의존성 추가

pom.xml

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

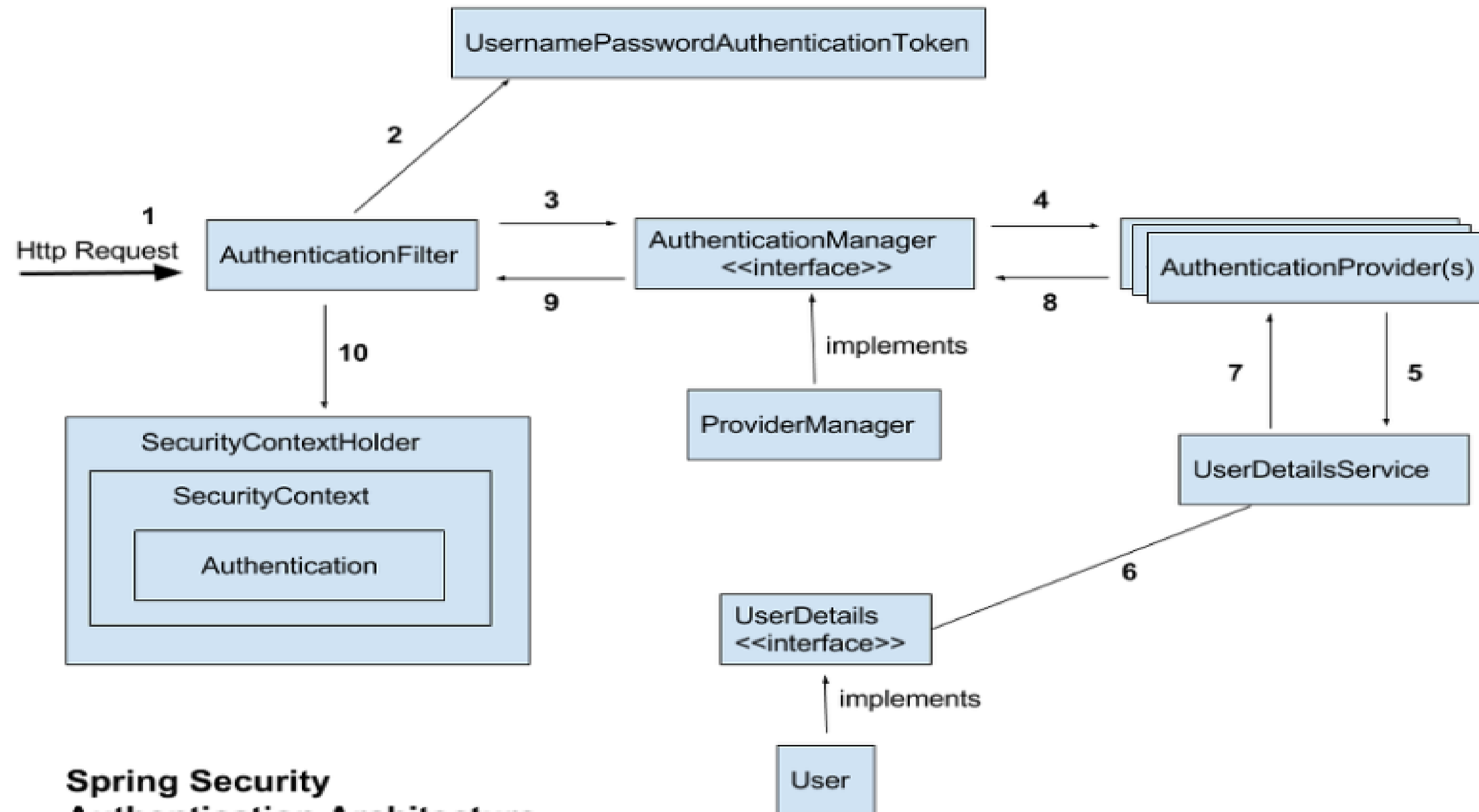
- index.html 요청 - 로그인 창이 자동으로 뜬다

Username : user

Password : 애플리케이션을 실행할 때 마다 랜덤 값 생성 (콘솔에 출력됨)

Spring Boot 시큐리티

■ 인증관련 Architecture



**Spring Security
Authentication Architecture**

Spring Boot 시큐리티

- 1) 사용자가 Form을 통해 로그인 정보를 입력하고 인증 요청을 보낸다.
- 2) UsernamePasswordAuthenticationFilter가 사용자가 보낸 아이디와 패스워드를 인터셉트한다.
HttpServletRequest에서 꺼내온 사용자 아이디와 패스워드를 진짜 인증을 담당할 AuthenticationManager 인터페이스(구현체 - ProviderManager)에게 인증용 객체(UsernamePasswordAuthenticationToken)로 만들어줘서 위임한다.
- 3) AuthenticationFilter에게 인증용 객체(UsernamePasswordAuthenticationToken)을 전달받는다.
- 4) 실제 인증을 할 AuthenticationProvider에게 Authentication객체(UsernamePasswordAuthenticationToken)을 다시 전달한다.
- 5) DB에서 사용자 인증 정보를 가져올 UserDetailsService 객체에게 사용자 아이디를 넘겨주고 DB에서 인증에 사용할 사용자 정보(사용자 아이디, 암호화된 패스워드, 권한 등)를 UserDetails라는 객체로 전달 받는다.
- 6) AuthenticationProvider는 UserDetails 객체를 전달 받은 이후 실제 사용자의 입력정보와 UserDetails 객체를 가지고 인증을 시도한다.
- 7) 7,8,9,10. 인증이 완료되면 사용자 정보를 가진 Authentication 객체를 SecurityContextHolder에 담은 이후 AuthenticationSuccessHandler를 실행한다.(실패시 AuthenticationFailureHandler를 실행한다.)

Spring Boot 시큐리티

2. Spring Security : Basic 인증 구현

- TemplateController 클래스에 메서드 추가
- static/index.html에 링크 추가
- templates/mypage.html 작성

src/main/java/com/basic/boot/controller/TemplateController.java

```
@Controller
public class TemplateController {

    @GetMapping("/mypage")
    public String mypage() {
        return "mypage";
    }
}
```

src/main/resources/static/index.html

```
<a href="/mypage">MyPage</a>
```

src/main/resources/templates/mypage.html

```
<h1>My Page</h1>
```


Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-1) 로그인 정보 페이지 작성하기

src/main/resources/templates/mypage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<meta charset="UTF-8">
<body>
<h1>My Page</h1>
  <div>
    Logged in user: <b th:inline="text" class="user"> [[${#httpServletRequest.remoteUser}]] </b>
    <form th:action="@{/app-logout}" method="POST">
      <input type="submit" value="Logout"/>
    </form>
  </div>
</body>
</html>
```

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-2) SecurityConfig 클래스 작성

: @Configuration 과 @EnableWebSecurity 어노테이션을 선언한다.

src/main/java/com/basic/boot/config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/mypage/**").authenticated()
            .antMatchers("/**").permitAll()
            .and()
            .formLogin()
            .and()
            .httpBasic();
        return http.build();
    }
}
```

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-3) 로그아웃 기능 추가

src/main/java/com/basic/boot/config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/mypage/**").authenticated()
            .antMatchers("/**").permitAll()
            .and()
            .formLogin()
            .and()
            .httpBasic()
            .and()
            .logout() //logout configuration
            .logoutUrl("/app-logout")
            .deleteCookies("JSESSIONID")
            .logoutSuccessUrl("/");
        return http.build();
    }
}
```

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-4) AccountService 클래스 작성

<https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#jc-authentication-userdetailsservice>

- UserDetailsService 인터페이스를 구현한 AccountService 클래스에 Account를 생성하는 createAccount() 메서드를 추가한다.

src/main/java/com/basic/boot/service/AccountService.java

```
import org.springframework.security.core.userdetails.UserDetailsService;
@Service
public class AccountService implements UserDetailsService {
    @Autowired
    private AccountRepository accountRepository;
    //Account 레코드 추가
    public Account createAccount(String username, String password) {
        Account account = new Account();
        account.setUsername(username);
        account.setPassword(password);
        return accountRepository.save(account);
    }
}
```

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-4) AccountService 클래스 작성

- AccountService Bean이 생성된 후에 바로 createAccount() 메서드가 호출되도록 @PostConstruct 어노테이션을 사용한다.

src/main/java/com/basic/boot/service/AccountService.java

```
@Service
public class AccountService implements UserDetailsService {
    @PostConstruct
    public void init() {
        Optional<Account> byUsername = accountRepository.findByUsername("myboot");
        if(!byUsername.isPresent()) {
            Account newAccount = this.createAccount("myboot", "1234");
            System.out.println(newAccount);
        }
    }
}
```

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-5) AccountService 클래스 작성 : UserDetailsService의 loadUserByUsername() 메서드 구현

src/main/java/com/basic/boot/service/AccountService.java

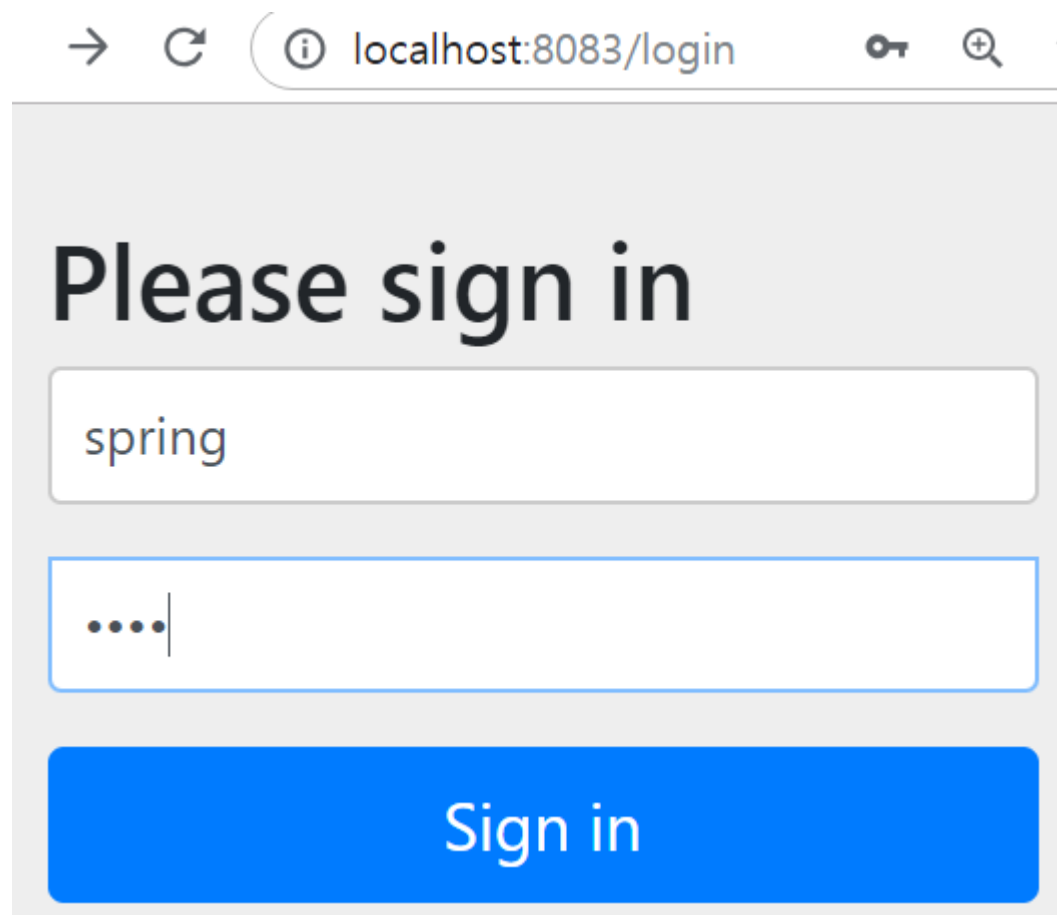
```
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
@Service
public class AccountService implements UserDetailsService {
    @Override
    //login 할때 사용자가 입력한 정보가 유효한지를 체크한다.
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<Account> byUsername = accountRepository.findByUsername(username);
        Account account = byUsername.orElseThrow(() -> new UsernameNotFoundException(username));
        return new User(account.getUsername(),
                        account.getPassword(), authorities());
    }
    //User 객체의 세번째 인자 USER라는 ROLE을 가진 사용자이다 라고 설정하는 부분
    private Collection<? extends GrantedAuthority> authorities() {
        return Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"));
    }
}
```

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-6) 로그인 실행

- Account에 등록된 username와 password로 로그인 한다.
- Password를 인코딩 하지 않아서 `IllegalArgumentException` 발생함



→ ↻ ⓘ localhost:8083/login 🔑 🔍

Please sign in

spring

...

Sign in

`java.lang.IllegalArgumentException:`
There is no PasswordEncoder mapped for the id "null"

Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-7) SecurityConfig 클래스에 PasswordEncoder를 Bean으로 등록

- PasswordEncoder

<https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#core-services-password-encoding>

- SecurityConfig 클래스에 passwordEncoder()를 bean으로 등록한다.

src/main/java/com/basic/boot/config/SecurityConfig.java

```
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
}
```


Spring Boot 시큐리티

3. Spring Security : Security 설정 커스터 마이징

3-8) AccountService 클래스에서 PasswordEncoder 사용

- 등록된 passwordEncoder bean을 주입 받아서 password를 인코딩한다.

src/main/java/com/basic/boot/service/AccountService.java

```
@Service
public class AccountService implements UserDetailsService {
    @Autowired
    private PasswordEncoder passwordEncoder;

    public Account createAccount(String username, String password) {
        Account account = new Account();
        account.setUsername(username);
        account.setPassword(passwordEncoder.encode(password));
        //account.setPassword(password);
        return accountRepository.save(account);
    }
}
```



판교 | 경기도 성남시 분당구 삼평동 대왕판교로 670길 유스페이스2 B동 8층 T. 070-5039-5805
가산 | 서울시 금천구 가산동 371-47 이노플렉스 1차 2층 T. 070-5039-5815
웹사이트 | <http://edu.kosta.or.kr> 팩스 | 070-7614-3450