



# SpringBoot 활용

---

플랫폼서비스팀

김설한

2021.08.11

---

## 01 기본 기능

- SpringApplication
- 외부설정 & Profile

## 02 기술 연동

- 스프링 웹 MVC
- 스프링 데이터
- 스프링 시큐리티

# 0) SpringBoot 프로젝트 생성

- Spring 이니셜라이저에서 사용할 dependency들을 추가한 뒤 프로젝트를 만들어준다.

※ spring 이니셜라이저 : <https://start.spring.io/>

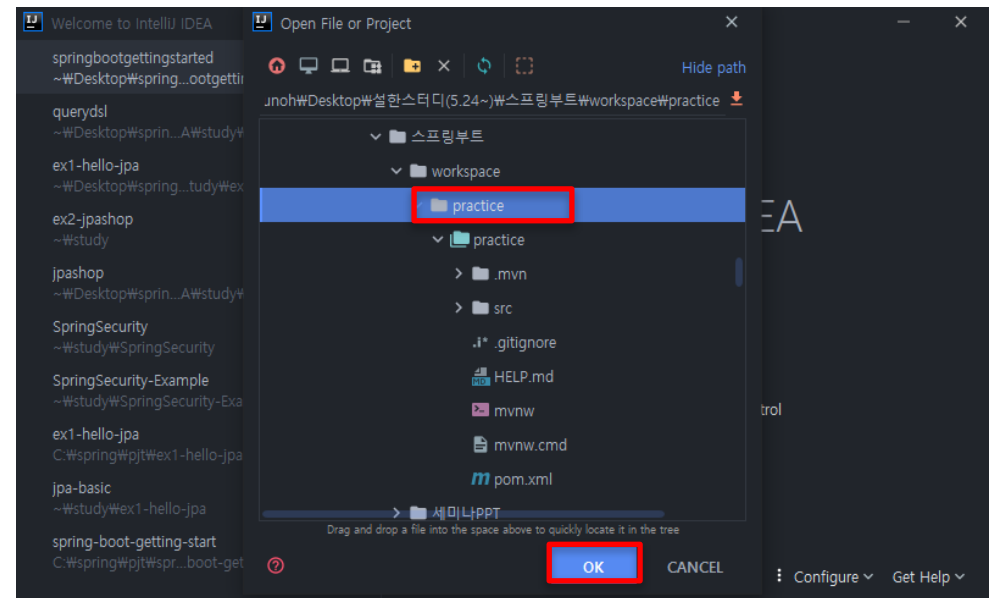
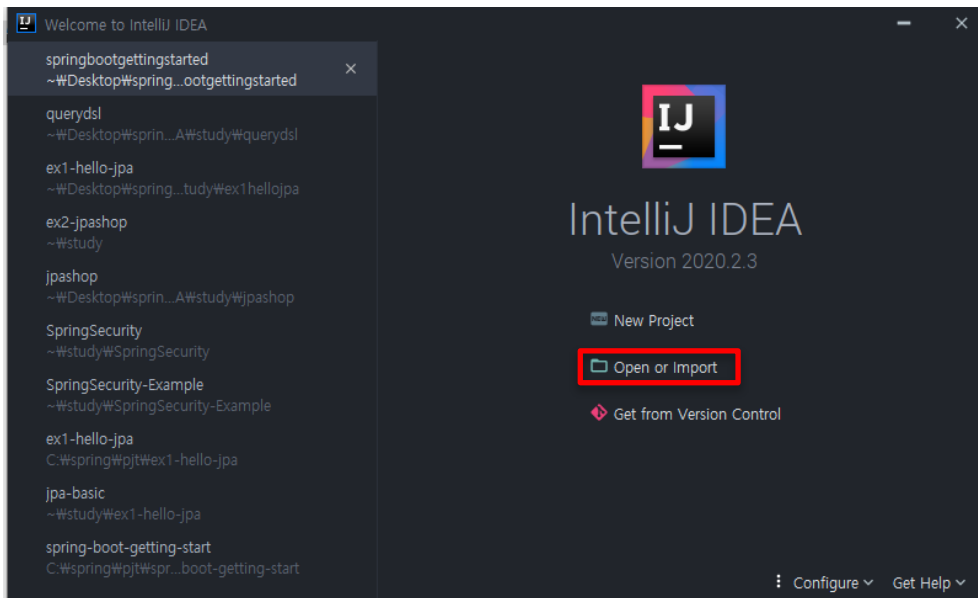
The screenshot displays the Spring Initializr web interface. On the left, there's a sidebar with a hamburger menu and social media icons. The main area is divided into several sections:   
1. **Project**: Includes radio buttons for 'Maven Project' (selected) and 'Gradle Project'.   
2. **Language**: Includes radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'.   
3. **Spring Boot**: Includes radio buttons for various versions, with '2.5.3' selected.   
4. **Project Metadata**: A form with fields for 'Group' (springboot.snowone), 'Artifact' (practice), 'Name' (practice), 'Description' (Demo project for Spring Boot), and 'Package name' (springboot.snowone.practice). It also has 'Packaging' (Jar selected, War unselected) and 'Java' version (16, 11, 8) options.   
5. **Dependencies**: A list of dependencies with checkboxes: 'Spring Web' (WEB), 'Spring Security' (SECURITY), 'H2 Database' (SQL), 'Spring Data JPA' (SQL), 'Lombok' (DEVELOPER TOOLS), and 'Thymeleaf' (TEMPLATE ENGINES). A button 'ADD DEPENDENCIES... CTRL + B' is present.   
6. **Actions**: At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.   
7. **Footer**: At the very bottom, there's a file download indicator 'practice.zip' and a '모두 표시' (Show all) button.

## 0) SpringBoot 프로젝트 생성

- [GENERATE]로 생성된 프로젝트 zip파일을 적당한 위치에 옮겨 압축을 풀어주고

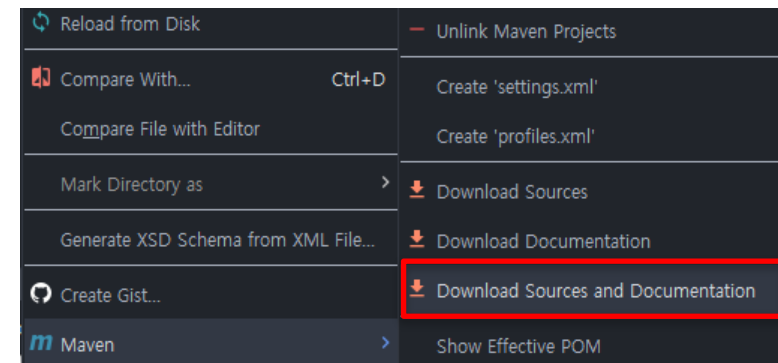
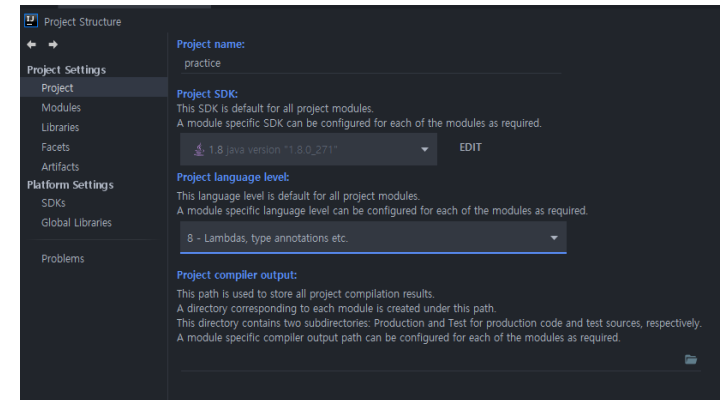
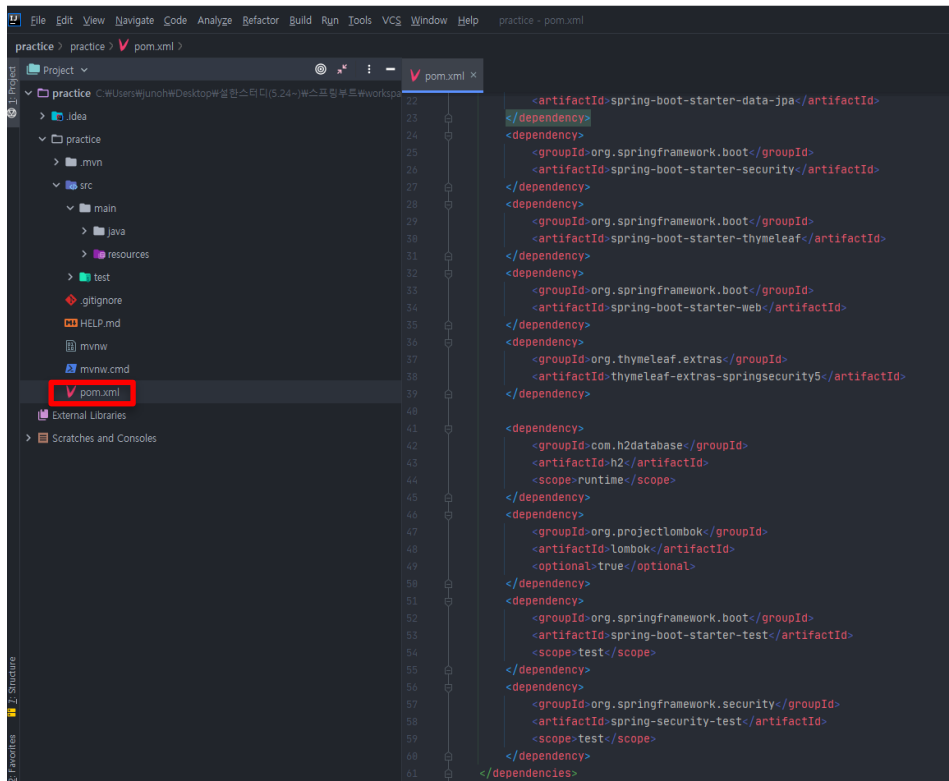
practice	2021-08-05 오전 9:54	파일 폴더	
practice.zip	2021-08-05 오전 9:48	압축(ZIP) 파일	57KB

- IDE 툴에서 해당 폴더를 선택해 import한다.



# 0) SpringBoot 프로젝트 생성

- import된 프로젝트의 pom.xml를 통해 initializer로 추가한 dependency들을 확인할 수 있다.
- 정상적으로 import가 됐다면 JDK 설정과 maven update를 해 준다.



01

# 기본 기능

---

- 1) SpringApplication
  - 2) 외부 설정 & profile
-

# 1) SpringApplication

- Spring Application의 부트스트랩을 편리하게 제공해 주는 class.

※부트스트랩 (한 번의 동작으로 알아서 실행되는 일련의 과정)

- Spring Application의 여러 기능들을 커스터마이징해 사용할 땐, 아래와 같이, 인스턴스를 생성해 실행하는 것이 좋다.

```
@SpringBootApplication
public class PracticeApplication {

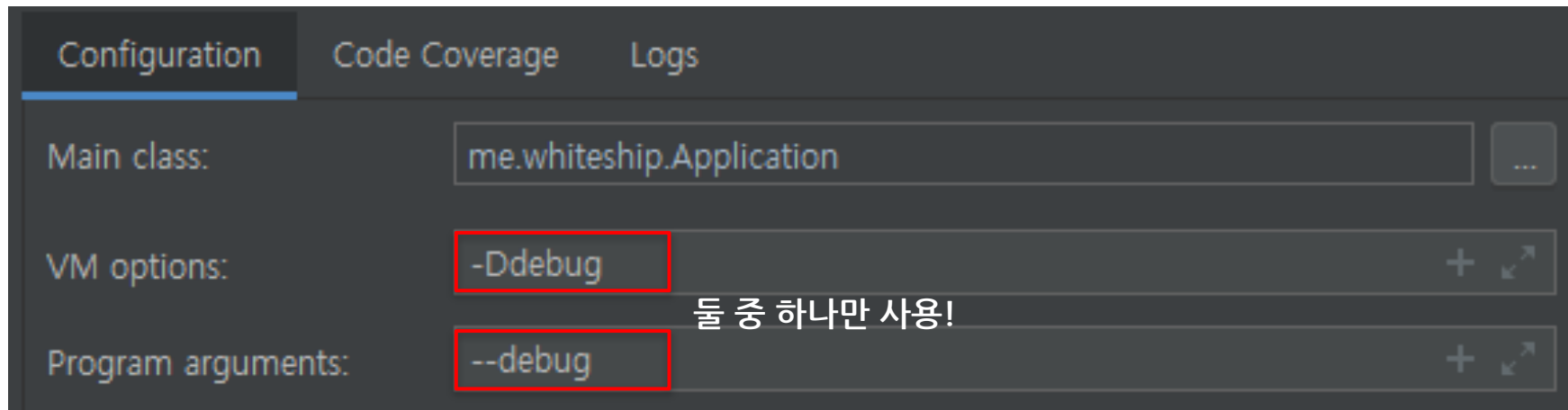
    public static void main(String[] args) {
        SpringApplication app = new SpringApplication(PracticeApplication.class);
        app.run();
    }
}
```

- Spring Application을 기본 설정으로 실행하면, 로그 레벨은 INFO로 설정된다.

```
INFO 5124 --- [main] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.context
INFO 5124 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
INFO 5124 --- [main] s.snowone.practice.PracticeApplication : Started PracticeApplication in 4.836 seconds (JVM running for 5.991)
```

# 1) SpringApplication

- Debug 모드로 실행시키고 싶다면, 실행 옵션(Edit Configurations...)의 VM options나 Program arguments로 조정 가능 하다.



※ VM options : JVM에 전달되는 인수로, JVM 실행 방법을 구성하고 조정하는 데 사용된다.

(-D 옵션: 자바 클래스에서 사용할 수 있는 시스템 속성. -Dkey=Value 쌍 형태로 System.getProperty(key)로 접근 가능.)

※ Program arguments : 애플리케이션에 전달되는 인수. main(String args[]) 의 args배열을 통해 접근 가능.

- Debug 모드로 실행하면, log는 자동설정(@EnableAutoConfiguration)의 적용/미적용의 이유들도 모두 출력해주어 추적 가능하다.



# 1) SpringApplication

- Spring Application의 시점에 따른 ApplicationEvent를 제공해주며, ApplicationListener를 implements해 이벤트 처리를 할 수 있다.

```
① @Component
public class AfterApplicationContextListener implements ApplicationListener<② ApplicationStartedEvent> {
    @Override ③
    public void onApplicationEvent(ApplicationStartedEvent applicationStartedEvent) {
        System.out.println("=====");
        System.out.println("PracticeApplication is started!");
        System.out.println("=====");
    }
}
```

- ① 리스너를 bean으로 등록해두면 컨테이너가 이벤트 시점에 찾아 자동 실행해준다.
- ② 대상 이벤트를 지정해주고, ③ onApplicationEvent 메서드를 override해 이벤트를 처리한다.

```
2021-08-05 11:08:16.964 INFO 11660 --- [main] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.context
2021-08-05 11:08:17.030 INFO 11660 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-08-05 11:08:17.038 INFO 11660 --- [main] s.snowone.practice.PracticeApplication : Started PracticeApplication in 4.831 seconds (JVM running for 5.326)
=====
PracticeApplication is started! // Application 실행 후 출력 (ApplicationStartedEvent 감지)
=====
```

## 1) SpringApplication

*But,*

- Context 생성 이전에 발생하는 Event Listener는 bean 등록으로 감지할 수 없다.

```
public class BeforeApplicationContextListener implements ApplicationListener<ApplicationStartingEvent> {
    @Override
    public void onApplicationEvent(ApplicationStartingEvent applicationStartingEvent) {
        System.out.println("=====");
        System.out.println("PracticeApplication is starting!");
        System.out.println("=====");
    }
}
```

애플리케이션 실행과 함께 발생(context 생성 전)

- Context 생성 이전에 발생하는 Event는 리스너가 bean으로 등록되기 전에 발생하기에, SpringApplication의 **addListeners**로 직접 등록해 줘야 한다.

```
@SpringBootApplication
public class PracticeApplication {

    public static void main(String[] args) {
        SpringApplication app = new SpringApplication(PracticeApplication.class);
        app.addListeners(new BeforeApplicationContextListener());
        app.run();
    }
}
```

```
=====
PracticeApplication is starting!
=====

      .   _--_       _         _--_ _
     /\\ / ___' _ -- _ _(_)_ -- _ \\ \\ \\
    ( ( ) \___ | '_ | '_ | | '_ V _' | \\ \\ \\
    \\/_ ___) | |_) | | | | | | (_| | | ) ) )
      ' |____| .__|_| | |_| | |_\\__, | / / / /
=====|_|=====|___/=/_/_/_/
:: Spring Boot ::                (v2.5.3)
```

## 2) 외부설정 & Profile

- Springboot는 `@ConfigurationProperties` 애노테이션을 통해 설정값들을 type safety하게 사용할 수 있도록 해준다.

```
seolhan.name = seolhan
seolhan.age = ${random.int[0,29]}
seolhan.fullName = ${seolhan.name} Kim
server.port = 9000
server.session.timeout=14400
logging.level.*=DEBUG
```

```
@Getter
@Setter
@ConfigurationProperties("seolhan")
public class SeolhanProperties {
    private String name;
    private int age;
    private String fullName;
}
```

```
@SpringBootApplication
@EnableConfigurationProperties(SeolhanProperties.class)
public class PracticeApplication {

    public static void main(String[] args) {
        val app = new SpringApplication(PracticeApplication.class);
        app.addListeners(new BeforeApplicationContextListener());
        app.run();
    }
}
```

- `@EnableConfigurationProperties` 애노테이션으로 properties 클래스를 등록해주면, 다른 빈들이 properties 빈을 주입받아 사용할 수 있다.

```
@Autowired
SeolhanProperties seolhanProperties;

@Override
public void onApplicationEvent(ApplicationStartedEvent applicationStartedEvent) {
    System.out.println("=====");
    System.out.println("★ My name is " + seolhanProperties.getFullName() + " ★");
    System.out.println("PracticeApplication is started!");
    System.out.println("=====");
}
```

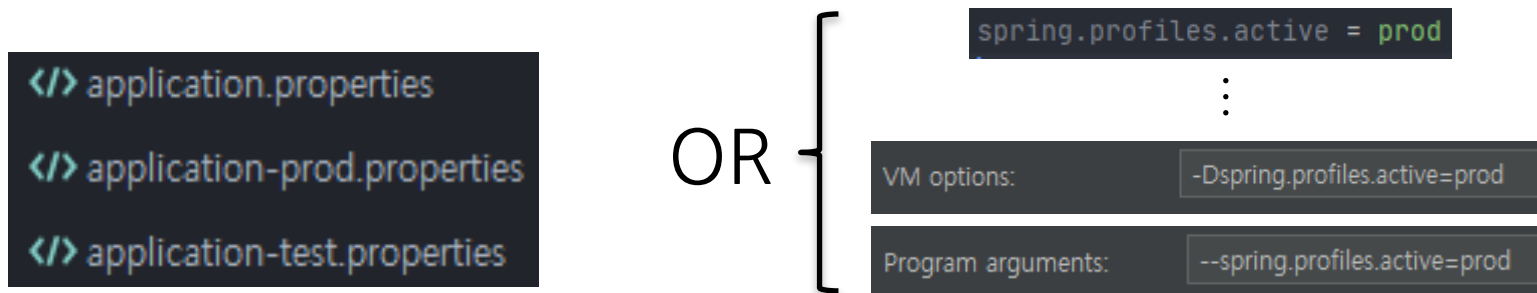
```
=====
★ My name is seolhan Kim ★
PracticeApplication is started!
=====
```

## 2) 외부설정 & Profile

- **@Profile** : 특정 Profile에서 대해 특정 빈/동작 설정을 다르게 하는 기능이다.
- 보통 @Configuration, @Component와 사용하며 특정 profile에만 특정 빈을 등록할 때 쓴다.

```
@Component
@Profile("prod")
public class AfterApplicationContextListener implements ApplicationListener<ApplicationStartedEvent> {
```

- application.properties 파일은 **application-{profile}.properties** 형식으로 파일들을 profile별로 설정하고 관리할 수 있는 기능을 제공한다.
- 이 때, application-{profile}.properties는 application.properties를 오버라이딩 한다.



- 활성 profile은 spring.profiles.active 값을 설정 하거나, @ActiveProfile을 통해 지정 할 수 있다.

02

# 기술 연동

- 
- 1) 스프링 웹 MVC
  - 2) 스프링 데이터
  - 3) 스프링 Security
-

# 1) 스프링 웹 MVC

- spring-boot-starter-web을 dependency 추가해주면 별도의 설정 없이 스프링 웹 MVC를 사용할 수 있다.

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

```
@RestController
@RequiredArgsConstructor
public class BaseController {

    private final BaseService baseService;

    @GetMapping("/fullname")
    public String getFullName() { return baseService.fullName; }

}
```

- 스프링 웹 MVC 관련 설정도 스프링 부트의 autoConfigure 모듈에 WebMvcAutoConfiguration class가 포함되어있기 때문에 자동 설정 된다.
- 스프링 부트가 제공하는 기본 설정을 확장해 사용하고 싶다면,  
@Configuration + implements WebMvcConfigurer로 추가 파일 생성이 가능하다.

# 1) 스프링 웹 MVC

## [1] HttpResponseMessage

- HTTP 요청 본문을 객체로 변경하거나, 객체를 HTTP 응답 본문으로 변경할 때 사용.
- @RequestBody
- @ResponseBody

```
@RestController
@RequiredArgsConstructor
public class BaseController {

    @PostMapping("/account/create")
    public @ResponseBody Account createAccount(@RequestBody Account account){
        return account;
    }
}
```

생략가능

- HttpResponseMessage는 요청/응답 데이터의 content-type이나 형식에 따라 적합한 종류의 컨버터로 동작한다.

※ 예 : JSON 본문 요청/응답일 경우 JsonMessageConverter , String 요청/응답일 경우 StringMessageConverter가 사용된다.

- 참고로 @RestController를 사용할 때는, @ResponseBody 생략이 가능하지만 @Controller를 사용할 때는 @ResponseBody를 써줘야 컨버터가 적용된다.

# 1) 스프링 웹 MVC

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

There was an unexpected error (type=Not Found, status=404).  
No message available

## [2] ExceptionHandler

- 스프링 부트 실행 시 BasicErrorController라는 기본 예외 핸들러가 등록되지만, @ExceptionHandler를 사용해 exception 처리를 커스터마이징 할 수 있다.

① 처리할 Exception을 extends한 Exception 클래스를 만들어 준다.

```
package springboot.snowone.practice.config;  
  
public class SeolhanException extends RuntimeException{  
}
```

② error 정보를 담을 커스텀 클래스를 만들어 준다.

```
@Getter  
@Setter  
public class SeolhanError {  
    private String message;  
    private String reason;  
}
```



# 1) 스프링 웹 MVC

- ③ 정의한 Exception이 발생할 때 쓰는 exception Handler를 @ExceptionHandler를 사용해 만들어주면, handler가 정상 동작함을 확인할 수 있다.

```
@RestController
@RequiredArgsConstructor
public class BaseController {

    @GetMapping("/seolhan")
    public String seolhan(){
        throw new SeolhanException();
    }

    @ExceptionHandler(SeolhanException.class)
    public @ResponseBody SeolhanError seolhanError(SeolhanException e){
        SeolhanError seolhanError = new SeolhanError();
        seolhanError.setMessage("seolhan.error");
        seolhanError.setReason("I don't know");
        return seolhanError;
    }
}
```

← → ↺ 🏠 ⓘ localhost:9000/seolhan

{"message":"seolhan.error","reason":"I don't know"}

# 1) 스프링 웹 MVC

- 전역으로 사용하고 싶을 땐 **@ControllerAdvice** 를 붙여 클래스를 따로 만들고, 그 안에 **@ExceptionHandler**를 정의하면 여러 컨트롤러에서 사용 가능하다.

```
package.springboot.snowone.practice.config;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(SeolhanException.class)
    public SeolhanError seolhanError(SeolhanException e){
        SeolhanError seolhanError = new SeolhanError();
        seolhanError.setMessage("seolhan.error");
        seolhanError.setReason("I don't know");
        return seolhanError;
    }
}
```

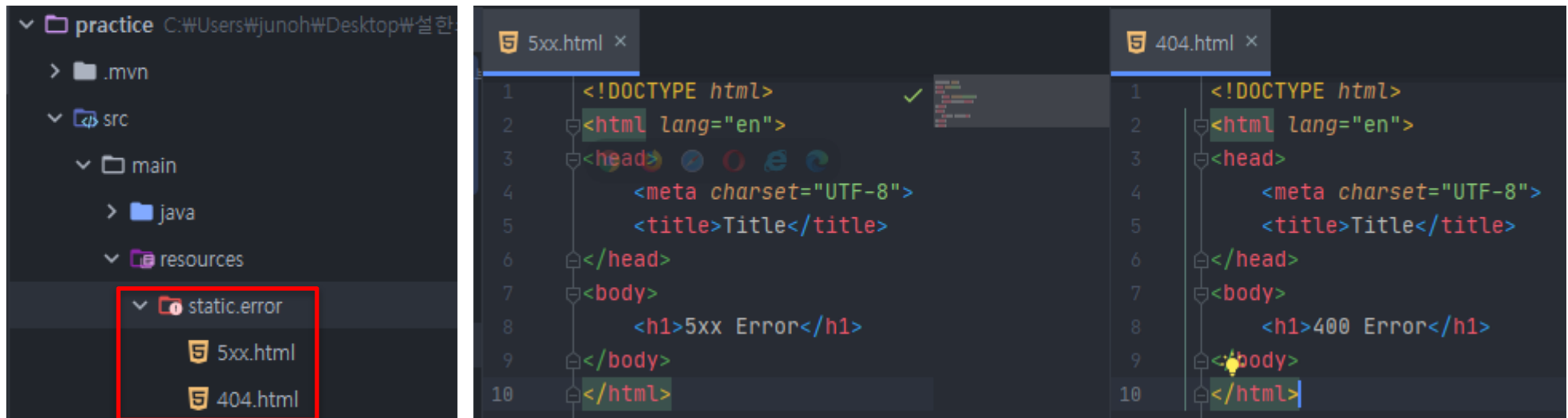
← → ↺ 🏠 ⓘ localhost:9000/fullname

{"message":"seolhan.error","reason":"I don't know"}

# 1) 스프링 웹 MVC

(+) 에러 페이지 커스터마이징

- src > main > resources > static 또는 templates에 error라는 디렉토리를 만들고, '상태코드 값.html' 파일을 만들어 에러 status 코드에 따라 다른 웹 페이지를 보여줄 수 있다.
- 상태 코드 값은 완전히 같게 하거나, 5xx 처럼 앞자리만 표시해줘도 된다.



The screenshot shows an IDE with a project named 'practice'. The file explorer on the left shows the directory structure: .mvn, src, main, java, resources, and a newly created 'static.error' directory containing '5xx.html' and '404.html'. The main editor displays the content of these files. The '5xx.html' file contains the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <h1>5xx Error</h1>
9 </body>
10 </html>
```

The '404.html' file contains the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <h1>400 Error</h1>
9 </body>
10 </html>
```

## 404 Error

# 1) 스프링 웹 MVC

## [3] CORS (Cross-Origin Resource Sharing)

- 다른 오리진끼리 리소스를 공유할 수 있는 방법을 제공하는 표준
- Origin = URI Schema + HostName + port
- @CrossOrigin : CORS를 적용해주는 애노테이션. 메소드/컨트롤러/웹 설정 파일에 적용.

```
@RestController
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:80")
public class BaseController {

    private final BaseService baseService;

    @GetMapping("/fullname")
    public String getFullName(){
        return baseService.getFullName();
    }
}
```

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry){
        registry.addMapping(pathPattern: "**")
            .allowedOrigins("*");
    }
}
```

- CORS 설정을 안하면 SOP(Single-Origin Policy)가 기본 적용되 오리진이 다른 클라이언트의 API 호출이 불가능하다.

## 2) 스프링 데이터

- 스프링 데이터는 SQL DB/ NoSQL 을 지원한다. 그 중 SQL DB 영역인 인메모리 데이터베이스 H2와 스프링 데이터 JPA를 소개한다.

SQL DB
<ul style="list-style-type: none"><li>- 인메모리 데이터베이스</li><li>- 스프링 데이터 JPA</li><li>- 데이터베이스 초기화</li><li>- 데이터베이스 마이그레이션 툴</li><li>- DataSource 설정</li><li>- JDBC 사용</li><li>...</li></ul>

※ 인메모리 데이터베이스 : 디스크가 아닌 메인 메모리에 모든 데이터를 보유하고 있는 데이터베이스.

### [1] H2 데이터 베이스

- 자바 기반 오픈 소스 RDBMS
- 스프링 부트는 내장 H2 데이터베이스를 지원해준다.

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

## 2) 스프링 데이터

- dependency를 추가해주면, 내장 설정 정보로 H2 데이터 베이스를 생성해주고 관련 설정 값들을 지정할 수 있게 해준다.

```
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console
```

localhost:9000/h2-console,

- 내장 DataSource 설정 정보도 주입받아 확인 가능하며, 해당 정보로 H2 콘솔에 접속 할 수 있다.

```
package.springboot.snowone.practice;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.ApplicationArguments;  
import org.springframework.boot.ApplicationRunner;  
import org.springframework.stereotype.Component;  
  
import javax.sql.DataSource;  
import java.sql.Connection;  
  
@Component  
public class PracticeApplicationRunner implements ApplicationRunner {  
  
    @Autowired  
    DataSource dataSource;  
  
    @Override  
    public void run(ApplicationArguments args) throws Exception {  
  
        val connection = dataSource.getConnection();  
        System.out.println("=====");  
        System.out.println("url : " + connection.getMetaData().url);  
        System.out.println("userName : " + connection.getMetaData().userName);  
        System.out.println("=====");  
  
        connection.close();  
    }  
}
```

```
=====  
url : jdbc:h2:mem:04ad6adb-4514-44a6-bb09-ac1312a77658  
userName : SA  
=====
```

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) [Save] [Remove]

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:04ad6adb-4514-44a6-bb09-ac1312a77658

User Name: SA

Password:

[Connect] [Test Connection]

## 2) 스프링 데이터

### [2] 스프링 데이터 JPA

- JPA 표준 스펙을 아주 쉽게 사용할 수 있게 스프링 데이터로 추상화 시켜놓은 것
- 의존성을 추가해 자동 설정 받을 수 있다.

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

- @Entity 등 JPA 애노테이션을 사용 할 수 있고, JpaRepository를 extends해 추상화된 기능도 사용할 수 있다. JpaRepository<[entity타입], [id 타입]>을 extends해준다.

```
@Getter
@Setter
@Entity
public class Account {

    @Id
    @GeneratedValue
    private Long id;

    private String username;

    private String password;
}
```

```
public interface AccountRepository extends JpaRepository<Account, Long> {

    Optional<Account> findByUsername(String username);
}
```

※ 스프링 데이터 JPA 제공 키워드 정리: <https://happygrammer.tistory.com/158>

### 3) 스프링 Security (기본 설정)

- 스프링 Security
  - : 보안 관련 인증(Authentication), 권한부여(Authorization) 기능 제공.
- 스프링 Security는 default로 basic Authentication이 적용되어,  
권한이 필요한 api에 접근하면 default 로그인 페이지로 redirect 한다.

```
@Test
public void hello() throws Exception {
    mockMvc.perform(get(urlTemplate: "/hello")
        .accept(MediaType.TEXT_HTML))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(view().name(expectedViewName: "hello"));
}
```

로그인 폼 인증 요구

```
MockHttpServletRequest:
    Status = 302
    Error message = null
    Headers = [X-Content-Type-Options:"nosniff", X-XSS-Protection:"1; mode=block"]
    Content type = null
    Body =
    Forwarded URL = null
    Redirected URL = http://localhost/login
    Cookies = []
```

localhost:9000/login

### Please sign in

Username

Password

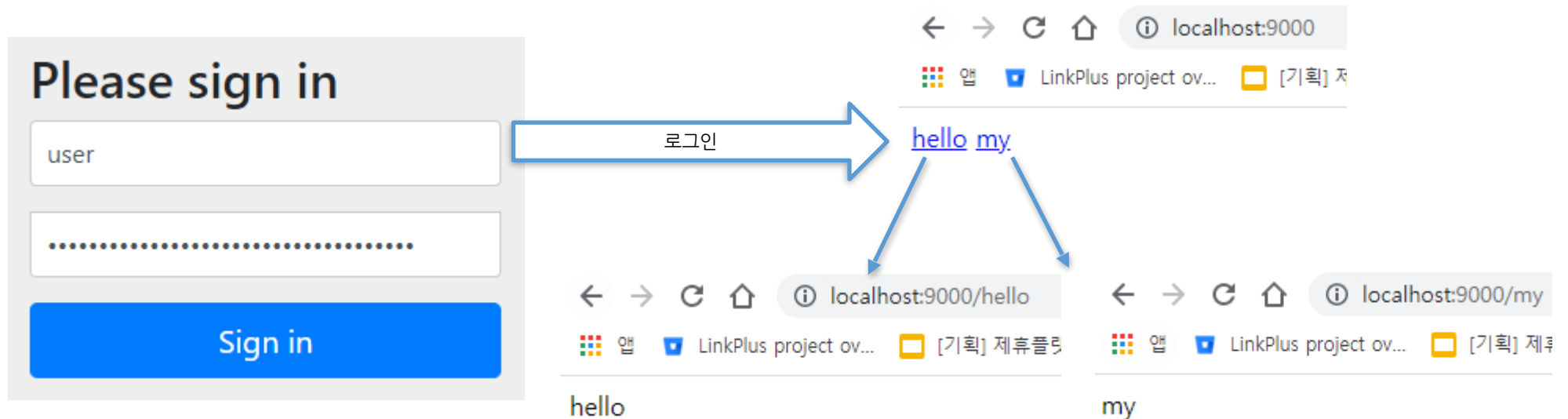
Sign in



### 3) 스프링 Security (기본 설정)

- Redirect 된 login페이지의 default 계정은 id 는 user이고, password는 애플리케이션 구동시마다 random 값으로 console창에 출력된다. ( security에 기본으로 설정된 기능 )

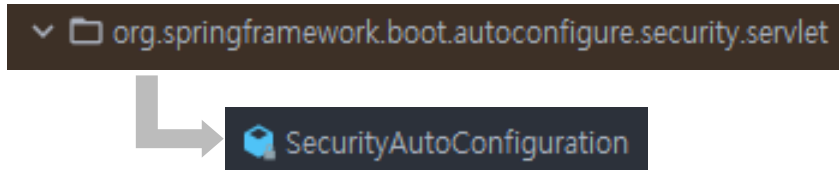
```
Using generated security password: 8a01744b-c6bf-446d-9e26-dd0a5e6d9ef7
```



- 로그인으로 인증을 마치면 페이지 접근이 가능해진다.

### 3) 스프링 Security (기본 설정)

- 스프링부트에서 security를 추가하면 SecurityAutoConfiguration이 자동 설정된다.



- SecurityAutoConfiguration에는 default 이벤트 퍼블리셔가 등록되어 있는데, 해당 퍼블리셔가 비밀번호 불일치/ 계정 만료/ 없는 아이디 사용 등을 체크 해준다.

```
this.addMapping(BadCredentialsException.class.getName(), AuthenticationFailureBadCredentialsEvent.class);
this.addMapping(UsernameNotFoundException.class.getName(), AuthenticationFailureBadCredentialsEvent.class);
this.addMapping(AccountExpiredException.class.getName(), AuthenticationFailureExpiredEvent.class);
this.addMapping(ProviderNotFoundException.class.getName(), AuthenticationFailureProviderNotFoundEvent.class);
this.addMapping(DisabledException.class.getName(), AuthenticationFailureDisabledEvent.class);
this.addMapping(LockedException.class.getName(), AuthenticationFailureLockedEvent.class);
this.addMapping(AuthenticationServiceException.class.getName(), AuthenticationFailureServiceExceptionEvent.class);
this.addMapping(CredentialsExpiredException.class.getName(), AuthenticationFailureCredentialsExpiredEvent.class);
```

- 이 때, 이벤트 핸들러를 등록해 이벤트 처리를 커스터마이징 해 줄 수 있다.
- 또한, 이벤트 퍼블리셔(DefaultAuthenticationEventPublisher)를 빈으로 등록해 이벤트 퍼블리셔 자체를 커스터마이징 할 수도 있다.

### 3) 스프링 Security (기본 설정)

- 기본 권한 설정은 WebSecurityConfigurerAdapter에 모든 요청은 권한을 체크하며, formLogin으로 리다이렉트 하도록 정의되어 있다.

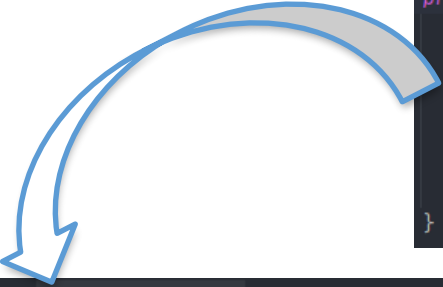
```
protected void configure(HttpSecurity http) throws Exception {  
    this.logger.debug("Using default configure(HttpSecurity). If subclass  
    http.authorizeRequests((requests) -> {  
        ((AuthorizedUrl)requests.anyRequest()).authenticated();  
    });  
    http.formLogin();  
    http.httpBasic();  
}
```

- UserDetailsServiceConfiguration
  - [1] 스프링부트 애플리케이션 시작 시 default user를 생성해준다
  - [2] AuthenticationManager / AuthenticationProvider / UserDetailsService 가 모두 빈으로 등록되지 않은 경우 적용되는 configuration이다.
  - [3] 스프링 security를 사용하는 경우, UserDetailsService를 필수로 구현하게 되기에 해당 configuration을 사용하게 되는 경우는 드물다.

### 3) 스프링 Security (커스터 마이징)

[1] WebSecurityConfigurerAdapter : URL 별 권한 설정.

```
protected void configure(HttpSecurity http) throws Exception {  
    this.logger.debug(o: "Using default configure(HttpSecurity). If subclass  
    http.authorizeRequests((requests) -> {  
        ((AuthorizedUrl)requests.anyRequest()).authenticated();  
    });  
    http.formLogin();  
    http.httpBasic();  
}
```



```
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
  
@Configuration  
// 해당 빈이 등록되면 기본 WebSecurityConfigurerAdapter 는 적용X  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http.authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry  
            .antMatchers( ...antPatterns: "/", "/hello").permitAll() // 루트와 /hello는 모두 접근 가능  
            .anyRequest().authenticated() // 나머지 모든 요청은 인증 필요  
            .and() HttpSecurity  
            .formLogin() FormLoginConfigurer<HttpSecurity>  
            .and() HttpSecurity  
            .httpBasic();  
    }  
}
```

### 3) 스프링 Security (커스터 마이징)

[2] UserDetailsService: 로그인 입력 값으로 DB에서 계정 정보를 가져오는 역할.

```
@Service
public class AccountService implements UserDetailsService {


    @Autowired
    private AccountRepository accountRepository;

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

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // 로그인 form에서 입력된 username이 파라미터로 들어옴
        Optional<Account> byUsername = accountRepository.findByUsername(username);
        Account account = byUsername.orElseThrow(() -> new UsernameNotFoundException(username));
        return new User(account.getUsername(), account.getPassword(), authorities());
    }

    // GrantedAuthority : spring security가 제공하는 권한 class
    private Collection<? extends GrantedAuthority> authorities() {
        return Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"));
    }
}
```

이후, AuthenticationProvider가  
여기서 return된 DB계정 정보와  
화면에서 입력된 로그인 정보를  
비교해 인증 절차 진행



### 3) 스프링 Security (커스터 마이징)

#### [3] PasswordEncoder 설정 및 사용

- PasswordEncoder는 비밀번호를 암호화하는 인터페이스 객체이다. 그래서 구현체를 대입해주고 이를 빈으로 등록하는 과정이 필요하다.

```
@Bean
public PasswordEncoder passwordEncoder(){
    return PasswordEncoderFactories.createDelegatingPasswordEncoder();
}
```

※ createDelegatingPasswordEncoder

- 여러 인코드 방식을 상황에 맞게 골라 쓸 수 있도록 지원. Default는 bcrypt.
- 암호화 시 인코드 방식을 명시해주어, 인코드 방식 변경 시 유연한 대응 가능.

- 서비스에서는 빈으로 등록된 PasswordEncoder를 주입받아 패스워드를 암호화 할 수 있다.

```
@Autowired
private AccountRepository accountRepository;

@Autowired
private PasswordEncoder passwordEncoder;

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



Thank you