



Application Framework Development Guide

II. SKIAF 따라하기

Index of Contents

II.1 개요	3
II.2 따라하기 준비	4
II.2.1 설치	4
II.2.2 template 가져오기	4
II.2.3 template 구성테스트	9
II.3 따라하기 절차	12
II.4 구현(로직개발)	13
II.4.1 Controller 구현	13
II.4.2 JPA 작업하기	17
II.4.3 SQL Mapper 사용하기	24
II.4.4 Thymeleaf 를 이용하기	29
II.4.5 프로그램 등록 및 메뉴 등록하기	35
Appendix. 프로젝트 수행시 고려사항	39

II. SKIAF 따라하기

II.1 개요

[목적]

본 문서는 Spring Boot기반의 SKIAF Framework 을 활용하여 어플리케이션 개발 시, 유의할 사항과 가이드라인을 제공한다.

또한 개발 방법을 가이드 하여 Framework 구성과 개발 절차를 보다 빠르게 익힐 수 있도록 하고자 한다.

체험하기를 수행한 개발자는 SKIAF 프레임워크로 개발할 기본준비가 완료 되었다고 판단 할 수 있다.

[문서의 범위]

본 문서에서는 다음의 사항을 기술한다.

- 개발환경
- 서비스개발
- 단위테스트
- 기본화면 개발

[사전 준비사항]

"I. SKIAF 개발환경 가이드" 문서를 필독하여 기본환경 세팅을 한다.

[선수 학습 대상]

skiaf를 커스터마이징하기 위해서 먼저 skiaf의 구조와 기능을 이해해야 하며, 아래와 같이 선수 학습이 필요하다.

선수 학습 대상	선수 학습 목표	필독 여부
I. SKIAF 개발환경 가이드	개발 환경	*
II. SKIAF 체험하기	skiaf 동작 경험	*
III. SKIAF Back-end 개발표준	아키텍처, 디렉토리 구조 이해	Mandatory
IV. SKIAF Front-end 개발표준	Thymeleaf 이해	Mandatory
V. SKIAF 개발가이드 - 시스템공통	보안, 메시지, Exception 처리 등에 대한 이해	Mandatory
VI. SKIAF 개발가이드 - 업무공통	메뉴, 권한 등의 공통기능 이해	Mandatory

* 에 해당하는 부분은 Spring 개발환경에 익숙한 개발자라면, 건너뛰어도 무방하다.

또한, skiaf는 Spring MVC와 Alopex를 기반으로 개발되었으므로 다음 기술을 필요로 한다.

구분	관련 기술

Back-End 기술	Java 관련 Spring Boot / MyBatis / JPA / Rest API
Front-End 기술	Thymeleaf
	Javascript / CSS / HTML / Ajax / jQuery
	Alopex UI / Alopex Grid

그 외, 3rd-party 라이브러리 사용경험이 있는 경우 커스터마이징에 유리하다 (ex: Spring Security, Lucy, LogBack 등)

[실습환경]

프로젝트 따라하기를 위한 개발자 PC 환경은 다음과 같이 준비 한다.

- OpenJDK 1.8, STS 3.9.4 을 사용한다.
- DBMS로는 H2 Embedded DB 를 활용한다.
- 형상/빌드/배포(SVN, JENKINS)는 본 과정에서는 제외된다

II.2 따라하기 준비

II.2.1 설치

[설치]

자신의 환경에 맞는 설치 파일을 관련 담당자에게 문의하거나 사이트에서 다운로드 받는다.

필요한 파일은

- JDK : 자세한 문서는 "[I. SKIAF 개발환경 가이드 > I.3 JDK 설치](#)" 참고
- STS : 자세한 문서는 "[I. SKIAF 개발환경 가이드 > I.4 Java IDE 설치 및 plugin 설정 > I.4.1 STS \(Spring Tool Suite\)](#)" 참고
- Lombok : 자세한 문서는 "[I. SKIAF 개발환경 가이드 > I.4 Java IDE 설치 및 plugin 설정 > I.4.2 Lombok](#)" 참고
- 기타플러그인 : 자세한 문서는 "[I. SKIAF 개발환경 가이드 > I.4 Java IDE 설치 및 plugin 설정 > I.4.6 플러그인 설정](#)" 참고
- Nexus설정 : 자세한 문서는 "[I. SKIAF 개발환경 가이드 > I.4 Java IDE 설치 및 plugin 설정 > I.5. Nexus 설정](#)" 참고

이다.

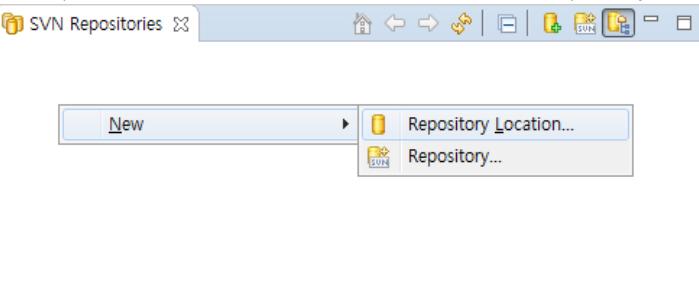
※ 설치가이드

설치관련 자세한 문서는 : "[I. SKIAF 개발환경 가이드](#)" 를 참조한다.

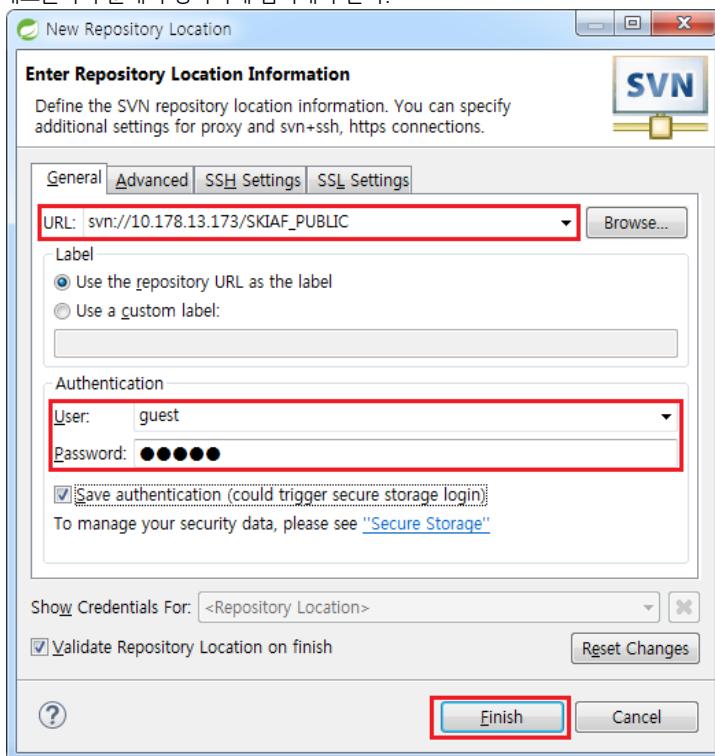
II.2.2 template 가져오기

[SVN에서 Export 하기]

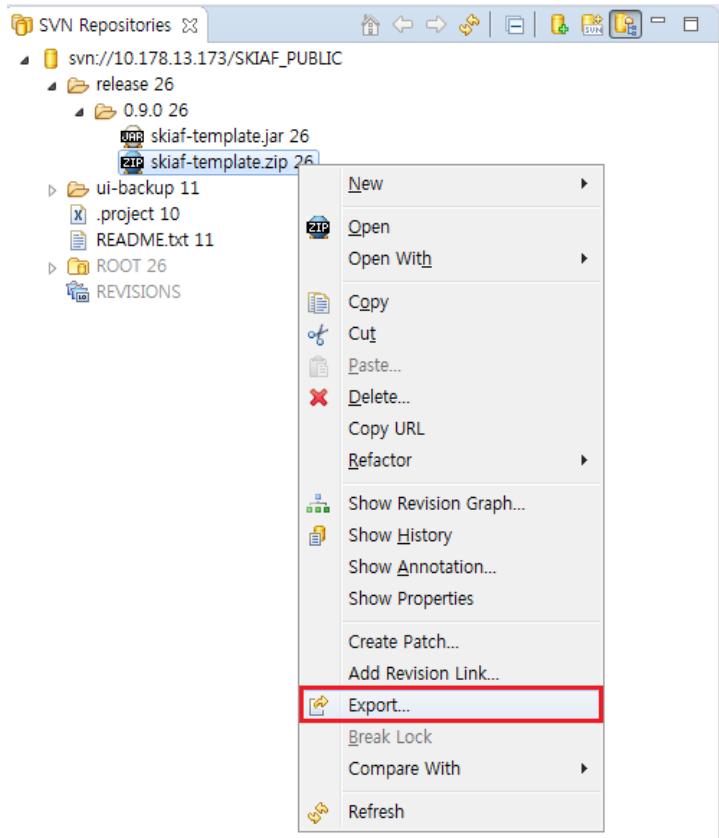
- SVN Repositories에서 빈공간에 오른쪽 마우스 클릭 - New - Repository Location 클릭



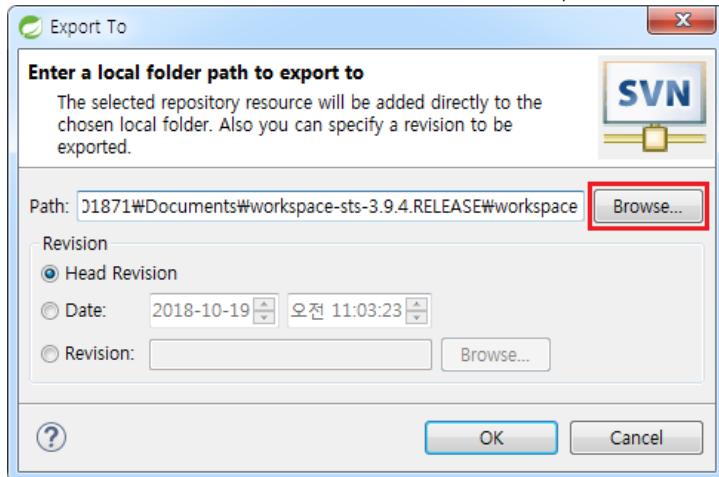
- SVN URL과 계정명 비밀번호를 입력하고 Finish를 클릭 (계정명 : guest, 비밀번호 : guest)
- SVN URL : svn://10.178.13.173/SKIAF_PUBLIC
- User : guest
- Password : guest
- 대소문자 구분해서 정확하게 입력해야 한다.



- SVN URL - release - 버전 - skiaf-template.zip 오른쪽 마우스 클릭 - Export 클릭



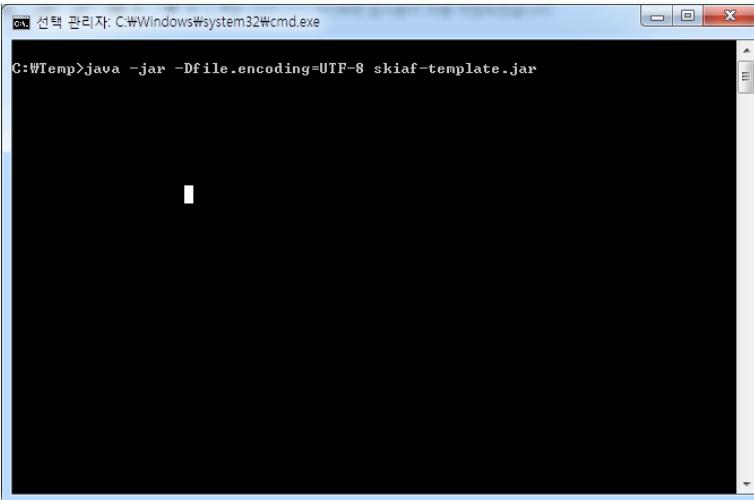
- Browse를 클릭해 원하는 폴더를 설정하고 OK버튼을 누르면 zip파일이 저장된다.



사전실행 해보기

테스트 용도로 실행하려면 아래와 같이 하면 실행해 볼 수 있다. 미리보기를 함으로써 필요한 기능이 다 있는지 체크하는 용도로 사용 가능하다.

- SVN에서 - release - 버전 - skiaf-template.jar 오른쪽 마우스 클릭 - Export 받아서
- windows cmd 창을 띄운 후 "java -jar -Dfile.encoding=UTF-8 skiaf-template.jar"를 실행하면 된다.

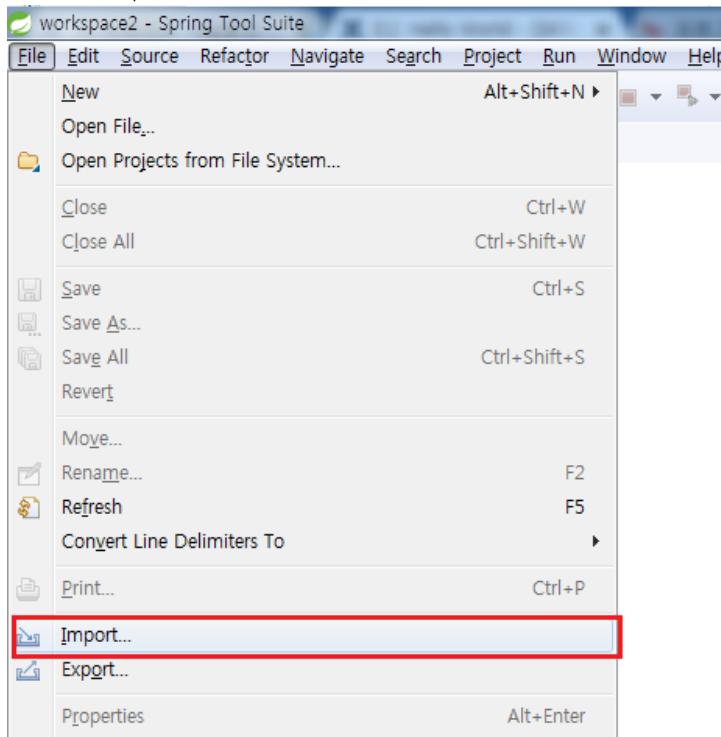


```
C:\선택 관리자: C:\Windows\system32\cmd.exe
C:\Temp>java -jar -Dfile.encoding=UTF-8 skiaf-template.jar
```

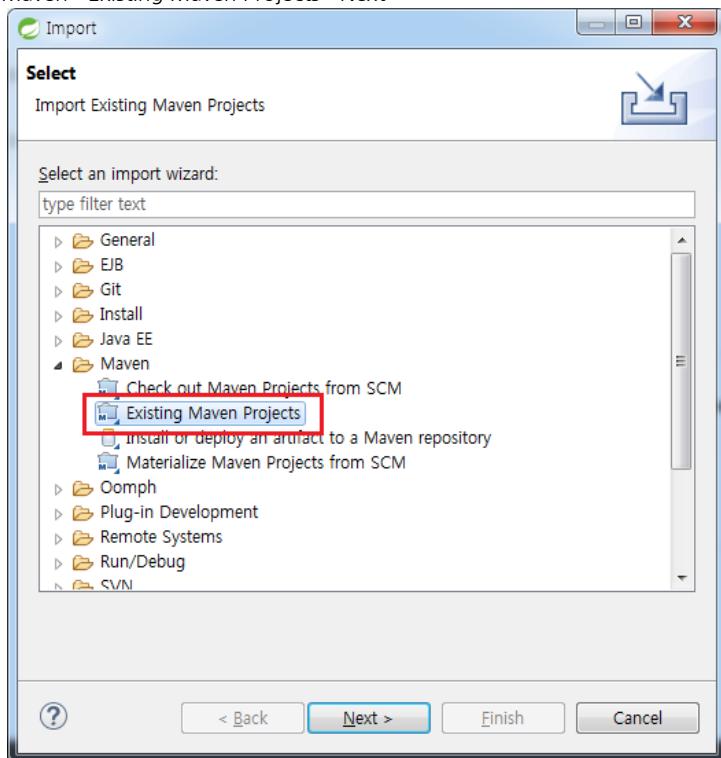
- 정상 기동 후 브라우저에서 "<http://localhost:8080>"을 넣으면 실행이 된다.

[STS에 Import 하기]

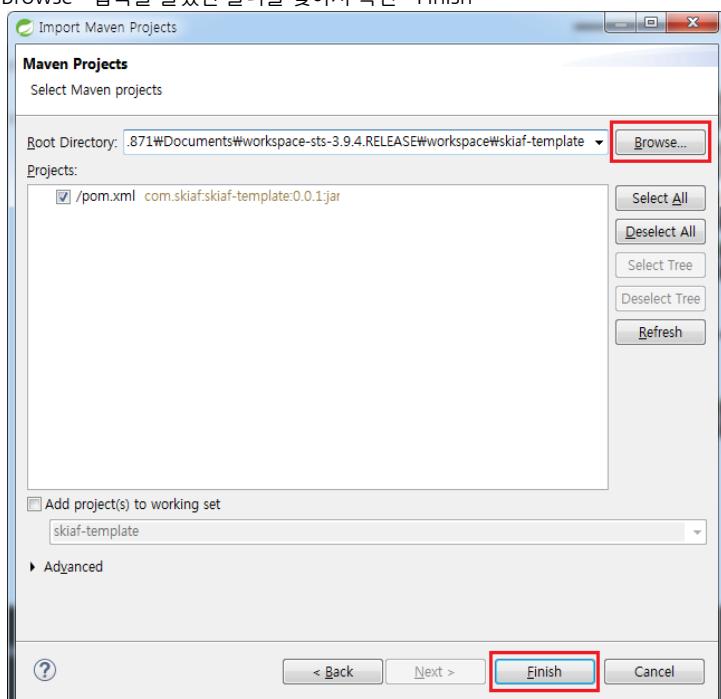
- Export 받은 skiaf-template.zip 파일을 사용할 workspace에 압축풀기
- STS - File - Import 클릭



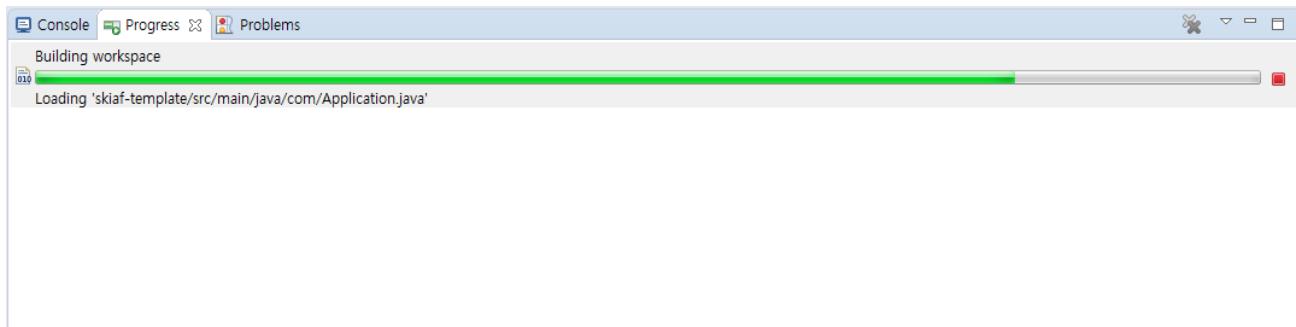
- Maven - Existing Maven Projects - Next



- Browse - 압축을 풀었던 폴더를 찾아서 확인 - Finish



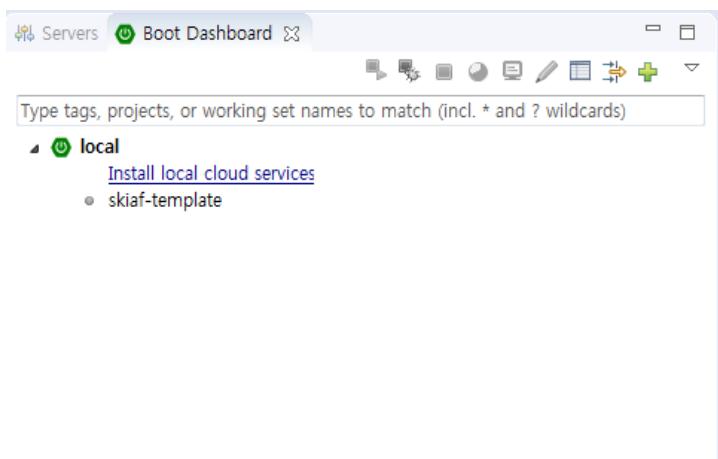
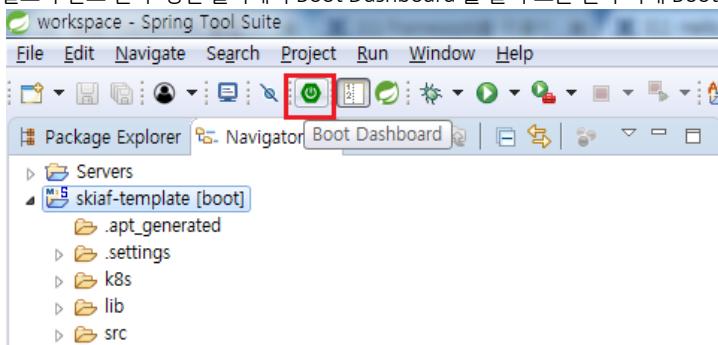
- Import한 프로젝트가 자동으로 빌드될때 까지 기다린다. 빌드가 되는 과정은 Progress 탭에서 확인 가능하다.



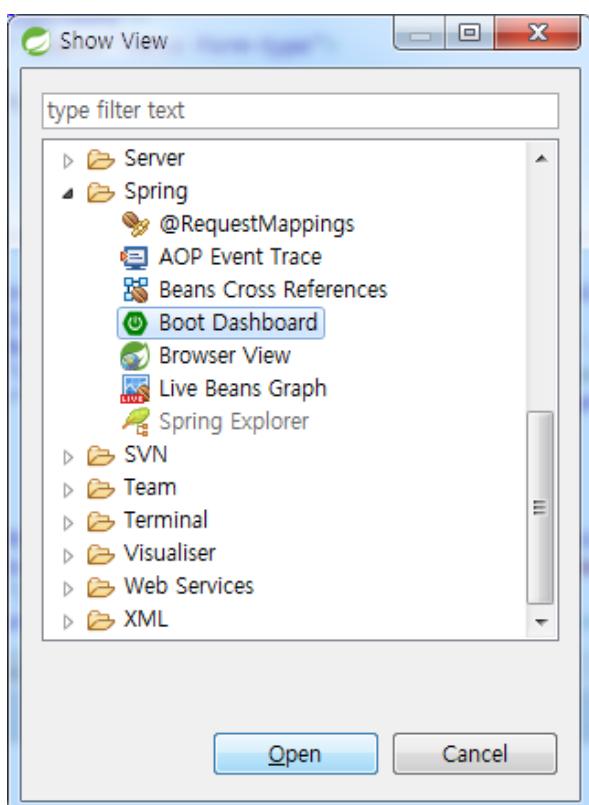
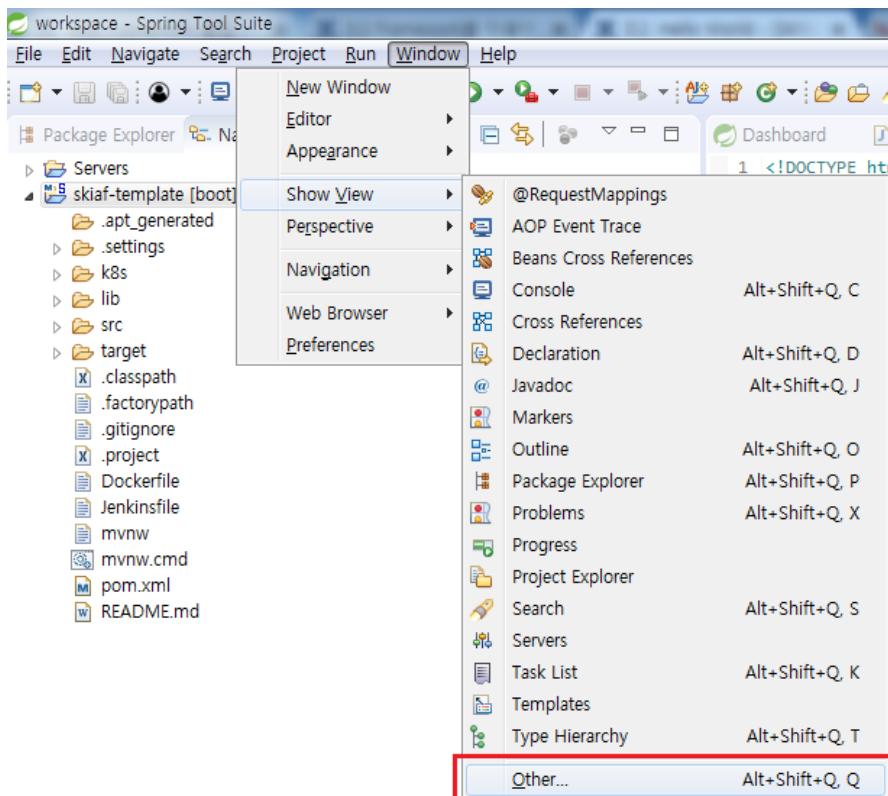
II.2.3 template 구성테스트

[template 구동하기]

- 빌드가 완료 된 후 상단 툴바에서 Boot Dashboard 를 클릭 또는 왼쪽 아래 Boot Dashboard 탭을 확인한다.

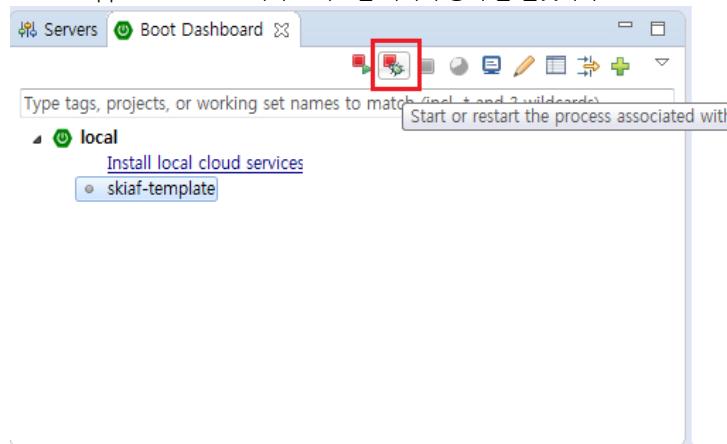


- Boot Dashboard가 없을 경우
Window - Show View - Other 클릭 Spring - Boot Dashboard - Open 하면 창을 볼 수 있다.



- 서버를 구동하려면 Boot Dashboard에서 skiaf-template을 선택하고 Start debug mode를 클릭한다. Console 창에서 구동여부를 확인할 수 있다.

Started Application in ... 이라고 나오면 서버 구동이 잘 된것이다.



```

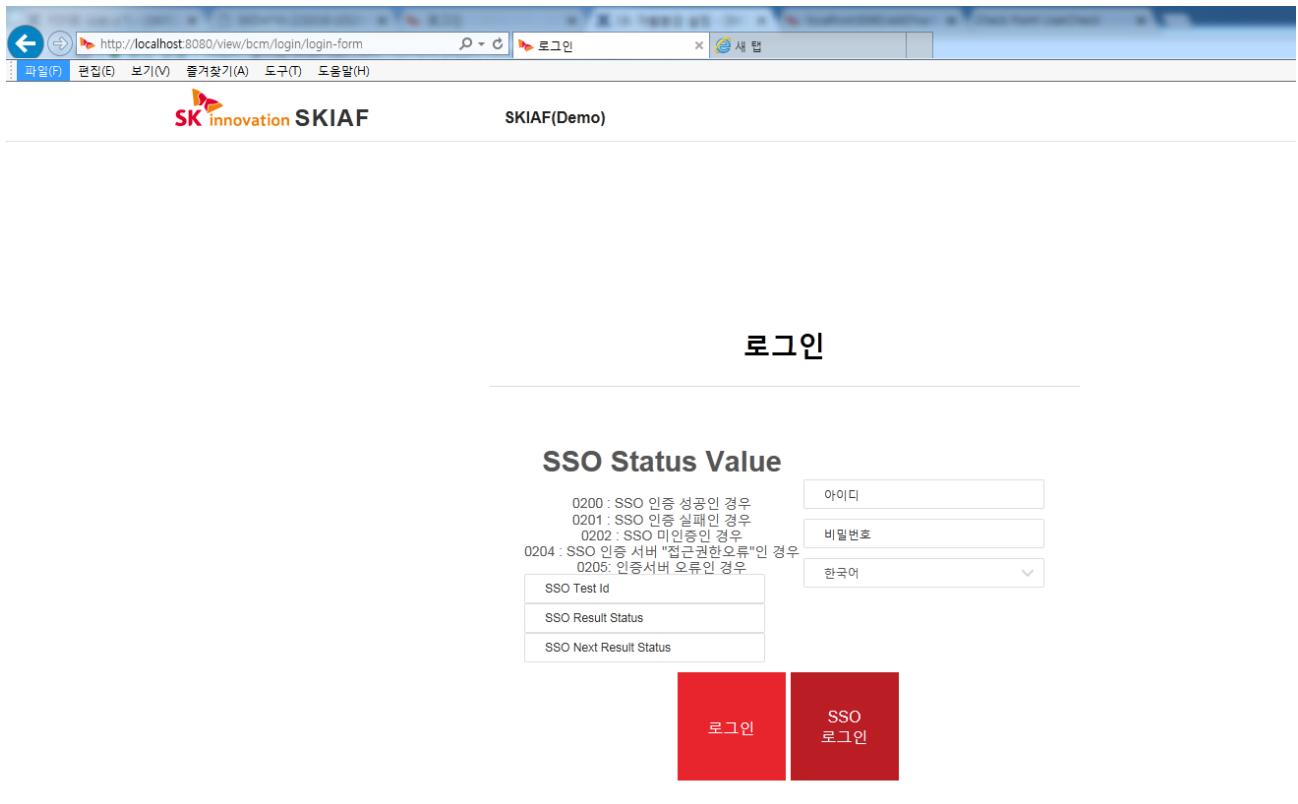
Console Progress Problems
skiaf-template - Application [Spring Boot App] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (2018. 10. 10. 오후 5:15:46)
2018-10-10 17:16:26.219 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: duplicateCheckU
2018-10-10 17:16:26.222 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: findOneUsingGET
2018-10-10 17:16:26.225 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: findOneDetailUs
2018-10-10 17:16:26.233 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: findQueryByKeyw
2018-10-10 17:16:26.247 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: updateUsingPUT_
2018-10-10 17:16:26.254 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: createUsingPOST
2018-10-10 17:16:26.258 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: duplicateCheckU
2018-10-10 17:16:26.260 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: findOneUsingGET
2018-10-10 17:16:26.265 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: findQueryByKeyw
2018-10-10 17:16:26.270 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: updateUsingPUT_
2018-10-10 17:16:26.273 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: deleteUserByUs
2018-10-10 17:16:26.277 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: saveUsersByUser
2018-10-10 17:16:26.283 INFO 8972 --- [main] .d.s.w.r.o.CachingOperationNameGenerator : Generating unique operation named: activeRoleListU
2018-10-10 17:16:26.397 INFO 8972 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2018-10-10 17:16:26.404 INFO 8972 --- [main] com.skiaf.demo.cache.CacheManagerCheck : 

=====
Using cache manager: org.springframework.cache.ehcache.EhCacheCacheManager
=====

2018-10-10 17:16:26.442 INFO 8972 --- [main] com.Application : Started Application in 36.725 seconds (JVM running

```

- 브라우저에서 확인하려면
브라우저 주소창에
`http://localhost:8080/`을 입력하면 된다.
- 서버가 정상적으로 작동된다면 다음과 같은 화면을 확인할 수 있다.



II.3 따라하기 절차

[기본지식]

SKIAF는 기본 Spring MVC 패턴을 기반으로 구성되어 있다. Spring DispatcherServlet을 통해 요청된 거래에 대해 URL 패턴을 분석하여 지정된 Controller에 정의된 MVC annotation의 Request Mapping과 일치하는 거래를 호출해주며,

거래가 완료되면 Request 시의 데이터 포맷과 일치하도록 변환한 후 Response를 하게 된다.

- MVC 패턴
 - Controller
 - 클라이언트의 요청한 거래에 Mapping이 되는 클래스로서, 보편적으로 한 화면당 한 개의 Controller가 발생하며, Controller 내부에는 여러 개의 메소드들이 있는데 통상적으로 화면에 존재하는 이벤트들과 1:1로 Mapping된다.
 - Service
 - 화면에서 요청된 거래에 대한 Business Logic을 처리하는 영역으로서, Controller에서 여러 개의 Service 메소드가 호출될 수 있다.
 - Repository(DAO)
 - DB를 Access하기 위한 Object로서 DB로부터 데이터를 취득하여 전달하는 영역을 말한다.
- TX 처리단위
 - @Transactional으로 클라이언트로부터 요청된 거래에 대한 Commit/Rollback 단위는 Service이다.
- MVC 패턴으로 개발 시 주의할 점
 - MVC 패턴의 기본 Flow는 Controller → Service → Repository(DAO)이다. 해당 Flow에 대한 역행이 존재할 경우에는 무한반복이

- 발생할 수 있기 때문에 절대로 구성하지 않도록 한다.
- 하나의 Controller 가 관련된 여러 개의 service 를 호출하는 구조로 개발되어야 한다. 그렇게 되기 위해서는 업무에 대한 세밀한 분석이 필요하다.

[개발순서]

상세 개발순서는 다음과 같다.

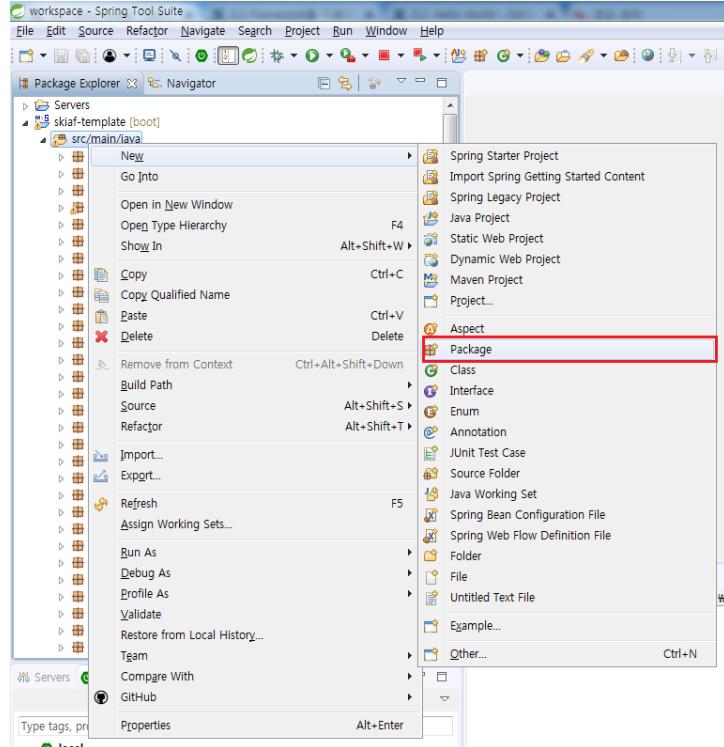
- 클래스 생성(DAO, Service, Controller)
- SQL 생성
- Controller 단위테스트
- 화면 생성
- 브라우저 테스트

II.4 구현(로직개발)

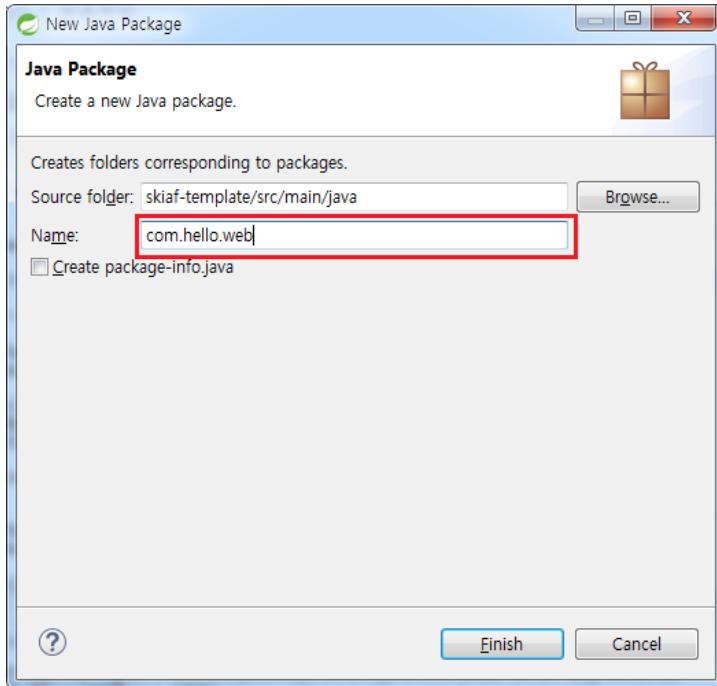
II.4.1 Controller 구현

[간단한 Hello Controller 만들기]

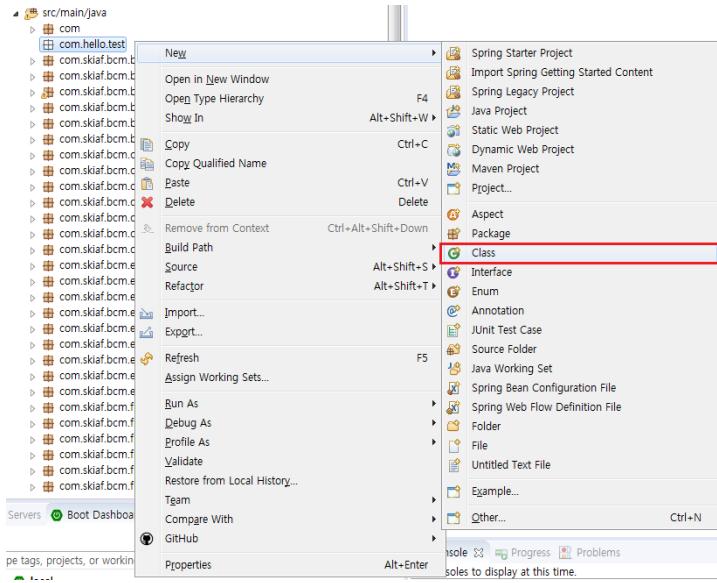
- 템플릿 프로젝트에서 src/main/java 오른쪽 마우스 클릭
New - Package를 클릭



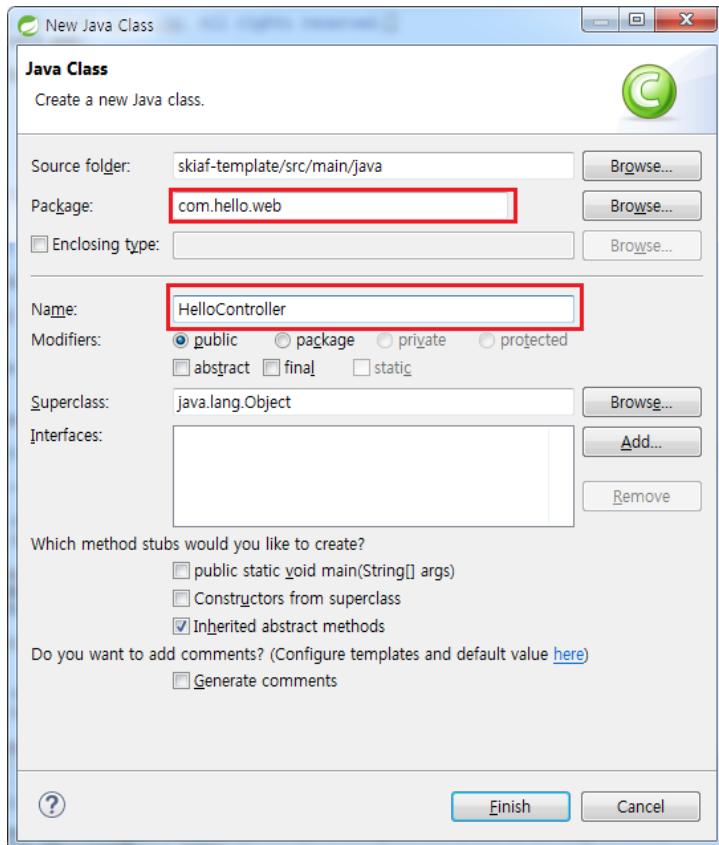
- Name에 com.hello.web 라고 입력 후 finish를 누른다. 패키지 이름은 소문자로 입력한다.



- 만들어진 패키지를 선택 - 우클릭
New - Class 클릭



- Package 명을 확인하고 Name에 HelloController 입력 후 Finish를 클릭



- 소스에 다음과 같이 입력한다.

```
HelloController

package com.hello.web;

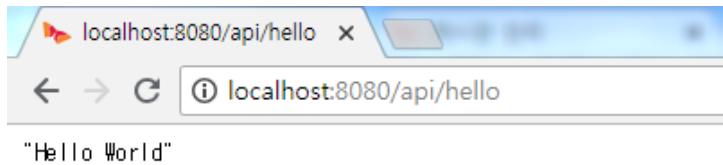
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import io.swagger.annotations.ApiOperation;

@RestController // RestController 어노테이션
public class HelloController {

    @ApiOperation(value = "헬로우 월드")
    @GetMapping("/api/hello") // Get Method 방식, 주소 입력
    public String hello() {
        return "Hello World"; // 출력값
    }
}
```

- 서버를 구동해 결과 값을 확인한다. 출력값은 Json 형태로 출력된다. 크롬에서는 화면에 출력되고 익스플로러에서는 Json파일이 다운로드된다.
- 주소 : http://localhost:8080/api/hello



- REST API의 모든 Method를 테스트 하기 위해선 Swagger나 PostMan과 같은 툴을 이용해 확인을 할 수 있다.
- SKIAF에서는 Swagger를 권장한다. Swagger에 대한 설정은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.2. 기본설정 (Profile / Config)" 을 참조 하면 확인 할 수 있다.
- 주소 : <http://localhost:8080/swagger-ui.html>

Hello Controller Hello Controller

GET /api/hello hello

Parameters

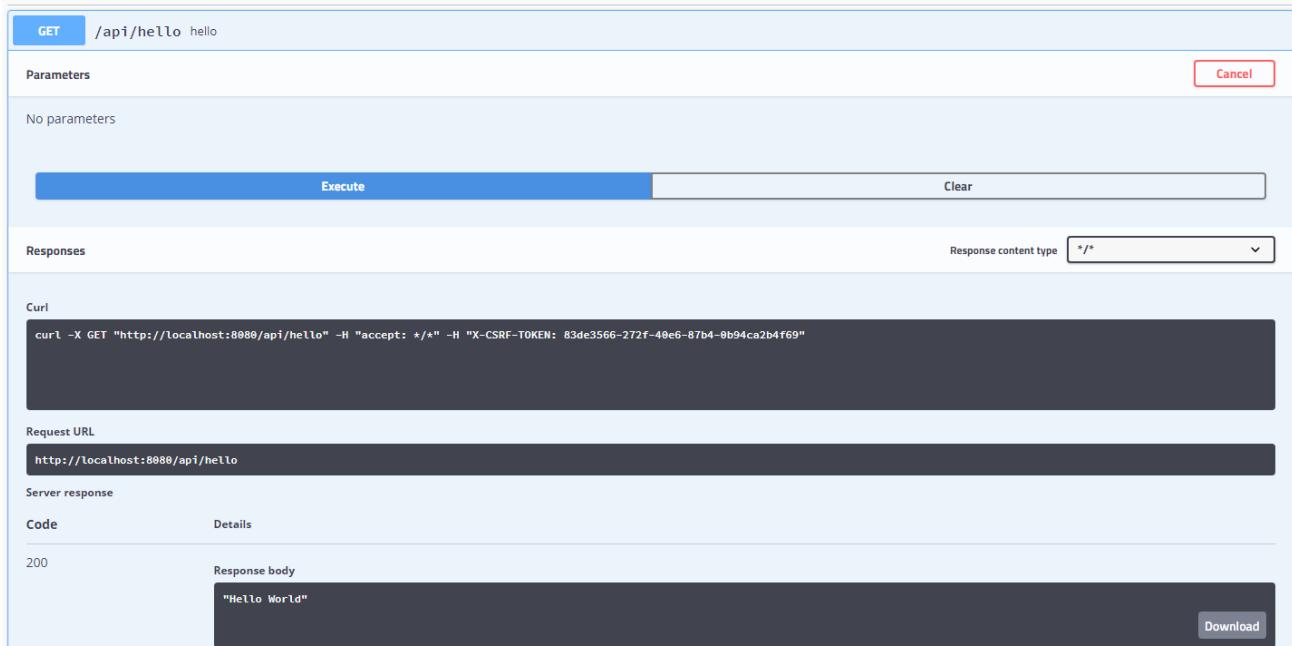
No parameters

Responses

Response content type */*

Code	Description
200	OK Example Value Model "string"
401	Unauthorized
403	Forbidden
404	Not Found

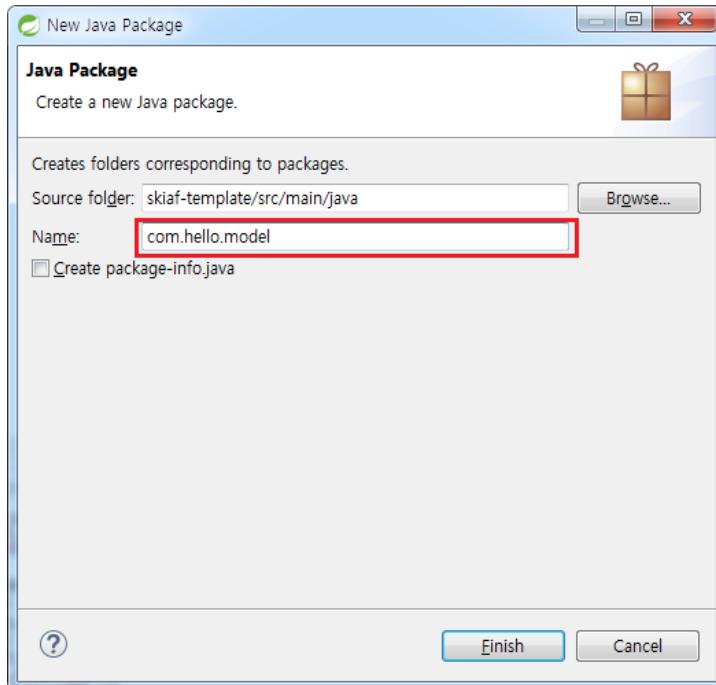
- Swagger 에 접속해 Hello Controller를 클릭하면, 만들어진 hello 메소드를 확인 할 수 있다.
- Try it out - Execute 를 클릭하면, 아래와 같이 결과를 확인 할 수 있다.

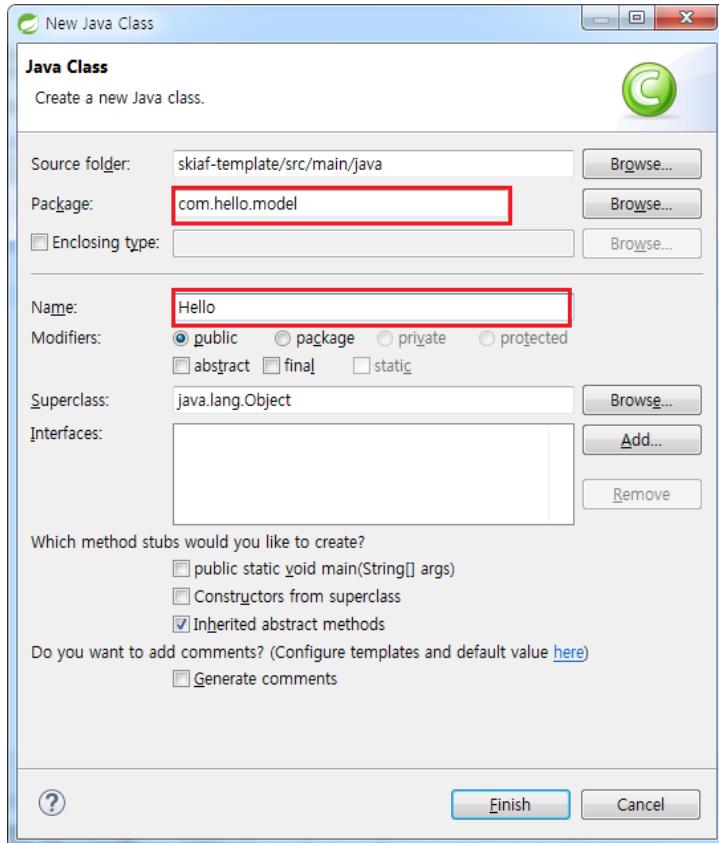


II.4.2 JPA 작업하기

[JPA를 이용해 DB 입출력 하기]

- 데이터 생성하기 위해 엔티티를 만들어야한다. 패키지를 새로 생성하고 Class를 만든다.





- 아래와 같이 Hello.java에 입력한다.
- 우리는 ID과 이름을 추가 할 것이다. ID는 자동으로 생성되고 이름은 입력받는 형식으로 만든다.

```
Hello.java

package com.hello.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data // lombok 어노테이션, getter, setter 등을 자동으로 생성해줌
@Entity (name = "HELLO")// JPA에서 테이블과 매핑할 클래스에 필수로 넣어야하는 어노테이션, name은 생성 할 테이블 이름
public class Hello {

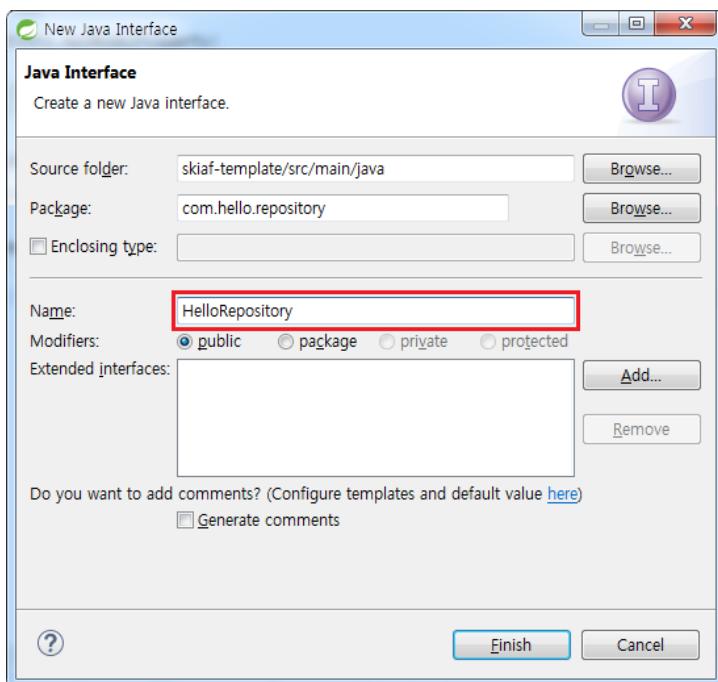
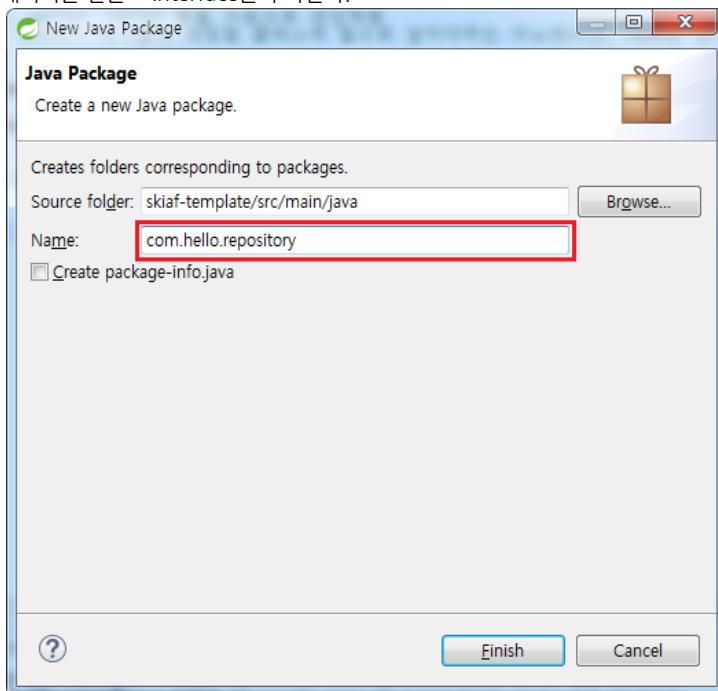
    @ApiModelProperty(hidden = true) //스웨거 모델 설정(값이 자동으로 생성되므로 숨기기처리)
    @Id // 키값
    @GeneratedValue(strategy = GenerationType.AUTO) // 시퀀스를 자동으로 생성해주는 어노테이션
    private int id;

    @ApiModelProperty(required = true, example = "이름") //스웨거 모델 설정(example은 예시)
    private String name;

}
```

- DB에 데이터를 입출력 하기 위해 Repository를 작성해야 한다.

- 패키지를 만들고 Interface를 추가한다.

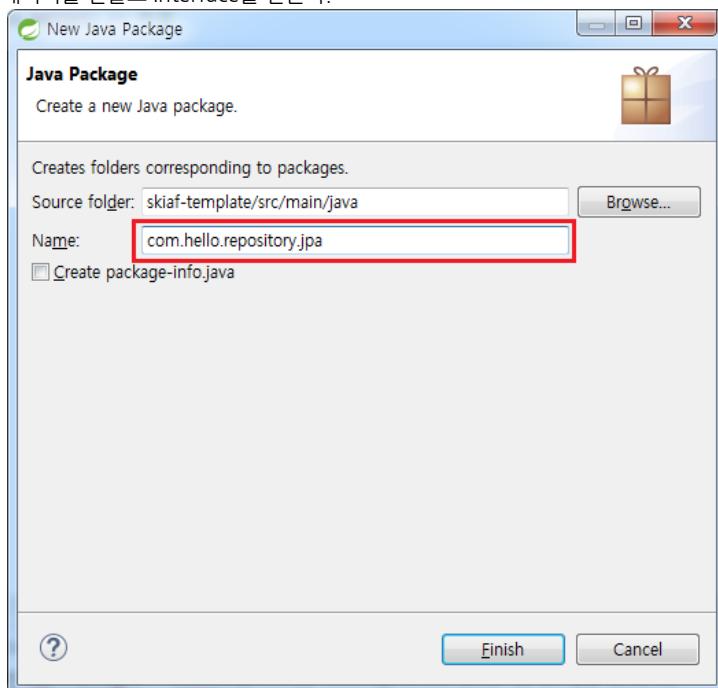


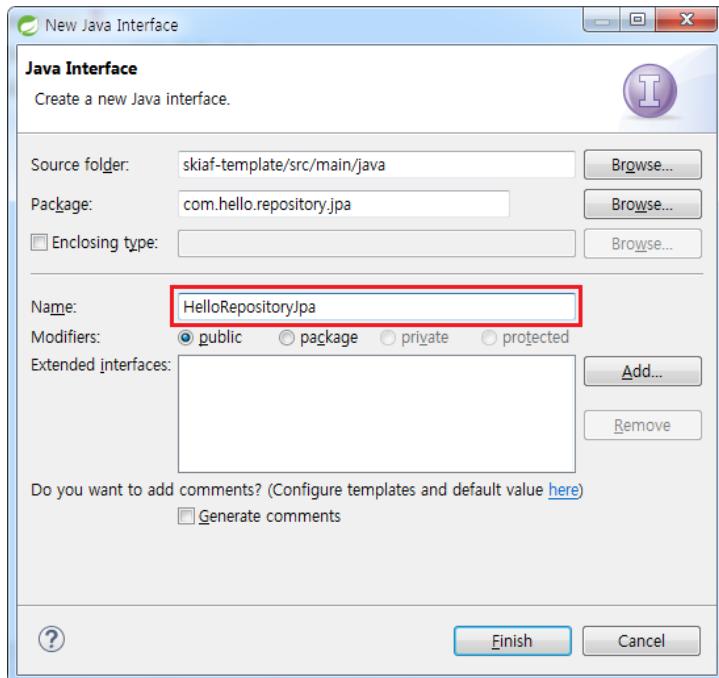
- 아래와 같이 HelloRepository.java를 입력한다.

HelloRepository.java

```
package com.hello.repository;  
  
import java.util.List;  
  
import org.springframework.stereotype.Repository;  
import com.hello.model.Hello;  
  
@Repository //Repository 어노테이션  
public interface HelloRepository {  
  
    public List<Hello> findAll(); // 전체 목록 검색  
  
    public Hello save(Hello hello); // 저장  
  
}
```

- JPA를 상속받을 Repository를 만들어야한다.
파키지를 만들고 Interface를 만든다.





- 아래와 같이 HelloRepositoryJpa.java를 입력한다.

```
HelloRepositoryJpa.java

package com.hello.repository.jpa;
import org.springframework.data.jpa.repository.JpaRepository;
import com.hello.model.Hello;
import com.hello.repository.HelloRepository;
public interface HelloRepositoryJpa extends HelloRepository, JpaRepository<Hello, Integer> {
}
```

- 기존에 만들었던 컨트롤러를 수정한다.

HelloController.java

```

package com.hello.web;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import com.hello.model.Hello;
import com.hello.repository.HelloRepository;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;

@Api(tags = "Hello Controller")
@RestController // RestController 어노테이션
public class HelloController {

    @Autowired
    HelloRepository helloRepository;

    @ApiOperation(value = "헬로우 월드")
    @GetMapping("/api/hello") // Get Method 방식, 주소 입력
    public String hello() {
        return "Hello World"; // 출력값
    }

    @ApiOperation(value = "헬로우 리스트")
    @GetMapping("/api/hellolist")
    public List<Hello> helloList() {

        return helloRepository.findAll();
    }

    @ApiOperation(value = "헬로우 추가")
    @PostMapping("/api/helloadd") // Post Method 방식
    public Hello helloName(Hello hello) {

        Hello helloData = helloRepository.save(hello);

        return helloData;
    }
}

```

- 프로젝트를 실행하고 결과값을 확인해 본다.
- Post 방식의 결과를 확인하기 위해선 Swagger를 이용해야한다.
- Post - 헬로우 추가를 누르고 Try it out을 누른다
- 추가할 데이터를 입력해야한다. 이름 부분에 임의의 값을 넣고 Execute를 누르면 추가된다

POST /api/helloadd 템포우 추가

Parameters

Name	Description
name * required string (query)	이름

Responses

Code	Description
200	OK
201	Created

Example Value | Model

```
{
  "name": "이름"
}
```

- 정상적으로 값이 추가된다면 아래와 같이 결과를 확인 할 수 있다.

name * required
string
(query)

이름

Execute Clear

Responses

Curl

```
curl -X POST "http://localhost:8080/api/helloadd?name=%EC%9D%B4%EB%A6%84" -H "accept: */*" -H "X-CSRF-TOKEN: 8bbbdeb7-703d-468a-8d84-5ba28379baeb"
```

Request URL

```
http://localhost:8080/api/helloadd?name=%EC%9D%B4%EB%A6%84
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 4, "name": "이름" }</pre> <p>Download</p> <p>Response headers</p> <pre>cache-control: no-cache, no-store, max-age=0, must-revalidate content-encoding: gzip content-type: application/json; charset=UTF-8 date: Wed, 24 Oct 2018 05:28:25 GMT expires: 0 pragma: no-cache transfer-encoding: chunked vary: Accept-Encoding x-content-type-options: nosniff</pre>

- 여러건을 입력 해 리스트를 확인해보자
- 일반 브라우저와 스웨거 두곳에서 모두 확인 할 수 있다.
- 주소 : <http://localhost:8080/api/hellolist>

GET /api/hellolist 헬로우 리스트

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X GET "http://localhost:8080/api/hellolist" -H "accept: */*" -H "X-CSRF-TOKEN: 8bbbdeb7-703d-468a-8d84-5ba28379baeb"
```

Request URL

<http://localhost:8080/api/hellolist>

Server response

Code	Details
200	<p>Response body</p> <pre>[{"id": 1, "name": "이름"}, {"id": 2, "name": "이름2"}, {"id": 3, "name": "이름3"}, {"id": 4, "name": "이름"}]</pre> <p>Download</p>

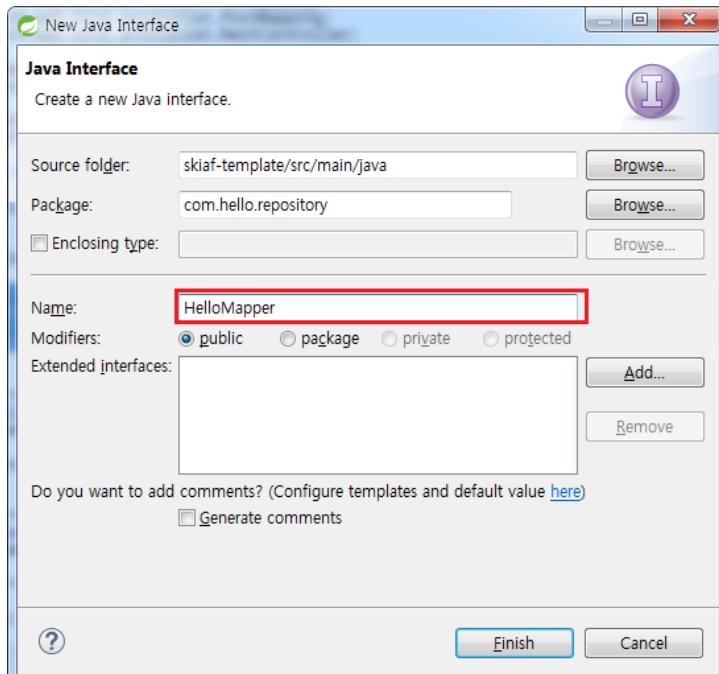
localhost:8080/api/hellolist

```
[{"id": 1, "name": "이름"}, {"id": 2, "name": "이름2"}, {"id": 3, "name": "이름3"}, {"id": 4, "name": "이름"}]
```

II.4.3 SQL Mapper 사용하기

[SQL Mapper(mybatis)를 이용해 출력하기]

- Controller와 Entity 모델은 그대로 사용하고 Mapper를 새로 추가 해야한다.
- repository 패키지에 HelloMapper Interface 파일을 만든다.

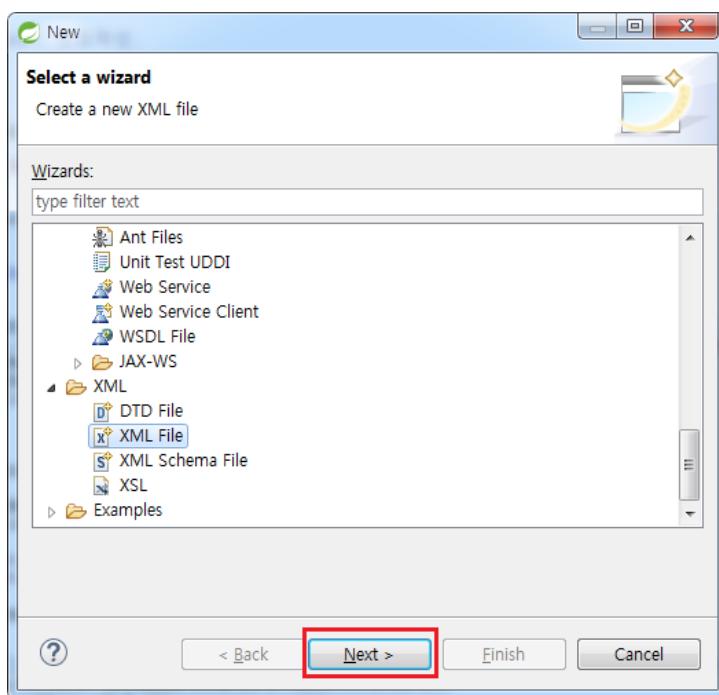
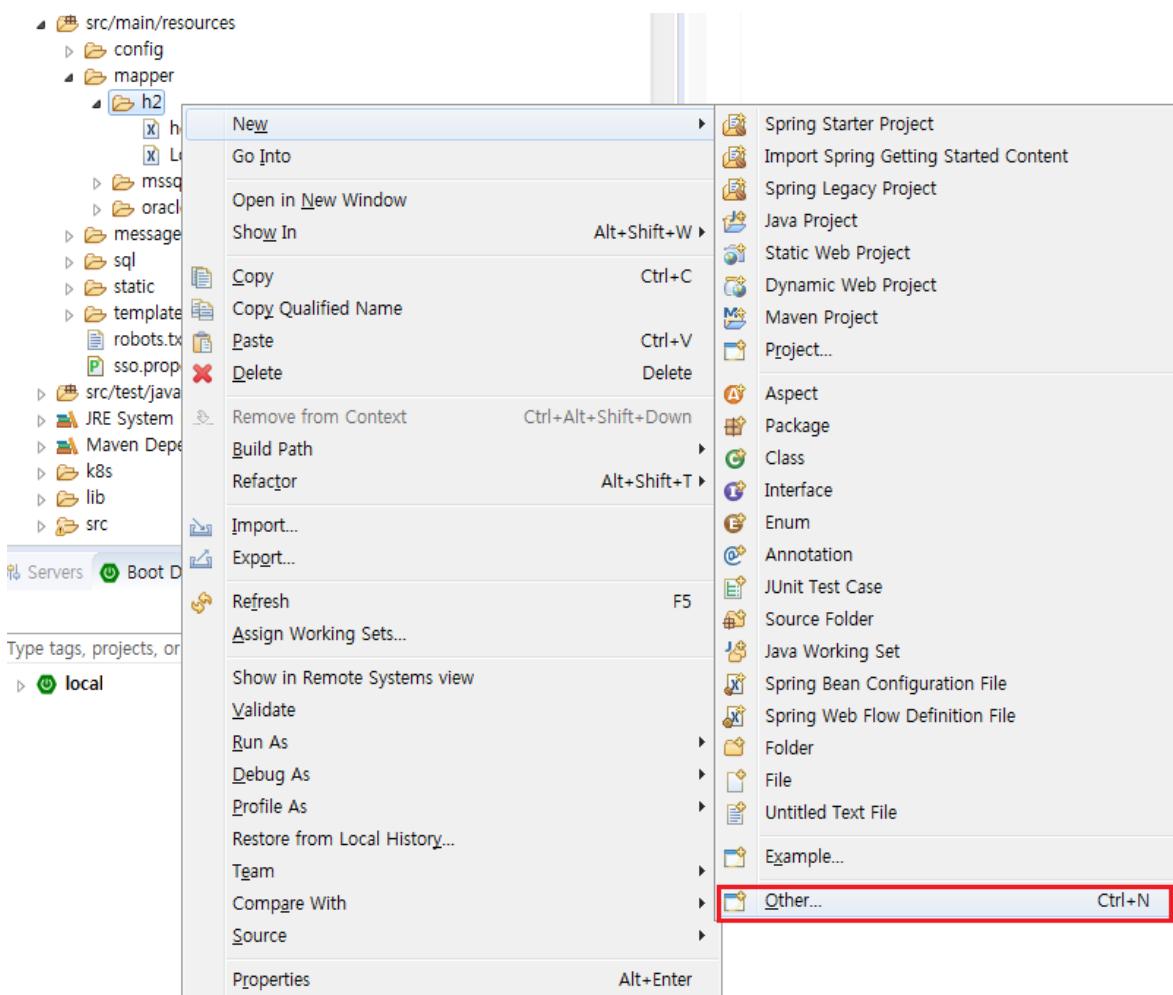


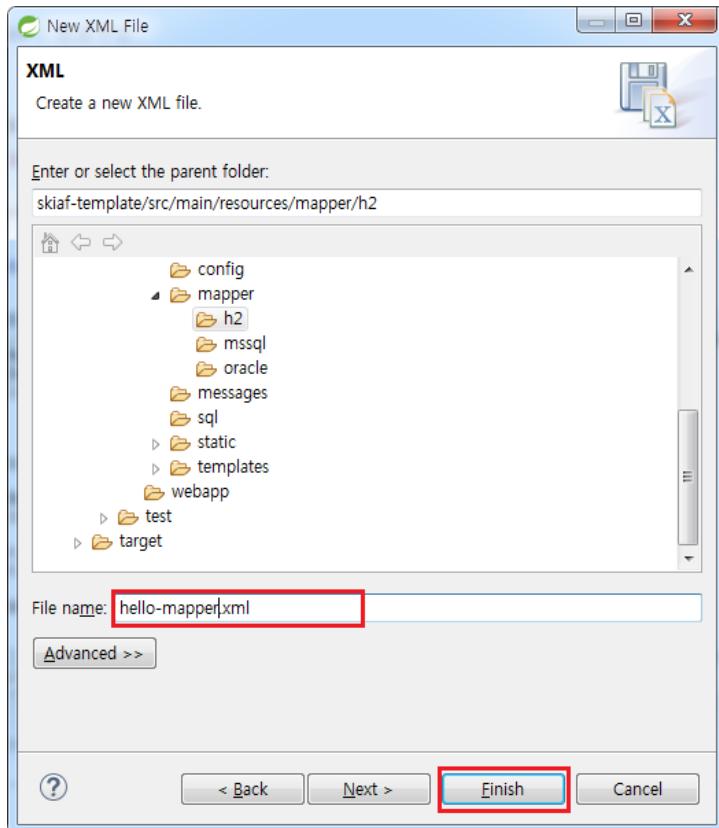
- 아래와 같이 코드를 입력한다.

HelloMapper.java

```
package com.hello.repository;
import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import com.hello.model.Hello;
@Mapper // Mapper 어노테이션
public interface HelloMapper {
    public List<Hello> selectHelloList(); // 전체 목록 검색
    public void insertHello(Hello hello); // 목록 저장
}
```

- hello-mapper.xml 파일을 만든다. 위치는 src/main/resources/mapper 이다.
- 각 DB별로 맞는 폴더에 XML파일을 만들면 된다. classpath에 관한 내용은 com.skiaf.core.config.MyBatisConfig.java 파일은 참조한다.





- hello-mapper.xml 파일과 HelloController.java 파일을 아래와 같이 입력한다.

hello-mapper.xml

```
<?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="com.hello.repository.HelloMapper"><!-- HelloMapper.java 패키지 --&gt;

    &lt;select id="selectHelloList" resultType="com.hello.model.Hello"&gt;<!-- Entity Model --&gt;
        SELECT * FROM HELLO
    &lt;/select&gt;

    &lt;insert id="insertHello" useGeneratedKeys="true" keyProperty="id" parameterType="com.hello.model.Hello"&gt;<!-- Sequence값 설정 포함 --&gt;
        INSERT INTO HELLO (NAME) VALUES (#{name})
    &lt;/insert&gt;

&lt;/mapper&gt;</pre>

```

HelloController.java

```

package com.hello.web;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import com.hello.model.Hello;
import com.hello.repository.HelloMapper;
import com.hello.repository.HelloRepository;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;

@Api(tags = "Hello Controller")
@RestController // RestController 어노테이션
public class HelloController {

    @Autowired
    HelloRepository helloRepository;

    @Autowired
    HelloMapper helloMapper;

    @ApiOperation(value = "헬로우 월드")
    @GetMapping("/api/hello") // Get Method 방식, 주소 입력
    public String hello() {
        return "Hello World"; // 출력값
    }

    @ApiOperation(value = "헬로우 리스트")
    @GetMapping("/api/hellolist")
    public List<Hello> helloList() {
        return helloRepository.findAll();
    }

    @ApiOperation(value = "헬로우 추가")
    @PostMapping("/api/helloadd") // Post Method 방식
    public Hello helloName(Hello hello) {
        Hello helloData = helloRepository.save(hello);
        return helloData;
    }

    /** SQL Mapper 관련 내용 추가 START ***/
    @ApiOperation(value = "myBatis 헬로우 추가")
    @PostMapping("/api/mybatisadd")
    public Hello insertHello(Hello hello) {
        helloMapper.insertHello(hello);
        return hello;
    }

    @ApiOperation(value = "myBatis 헬로우 리스트")
    @GetMapping("/api/mybatislist")
    public List<Hello> helloList() {
        List<Hello> hello = helloMapper.selectHelloList();
        return hello;
    }
    /** SQL Mapper 관련 내용 추가 END ***/
}

```

- JPA와 같은 방법으로 결과를 확인 할 수 있다.
- Post Method는 스웨거로 테스트 한다.

POST /api/mybatisadd myBatis 웹로우 추가

Parameters

Name	Description
name * required string (query)	이름

Execute Clear

Responses

Curl

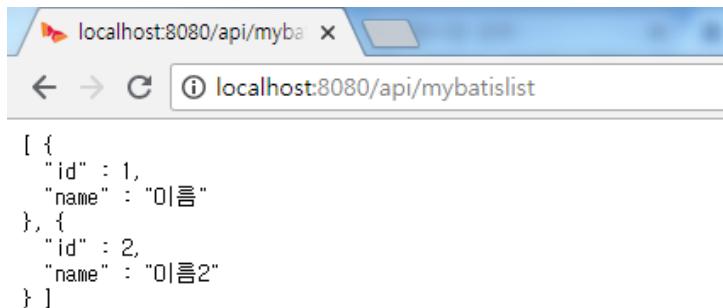
```
curl -X POST "http://localhost:8080/api/mybatisadd?name=%EC%9D%B4%EB%A6%84" -H "accept: */*" -H "X-CSRF-TOKEN: fcb5ce93-bb91-4cb5-a8d7-7d111fae50bd"
```

Request URL

<http://localhost:8080/api/mybatisadd?name=%EC%9D%B4%EB%A6%84>

Server response

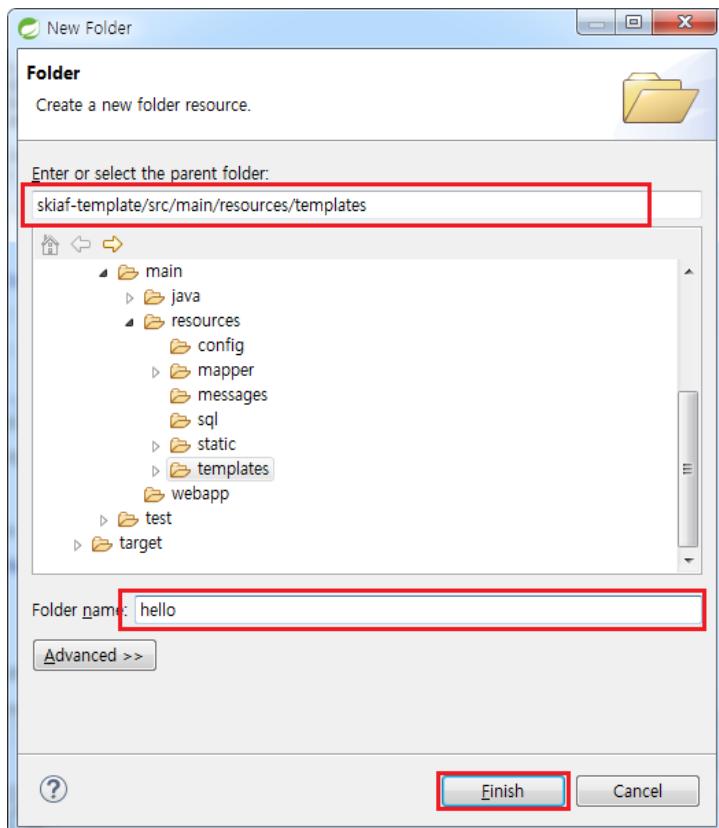
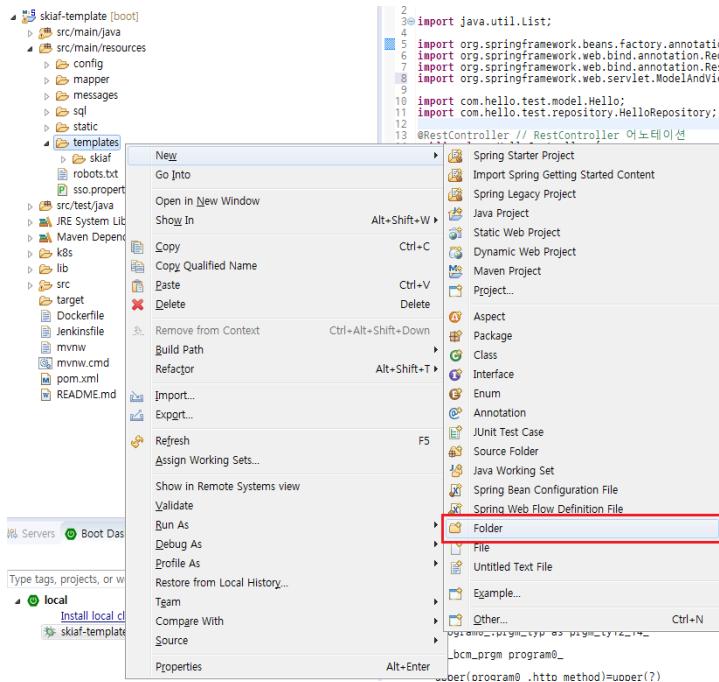
Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "name": "이름" }</pre> <p>Download</p>



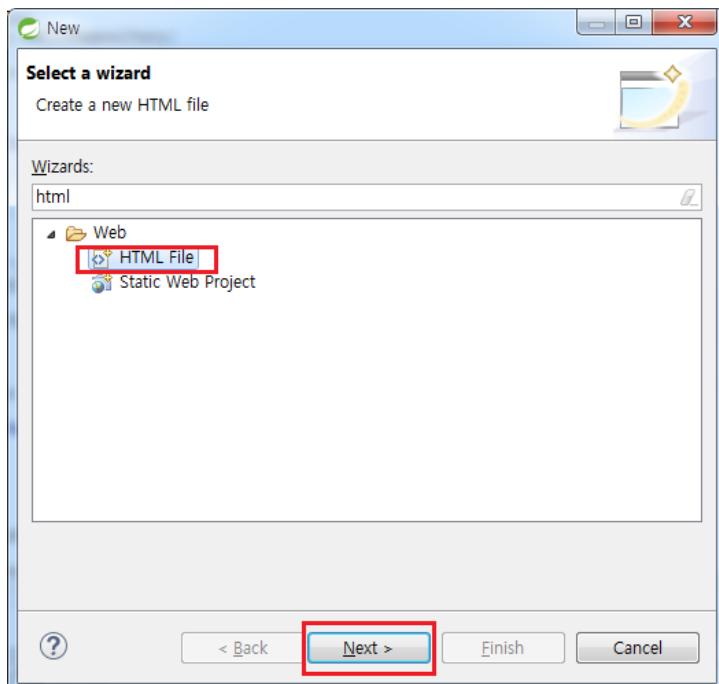
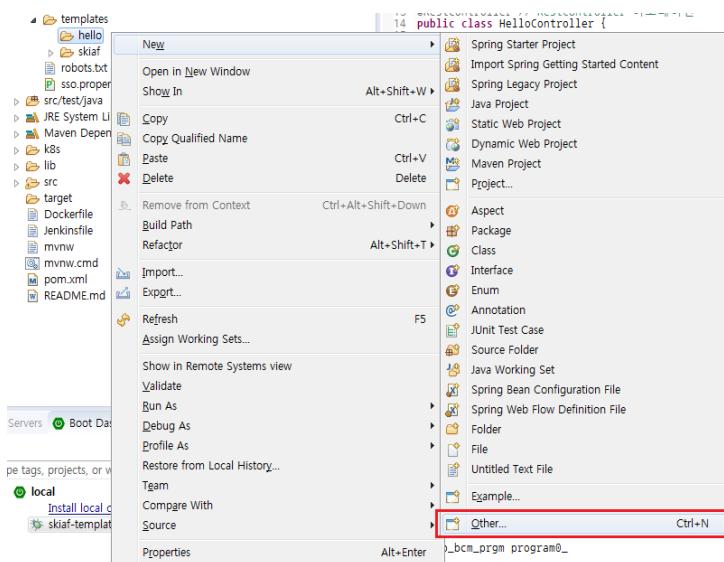
II.4.4 Thymeleaf 를 이용하기

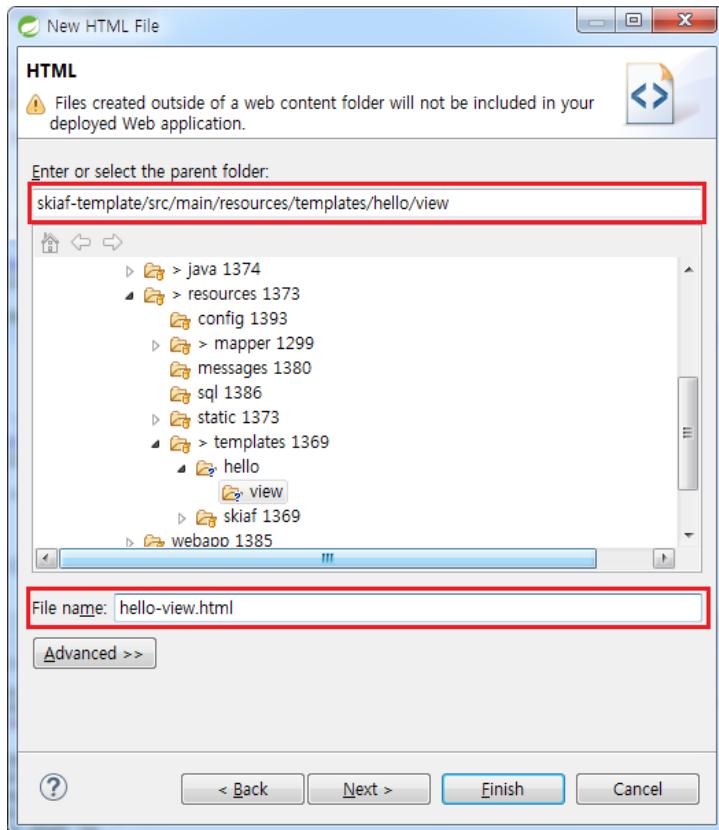
[Thymeleaf 를 이용해 출력하기]

- Thymeleaf를 이용해 helloList를 출력한다.
- html파일을 넣을 /templates/hello 폴더와 /templates/hello/view 폴더 두개를 생성한다.



- Thymeleaf 파일의 확장자는 html이다.
- /templates/hello/view 폴더 안에 html 파일을 생성한다.





- 아래와 같이 입력한다.

```
hello-view.html

<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">

    <h2> hello </h2>
    <table border="0">
        <tr>
            <td>ID</td>
            <td>NAME</td>
        </tr>
        <tr th:each="list:${data}">
            <td th:text="${list.id}"></td>
            <td th:text="${list.name}"></td>
        </tr>
    </table>
</html>
```

- 컨트롤러 파일을 통해 html 파일을 불러와야 하니 컨트롤러 파일을 수정한다. JPA방식으로 부를 view와 myBatis로 부를 view 2개의 method를 추가한다.

HelloController.java

```

package com.hello.web;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

import com.hello.model.Hello;
import com.hello.repository.HelloMapper;
import com.hello.repository.HelloRepository;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;

@Api(tags = "Hello Controller")
@RestController //RestController 어노테이션
public class HelloController {

    @Autowired
    HelloRepository helloRepository;

    @Autowired
    HelloMapper helloMapper;

    @ApiOperation(value = "헬로우 월드")
    @GetMapping("/api/hello") // Get Method 방식, 주소 입력
    public String hello() {
        return "Hello World"; // 출력값
    }

    @ApiOperation(value = "헬로우 리스트")
    @GetMapping("/api/hellolist")
    public List<Hello> helloList() {
        return helloRepository.findAll();
    }

    @ApiOperation(value = "헬로우 추가")
    @PostMapping("/api/helloadd") // Post Method 방식
    public Hello helloName(Hello hello) {
        Hello helloData = helloRepository.save(hello);
        return helloData;
    }

    /** SQL Mapper 관련 내용 추가 START ***/
    @ApiOperation(value = "myBatis 헬로우 추가")
    @PostMapping("/api/mybatisadd")
    public Hello insertHello(Hello hello) {
        helloMapper.insertHello(hello);
        return hello;
    }

    @ApiOperation(value = "myBatis 헬로우 리스트")
    @GetMapping("/api/mybatislist")
    public List<Hello> hellolist() {
        List<Hello> hello = helloMapper.selectHelloList();
        return hello;
    }
    /** SQL Mapper 관련 내용 추가 END ***/
}

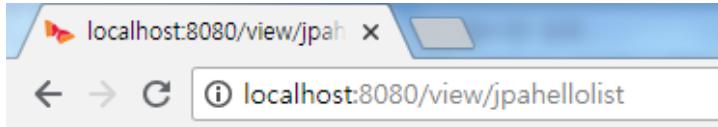
@GetMapping("/view/jpahellolist") // view 는 GetMapping을 사용한다.

```

```
public ModelAndView jpaHelloListView() {  
    ModelAndView modelAndView = new ModelAndView();  
  
    modelAndView.addObject("data", helloRepository.findAll()); //jpa의 findAll을 이용해 리스트 가져오기  
    modelAndView.setViewName("hello/view/hello-view");  
  
    return modelAndView;  
}  
  
@GetMapping("/view/mybatisHelloList") // view 는 GetMapping을 사용한다.  
public ModelAndView mybatisHelloListView() {  
  
    ModelAndView modelAndView = new ModelAndView();  
  
    modelAndView.addObject("data", helloMapper.selectHelloList()); //mybatis mapper의 selectHelloList를 이용해 리스트  
    가져오기  
    modelAndView.setViewName("hello/view/hello-view");  
  
    return modelAndView;  
}
```

```
    }
```

- 이전에 만든 리스트를 저장하는 주소를 통해 데이터를 추가하고 View를 통해 확인해보자.
- JPA를 이용해 view 확인 : <http://localhost:8080/view/jpahellolist>

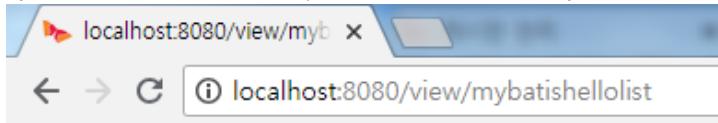


hello

ID NAME

- 1 이름1
- 2 이름2
- 3 이름3

- myBatis를 이용해 view 확인 : <http://localhost:8080/view/mybatishellolist>



hello

ID NAME

- 1 이름1
- 2 이름2
- 3 이름3

II.4.5 프로그램 등록 및 메뉴 등록하기

참고사항

로그인 세션 관련 AOP 처리 내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.6 MVC"의 AOP 부분을 참조한다.

본 예제는 프로그램 등록과 메뉴 등록에 관한 내용만 포함한다.

[프로그램 등록하기]

- 실습을 통해 만든 화면 2개(JPA Hello List, myBatis Hello List)를 SKIAF에 프로그램으로 등록한다.
- 프로그램으로 등록하고 권한을 추가하면 로그인을 해야만 view 페이지를 확인 할 수 있게된다.

- skiaf-template 메인 화면으로 들어가 관리자 계정으로 로그인을 한다.
- 업무 공통 > 프로그램 관리 페이지로 들어가 등록 버튼을 클릭한다.
- 등록 화면에서 다음과 같이 입력 후 아래 저장 버튼을 눌러 저장하면 프로그램이 등록 된것이다.

프로그램 등록

유형	<input checked="" type="radio"/> VIEW <input type="radio"/> SERVICE
대표 프로그램 ID	PVHELLO × 중복체크
사용 가능한 대표 프로그램 ID입니다. 대표 프로그램 ID는 최대 10자까지입니다.	
대표 프로그램 명	헬로우 월드 ×

선택	프로그램 ID	프로그램 명	설명	메소드	경로	사용여부
<input checked="" type="checkbox"/>	PVHELLO-01	JPA 헬로우 월드	JPA 헬로우 리스트	GET	/view/jpahellolist	Y
<input checked="" type="checkbox"/>	PVHELLO-02	myBatis 헬로우 월드	myBatis 헬로우 리스트	GET	/view/mybatishellolist	Y

+

[메소드]

GET : 특정 리소스의 표시를 요청합니다. GET을 사용하는 요청은 오직 데이터를 받기만 합니다.
 HEAD : GET 메소드의 요청과 동일한 응답을 요구하지만, 응답 본문을 포함하지 않습니다.
 POST : 특정 리소스에 엔티티를 제출할 때 쓰입니다. 이는 종종 서버의 상태의 변화나 부작용을 일으킵니다.
 PUT : 목적 리소스 모든 현재 표시를 요청 payload로 바꿉니다.
 DELETE : 특정 리소스를 삭제합니다.
 CONNECT : 목적 리소스로 식별되는 서버로의 터널을 맺습니다.
 OPTIONS : 목적 리소스의 통신을 설정하는 데 쓰입니다.
 TRACE : 목적 리소스의 경로를 따라 메시지 loop-back 테스트를 합니다.
 PATCH : 리소스의 부분만을 수정하는 데 쓰입니다.

- 프로그램을 등록 했으니 권한을 부여해야 한다.
- 등록한 프로그램 중 하나를 선택하고 권한 관리 버튼을 누른다.

프로그램 관리

전체 프로그램 ID, 프로그램 명을 입력하십시오. 검색 미사용 포함

수정 등록 **권한 관리**

선택	프로그램 ID	프로그램 명	유형	설명	메소드	경로	사용 여부	수
<input type="checkbox"/>	PVHELLO-02	myBatis 헬로우 월드	VIEW	myBatis 헬로우 리스트	GET	/view/mybatisHelloList	Y	skia
<input checked="" type="checkbox"/>	PVHELLO-01	JPA 헬로우 월드	VIEW	JPA 헬로우 리스트	GET	/view/jpaHelloList	Y	skia
<input type="checkbox"/>	PSROLE-05	Role 상세조회(매핑...)	SERVICE	권한 Id가 일치하는 권한에 ...	GET	/api/roles/{roleId}/commands...	Y	ac
<input type="checkbox"/>	PSROLE-04	Role 수정	SERVICE	권한 Id가 일치하는 권한정보...	PUT	/api/roles/{roleId}	Y	ac
<input type="checkbox"/>	PSROLE-07	Role에 사용자그룹...	SERVICE	권한 사용자 그룹 연결 목록...	POST	/api/roles/{roleId}/map/usergr...	Y	ac
<input type="checkbox"/>	PSROLE-02	Role 등록	SERVICE	권한을 저장한다.	POST	/api/roles	Y	ac
<input type="checkbox"/>	PSELMT-07	프로그램 요소 순번...	SERVICE	프로그램 요소 순번으로 조회	GET	/api/elements/{elementSeq}/...	Y	ac
<input type="checkbox"/>	PSELMT-04	프로그램 요소 권한...	SERVICE	프로그램 요소와 권한을 저장	POST	/api/elements/programs/{pro...	Y	ac
<input type="checkbox"/>	PSELMT-06	프로그램 요소 중복...	SERVICE	프로그램 요소에 대한 중복 ...	GET	/api/elements/{elementKey}/...	Y	ac
<input type="checkbox"/>	PSROLE-01	Role 검색 페이징 ...	SERVICE	검색 조건, 페이지 설정에 따...	GET	/api/roles	Y	ac

총 건수 : 151

1 2 3 4 5 6 7 8 9 10 > >>

10 ▾



- 지금은 추가된 권한이 없기 때문에 목록이 나오지 않는다.
- 왼쪽 아래 권한 추가 버튼을 눌러 아래와 같이 권한을 추가한다.
- 권한은 게시판 관리 권한을 부여했다.

프로그램 권한 선택



적용 대상 프로그램 선택

선택	프로그램 ID	프로그램 명	유형
<input checked="" type="checkbox"/>	PVHELLO-02	myBatis 헬로우 월드	VIEW
<input checked="" type="checkbox"/>	PVHELLO-01	JPA 헬로우 월드	VIEW

적용 대상 권한 선택

권한 ID, 권한명을 입력하십시오. 검색

선택	권한 ID	권한명	권한설명
<input type="checkbox"/>	APBCM00001	PRO_USER	사용자 관리 권한
<input type="checkbox"/>	APBCM00002	PRO_MENU	메뉴 관리 권한
<input type="checkbox"/>	APBCM00003	PRO_PROGRAM	프로그램 관리 권한
<input type="checkbox"/>	APBCM00004	PRO_ROLE	권한 관리 권한
<input type="checkbox"/>	APBCM00005	PRO_CODE	코드 관리 권한
<input type="checkbox"/>	APBCM00006	PRO_MESSAGE	메시지 관리 권한
<input checked="" type="checkbox"/>	APBCM00007	PRO_BOARD	게시판 관리 권한
<input type="checkbox"/>	APBCM00008	PRO_VIEW_LOG	로그조회 권한
<input type="checkbox"/>	APBCM00009	PRO_BOARD_BB2	bb2 권한

취소

적용

- 추가 된 목록을 확인하고 저장 버튼을 누르면 프로그램에 권한이 추가 된다.

프로그램 권한 관리

선택

대표 프로그램 ID	PVHELLO	유형	VIEW	▼			
프로그램 권한 목록							
선택	권한 ID	권한명	프로그램 ID	프로그램 명	유형	메소드	경로
<input checked="" type="checkbox"/>	APBCM00007	PRO_BOARD	PVHELLO-02	myBatis 헬로우 월드	VIEW	GET	/view/mybatishellolist
<input checked="" type="checkbox"/>	APBCM00007	PRO_BOARD	PVHELLO-01	JPA 헬로우 월드	VIEW	GET	/view/jpahellolist

+ (추가)
이전
삭제
저장

- 권한이 추가 되면 APBCM00007 이라는 권한을 가지고 있는 사용자만 헬로우 예제를 확인 할 수 있고, 권한이 없는 다른 사용자는 확인 할 수 없게된다.
- 로그아웃을 하고 예제 URL을 입력해 들어가면 로그인을 하라는 오류 페이지를 확인 할 수 있다.
- 프로그램 관리에 대한 자세한 내용은 "V. SKIAF 개발자매뉴얼 - 업무공통기능 > V4. 프로그램 관리" 를 참조한다.

[메뉴 등록하기]

- 프로그램을 등록 했다면 좌측 메뉴에 메뉴 등록을 할 수 있다.
- 왼쪽 메뉴에서 개발용 폴더에 메뉴를 추가한다.
- 업무공통 - 메뉴관리를 클릭하고 등록 버튼을 클릭한다.
- 아래와 같이 입력하고 저장한다.

메뉴 등록

메뉴 ID*	MBCM00405	<input type="button" value="중복체크"/>
사용 가능한 메뉴 ID입니다. * 영문, 숫자와 특수기호(.)_(.)만 사용 가능합니다. (최대 10자) * 예) MENU001		
메뉴명 (ko)*	myBatis 헬로우 월드	<input type="button" value="X"/>
메뉴명 (en)		
메뉴명 (zh)		
메뉴 설명		
메뉴 유형*	<input type="radio"/> 폴더 <input checked="" type="radio"/> 프로그램 <input type="radio"/> URL	
프로그램 ID*	PVHELLO-02	<input type="button" value="프로그램 선택"/> <input type="button" value="X"/>
오픈 유형*	현재창	
* 현재창 : 열려 있는 현재창이 변경됩니다. * 새탭 : 현재창을 유지하고, 현재창 우측 새탭으로 오픈됩니다. * 새창 : 현재창을 유지하고, 브라우저 새창으로 오픈됩니다.		
상위메뉴*	개발용	<input type="button" value="X"/> <input type="button" value="상위메뉴선택"/> <input type="button" value="Search"/>
정렬 순번*	5	<input type="button" value="X"/>
* 숫자만 입력 가능합니다. * 같은 레벨의 메뉴가 순번이 같을 경우 메뉴 ID 기준으로 정렬 됩니다.		

- 예제에서 만든 2개의 프로그램을 메뉴로 모두 등록 했다면 아래와 같이 메뉴에서 확인 할 수 있다.



- 메뉴를 클릭해 페이지가 제대로 나오는지 확인한다.
- 메뉴 관리에 대한 자세한 내용은 "V. SKIAF 개발자매뉴얼 - 업무공통기능 > V6. 메뉴 관리"를 참조한다.

Appendix. 프로젝트 수행시 고려사항

[문서의 목적]

skiaf는 binary 형태의 library가 아닌 Open Source 형태로 제공되는 F/W이며, 프로젝트 요구사항에 따라 다양한 커스터마이징이 가능하다.

프로젝트를 수행하려는 개발 책임자 또는 프로젝트 아키텍트는 skiaf를 어떻게 적용해야 할지를 결정하고, 그에 따라 프로젝트 자체 표준 및 개발 절차를 정의해야 한다.

따라서, 본 문서는 개발자를 위한 프로젝트 수행절차라기 보다는 각 프로젝트의 개발 책임자 또는 아키텍트 담당자가 skiaf를 어떻게 커스터마이징해야 할지에 대한 간략한 지침을 제공한다.

[커스터마이징 전략]

프로젝트의 개발 책임자는 개발을 시작하기 전에 skiaf에 대한 커스터마이징 목표와 범위를 정해야 한다.

즉, skiaf 기능을 있는 그대로 모두 활용하여 프로젝트의 요구사항과 관계된 기능만 추가할 것인지, 시스템/업무공통기능은 활용하되 UI 부분만 바꿀 것인지, 일부의 기능만을 바꿀 것인지 등의 커스터마이징 목표에 따라 커스터마이징 대상이 달라진다.

1) UI / UX 커스터마이징

구분	변경 영향도	커스터마이징 범위 / 전략
Grid, Tree 등 각종 컴포넌트	매우 큼	Alopex 대신 대체 솔루션을 찾아야 하며, skiaf UI 표준 및 관련 리소스 (css, js, image파일) 등을 사용할 수 없다.
전반적인 화면 구성 및 레이아웃	약간 큼	UI/UX 표준은 활용할 수 없으나 skiaf UI 관련 리소스는 활용 가능함
버튼 Image, Color 등 각종 스타일	작음	UI 표준 및 스타일 가이드를 참고하여 css 파일 및 image를 수정
GNB / LNB 메뉴, 메인, 로그인 화면	적정	관련 프로그램 또는 파일을 찾아 해당 부분만 수정 LNB의 경우, 메뉴관리 기능을 이용하여 구성 가능함

2) 시스템 공통 커스터마이징

시스템공통 기능에 대해서는 skiaf 본연의 핵심 기능이므로, 코딩 레벨의 커스터마이징을 권고하지 않는다.

단, 시스템의 핵심 기능을 이해하고 옵션의 변경 및 추가사항 반영 등을 위해서는 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능"을 참고하여, 프로젝트에 필요한 내용을 변경하도록 한다.

구분	고려사항	커스터마이징 범위 / 전략
세션관리	Redis 사용여부	skiaf의 세션관리 기능은 Redis를 지원한다. Redis 사용을 원치 않을 경우 application.properties 수정이 필요하며 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.2 기본설정 (Profile / Config)"을 참고한다.

보안필터		<p>skiaf는 XSS, CORS, CSRF를 지원한다.</p> <p>XSS의 경우 기본적으로 Dome-based XSS 방식을 통해 XSS를 방어 하였고, html 메세지 필요한 경우, Lucy-xss Filter를 사용하여 필터링이 필요한곳 (메세지, 게시판)에 설정되어 있다.</p> <p>이후 프로젝트별로 필터링 예외 대상을 정의하고 Filter를 설정한다.</p> <p>상세내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.3 보안 > XSS" 를 참조한다.</p>
DB연결		<p>Oracle, MS-SQL, h2 DB는 모두 사용 가능하다.</p> <p>그 외의 DB를 사용할 경우, 각 쿼리가 맞게 수행되는지 SQL 검증이 필요하다.</p> <p>상세내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.5 데이터 연결" 을 참조한다.</p>
SQL 매퍼	JPA, MyBatis	<p>JPA, Mybatis 동시사용을 지원하며 그 외의 Mapper 사용시에는 커스터마이징을 해야 한다.</p> <p>상세내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.5 데이터 연결" 을 참조한다.</p>
로그	logback-spring.xml	<p>skiaf 로그처리는 Spring Logback을 사용하였다.</p> <p>로그수준 및 로그저장 방식(DB, File, Syslog 프로토콜) 등에 대해 프로젝트별 로그정책을 수립하고 필요한 경우 관련 프로그램을 수정해야 한다.</p> <p>상세내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.8 Logging" 을 참조한다.</p>
사용자 인증방식	SSO, 공인인증서 사용 여부	<p>SSO 또는 공개키인증을 사용할 것인지에 따라 사용자 인증 로직을 수정한다.</p> <p>단, SSO 인증 기능은 skiaf에서 제공되나 공개키인증은 지원하지 않는다.</p> <p>상세내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.3 보안 > 인증 (Authentication) " 를 참조한다.</p>
Cache	캐쉬설정 사용여부	<p>skiaf는 기본적으로 캐쉬를 사용하며(메시지, 메뉴정보), 추가적으로 캐쉬 적용할 대상을 설정할 수 있다.</p> <p>skiaf의 캐쉬 관리는 EHCache, Redis 사용 설정을 제공하며, 그 외 다른 메모리DB를 사용하고자 할 경우에는 커스터마이징이 필요하다.</p> <p>상세내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.9 Cache" 를 참조한다.</p>

3) 업무공통 커스터마이징

skiaf 업무공통 기능은 각각 다음과 같은 특징을 가지며, 부분적인 커스터마이징을 원할 경우 관련 기능과의 연관성, 영향 범위를 고려하여 변경해야 하므로 가급적 해당 기능을 모두 활용할 것을 권고한다.

구분	변경 영향도	커스터마이징 범위 / 전략
사용자	중간	<p>사용자 정보는 업무 요구사항에 따라 다양한 커스터마이징이 필요한 영역이다.</p> <p>단, User ID와 Login ID, 회사코드 속성은 권한처리와 관련되어 있으므로 유지해야 하는 속성이다.</p> <p>그 외 자세한 내용은 V.SKIAF개발가이드-업무공통 기능을 참고한다.</p>
코드관리	작음	<p>skiaf는 도메인 중심 사상을 부분적으로 지향하여 개발되었으며, 통합 DB 중심의 중앙집권적 코드사용을 권장하지 않는다. (JP/오브젝트를 사용함)</p> <p>그러나 범용 목적으로 코드 관리 기능을 제공하며, 각자 프로젝트 방식에 따라 코드관리 기능을 커스터마이징하도록 한다.</p>
게시판관리	중간	<p>공지사항, 자유게시판 등 목적별/용도별로 게시판을 추가할 수 있다.</p> <p>단, 게시판별로 권한설정을 하려면 권한관리 기능을 이해하고 각 게시판별 프로그램ID를 등록하여 필요한 권한을 부여해야 한다.</p> <p>skiaf에서 제공하는 게시판 형태를 바꾸고 싶은 경우에는 해당 소스를 참고하여 신규 개발한다.</p>

그 외, 사용자그룹관리/메뉴관리/프로그램관리/프로그램요소관리/권한관리 등의 기능은 프로그램에 대한 접근 및 사용권한 처리 등을 목적으로 강하게 결합되어 있으므로 가급적 커스터마이징하지 않기를 권고하지 않는다.

[프로젝트용 개발 구조 및 표준 정의]

위에 언급된 커스터마이징 목표와 범위/전략에 따라 자체 프로젝트의 개발 구조와 표준을 정의하도록 한다.

프로젝트 자체 표준에 포함되어야 할 내용은 다음과 같다.

- skiaf 의 디렉토리 및 패키지 구조를 참고하여 자체 표준 디렉토리 / 패키지 구조
- Java 파일, HTML, JS 등 프로젝트에서 사용할 각종 유형별 파일에 대한 명명 규칙
- UI 레이아웃 표준, 코딩표준 및 주석처리 등에 대해 프로젝트용 표준과 가이드
- 프로젝트를 수행하는 DevOps 등의 개발환경 정의

자체 표준을 정의하기 어려울 경우 skiaf의 디렉토리/패키지 구조, 가이드와 표준을 그대로 이용해도 무방하다.

단, skiaf 소스와 프로젝트 자체 소스가 구분되도록 명명 규칙은 필요하다.

이상과 같이 프로젝트 개발 준비를 마쳤으면, 개발자에게 개발가이드를 학습시키고 프로젝트를 수행한다.

주의사항

- "II. SKIAF 따라하기"에서 만든 HelloWorld 예제는 Sample 예제이다.
따라하기 예제와 다른 점은 PaaS에서 구동을 하기 위한 추가적인 내용들이 들어있다. (HelloConfig.java 등)
- HelloConfig.java내의 Entity Scan 범위설정이 com.skiaf로 설정되어있어 신규 패키지로 만든 작업들이 실행 되지 않을 수 있다.(@EntityScan(basePackages = {"com.skiaf"}))
따라서, 실제 프로젝트 개발을 할 시에는 반드시 com.hello.* 프로젝트를 삭제 해서 프로젝트 개발을 수행하는것을 권고한다.
- 자세한 내용은 skiaf-template 내 HelloConfig.java의 주석을 참조한다.

