# 강의 링크

<https://www.inflearn.com/course/%EC%8A%A4%ED%94%84%EB%A7%81-%EC%9E%85%EB%AC%B8-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8>

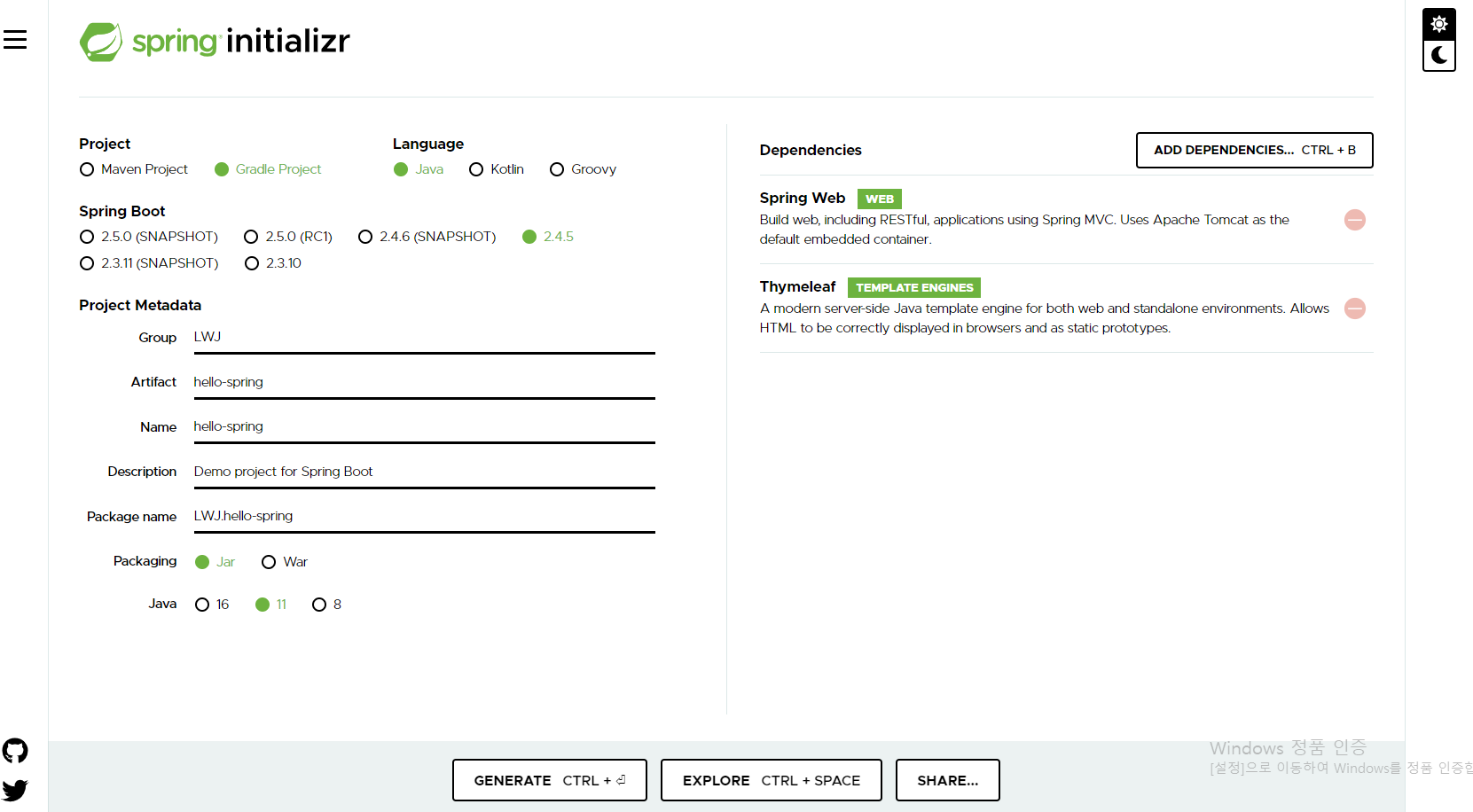
(김영한님)

# 환경 설정

1. java 11 설치 (oracle 가입 필요)
2. IntelliJ 설치
   1. 프로젝트 열고 빌드 시 - 환경설정에서 gradle 검색해보면 build Tools 아래에 Gradle 나올 것, 여기서 Build and run using, Run tests using 모두 IntelliJ IDEA 로 변경해야 빌드 빨리될 것.

# 프로젝트 생성

1. start.spring.io



1. project : Gradle 선택
2. Spring Boot

SNAPSHOT 은 아직 만들고있는 버전

다른것도 아직 정식 릴리즈된 버전이 아님

아무것도 안적혀있는 최신으로.

1. Project Metadata

Group 에 보통 기업 명

Artifact 는 빌드되어 나오는 결과물

1. Dependencies

ADD DEPENDENCIES 클릭해서

WEB - web 검색 후 SPRING WEB (tomcat 내장)

TEMPLETE ENGINES - Thymeleaf (html 관련)

1. GENERATE 클릭하면 압축파일 받아지고, 이걸 압축 풀면됨
2. IntelliJ

open 눌러서 다운받은 폴더 가서 build.gradle 찾아서 이걸 오픈 (Open as Project)

처음 오픈하면 이것저것 다운로드 받을거 (네트워크 연결 필요)

1. 실행

src/main/java 아래에 코드들 있을거.

src/test 는 테스트코드

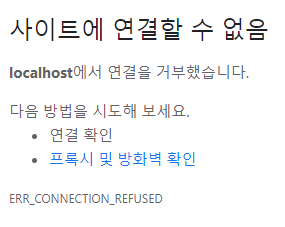
src/main/java 아래에 HelloSpringApplication 에 main 있을거고, 실행하면

~embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)

이런거 나올거고, 인터넷으로 localhost:8080 들어가보면



이런거 나올 것. 아무것도 없으므로 이렇게 나오는게 성공.



이게 실패

1. gradle

build.gradle 에 보면 dependencies 나옴.

# 첫 화면 생성

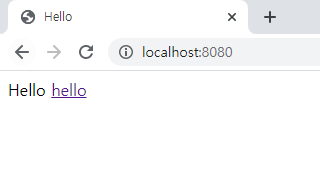
1. Welcome Page

resources/static/index.html 생성

<index.html>

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>Hello</title>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  </head>  <body>  Hello  <a href="/hello">hello</a>  </body>  </html> |

후 실행하면 localhost:8080 에서 아래와 같이 뜰 것.



스프링 부트가 제공하는 welcome page 기능.

resource/static/index.html 을 만들면 이걸 참조해서 뿌려줌.

이러한 기능들 찾아서 할 수 있음. (spring.io 들어가서 project/spring boot/learn 의 reference doc 들어가면 spring boot feature 에서 welcome 또는 index.heml 검색)

1. 템플릿 엔진 (thymeleaf) / controller

controller package 하나 만들고, 아래에 HelloController 생성

|  |
| --- |
| package LWJ.hellospring.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.GetMapping;  @Controller  public class HelloController {  @GetMapping("hello") // localhost:8080/hello 에서의 hello 를 의미함.  public String hello(Model model) {  model.addAttribute("data", "hello!!");  return "hello"; // templates/hello.html 를 찾아감.  }  } |

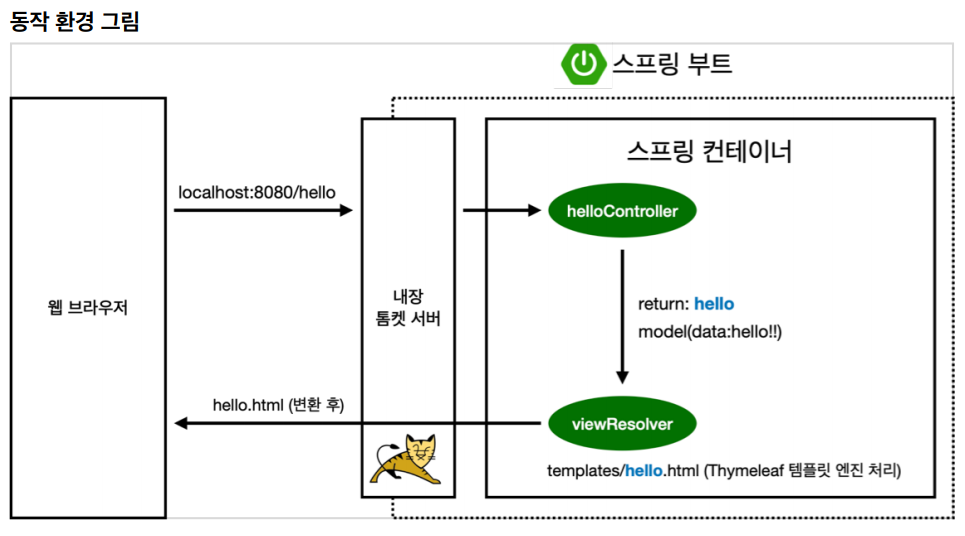
@Controller -> 이거 annotation 적어줘야함

@GetMapping -> 이게 사이트 주소 매핑해주는 역할

@GetMapping(“asdf”) 로 하면 localhost:8080/asdf 와 연결됨

return “hello” 는 templetes/hello.html 을 찾을 수 있도록 함.

<참고 그림>



# 빌드 및 실행

1. jar 배포 및 실행

local 에서 cmd 실행 후 hello-spring 에서

(window) ./gradlew.bat build

(linux) ./gradlew build

실행하면 빌드될거고, build/hello-spring-0.0.1-SNAPSHOT.jar 가 생성됨.

그러면

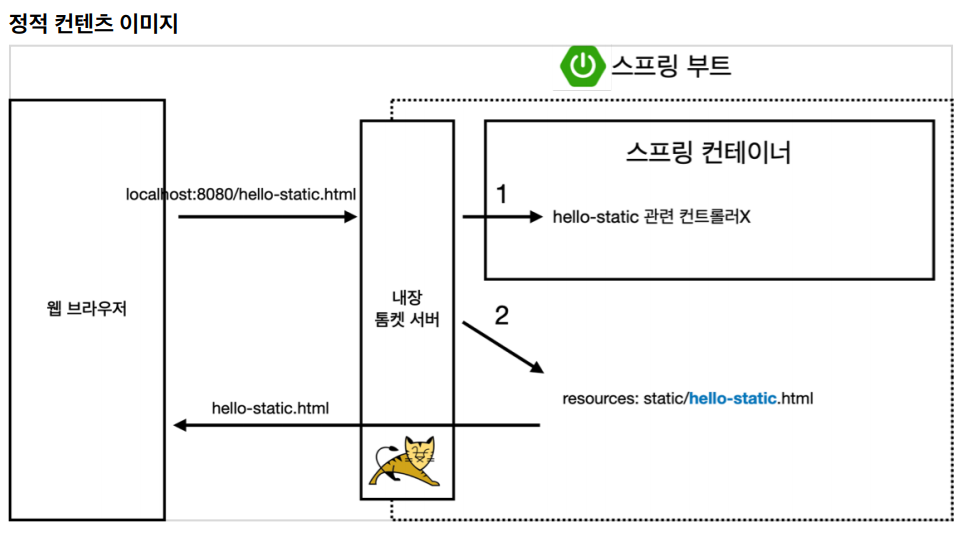
java -jar hello-spring-0.0.1-SNAPSHOT.jar 하면 실행되어서 localhost:8080 가면 페이지 나올 것.

**=> 따라서 배포 시 .jar 파일만 서버에 넣고 서버에서 이 파일 실행시키면 됨.**

# 분류

1. 정적 컨텐츠

파일을 그대로 전달하는 것



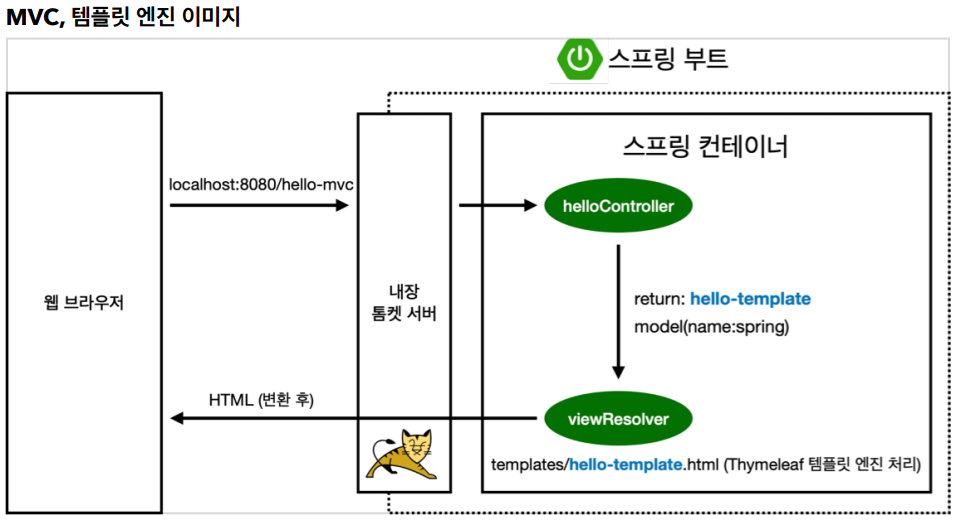
컨트롤러 탐색이 우선순위

없으면 static에서 탐색

1. MVC (Model View Controller)

html 을 뭔가 프로그래밍해서 변경해서 동적으로 전달하는 것

* 1. controller 는 내부 처리에 집중, view 는 화면에 보여지는 것에 집중



1. API

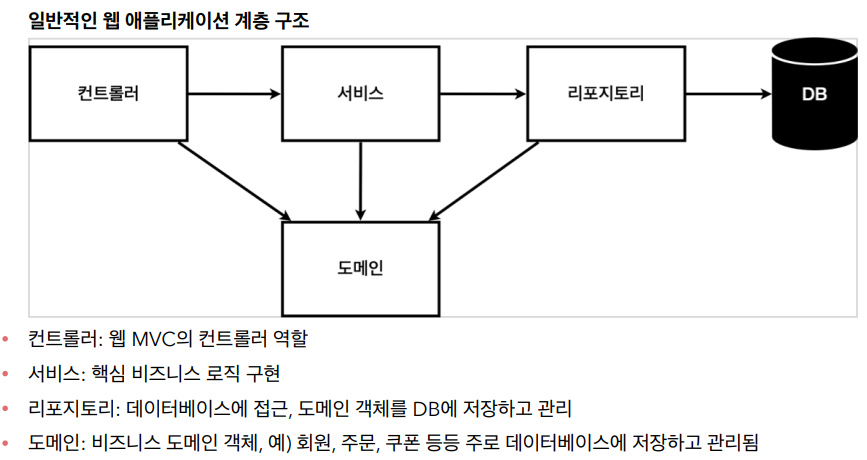
서로 통신하는 것. 어쨌든 data 전달. 이걸 front로 보내서 html을 만들던가 서버로 보내서 뭔가 처리를 하던가 등등.

1. 코드 (정적 컨텐츠 / MVC / API)

|  |
| --- |
| package LWJ.hellospring.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.ResponseBody;  @Controller  public class HelloController {  @GetMapping("hello") // localhost:8080/hello 에서의 hello 를 의미함.  public String hello(Model model) {  model.addAttribute("data", "hello!!");  return "hello"; // templates/hello.html 를 찾아감.  }  @GetMapping("hello-mvc")  public String helloMVC(@RequestParam("name") String name, Model model) {  // RequestParam"name" 으로 받는건 주소에서 들어옴.  // localhost:8080/hello-mvc?name=LWJ => 여기에서 ?name=LWJ 부분.  // 이 name(LWJ) 을 동일하게 value에 담아서 model에 담아서 보냄  model.addAttribute("name", name);  return "hello-template";  }  @GetMapping("hello-string")  @ResponseBody // html에 body 부분에 이걸 직접 넣어주겠다는 의미  public String helloString(@RequestParam("name") String name) {  return "hello " + name;  // 이건 view 이런게 없음. 그냥 문자가 그대로 내려감.  // 그래서 소스보기 하면 html 같은거 없이 진짜 그냥 그대로 내려감. (소스보기 하면 그냥 hello LWJ 로 나올 것)  }  @GetMapping("hello-api")  @ResponseBody // 객체를 반환하면 기본으로 json으로 반환함.  public Hello helloApi(@RequestParam("name") String name) {  Hello hello = new Hello();  hello.setName(name);  return hello;  }  static class Hello {  private String name;  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  }  } |

# 회원 관리 예제

1. 구현



<https://github.com/wjlees/LWJ/tree/main/SPRING/hello-spring> (구현 및 테스트코드)

<https://github.com/wjlees/LWJ/commit/5f84c9ec3e4d7a3741a2ce4e363e3280e0b93d06>

Optional 은 null 일 가능성이 있을 때 Optional 로 감싸주기 위해서 붙임

Optional<Member> result = memberRepository.findByName(member.getName());

이렇게 하면 결과값이 null 일 수 있음.

이게 결과값을 Optional 로 한번 감싸주는게 되는데, 그럼으로써 Optional result 의 많은 함수들을 사용할 수 있게 된다.

if (result != null) {~~} 이 아니라, result.ifPresent( m->{~~}); 로 사용 가능해진다.

result.get() 으로 꺼내는건 optional 으로 감싸져있는걸 꺼내는 것.

최종적으로는

memberRepository.findByName(member.getName())

.ifPresent( m -> {

throw new IllegalStateException(“~!”);

});

로 한다.

1. 테스트 코드 작성

테스트 코드는 src/test 아래에 작성하고, naming 규칙은 패키지의 경우 동일하게, 클래스의 경우 동일 + Test 를 붙인다.

보통 assertThat 을 이용하여 확인한다.

Assertions.assertThat(result).isEqualTo(member1);

같으면 그냥 넘어갈거고 다르면 뭔가를 띄울 것.

[ Ctrl + Shift + t ] : 해당 코드에 맞는 테스트 코드 틀 자동 생성

//given

//when

//then

이 주석부터 달고, 이거 기반으로 채우면 보기 쉬움

1. Throw Exception의 Test

IllegalStateException e = assertThrows(IllegalStateException.class, () -> memberService.join(member2));

Assertions.assertThat(e.getMessage()).isEqualTo("이미 존재하는 회원입니다.");