**프로젝트 환경설정**

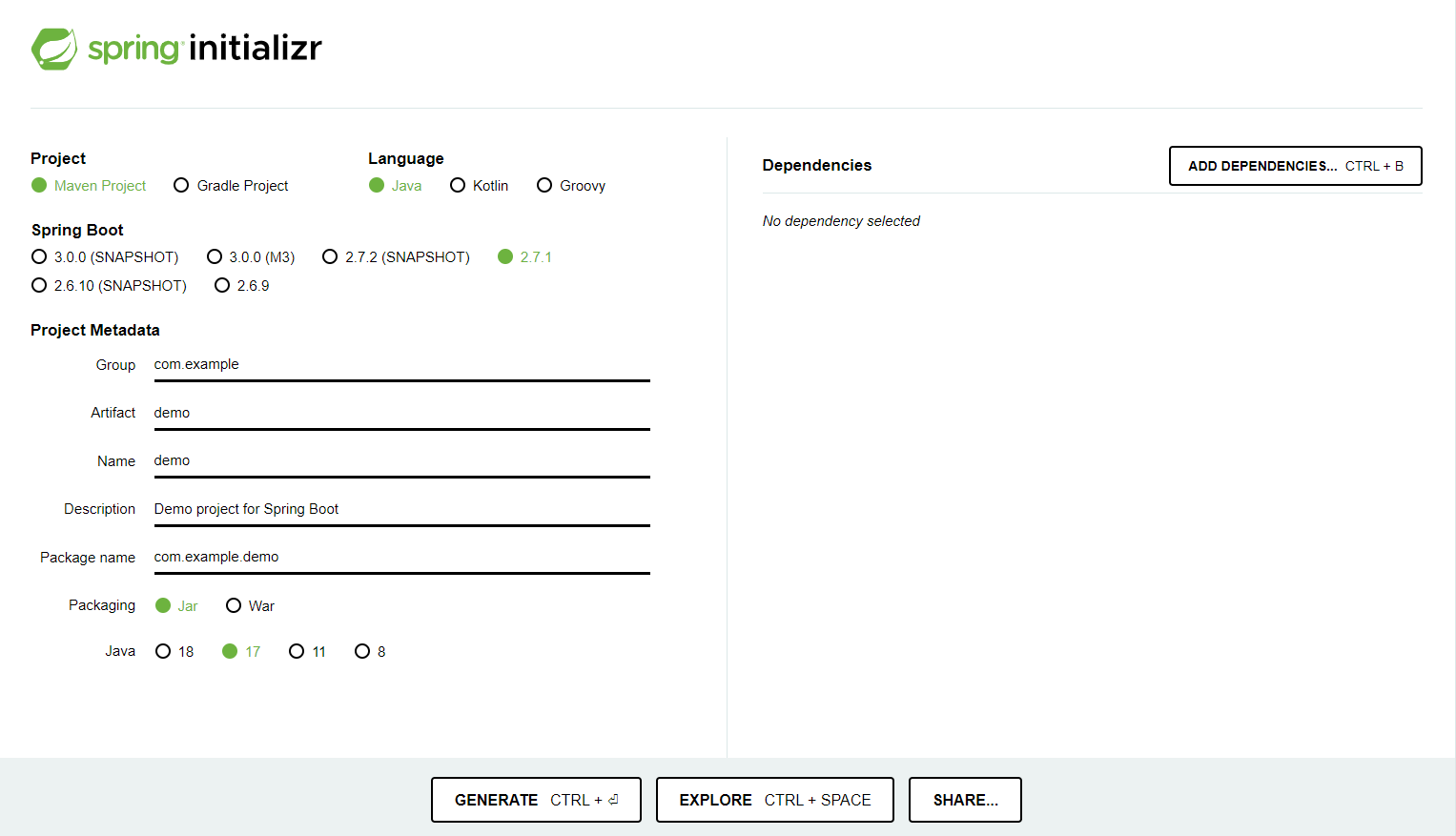
**📒 프로젝트 생성**

프로젝트를 생성하기 앞서 **Java 17**과 **IntelliJ** 설치해야 한다.

**🖋️ 스프링 부트 스타터**

다 설치했다면, 이제 본격적으로 시작하기 위해 [**스프링 부트 스타터 사이트**](https://start.spring.io/)를 사용해 **스프링 프로젝트**를 생성해야 된다.

[**스프링 부트 스타터 사이트**](https://start.spring.io/)는 스프링 부트 기반으로 스프링 관련 프로젝트를 만들어 주는 사이트이며, 해당 사이트에 들어가면 아래 사진과 같은 화면이 나온다.



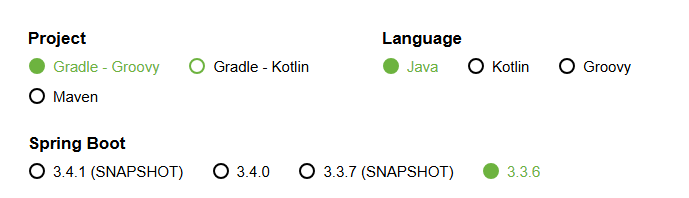
**🖋️ 스프링 부트 스타터 설정**

[**스프링 부트 스타터 사이트**](https://start.spring.io/)에서 **Project**는 **Gradle Project**로 선택한다.



Maven과 Gradle은 필요한 라이브러리를 땡겨와서 빌드하는 라이프 사이클까지 관리해주는 Tool이며, 과거에는 Maven을 자주 사용하였지만 요즘에는 주로 **Gradle**를 사용하므로 **Gradle**를 선택한다.

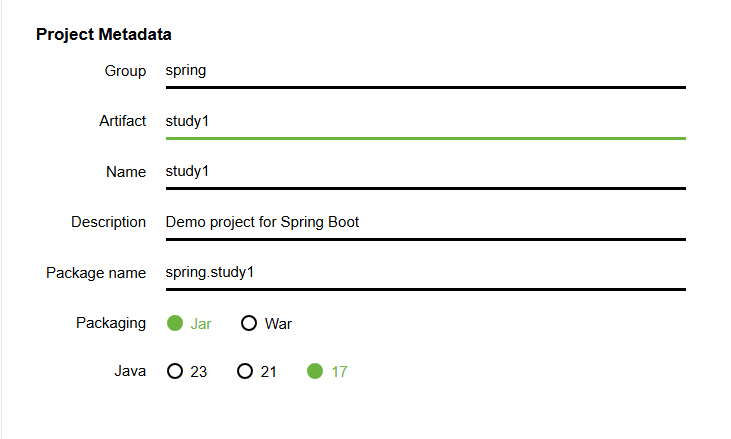
**Language**는 **Java**를 선택해준다.  

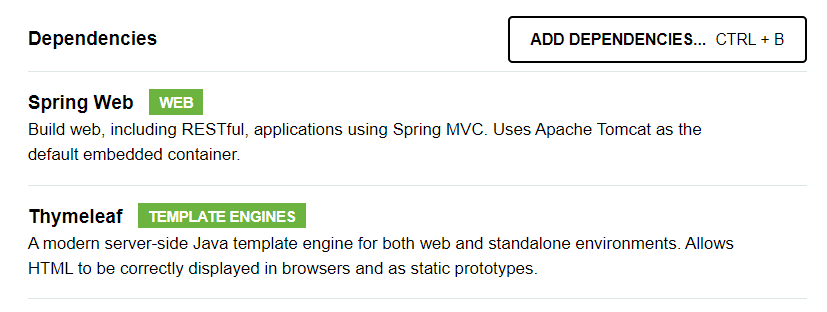

**Spring Boot**는 **2.7.1**을 선택해준다.  


여기서 SNAPSHOT은 아직 만들고 있는 버전이며, M3는 정식으로 릴리즈 되지 않은 버전이므로, 정식 버전 중에 가장 버전이 높은 것을 선택해야 한다.

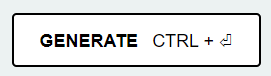
**Project Metadata**의 내용은 아래 사진과 같이 작성할 것이다.

**Group**은 보통 기업명을 쓰는데 지금은 아무 단어를 쓸 것이고, **Artifact**는 빌드되어 나올 때 결과물 이름으로 즉 프로젝트 명이다.  
**Name**, **Description**, **Package name**은 위에 작성한 내용대로 바뀐 상태 그대로 두며, **Packaging**은 **Jar**, **Java**는 **17**을 선택해준다.



**Dependencies**은 **Spring Web**와 **Thymeleaf**를 선택한다.  


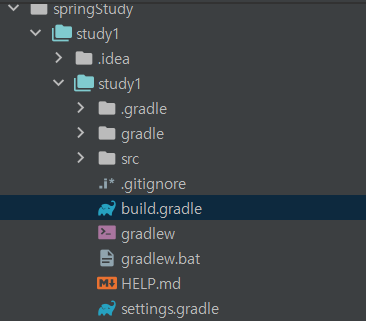
**Dependencies**은 어떤 라이브러리를 땡겨서 사용할 것인가에 대한 것을 의미하며,  
spring 기반의 web 프로젝트를 만들기 위해 **Spring Web**, html을 만들어주는 템플릿 엔진인 **Thymeleaf** 라이브러리를 선택해준다.

설정을 다 한 후에 **GENERATE** 버튼을 누르면 프로젝트가 다운 받아지며, 다운 받은 프로젝트를 원하는 폴더에 압축을 풀어 놓는다.  


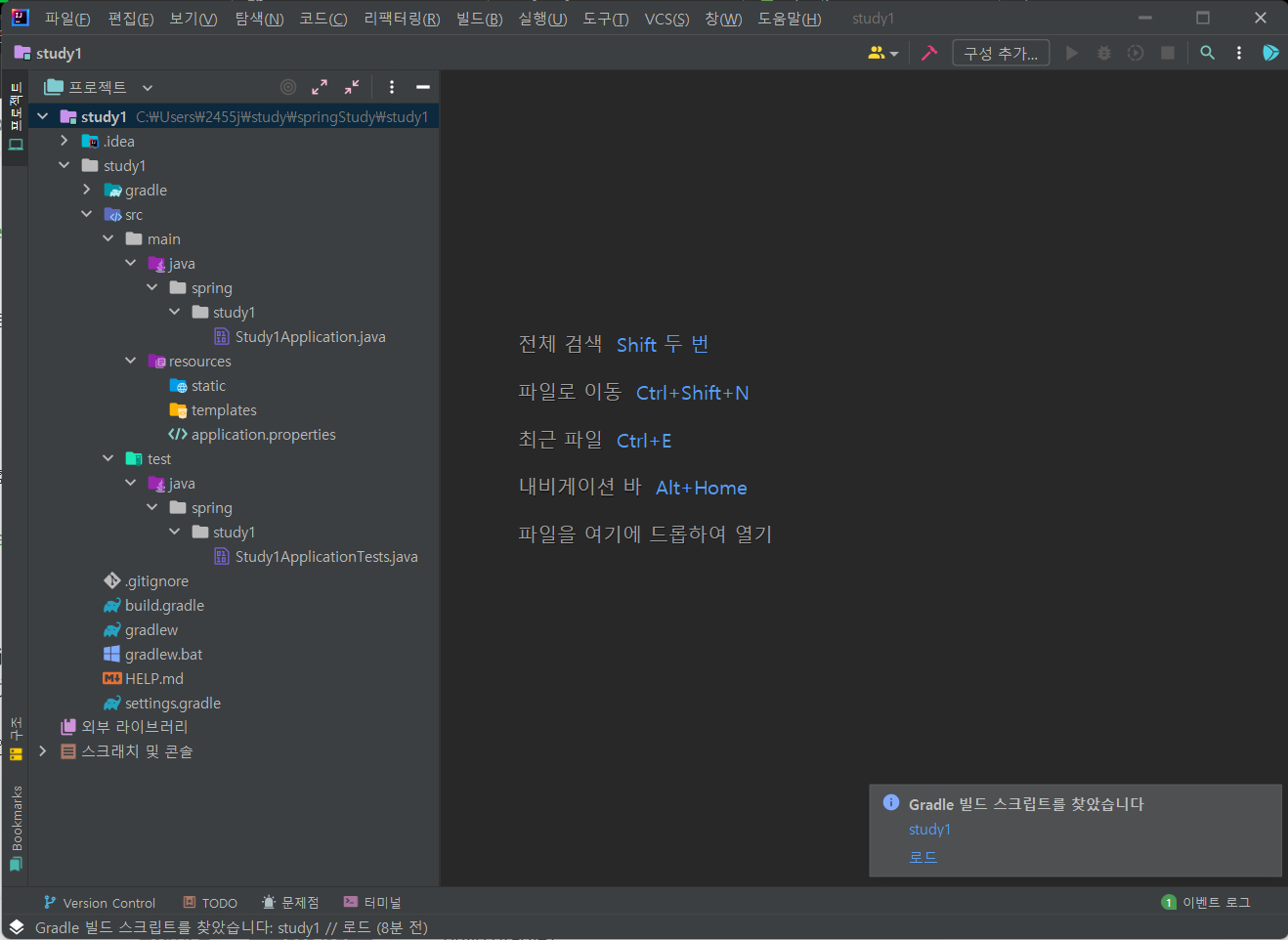
C:\study1에 붙여넣는다.

**🖋️ 프로젝트 설명**

프로젝트를 실행하기 위해 **IntelliJ**를 실행하여 압축된 폴더의 **build.gradle** 파일을 선택하여 프로젝트를 연다.



프로젝트를 열면 아래 화면처럼 나오는데 **src**폴더를 열면 안에 **main**과 **test**폴더가 있다.  
**main**폴더 안에는 **java**와 **resources**폴더가 있고, **java**폴더 안에는 실제 소스파일과 패키지가 들어있으며, **resources**폴더 안에는 실제 java코드 파일을 제외한 설정파일 등이 들어있다.  
**test**폴더 안에는 test코드와 관련된 파일들이 들어있으며 개발을 할 때 매우 중요하다.



**build.gradle**에는 간단하게 설명하자면 버전을 설정하고 라이브러리를 땡겨오는 역할을 하는 내용들의 코드가 적혀있다.

plugins {

id 'org.springframework.boot' version '3.3.6'

id 'io.spring.dependency-management' version '1.0.11.RELEASE'

id 'java'

}

group = 'spring'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '17'

repositories { // mavenCentral에서 아래 라이브러리들을 다운받음

mavenCentral()

}

dependencies { // 라이브러리

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

implementation 'org.springframework.boot:spring-boot-starter-web'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

}

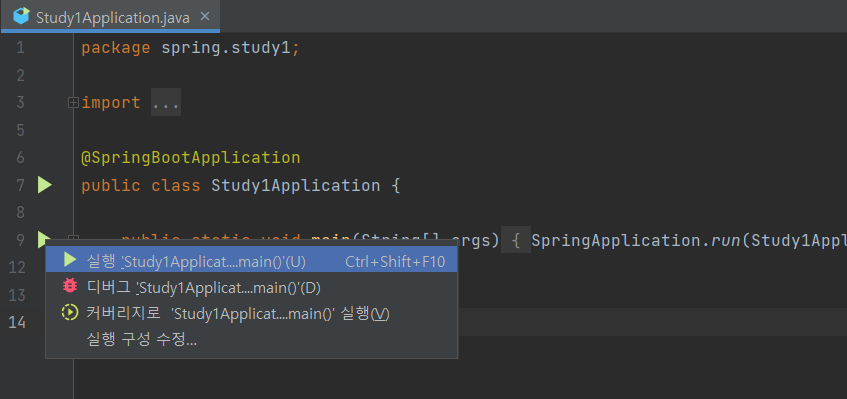
tasks.named('test') {

useJUnitPlatform()

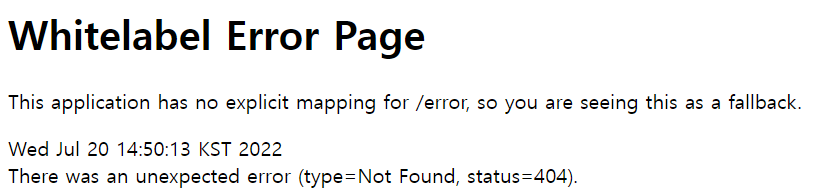
}

**🖋️ 프로젝트 실행**

본격적으로 프로젝트를 실행하기 위해 main/java/spring/study1/**Study1Application.java** 파일을 선택하여 실행을 하면 **spring boot application**이 실행된다.



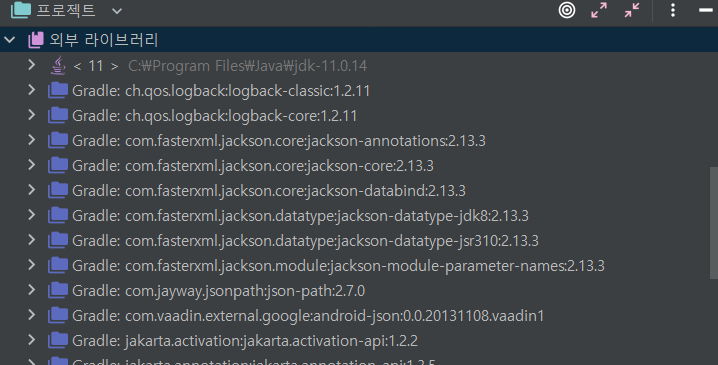
실행을 한 후 [http://localhost:8080](http://localhost:8080/)에 들어가서 아래 화면처럼 뜨면 프로젝트로 정상적으로 실행이 된 것이다.



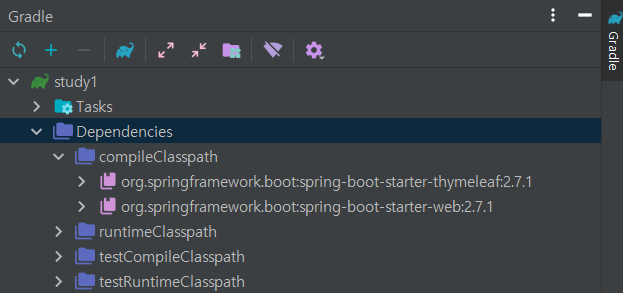
**📒 라이브러리 살펴보기**

**🖋️ 라이브러리**

프로젝트를 만들면서 **라이브러리**로 Spring Web, Thymeleaf 2개밖에 가져오지 않았지만 실제로는 매우 많은 라이브러리를 가져오게된다.  
이러한 이유는 라이브러리들이 **의존관계**로 엮여있어 한 개의 라이브러리를 사용할 때 필요한 라이브러리들을 자동적으로 가져오기 때문이다.

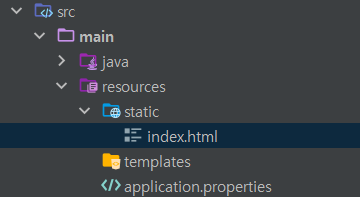


라이브러리들의 의존관계는 화면 오른쪽 상단에 있는 **Gradle**을 누르면 볼 수 있다.



**📒 View 환경설정**

**🖋️ 정적 page 만들기**

[http://localhost:8080](http://localhost:8080/)에 들어가면 아무런 페이지가 만들어 있지 않아 error 페이지가 뜬다.  
그래서 페이지를 직접 만들어주기 위해 src/main/resources/static 폴더에다가 **index.html** 파일을 새로 만들어 준다. 그러면 이 **index.html**이 첫 화면 페이지가 된다.  


**index.html**에 아래와 같은 html 코드를 작성한다.

<!DOCTYPE HTML>

<html>

<head>

<title>Hello</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

</head>

<body>

<h1>Hello</h1>

<a href="/hello">hello</a>

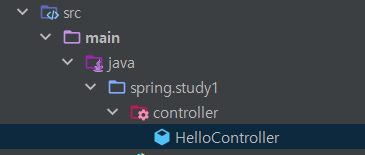
</body>

</html>

작성 후에 다시 프로젝트를 실행하여 [http://localhost:8080](http://localhost:8080/)에 들어가면 아래와 같은 화면이 뜨는 것을 확인할 수 있다.  


**🖋️ 동적 page 만들기 - thymeleaf 템플릿 엔진**

정적인 page가 아니고 동작하고 프로그래밍 되는 page를 만들어 볼 것이다.  
그래서 src/main/java/spring.study1 폴더에 **controller** 라는 **패키지**를 만들고 그 안에 **HelloController** 라는 java class 파일을 만들어준다.



**HelloController**에 아래와 같은 코드를 작성한다.

package spring.study1.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

@Controller // spring framework 를 사용하기 위해

public class HelloController {

@GetMapping("hello") // 웹 어플리케이션에서 /hello로 들어오면 해당 메서드 호출

public String hello(Model model) {

model.addAttribute("data","hello!!");

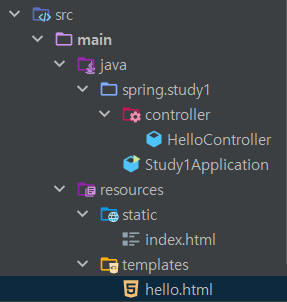
// attribute 의 이름이 "data"인 곳에 value 로 "hello!!" 가 들어감

return "hello";

}

}

그리고 src/main/resources/template 폴더에다가 **hello.html** 파일을 새로 만들어 준다.



hello.html에 아래와 같은 코드를 작성하며, 위에서 만든 **HelloController**라는 컨트롤러가 **hello.html** 파일을 controller 한다.

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org"> <!--템플릿 엔진으로 thymeleaf 사용할 수 있게 함-->

<head>

<title>Hello</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

</head>

<body>

<p th:text="'안녕하세요. ' + ${data}" >안녕하세요. 손님</p>

<!--th: thymeleaf, HelloController의 value 값이 data에 들어감 -->

</body>

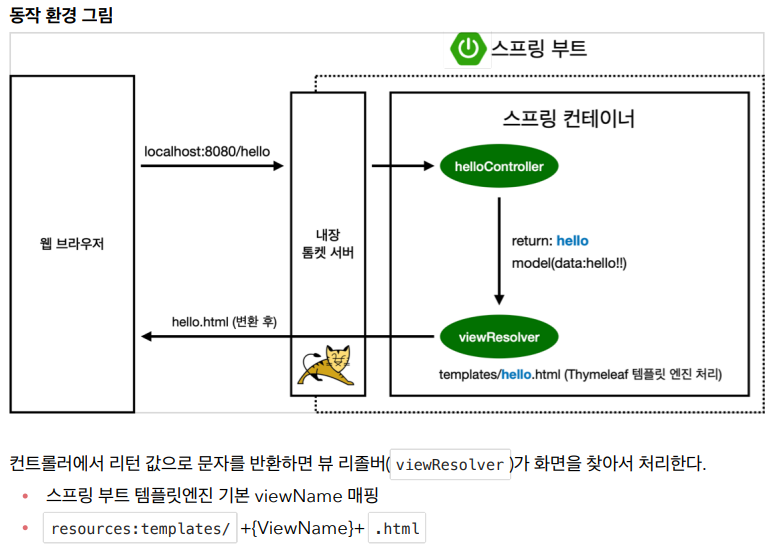
</html>

작성 후에 다시 프로젝트를 실행하여 <http://localhost:8080/hello>에 들어가면 아래와 같이 ${data} 자리에 hello!! 가 들어간 것을 확인할 수 있다.



**🖋️ 동작 원리**

위에 실행한 프로젝트처럼 작동하는 원리는 아래 사진에 간단하게 설명되어 있다.



웹 브라우저에서 <http://localhost:8080/hello> 에 들어가면 스프링부트는 **톰켓**이라는 웹서버를 내장하고 있다.  
**/hello** 페이지는 **get** 방식으로 들어왔기 때문에 **@GetMapping("hello")** 코드를 가지고 있는 **HelloController**를 **톰켓**을 통해 부르며, 이 컨트롤러 안에 있는 메서드가 실행된다.  
해당 메서드에서 **key**는 **"data"**, **value**는 **"hello!!"** 라는 **model**을 만들고 **return "hello";** 를 통해 값이 들어간 **model**을 가지고 **hello.html** 의 화면을 **렌더링** 시켜준다.