

# SPRING boot

## 웹프로그래밍

발행일 : 2021.01.01

변경일 : 2021.09.23 (2본 시작일)

저자 : 박수민

이메일 : [foxman12@hanmail.net](mailto:foxman12@hanmail.net)

발행처 : 휴먼교육

# 목차

---

<b>01. spring boot</b>	<b>3</b>
> 01. 프로그램 설치	3
> 02. 프로젝트 생성하기	7
> 02. static 폴더 helloworld 출력	15
> 03. 데이터 전송방식 get과 post	17
> 04. MVC 패턴 사용하기	18
> 05. 사용자 입력을 view에 보내주기	22
> 06. 객체형태 데이터처리하기	25
> 07. Mustache arrayList 출력하기	29
> 08. mustache 분할코드 화면을 만들어 보자.	30
> 09. JPA를 이용한 crud작업	31
> 10. 오라클로 변경하기	44
> 11.	44
> 12.	44

# 01. spring boot

## > 01. 프로그램 설치

-jdk 설치

windows + r 컨솔창에 java를 입력해보고 패스가 잡혀 있지 않다면 openjdk를 설치하여 패스를 잡아보자.

이미 설치되어 있는 java의 패스를 잡고 싶다면 웹에 검색해서 잡아보자.

설치와 함께 패스 잡는법은 다음과 같다.

1. [www.adoptium.net](https://www.adoptium.net) 사이트에 들어간다.



## Prebuilt OpenJDK Binaries for Free!

Java™ is the world's leading programming language and platform. The Adoptium Working Group promotes and supports high-quality, TCK certified runtimes and associated technology for use across the Java ecosystem. Eclipse Temurin is the name of the OpenJDK distribution from Adoptium.

Download Temurin™ for Windows x64



2. 중간에 체크되어 있는 other platforms and versions를 선택한다.

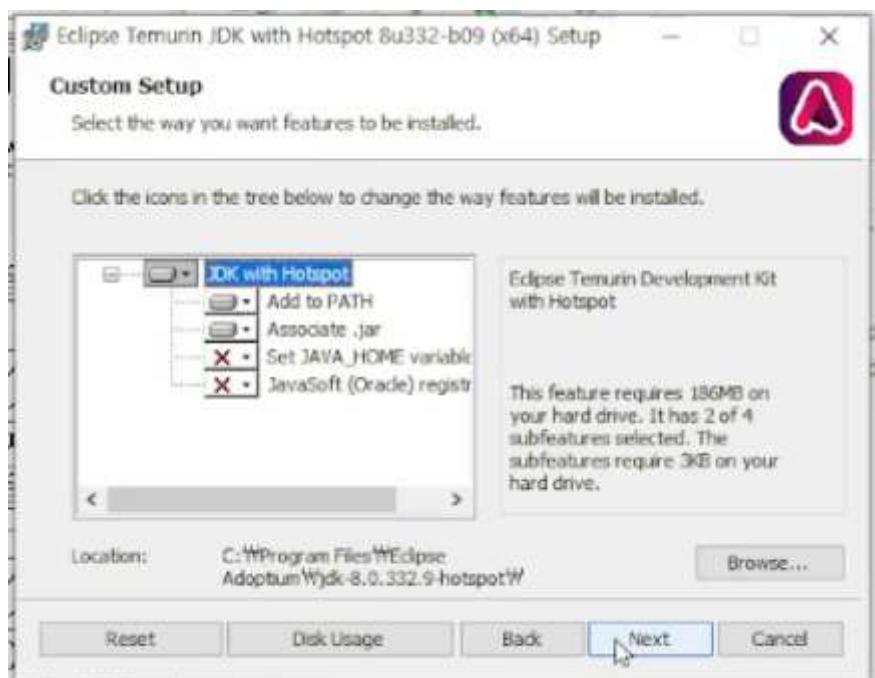
3. 다음 이미지 처럼 windows x64 jdk 8를 선택한다.

Use the drop-down boxes below to filter the list of current releases.

Operating System	Architecture	Package Type	Version
Windows	x64	JDK	8
<b>jdk8u332-b09</b> Temurin <input checked="" type="checkbox"/> <input type="radio"/> May 5, 2022		Windows	x64
		JDK - 90 MB <a href="#">.msi</a> <a href="#">Checksum</a>	<a href="#">.msi</a>
		JDK - 104 MB <a href="#">.zip</a> <a href="#">Checksum</a>	<a href="#">.zip</a>

4..msi 버튼을 클릭하여 선택된 버전을 다운로드 한다.

5. 기본값으로 설정하고 설치 한다.



## 인텔리제이 설치하기

1. <https://www.jetbrains.com/ko-kr> 사이트에서 개발자 도구 intelliJ를 선택한다.

IDE	플러그인 및 전체 플러그인
AppCode	IntelliJ IDEA
CLion	PhpStorm
DataGrip	PyCharm
DataSpell	Rider
Fleet	RubyMine
GoLand	WebStorm
IntelliJ IDEA	Space
PhpStorm	AppCode
PyCharm	CLion
Rider	DataGrip
RubyMine	DataSpell
WebStorm	Fleet

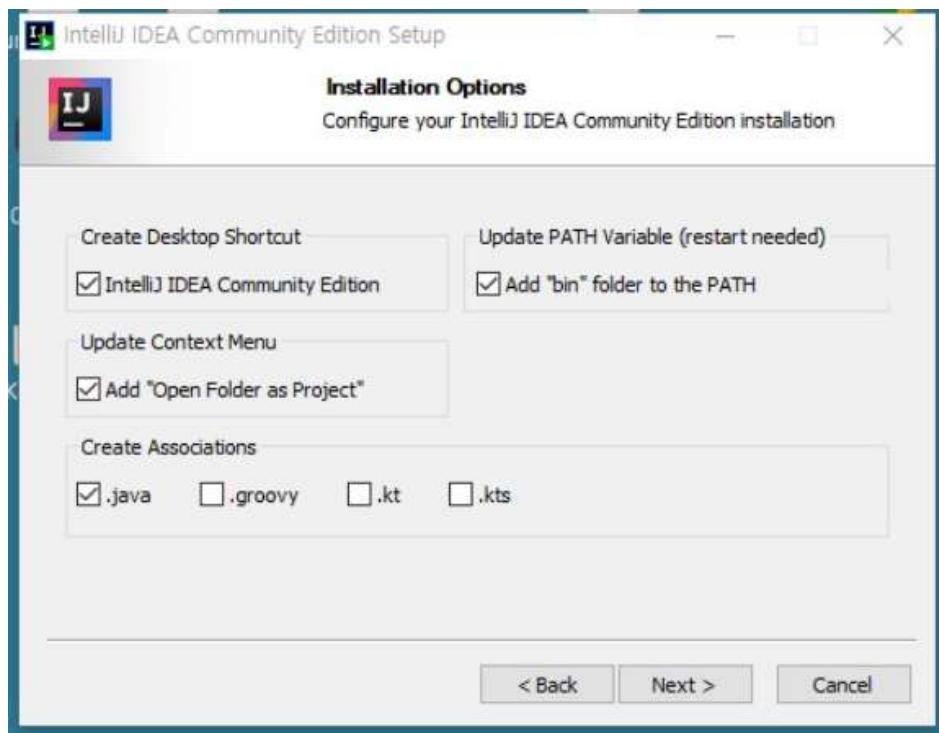
2. 가운데에 다운로드 버튼을 누른다.



3. community 버전을 다운로드한다.

The image displays the download page for the IntelliJ IDEA Community edition. On the left, there's a large logo consisting of overlapping colored shapes (red, blue, yellow) forming a hexagonal pattern with the letters 'IJ' in the center. To the right of the logo, the text '다운로드 IntelliJ IDEA' is displayed above three download links: 'Windows', 'macOS', and 'Linux'. The 'Windows' link is underlined with a blue line, indicating it is the active or selected version. Below these links are two sections: 'Ultimate' and 'Community'. The 'Ultimate' section is for '웹 및 엔터프라이즈 개발용' (Web and Enterprise development) and includes a '다운로드' button and a note about a 30-day trial. The 'Community' section is for 'JVM 및 Android 개발용' (JVM and Android development) and includes a '다운로드' button and a note about being '무료, 오픈 소스로 빌드됨' (Free, built with open source).

4. 다음과 같이 설정하고 설치를 이어 나가자.



## > 02. 프로젝트 생성하기

<https://start.spring.io/>에 접속해서 다음과 같이 프로젝트를 추가한다.

만들고자 하는 프로젝트를 해당 사이트에서 생성해서 인텔리제이에서 불러오면 프로젝트 세팅이 가능하다.

다음과 같이 설정을 한다음 하단의 generate 버튼을 클릭하면 설정된 신규 프로젝트 환경이 세팅되어 다운로드 된다.

1. gradle, java, 2.7.2 을 선택한 다음 project artifact와 name를 demo에서 sample로 변경하고, jar 8 버전을 선택하였다.

The screenshot shows the Spring Initializr interface. On the left, under 'Project', 'Gradle Project' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '2.7.2' is selected. In the 'Project Metadata' section, 'Group' is set to 'com.example', 'Artifact' to 'sample', 'Name' to 'sample', 'Description' to 'Demo project for Spring Boot', and 'Package name' to 'com.example.sample'. Under 'Packaging', 'Jar' is selected. Under 'Java', '8' is selected. On the right, the 'Dependencies' section is expanded, showing several optional add-ons: 'Spring Boot DevTools' (selected), 'Lombok', 'Spring Web' (selected), 'Thymeleaf', 'Mustache', 'Spring Data JPA', and 'H2 Database'. At the bottom, there are social sharing icons (Facebook, Twitter) and three buttons: 'GENERATE' (with a red checkmark), 'EXPLORE', and 'SHARE...'.

2. 왼쪽 상단 add dependencies 버튼을 눌러서 다음 항목들을 추가하였다.

### Spring Boot DevTools

기본적을 개발할때 추가적으로 도움을 주는 의존객체 어플리케이션 리스트를 빠르게하고 라이브 리로드 등을 지원

### Lombok

setter, getter, 생성자, toString 등 자동 생성해주는 의존객체

### spring web

웹 제작, restful, spring mvc, 아파치 톰캣을 포함하는 의존객체

`thymeleaf, mustache`

jstl과 같은 일을 한다. 동적으로 서버페이지 작성하는 스크립트

`spring data jpa`

데이터베이스를 객체형태로 crud 처리하는 jpa 의존객체

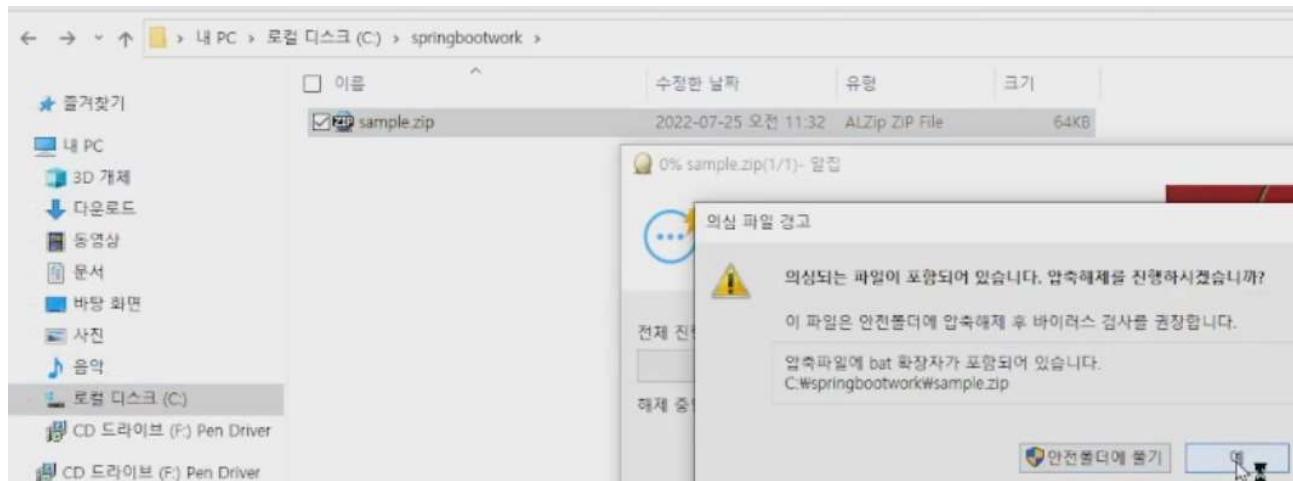
`h2 database`

spring boot에서 메인으로 제공해주는 database 특징은 메모리에 생성되어서 프로그램이 종료되면 사라진다.

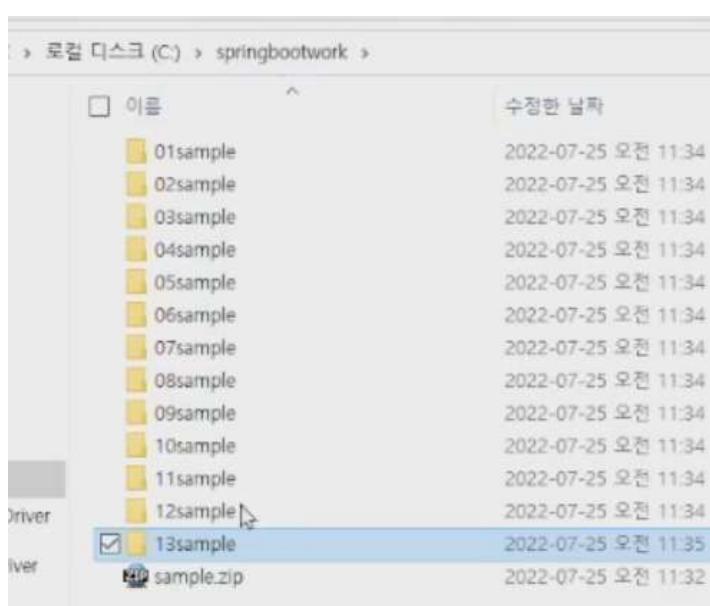
`oracle driver`

오라클을 사용할수 있는 jdbc 의존객체

3. 하단의 generate버튼을 클릭하여 신규 프로젝트 작성 파일을 다운로드 받은 다음 c폴더에 springbootwork폴더를 만들고 해당 파일을 복사한 다음에 해당 폴더에 압축을 풀자.

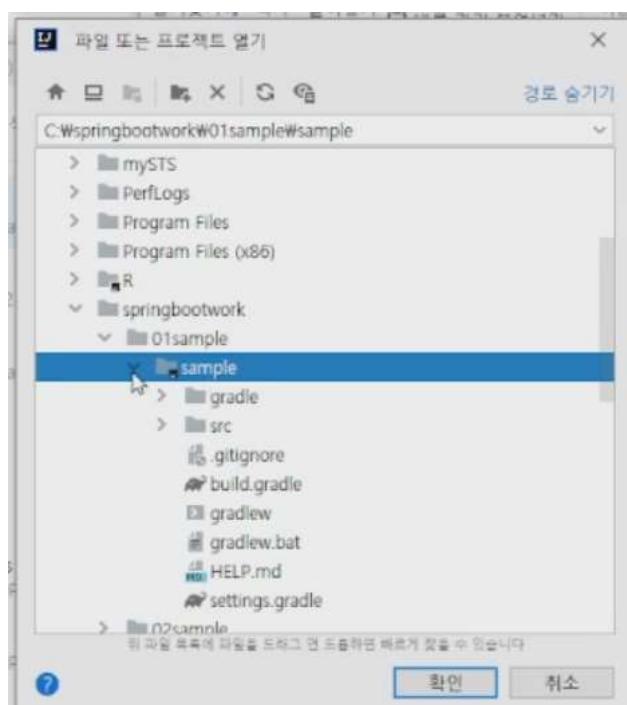
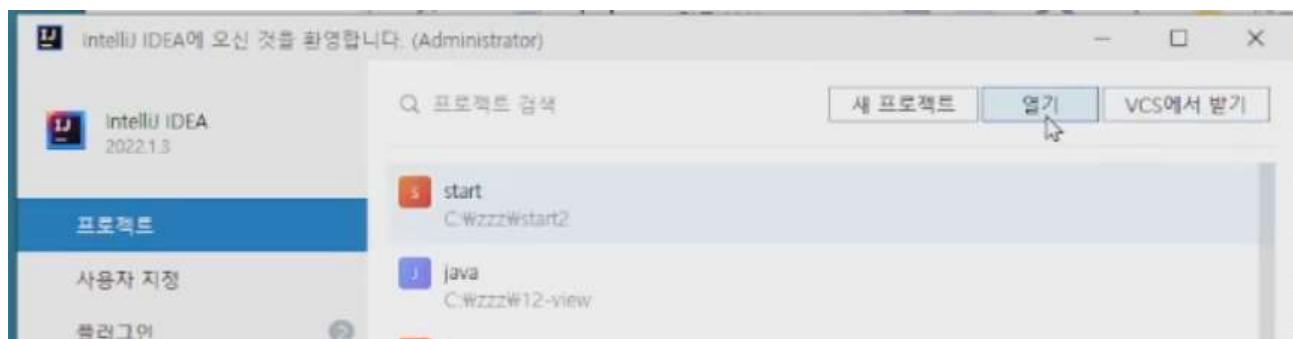


4. 같은 형태의 프로젝트를 여러개 만들 예정 이어서 압축을 푼 파일을 여러개 복사하여 폴더 이름을 변경 하여 추가해 보자.



5. 인텔리제이 아이콘에서 오른쪽 클릭해서 관리자 권한으로 실행한다.

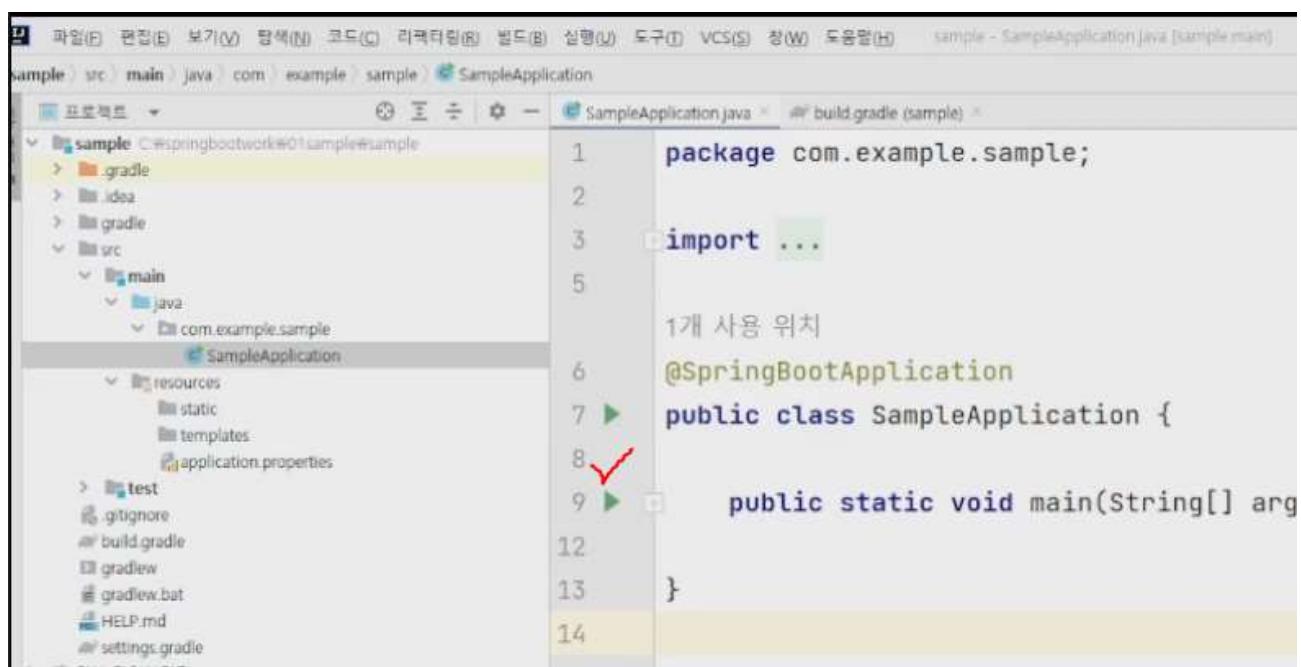
6. 열기 버튼을 클릭해서 springbootwork폴더에 01sample 폴더를 선택한다.



7. 한참을 기다리면 다운로드가 끝난다.

8. 다운로드가 완료되면 아래 이미지처럼 SampleApplication파일을 선택하여 화살표가 존재하는지 확인해 보자.

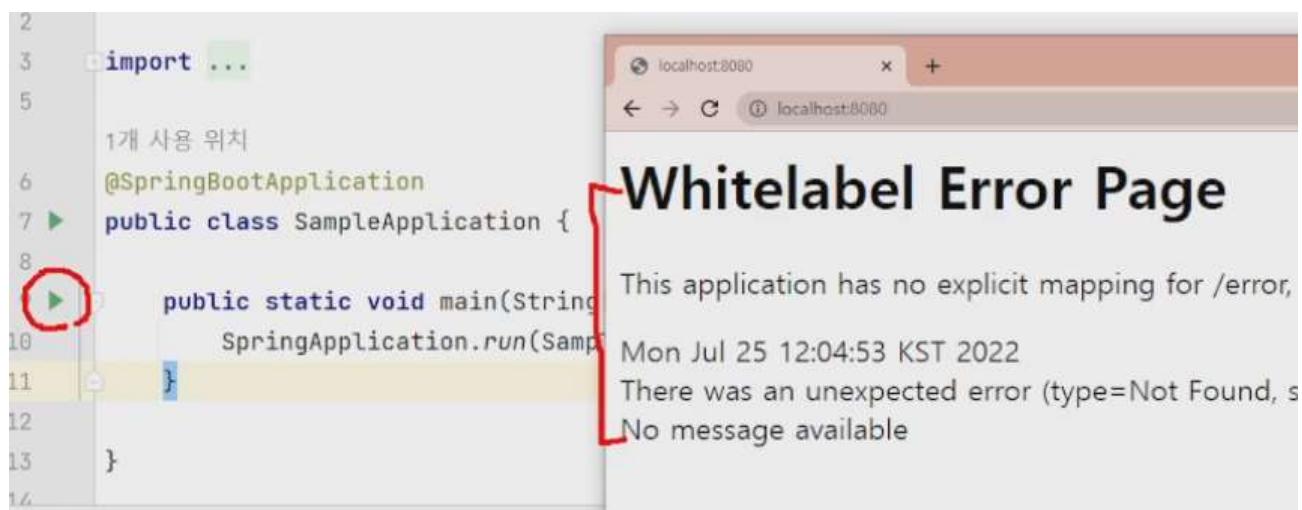
9. 폴더구조와 프로젝트를 확인하자.



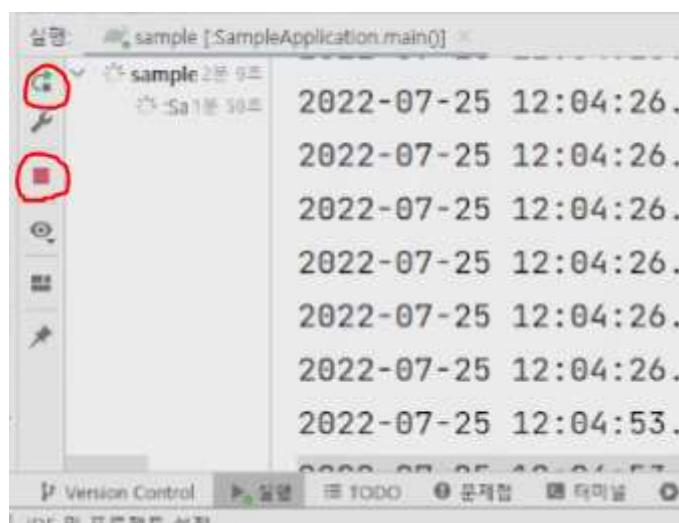
10. 화살표가 존재하지 않으면 문제가 발생한 상태이다. 주의 알림 표시를 잘 찾아보면 에러를 확인할 수 있다. 아래이미지 처럼 main부분을 클릭하고 전구모양을 클릭해서 자바를 설치하던지 버전을 올바르게 선택하면 문제가 해결된다.

```
@SpringBootApplication  
public class SmampleApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(SmampleApplication.class, args);  
    }  
}
```

11. 아래도 문제가 해결되지 안으면 컴퓨터를 리부팅, 인텔리제이를 재설치, 윈도우즈를 밀고 다시 설치하는 등 다양한 방법이 필요하다.  
설치가 완료되면 상단 9번줄 체크되어 있는 화살표를 클릭해서 실행해 보자.

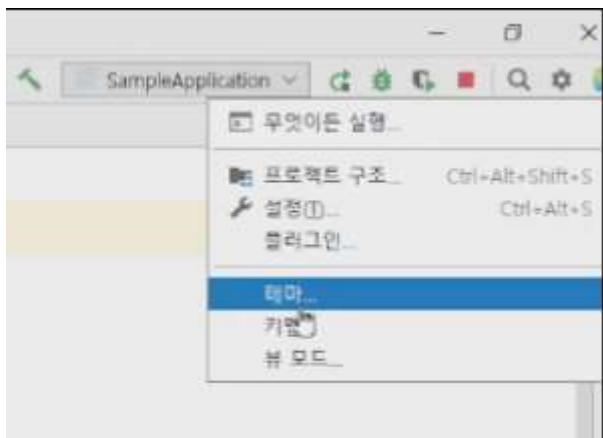


화살표를 누르고 브라우저를 실행시킨 다음 <https://localhost:8080> 상위 이미지처럼 메시지가 나오면 정상 동작한 것이다.



spring boot는 톰캣 같은 웹서버가 내장되어 있다. 코드를 수정하고 적용하려면 웹서버를 재시작 해야 적용된다. 왼쪽의 첫번째 동그라미안의 아이콘을 클릭하면 웹서버가 재시작 된다.

두번째 동그라미를 클릭하면 웹서버를 멈출수 있다.

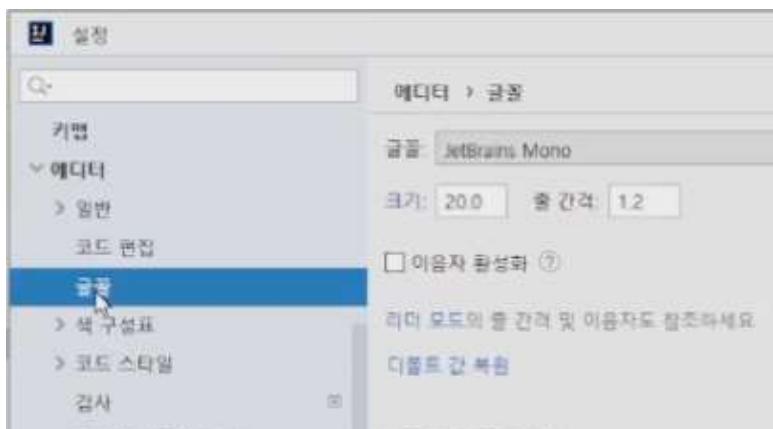


테마를 사용해서 에디터의 색상을 변경하기 싶으면 왼쪽 상단 텁니바퀴 모양의 아이콘을 클릭하고 테마를 선택한다음 원하는 색상을 선택하면 된다.

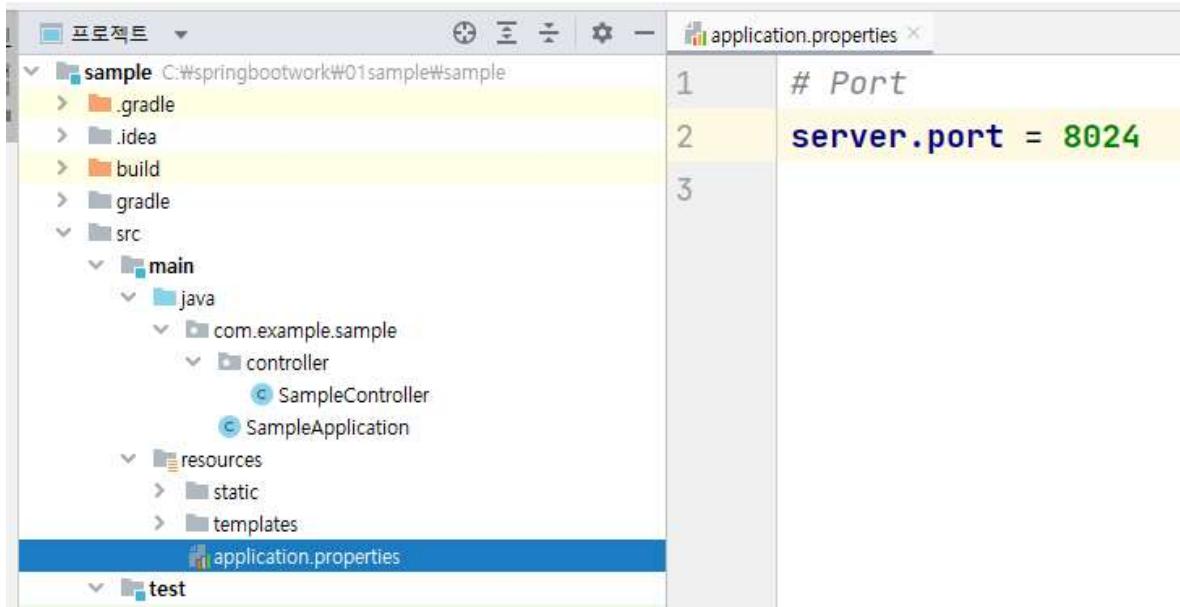
현재 사용하고 있는 테마는 windows 10 light이다.



파일의 설정에 폰트를 변경할 수 있다. 아래 화면이 뜨면 에디터의 글꼴을 선택해서 폰트 크기를 변경할 수 있다. 글꼴은 기본적으로 제공하는 글꼴을 사용하는 것이 좋다.



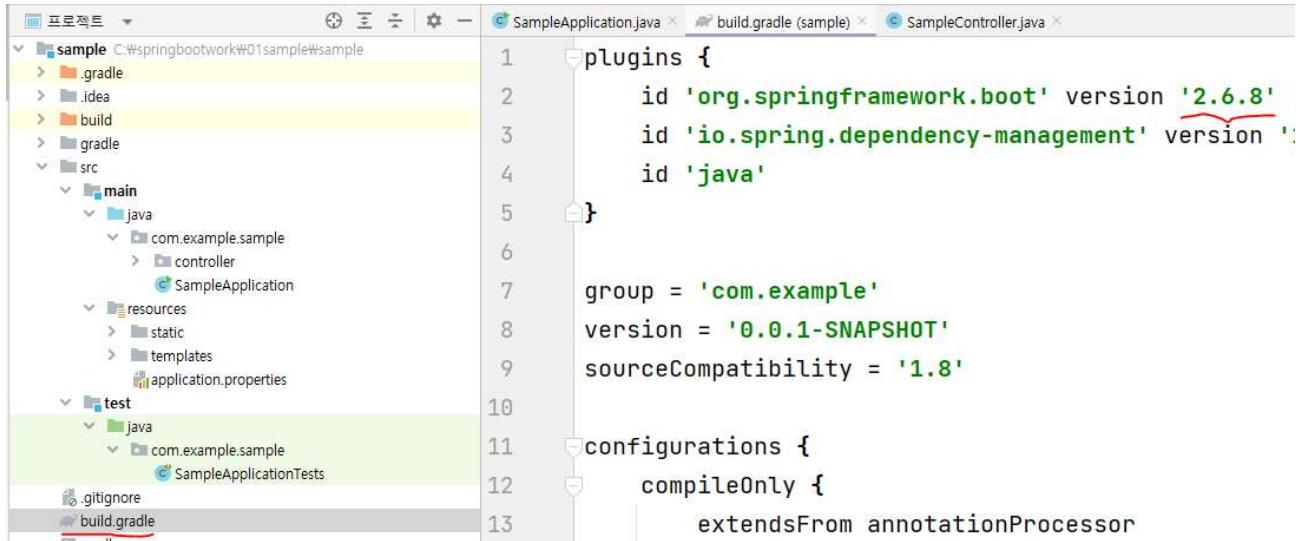
application.properties 파일에서 프로젝트관련 설정을 할 수 있다.



# 포트를 변경하고 싶다면 다음과 같이 기술하면 된다.

# Port

```
server.port = 8024
```



build.gradle 파일에서 현재 프로젝트 세팅 상태를 확인할 수 있다.

mustache에서 한글이 깨지는 현상이 나타나서 spring boot version을 2.6.8로 낮췄다.

```

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-mustache'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.h2database:h2'
    runtimeOnly 'com.oracle.database.jdbc:ojdbc8'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

```

파일 아래부분에 상위와 같이 설치되어 있는 의존객체들이 나열되어 있다. 기존 아래 코드에서 mustache와 thymeleaf가 같은 일을 해서 둘다 설치되어 있으면 설정에 문제가 생겨 thymeleaf관련 의존객체를 삭제했다.

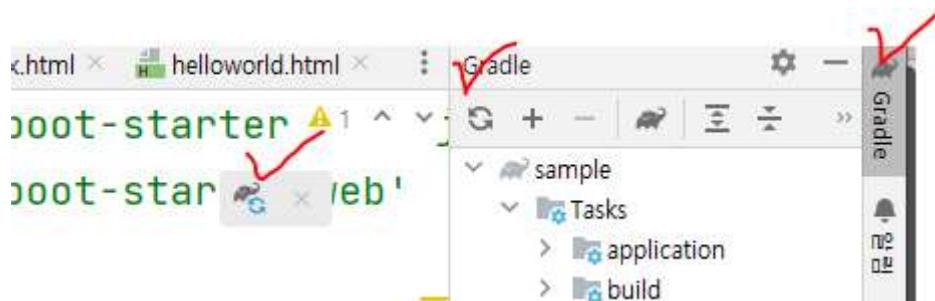
```

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-mustache'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.h2database:h2'
    runtimeOnly 'com.oracle.database.jdbc:ojdbc8'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

```

특정 기능을 의존 객체로 추가 삭제하고 싶다면 관련 단어가 기술되어 있는 줄들을 추가 삭제하면 된다.

build.gradle이 변경되면 아래 체크되어 있는 부분을 클릭해서 설정을 리로드 하여야 한다.



프로그램관련 직종을 확인해 보자.

웹디자이너

포토샵, 일러스트, 웹사이트를 그림 형태로 웹페이지를 제작

웹퍼블리셔

실제 원본 디자인을 웹사이트에 운영되는 html형태로 변경하는 작업

front-end developer

html, css, javascript, react, angular, jquery, vue 데이터를 이용해 동적인 클라이언트 화면을 제작

back-end developer

jsp, spring, asp, php, jango 서버 페이지를 동적으로 제작

서버란? 데이터를 제공해주는 컴퓨터

클라이언트란? 서버에 데이터를 요청해서 데이터를 제공받는 컴퓨터

네이버사이트 서버 본인 컴퓨터 클라이언트에 해당한다.

웹 프로그래밍이란? 사용자가 브라우저를 통해서 주소로 데이터를 요청하면 원하는 처리를 한후 html파일을 사용자에게 제공해 주는 프로그램을 의미한다. 클라이언트는 브라우저를 사용하고 서버는 톰캣 같은 웹서버 전문 프로그램을 사용한다.

웹서버? html데이터를 좀더 효과적으로 제공해 주기 위해서 서버를 극대화 하기 위해서 톰캣 등 의 전문 웹서버 프로그램을 설치 한다.

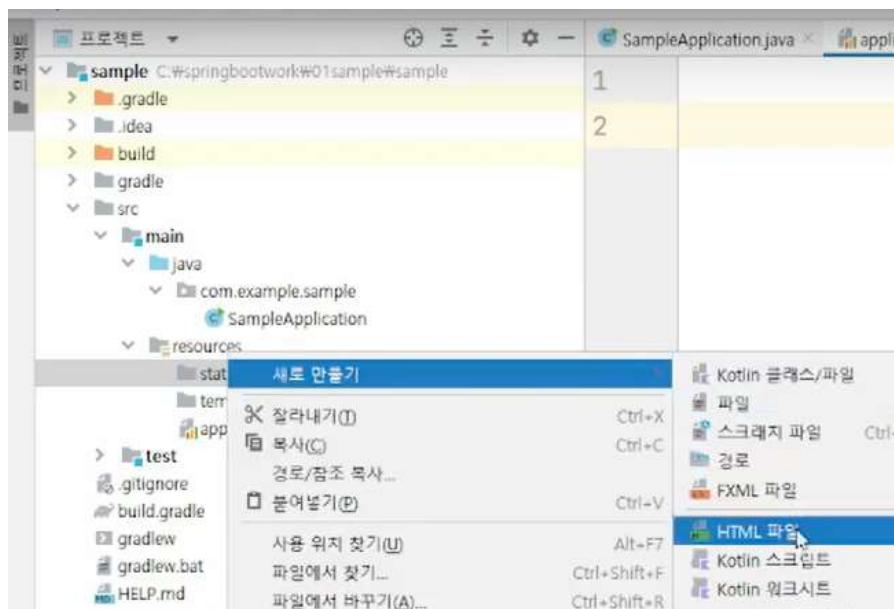
spring boot를 실행시킴으로써 본인 컴퓨터는 서버가 되고 브라우저는 클라이언트가 되어 주소로 원하는 데이터를 주고 받을 수 있게 된다.

이때 서버 클라이언트간에 사용하는 프로토콜이 http이다.

http는 request와 response로 간단하게 구성되어 있어 주소로 요청하고 html파일로 응답한다.

spring boot를 이용하여 사용자가 필요한 웹프로그램을 만드는 것이 이 책의 목표이다.

### > 03. static 폴더 helloworld 출력



왼쪽 static 폴더에서  
오른쪽 클릭을 하고  
helloworld.html 코드를  
생성한 다음 html 코드를  
입력한다.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
Hello World!
반갑습니다.
</body>
</html>
```

다음 코드의 화살표부분을 클릭해서 프로그램을 실행 시키자.

```
6     @SpringBootApplication
7 ►  public class SampleApplication {
8 ►      public static void main(String[] args) {
9          SpringApplication.run(SampleApplication.class, args);
10         }
11     }
```

The screenshot shows a Java IDE interface. In the top tab bar, there are three tabs: 'SampleApplication.java', 'application.properties', and 'build.gradle'. The 'application.properties' tab is currently active. The code in this file is:

```

1 # Port
2 server.port = 8080
3

```

Below the code editor is a browser window titled 'Title'. The address bar shows 'localhost:8080/helloworld.html'. The page content is 'Hello World! 방갑습니다.'

<http://localhost:8080/helloworld.html>  
를 브라우저 주소창에 기술하면 우리가 작성한 html파일을 확인할 수 있다.

static 폴더에 index.html를 다음과 같이 만들어 놓고 브라우저에  
<http://localhost:8080/> 를 입력하면 자동으로 index.html의 내용이 출력되는 것을 확인 할 수 있다.

The screenshot shows a Java IDE interface with a project tree on the left and a code editor on the right.

**Project Tree:**

- sample (C:\springbootwork\#01sample\sample)
  - .gradle
  - .idea
  - build
  - gradle
  - src
    - main
      - java
      - resources
        - static
          - helloworld.html
          - index.html
      - templates
    - application.properties
  - test
  - .gitignore
  - build.gradle

**Code Editor (index.html):**

```

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

```

이전에 설명한 서버를 재시작 하는 것을 잊지 말자 내용이 변경되면 반드시 서버를 재시작 해야 한다.

### 문제 1)

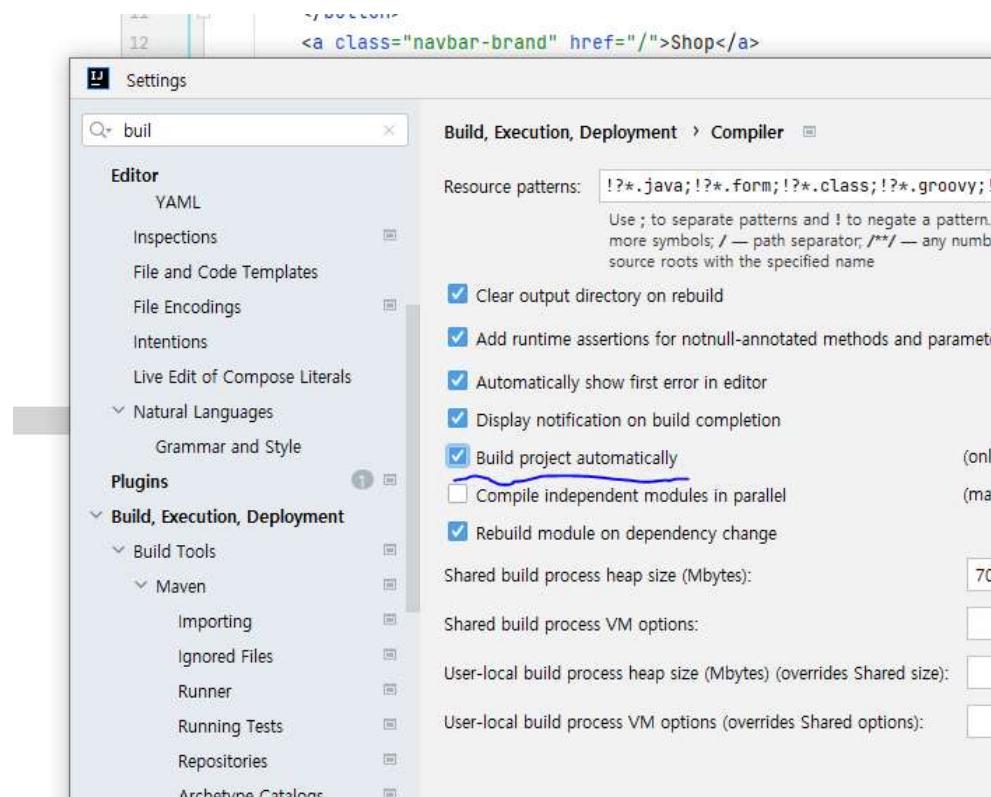
<http://localhost:8888/myName.html>를 입력 받아 화면에 본인 정보를 출력하는 프로그램을 구현해 보자.

자동화면 갱신하기

File-> Settings

Build, Execution, Deployment >> compiler

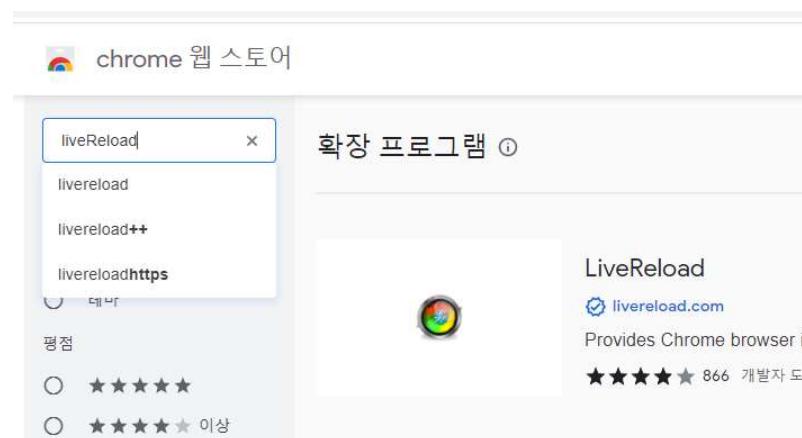
build projectAutomatically 를 체크한다.



application.properties 파일에 다음을 추가한다.

```
#live reload
spring.devtools.livereload.enabled=true
```

구글 클롬 웹스토어에서 livereload를 설치한다.





html를 수정하고 왼쪽 아이콘중 적용하고 싶은 화면을 클릭하면 적용된다.

```
#thymeleaf cache 사용중지
```

```
spring.thymeleaf.cache=false
```

---

## > 04. 데이터 전송방식 get과 post

---

클라이언트의 데이터를 서버로 전송하는 방식은 크게 get과 post가 있다.

get 방식은 주소에 쿼리 스트링을 이용하여 눈에 보이게 전송하는 방식이다.

쿼리 스트링은 주소+?key=value&key=value와 같은 형태로 주소 다음에 물음표를 붙이고 key와 value를 =으로 짹을 지어 &로 연결하여 원하는 데이터들을 전송하는 방법이다.

브라우저에 입력하는 방식, a 태그를 사용하는 방식, form태그의 method의 속성 값으로 get를 넣는 방법이 get방식이다.

post 방식은 전송할 데이터를 가지고 있지만 주소에서 확인할 수 없어 비밀번호 입력이나 대용량 파일 전송에 사용한다. form태그의 method의 속성 값으로 post를 넣는 방법이 유일한 post방식이다.

## > 05. MVC 패턴 사용하기

Controller란? 사용자가 요청한 주소와 전송방식을 확인해서 특정 자바코드를 실행 시켜주는 역할을 한다.

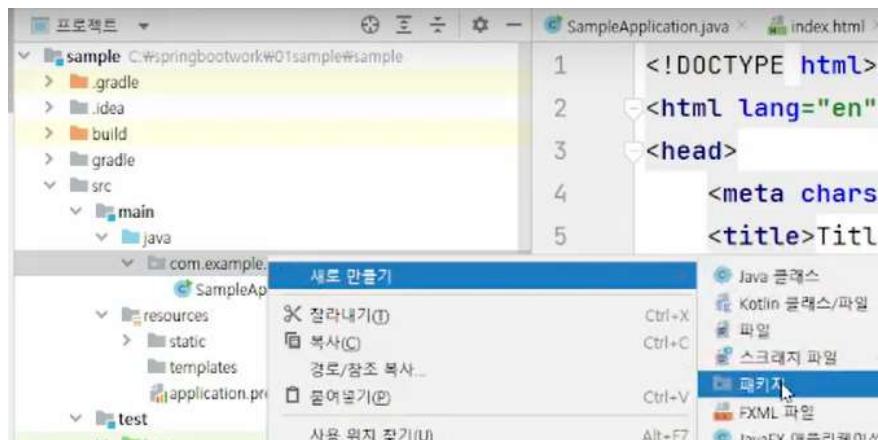
Model이란? 사용자가 원하는 데이터를 담아서 보관하는 역할을 한다.

view란? 모델을 이용해서 사용자가 원하는 화면을 만들어 보여주는 역할을 한다.

클라이언트 사용자가 서버쪽에 주소와 데이터를 전송해 주면 컨트롤러가 이를 받아 주소와 데이터를 확인하여 어떤 비즈니스 로직(자바 코드)를 실행 시켜 데이터를 모델에 담아 뷰를 선택해서 보내주면 뷰에서 모델을 이용해서 html를 만들어 요청한 클라이언트에게 데이터를 보내준다. 보통 이런 형태를 이용해서 웹프로그램을 만드는데 이를 MVC패턴이라고 한다.

컨트롤러 만들기

com.example.sample 패키지에 controller 패키지를 생성해보자.



다음 이미지 처럼 새로 만든 controller 패키지에서 마우스 오른쪽 클릭을 통해서 SampleController.java 자바 클래스를 만들어 보자.



SampleController.java 파일을 클릭해서 다음 이미지와 같은 코드를 입력해 보자.  
작업중 빨간색 글씨가 뜨면 마우스를 올려 놓고 클릭을 통해서 자동으로 필요한 패키지를 임포트하면 된다.

@Controller는 해당 자바클래스를 컨트롤러로 사용하겠다는 의미이다.

@GetMapping("/01helloWorld")의 의미는 클라이언트가 get방식으로 <http://localhost:8080/01helloWorld>를 입력하면 해당 주소 다음에 기술되어 있는 메소드가 실행된다는 의미이다. 메소드 이름은 큰 의미가 없으니 다른 메소드와 이름이

겹치지 않도록 적절하게 기술하면 된다.

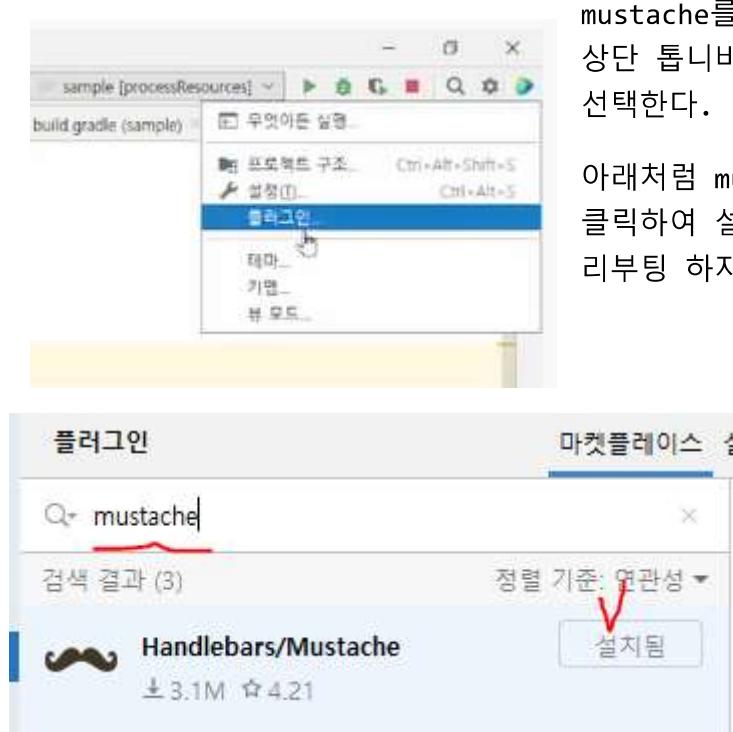
```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class SampleController {

    @GetMapping("/01helloWorld")
    public String helloWorldGet1(){
        return "01helloWorld";
    }
}
```

`return "01helloWorld";`에서 리턴된 문자열이 view의 이름이 된다. 우리는 mustache를 사용 할 예정이어서 연결할 view의 이름은 `01helloWorld.mustache`가 된다.

과거 jsp에서 view는 jstl로 만들었으나 spring boot는 thymeleaf, mustache 등 아주 다양한 형태로 만들 수 있다. 일단 사용할 view로 mustache를 사용해볼 예정이다.



mustache를 틀이 인식하게 하고 싶다면 오른쪽 상단 톱니바퀴를 클릭한 다음 플러그인을 선택한다.

아래처럼 mustache를 검색해서 설치 버튼을 클릭하여 설치 하자. 설치후 인식이되지 않는다면 리부팅 하자.

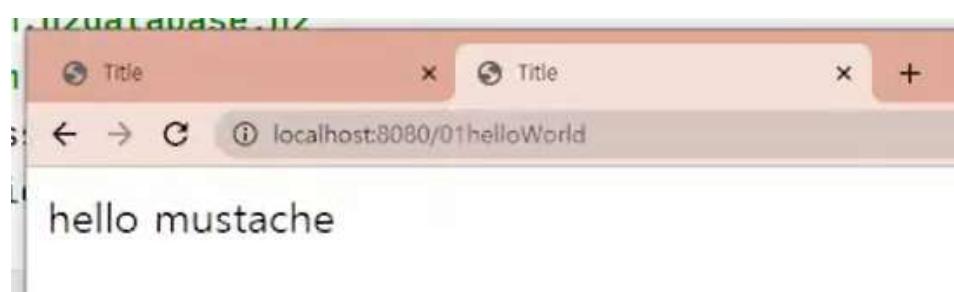
01helloWorld.mustache view 파일을 만들 위치는 resources 밑에 template에 만들면 된다 해당 폴더에서 마우스오른 쪽을 클릭해서 파일을 선택한후 01helloWorld.mustache 이름을 추가하여 해당 파일을 생성한다.



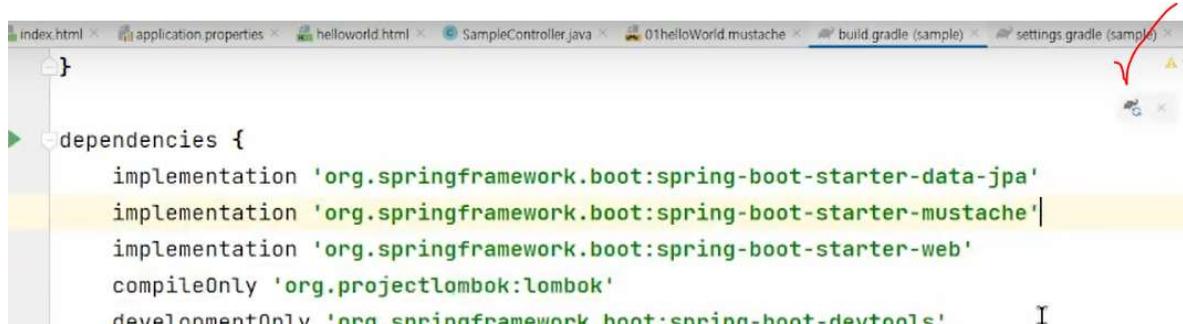
파일을 선택해서 다음과 같이 입력한다.

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>Title</title>  
</head>  
<body>  
    hello mustache  
</body>  
</html>
```

SampleApplicaton.java파일을 실행시키고 브라우저를 열어서 <http://localhost:8080/01helloWorld> 를 입력하고 실행시키면 다음 이미지 처럼 방금 만든 html내용을 확인할 수 있다.



thymeleaf와 mustache 의존객체가 2개 들어가서 문제가 발생할 수 있다. 문제 발생시 build.gradle파일의 dependencies부분에서 thymeleaf부분을 삭제하고 코끼리 아이콘을 클릭해서 재설정 한다음 다시 실행해 보자.



html 소스코드 부분만 다시 설정하고 싶다면  
`processResources`를 선택해서 그부분만 재실행하면 화면이  
 갱신된다.

자바코드 부분이 수정되었다면 이전처럼 모두 재시작해야  
 한다.

get방식에 대한 주소 매핑을 하는 방법은  
`@GetMapping`이나 `@RequestMapping`에 `method=RequestMethod.GET` 설정을 사용하는 것이다.

post방식에 대한 주소 매핑을 하는 방법은  
`@PostMapping`이나 `@RequestMapping`에 `method=RequestMethod.POST` 설정을 사용하는 것이다.

```

@Controller
//{@RequestMapping(value="/folder")}
public class SampleController {
    @GetMapping("/01helloWorld")
    public String helloWorldGet1(){
        return "01helloWorld";
    }
    @RequestMapping(value="/02helloWorld",method= RequestMethod.GET)
    public String helloWorldGet2(){
        return "01helloWorld";
    }
}
  
```

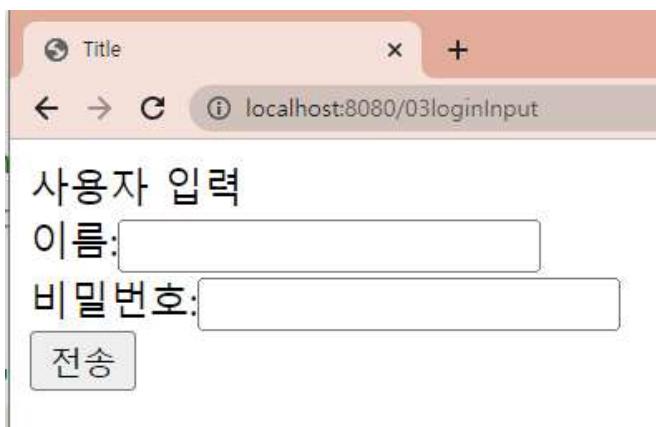
상위 코드를 추가한 후 <http://localhost:8080/02helloWorld> 를 주소창에 다음과 같이  
 입력하면 아까와 동일한 화면이 나온다.

```
//{@RequestMapping(value="/folder")}
```

상위 매핑의 주석을 풀고 실행하면 전체 실행주소는 /folder + 클래스의 메소드에 선언된  
 매핑 주소가 되어 다음과 같은 매핑 주소를 가진다.

<http://localhost:8080/folder/02helloWorld>

## > 06. 사용자 입력을 view에 보내주기



<http://localhost:8080/03loginInput> 과 같이 입력하면 왼쪽과 같은 사용자 입력을 받는 화면을 보여주게 만들어보자.

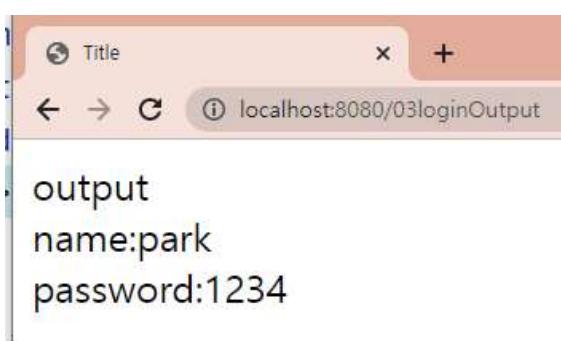
SampleController 클래스에 다음과 같은 주소 맵핑 코드를 추가 하자.

```
@GetMapping("/03loginInput")
public String loginInput() {
    return "02loginInput";
}
```

template 풀더에 02loginInput.mustache 파일을 만들고 다음과 같이 입력한다.

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    사용자 입력<br>
    <form action="/03loginOutput" method="post">
        이름:<input type="text" name="name" ><br>
        비밀번호:<input type="text" name="password" > <br>
        <input type="submit" value="전송">
    </form>
</body>
</html>
```

프로그램을 실행 상위 사용자 입력화면이 나오록 실행 해보자.



상위 입력 부분에 park과 1234를 입력한 다음 전송 버튼을 누르면 왼쪽과 같은 결과 화면이 나오도록 프로그램 작성은 해보자.

이전 html 폼 태그

```
<form action="/03loginOutput"
method="post">
    에서 전송방식은 post 맵핑 주소는
    /03loginOutput 이고 컨트롤러에 맵핑 주소와
```

사용자 입력 데이터를 처리하는 로직을 추가해 보면 다음과 같다.

```
사용자 입력<br>
<form action="/03loginOutput" method="post">
    이름:<input type="text" name="name" ><br>
    비밀번호:<input type="text" name="password" > <br>
    <input type="submit" value="전송">
</form> // @RequestMapping(value="/03loginOutput", method= RequestMethod.POST)
@PostMapping("/03loginOutput")
public String loginOut(
    @RequestParam(value="name", required=false) String name
    , @RequestParam String password
    , Model model){
    System.out.println(name);
    System.out.println(password);

    model.addAttribute(attributeName: "name", name);
    model.addAttribute(attributeName: "password", password);

    return "03loginOutput";
}
```

form 태그안에 기술된 사용자 입력을 컨트롤러에서 처리하기 위해서는 컨트롤러로 매핑된 메서드의 매개변수에 `@RequestParam` 를 붙이고 form태그에서 사용자가 입력한 데이터의 name속성과 같은 이름의 value 값이나 변수명을 선언하면 자동으로 사용자가 입력한 값이 컨트롤러의 해당 매개 변수에 들어간다.

<input type="text" name="name" >에 입력한값은  
`@RequestParam(value="name", required=false) String name` 의 name에  
<input type="text" name="password" > 에 입력한값은  
`@RequestParam String password` 의 password 에 담겨 해당 메소드에서 사용할 수 있다.

`return "03loginOutput";`은 컨트롤러를 실행후 뷰를 `03loginOutput.mustache`로 연결하라는 의미이다.

`required=false` 의 의미는 사용자 입력이 반드시 필요한 데이터는 `true`, 없어도 되면 `true`이고 기본값은 `true`이다.

모델에 key value형태로 담은 데이터를 key를 통해서 읽어와 view에서 사용할 수 있다.  
컨트롤의 매개변수로 Model model을 선언하면 컨트롤 안에서 model을 사용할 수 있고  
다음과 같이 데이터를 model에 담아 view에서 사용할 수 있다.

`model.addAttribute("name", name); "name" key에 값으로 name변수를 넣음`

```
//@RequestMapping(value="/03loginOutput", method= RequestMethod.POST)
@PostMapping("/03loginOutput")
public String loginOut(
    @RequestParam(value="name", required=false) String name
    , @RequestParam String password
```

```

    , Model model){
System.out.println(name);
System.out.println(password);
model.addAttribute("name",name);
model.addAttribute("password",password);
return "03loginOutput";
}

```

모델에 담긴 데이터를 view에서 확인하는 방법은 {{ 이중 중괄호 안에 모델에 담은 데이터의 키값을 기술하면 실행시 그 부분이 키와 짹을 이룬 value가 출력된다. 다음 코드를 확인해 보자.

`03loginOutput.mustache` 파일을 resources > templates 폴더에서 오른쪽 마우스 클릭하여 다음과 같이 추가하자.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
output<br>
name: {{name}} <br>
password: {{password}}<br>
</body>
</html>

```

---

## > 07. 객체형태 데이터처리하기

---

@RequestParam String password 와 같은 형태로 사용자 입력을 일일이 처리할 수 있지만 객체를 이용해서 처리하는 방법도 있다.

다음처럼 name,password를 담을 수 있는 객체를 선언한다.

```
public class LoginDto {  
    4개 사용 위치  
    private String name;  
    4개 사용 위치  
    private String password;
```

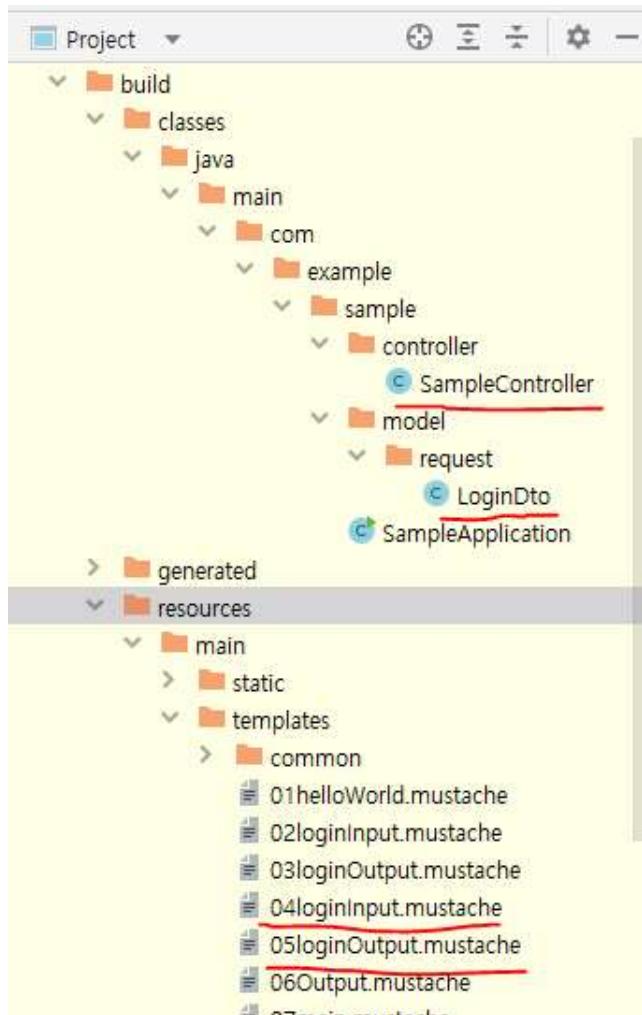
왼쪽 처럼 기술하고 setter, getter, 생성자, toString등을 오른쪽 마우스 클릭 generate를 선택하여 추가할 수 있다. 클래스 안에서 추가하고 싶은 항목을 선택하면 된다.  
컨트롤러 메서드의 매개변수를 다음처럼 클래스로 생성하면 name과 password를 담을 수 있다.

```
@PostMapping("/05loginOutput")  
public String loginDtoOutput(LoginDto dto, Model model){  
    System.out.println(dto);  
    model.addAttribute(attributeName: "dto", dto);  
    return "/05loginOutput";  
}
```

뷰에서 다음처럼 객체정보를 출력할 수 있다.

```
{{#dto}}  
    name: {{name}} <br>  
    password: {{password}}<br>  
{{/dto}}
```

다음 순서대로 예제를 작성해보자.



1. model 밑에 request 패키지를 만들고 LoginDto 클래스를 만든다.

```

package com.example.sample.model.request;
public class LoginDto {
    private String name;
    private String password;
    public LoginDto() {}
    public LoginDto(String name, String password) {
        this.name = name;
        this.password = password;
    }
    @Override
    public String toString() {
        return "LoginDto{" +
            "name='" + name + '\'' +
            ", password='" + password +
            '\'';
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

2. name password를 입력 화면관련 코드를 Samplecontroller에 추가 한다.

```
@GetMapping("/04loginInput")  
  
public String loginDtoInput() {  
  
    return "/04loginInput";  
  
}
```

3. <http://localhost:8080/04loginInput>라 요청하였을때 보여줄 화면 파일 **04loginInput.muscatch** 파일을 resources 밑에 templates 밑에 생성하고 다음 코드를 넣는다.

```
<!DOCTYPE html>  
  
<html lang="ko">  
  
<head>  
    <meta charset="UTF-8">  
    <title>Title</title>  
</head>  
  
<body>  
    사용자 입력<br>  
  
<form action="/05loginOutput" method="post">  
    이름:<input type="text" name="name" ><br>  
    비밀번호:<input type="text" name="password" > <br>  
    <input type="submit" value="전송">  
</form>  
</body>  
</html>
```

4. submit버튼을 눌러 <http://localhost:8080/05loginOutput> 주소를 통해 컨트롤러에서 사용자 입력을 받아 처리할 컨트롤러 코드를 추가해 보자.

Model과 ModelAndView 를 사용하는 방법을 확인해 보자. 하나의 응용프로그램에서 맵핑 주소가 같은 코드가 2개 존재하면 문제가 발생한다. 아래 2개의 코드중 하나는 반드시 주석을 해야 실행 가능한다.

Caused by: java.lang.IllegalStateException: Ambiguous mapping. Cannot map 'sampleController' method 과같은 에러는 주소가 겹쳤을때 발생하는 에러이다.

```
@PostMapping("/05loginOutput")  
  
public String loginDtoOutput(LoginDto dto, Model model) {
```

```

        System.out.println(dto);
        model.addAttribute("dto", dto);
        return "/05loginOutput";
    }

    @PostMapping("/05loginOutput")
    public ModelAndView loginDtoOutput(LoginDto dto) {
        ModelAndView model=new ModelAndView();
        System.out.println(dto);
        model.addObject("dto", dto);
        model.setViewName("/05loginOutput");
        return model;
    }
}

```

5. <http://localhost:8080/05loginOutput> 요청시 처리 화면 view 05loginOutput.mustache 파일을 다음과 같이 만든다.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    output<br>
    {{#dto}} <!-- 모델에 담겨있는 객체 이름을 왼쪽과 같이 기술 한다. -->
    name:{{name}} <br><!-- dto 객체안의 필드 이름을 -->
    password:{{password}}<br>
    {{/dto}}
</body>
</html>

```

---

## > 08. Mustache arrayList 출력하기

---

1. <http://localhost:8080/06arrayList> 를 요청하면 컨트롤러의 dtos에 데이터를 담아 06Output.mustach 의 뷰로 보여주는 다음 코드를 확인해 보자.

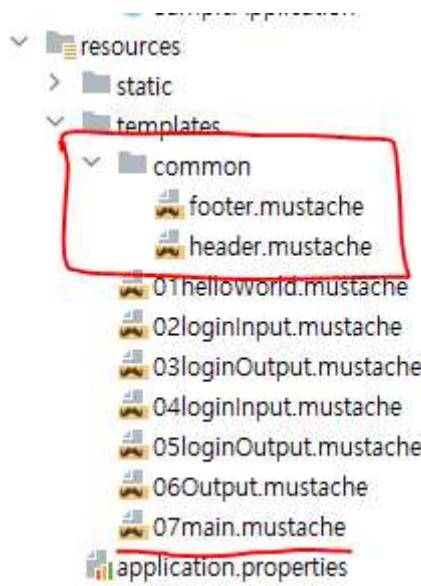
```
@GetMapping("/06arrayList")
public ModelAndView arrayListM() {
    ArrayList<LoginDto> dtos=new ArrayList<>();
    dtos.add(new LoginDto("1","1"));
    dtos.add(new LoginDto("2","2"));
    dtos.add(new LoginDto("3","3"));

    ModelAndView model=new ModelAndView();
    model.addObject("dtos",dtos);
    model.setViewName("/06Output");
    return model;
}

//06Output.mustach
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    output<br>
    {{#dtos}}
        name:{{name}}<br>
        password:{{password}}<br>
    {{/dtos}}
</body>
</html>
```

## > 09. mustache 분할코드 화면을 만들어 보자.

- 컨트롤러에 <http://localhost:8080/07> 를 입력하면 07main.mustache 뷰가 호출되도록 컨트롤에 다음 코드를 추가한다.



```
@GetMapping("/07")
public String mustacheCommon() {
    return "/07main";
}
```

- 07main.mustach 내용은 다음과 같다.  
{ {>common/header} }  
메인 부분  
{ {>common/footer} }

{ {>mustache 파일이름} } 과 같은 형태로 부분 파일이름을 기술한다.

- common폴더를 만들고 footer.mustache header.mustache 파일을 만들고 각각의 파일에 다음과 같은 코드를 넣는다.

### 4. header.mustache

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
여기는 헤더
```

### 5. footer.mustache

```
여기는 footer
</body>
</html>
```

- 07main.mustach에서 부분파일이 합쳐지면 하나의 html코드가 된다.

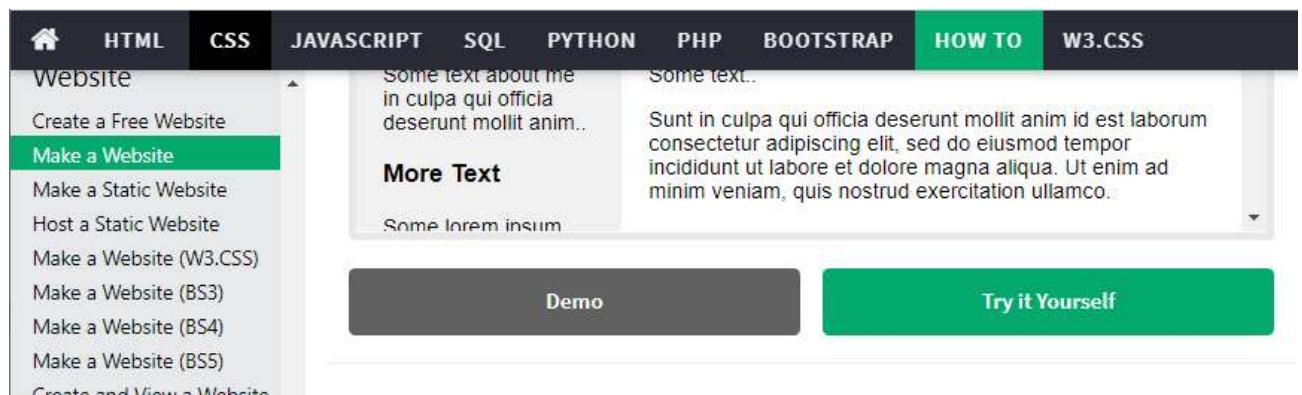
## > 10. JPA를 이용한 crud작업

sample02에서 jpa작업을 할 예정이다 인텔리제이를 이용해서 해당 파일을 열어 보자.

예전 spring legacy 프로젝트에서는 DB를 mybatis를 이용하여 xml로 조작 하였다. spring boot에서는 jpa를 이용해서 자바 객체형태로 DB에 직접 조작 한다.

작업할 화면을 템플릿 형태로 만들어볼 예정이다.

[https://www.w3schools.com/howto/howto\\_make\\_a\\_website.asp](https://www.w3schools.com/howto/howto_make_a_website.asp) 사이트에 들어가 왼쪽 매뉴에서 make a website를 클릭하면 오른쪽과 같은 화면이 나온다. try it yourself를 클릭하면 우리가 작업할 화면 소스코드가 나타난다.

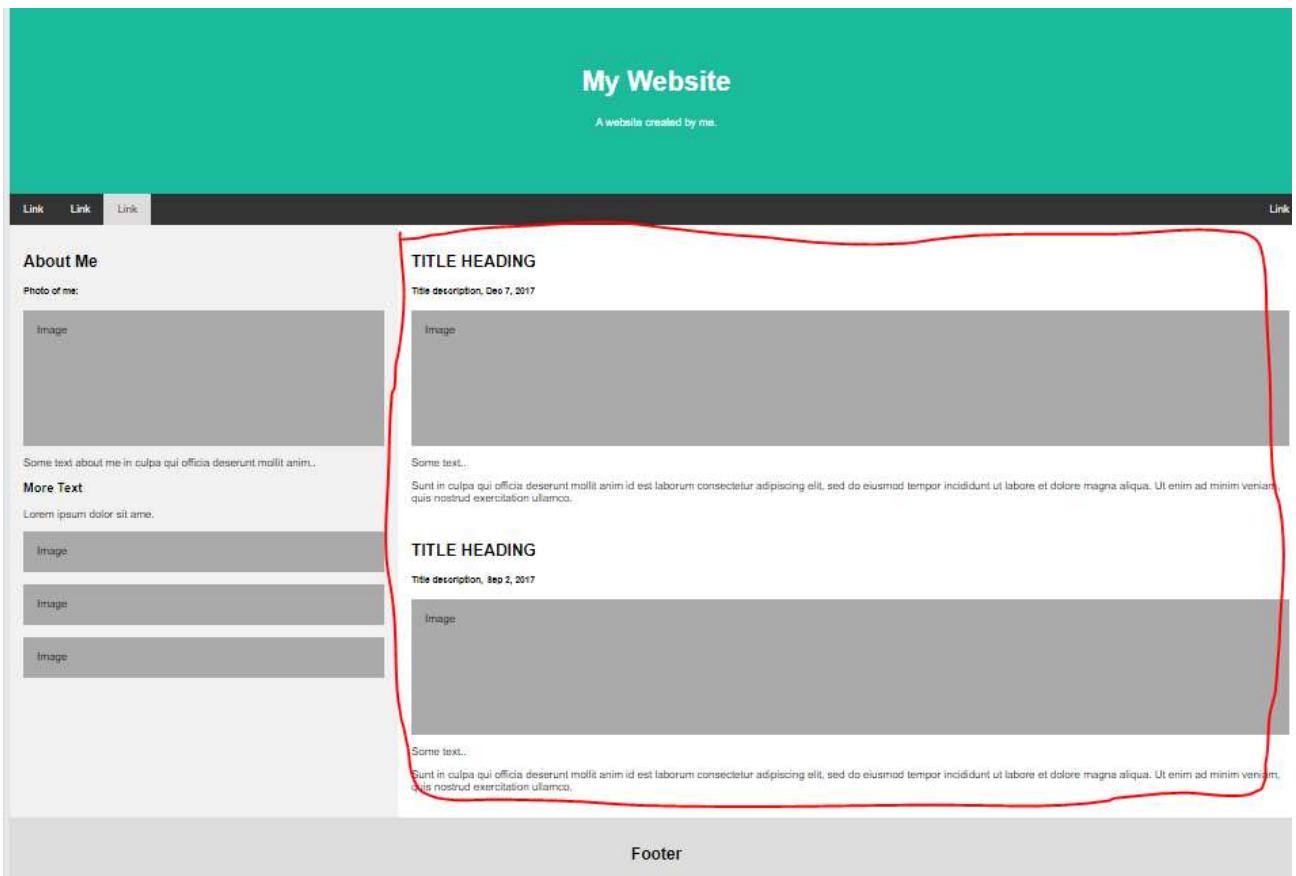


실행된 페이지를 확인해 보면 아래와 같은 화면이 보이는데 insert, update, select, delete 요청시 빨간색 박스 부분만 변경하여 처리할 예정이다.

빨간박스 위부분 html은 head.mustache파일에 기술한다.  
빨간박스 부분은 요청 종류에 따라 insert.mustache, update.mustache, selectOne.mustache, selectAll.mustache 중 하나의 화면을 제공한다.

빨간 박스 아래 부분은 footer.mustache에 기술한다.

select의 경우 하나의 데이터를 읽어오는 selectOne.mustache와 여러개의 데이터를 처리하는 selectAll.mustache를 추가하였다. delete.mustache는 삭제시 selectOne.mustache 상세 보기 화면(하나의 데이터가 검색된 화면)에서 하고 있어서 여기에서는 존재하지 않는다.



```
//head.mustache
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Page Title</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    * {
      box-sizing: border-box;
    }

    /* Style the body */
    body {
      font-family: Arial, Helvetica, sans-serif;
      margin: 0;
    }

    /* Header/logo Title */
    .header {
      padding: 80px;
      text-align: center;
      background: #1abc9c;
      color: white;
    }

    /* Increase the font size of the heading */
    .header h1 {
      font-size: 40px;
    }

    /* Style the top navigation bar */
    .nav-bar {
      background-color: #f2f2f2;
      overflow: hidden;
    }
  
```

```

.navbar {
  overflow: hidden;
  background-color: #333;
}

/* Style the navigation bar links */
.navbar a {
  float: left;
  display: block;
  color: white;
  text-align: center;
  padding: 14px 20px;
  text-decoration: none;
}

/* Right-aligned link */
.navbar a.right {
  float: right;
}

/* Change color on hover */
.navbar a:hover {
  background-color: #ddd;
  color: black;
}

/* Column container */
.row {
  display: -ms-flexbox; /* IE10 */
  display: flex;
  -ms-flex-wrap: wrap; /* IE10 */
  flex-wrap: wrap;
}

/* Create two unequal columns that sits next to each other */
/* Sidebar/left column */
.side {
  -ms-flex: 30%; /* IE10 */
  flex: 30%;
  background-color: #f1f1f1;
  padding: 20px;
}

/* Main column */
.main {
  -ms-flex: 70%; /* IE10 */
  flex: 70%;
  background-color: white;
  padding: 20px;
}

/* Fake image, just for this example */
.fakeimg {
  background-color: #aaa;
  width: 100%;
  padding: 20px;
}

/* Footer */
.footer {
  padding: 20px;
  text-align: center;
  background: #ddd;
}

/* Responsive layout - when the screen is less than 700px wide, make the
two columns stack on top of each other instead of next to each other */
@media screen and (max-width: 700px) {

```

```

.row {
  flex-direction: column;
}

/* Responsive layout - when the screen is less than 400px wide, make the
navigation links stack on top of each other instead of next to each other
*/
@media screen and (max-width: 400px) {
  .navbar a {
    float: none;
    width: 100%;
  }
}
</style>
</head>
<body>

<div class="header">
  <h1>My Website</h1>
  <p>A website created by me.</p>
</div>

<div class="navbar">
  <a href="#">Link</a>
  <a href="#">Link</a>
  <a href="#">Link</a>
  <a href="#" class="right">Link</a>
</div>

<div class="row">
  <div class="side">
    <h2>About Me</h2>
    <h5>Photo of me:</h5>
    <div class="fakeimg" style="height:200px;">Image</div>
    <p>Some text about me in culpa qui officia deserunt mollit anim..</p>
    <h3>More Text</h3>
    <p>Lorem ipsum dolor sit ame.</p>
    <div class="fakeimg" style="height:60px;">Image</div><br>
    <div class="fakeimg" style="height:60px;">Image</div><br>
    <div class="fakeimg" style="height:60px;">Image</div>
  </div>
//footer.mustache
</div>

<div class="footer">
  <h2>Footer</h2>
</div>

</body>
</html>

//insert.mustache
{{>common/head}}
<div class="main">
  <h2>TITLE Insert</h2>
  <form action="/insert" method="post">
    <table width="80%">
      <tr>
        <td>name:</td><td><input type="text" name="name"></td>
      </tr>
      <tr>
        <td>password:</td><td><input type="text" name="password"></td>
      </tr>
      <tr>
        <td colspan="2"><a href="/selectAll">listAll</a></td>
      </tr>
    </table>
  </form>
</div>

```

```

        type="submit" value="submit">></td>
    </tr>
</table>
</form>
</div>
{{>common/footer}}}

//update.mustache
{{>common/head}}
<div class="main">
    <h2>TITLE update</h2>

    {{#dto}}
        <form action="/update" method="post">
            <table width="80%">
                <tr>
                    <td>id:</td><td>{{id}}</td>
                </tr>
                <tr>
                    <td>name:</td><td><input type="text" value="{{name}}"/>
                </tr>
                <tr>
                    <td>password:</td><td><input type="text" value="{{password}}"/>
                </tr>
                <tr>
                    <td colspan="2"><a href="/selectAll">list all</a>
                        <input type="submit" value="update"></td>
                </tr>
            </table>
        </form>
    {{/dto}}
</div>
{{>common/footer}}}

//selectOne.mustache
{{>common/head}}
<div class="main">
    <h2>TITLE selectOne</h2>

    {{#dto}}
        <table width="80%">
            <tr>
                <td>id:</td><td>{{id}}</td>
            </tr>
            <tr>
                <td>name:</td><td>{{name}}</td>
            </tr>
            <tr>
                <td>password:</td><td>{{password}}</td>
            </tr>
            <tr>
                <td colspan="2"><a href="/selectAll">list all</a>
                    <a href="/delete/{{id}}">delete</a>
                    <a href="/update/{{id}}">update</a></td>
                </tr>
            </table>
    {{/dto}}
</div>
{{>common/footer}}}

//selectAll.mustache
{{>common/head}}

```

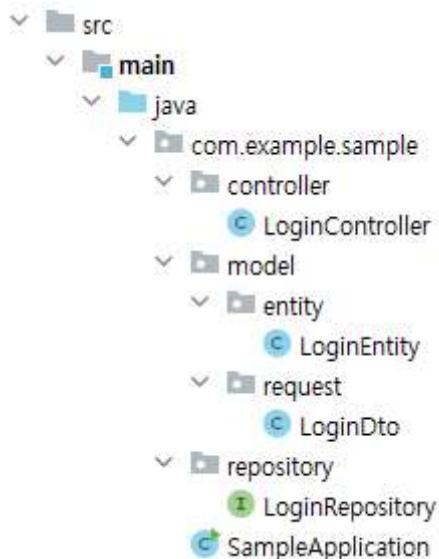
```

<div class="main">
    {{#msg}}
        <h2>Message: {{msg}}</h2>
    {{/msg}}
    <h2>TITLE select ListAll</h2>

    <table width="80%">
        {{#dtos}}
            <tr>
                <td>id:</td><td>{{id}}</td>
            </tr>
            <tr>
                <td>name:</td><td><a href="/selectOne/{{id}}">{{name}}</a></td>
            </tr>
            <tr>
                <td>password:</td><td>{{password}}</td>
            </tr>
        {{/dtos}}
        <tr>
            <td colspan="2"><a href="/insert">new data</a></td>
        </tr>
    </table>

</div>
{{>common/footer}}

```



boot에 기본으로 세팅 되어 있는 데이터 베이스의 경우 테이블을 따로 만들지 않아도 알아서 들어 간다.

```

package com.example.sample.model.entity;

import lombok.*;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;

```

//model 밑에 entity 밑에 LoginEntity.java

jpa로 데이터베이스에 데이터를 넣으려면 클래스로 만들어서 db에 저장해야 한다. 이때 필요한 클래스가 Entity클래스이다. LoginEntity 클래스는 해당 인스턴스 값 그대로 db에 데이터를 조작할때 사용한다.

여기서 특별히 룸복을 사용하였다. 룸복은 어노테이션을 이용하여 setter,getter,생성자,toString등을 일어 기술하지 않고 기술한것 처럼 사용할 수 있다. 각설명은 아래 예제 주석으로 기술하였으니 확인해보자.

entity클래스로 사용하려면 테이블에서 식별가능한 컬럼은 @Id, 자동 증가 컬럼은 @GeneratedValue, 일반 컬럼은 @Column으로 기술하면 h2 DB 같은 경우 spring

```

import javax.persistence.Id;

@Entity //jpa에서 사용할 클래스에 반드시 추가한다.
@AllArgsConstructor //모든 필드를 매개변수로 가지는 생성자 생성
@NoArgsConstructor //디폴트 생성자 생성
@ToString //객체의 내용을 문자열로 출력 한다.
@Getter // getter setter 자동 추가 하고 싶을때 사용
@Setter

public class LoginEntity {

    @Id //테이블에서 유일한 값을 가지는 컬럼에 추가 할때 사용.
    @GeneratedValue //해당 컬럼 자동 증가
    private Integer id;

    @Column //해당 테이블의 컬럼으로 사용하고자 할때 사용
    private String name;

    @Column
    private String password;

}

```

// model 밑에 request 밑에 LoginDto.java

1.파일에 id 필드를 추가하고 id필드와 관련된 로직을 추가 하였다.

jpa를 사용하려면 테이블의 유일한 값을 컬럼으로 가지고 있어야 해서 추가 하였다.

2.jpa로 데이터베이스에 데이터를 넣으려면 데이터구조와 동일한 Entity클래스가 필요한데 사용자입력을 받은 dto클래스의 필드를 이용해서 자동으로 Entity클래스를 만드는 아래와 같은 필드가 필요하다.

```

public LoginEntity toEntity() {
    return new LoginEntity(id, name, password);
}

```

과 같은 필드를 추가하였다.

```

package com.example.sample.model.request;
import com.example.sample.model.entity.LoginEntity;
public class LoginDto {
    private Integer id;
    private String name;
    private String password;
    public LoginEntity toEntity() {
        return new LoginEntity(id, name, password);
    }
    public LoginDto() {}
    public LoginDto(String name, String password) {
        this.name = name;
    }
}

```

```

        this.password = password;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    @Override
    public String toString() {
        return "LoginDto{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", password='" + password + '\'' +
            '}';
    }
    public LoginDto(Integer id, String name, String password) {
        this.id = id;
        this.name = name;
        this.password = password;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

//repository 폴더 밑에 LoginRepository.java

해당 클래스는 Entity클래스를 DB에 넣을때 필요한 작업을 구현해 놓은 클래스이다.  
 CrudRepository 클래스를 상속받으면 기존 적인 작업은 이미 다 만들어져 있어서 특별히  
 작업할 내용이 없다. CrudRepository 클래스에 없는 기능이나 이미 있지만 재정의 하여  
 사용하고 싶다면 LoginRepository 인터페이스 안에 내용을 재정의 하여 사용한다.

CrudRepository<Entity클래스, 해당Entity클래스의식별ID자료형> 형태로 사용한다.

```

package com.example.sample.repository;

import com.example.sample.model.entity.LoginEntity;
import com.example.sample.model.request.LoginDto;

```

```

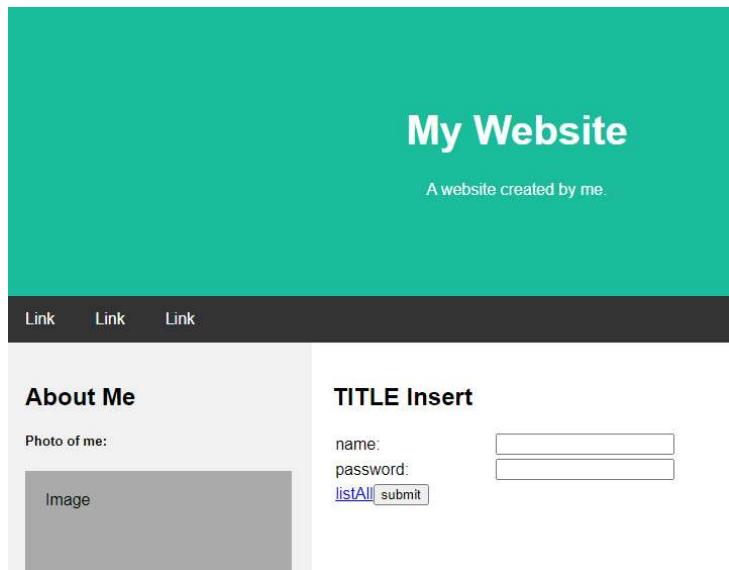
import org.springframework.data.repository.CrudRepository;

import java.util.ArrayList;

public interface LoginRepository extends
CrudRepository<LoginEntity, Integer> {
    @Override
    ArrayList<LoginEntity> findAll();
}

// findAll 메소드는 이미 부모에 재정의 되어 있는 메소드 이지만 리턴값을
// 다르게하여 재정의한 메소드이다. 이처럼 기존 존재하는 메소드와 다르게 만들어 사용하고
// 싶다면 이곳에 정의하여 추가하면 된다.
}

```



// <http://localhost:8080/insert>  
 주소로 들어와 insert.mustache  
 파일에 연결하는 컨트롤러를  
 LoginController에 추가한다.

```

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

```

실행후 왼쪽화면이 나오면 정상  
 적으로 작성된 것이다.

## TITLE selectOne

id:	1
name:	park
password:	123
<a href="#">list all</a> <a href="#">delete</a> <a href="#">update</a>	

상위 이미지에 park, 123 데이터를  
 넣고 submit 버튼을 누르면  
<http://localhost:8080/insert>  
 형태로 post 방식으로 전송되어 왼쪽과  
 같은 화면이 나온다.  
 컨트롤러에 다음과 같은 코드를  
 입력하면 된다. 주석을 확인해 보자.

```

@Controller
@Slf4j // log.info() 로그 메소드를 사용하기 위해서 사용
public class LoginController {
    @Autowired // db조작을 위해서 Repository 객체를 주입하였다.
    private LoginRepository loginRepository;
}

```

클래스 시작부분에 상위와 같이 추가한다. 로그란? 디버깅시 화면에 출력할 메시지를  
 관리하는 프로그램이다. System.out.println과 차이점은 로그는 디버깅시에만

출력이되고 System.out.println은 디버그 뿐만 아니라 프로그램 배포시에도 출력이 되어 배포시 주석처리하여 출력하지 못하게 하는 번거러움이 있다.

```
@PostMapping("/insert")
public String insert(LoginDto dto, Model model) {
    System.out.println(dto);
    //DB 넣는 작업 JAP 진행
    //toEntity메소드를 사용하지 않고 만들 경우
    //LoginEntity entity=new
    LoginEntity(null,dto.getName(),dto.getPassword());
    //LoginEntity entitySaved=loginRepository.save(entity);
    //toEntity메소드를 사용하여 만든 경우
    LoginEntity entitySaved=loginRepository.save(dto.toEntity());
    //loginRepository.save메소드 다음에 오는 객체가 db에 들어가고 db에 들어간 데이터 리턴한다.
    System.out.println(entitySaved); //콘솔에 직접 찍는 방법
    log.info("log"+entitySaved.toString()); //로그에 찍는 방법 @Slf4j
    // return "insert";
    return "redirect:/selectOne/"+entitySaved.getId();
}
```

## TITLE selectOne

id:  
name:  
password:  
[list all](#) [delete](#) [update](#)

1  
park  
123

상위 이미지에서 submit 버튼을 누르면 다음주소로 redirect된다.

<http://localhost:8080/selectOne/1>

맨끝에 1은 보여주고 싶은 데이터의 id값이어서 그때 그때 다르다. 아래코드를 추가하고 왼쪽과 같은 화면이 나오는지 확인해 보자.

```
//selectOne/1 1번글 읽어오기
@GetMapping("/selectOne/{id}")
public String selectOne(@PathVariable Integer id, Model model) {
    LoginEntity dto=loginRepository.findById(id).orElse(null);
    model.addAttribute("dto", dto);
    System.out.println(dto);
    return "selectOne";
}
```

## TITLE select ListAll

id:  
name:  
password:  
id:  
name:  
password:  
id:  
name:  
password:  
[new data](#)

1  
park  
123  
2  
pakr2  
2222  
3  
pakr2  
2222

상위 이미지에서 listAll a태그를 클릭하면

<http://localhost:8080/selectAll>

형태의 주소를 컨트롤에게 요청하게 되어 다음 코드가 실행되고 왼쪽 화면이 나온다.

데이터를 몇개더 입력하고 실행한 화면이다.

```

@GetMapping("/selectAll")
public String selectAll(Model model) {
//loginRepository.findAll() 메소드를 이용하여 데이터 베이스에 들어 있는 모든
데이터를 리턴하였다. 상속받은 CrudRepository<LoginEntity, Integer>
에 있는 findAll()과 리턴값이 달라서 재정의 하였다.
    ArrayList<LoginEntity> dtos=loginRepository.findAll();
    model.addAttribute("dtos",dtos);
    System.out.println(dtos);
    return "selectAll";
}

```

상위 이미지에서 new data를 클릭하면 <http://localhost:8080/insert> 주소가 요청되어 이전에 구현한 insert페이지로 이동한다.

## TITLE selectOne

id:  
 name:  
 password:  
[list all](#) [delete](#) [update](#)

1  
 park  
 123

왼쪽에서 delete를 클릭하면  
<http://localhost:8080/delete/1>  
 주소에 해당하는 다음 코드가  
 컨트롤러에서 실행되어 데이터가  
 삭제된다.

```

@GetMapping("/delete/{id}")
public String delete(@PathVariable Integer id,
                     RedirectAttributes rttr){
  LoginEntity dto=loginRepository.findById(id).orElse(null);

  System.out.println(dto);
  if(dto!=null) {
    loginRepository.delete(dto);
    rttr.addFlashAttribute("msg", "success");
  }
  return "redirect:/selectAll";
}

```

## TITLE update

id: 1  
 name: park  
 password: 123  
[list all](#) [update](#)

상위 이미지에서 update 버튼을  
 클릭하면  
<http://localhost:8080/update/1>  
 주소로 컨트롤러에 이동하고 다음 코드가  
 실행 되어 왼쪽 화면 처럼 수정 가능한  
 화면이 나온다.

```

@GetMapping("/update/{id}")
public String update(@PathVariable Integer id,
                     Model model){
  LoginEntity dto=loginRepository.findById(id).orElse(null);

```

```

if(dto==null) {
    System.out.println("데이터가 없음");
} else{
    model.addAttribute("dto", dto);
}
return "update";
}

```

상위 창에 수정한 후 update 버튼을 누르면 수정한 내용이 적용된다.

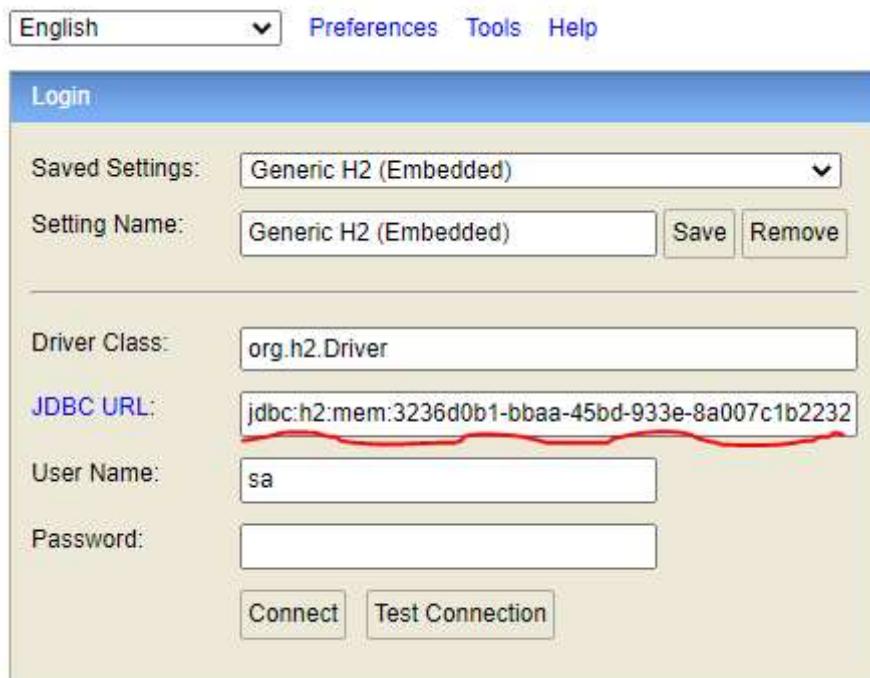
<http://localhost:8080/update> 주소로 post방식으로 요청되고 컨트롤러에 다음 코드가 실행되어 데이터베이스에 수정된다.

```

@PostMapping("/update")
public String update( LoginDto dto){
    LoginEntity entity=dto.toEntity();
    System.out.println(entity.toString());
    loginRepository.save(entity);
    return "redirect:/selectOne/"+entity.getId();
}

```

기본으로 사용하고 있는 h2 DB는 메모리에서 동작하고 있기 때문에 서버를 재시작하면 그동안 작업한 내용이 초기화 된다. 실행중 웹에서 확인하려면 브라우저에 <http://localhost:8080/h2-console> 과 같이 입력하면 다음 창이 뜨는데 JDBC URL경로에 서버 실행시 콘솔에 출력된 url경로를 복사 붙여 넣기 한다음 connect 버튼을 클릭하면 접근할 수 있다.



```

Run: sample [SampleApplication.main()]
sample [SampleApplication] 4 hr, 8 min
  sample [SampleApplication.main()] 4 hr, 6 min
    ita repository scanning in 141 ms. Found 1 JPA repository interfaces.
    with port(s): 8080 (http)
    Tomcat]
    engine: [Apache Tomcat/9.0.65]
    g embedded WebApplicationContext
    nContext: initialization completed in 5219 ms
    rting...
    rt completed.
    le at '/h2-console'. Database available at 'jdbc:h2:mem:3236d0b1-bbaa-45bd-933e-8a007c1b2232'
    ing PersistenceUnitInfo [name: default]

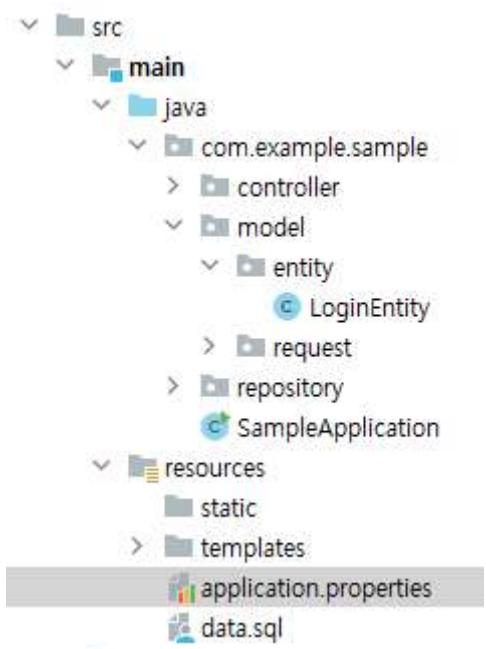
```

edit 박수에 sql를 입력하고 run버튼을 누르면 원하는 결과를 얻을 수 있다.  
 테이블 이름을 확인해 보면 java에서 camelCase형태로 loginEntity라 기술하면  
 데이터 베이스에서 snakeCase형태로 login\_entity란 이름으로 생성된다. 이는  
 컬럼도 마찬가지이다.

ID	NAME	PASSWORD
1	park	123
2	pakr2	2222
3	pakr2	2222

(3 rows, 68 ms)

## > 12. 오라클로 변경하기



이전 프로그램을 실행시키면 기본으로 설정되어 있는 h2 DB가 사용된다. 설정을 변경하여 오라클DB에서 동작하게 할 수 있다.

application.properties 파일을 선택하여 아래의 환경설정 내용을 기술하고 실행하면 된다. username과 password를 확인해 보자.

```
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=c##human
spring.datasource.password=human
```

application.properties 파일 말고 application.yml 파일을 사용하여 설정을 하기도 한다. yaml 사용방법은 폴더 별로 .으로 연결하던것을 : 과 줄바꿈으로 기술하면된다. 두 파일을 잘 비교해보자.

application.properties 파일을 삭제하고 application.yml파일을 만든 다음 아래 내용을 추가 하면 동일하게 동작한다.

```
spring:
  datasource:
    driver-class-name: oracle.jdbc.driver.OracleDriver
    url: jdbc:oracle:thin:@localhost:1521:xe
    username: c##human
    password: human
```

h2 디비는 테이블 생성없이 바로 처리가 되지만 오라클의 경우는 관련 테이블과 sequence를 생성해 주어야 한다. 다음 sql를 작성해 보자.

```
create table login_entity(
  id number,
  name nvarchar2(100),
  password nvarchar2(100)
);
```

```

create sequence login_id_seq;

insert into login_entity values(login_id_seq.nextVal, 'park1', '11234');
insert into login_entity values(login_id_seq.nextVal, 'park2', '21234');
insert into login_entity values(login_id_seq.nextVal, 'park3', '31234');
commit;
select * from login_entity;

```

다음은 loginEntity 클래스에 오라클에서 사용할 seq를 id에 연결한 코드이다. 다음을 추가하고 실행해보자.

```

@SequenceGenerator(
    name="LOGIN_ID_SEQ_GEN", //시퀀스 제너레이터 이름
    sequenceName="LOGIN_ID_SEQ", //시퀀스 이름
    initialValue=1, //시작값
    allocationSize=1 //메모리를 통해 할당할 범위 사이즈
)
public class LoginEntity {
    @Id
    @GeneratedValue(
        strategy=GenerationType.SEQUENCE, //사용할 전략을 시퀀스로 선택
        generator="LOGIN_ID_SEQ_GEN" //식별자 생성기를 설정해놓은
    LOGIN_ID_SEQ_GEN으로 설정
    )
    private Integer id;
}

```

프로젝트를 실행해보면 이전과 동일하게 실행되는 것을 확인할 수 있고 sql developer를 이용하여 데이터 베이스에 제대로 데이터가 저장되는지 확인해 보자.

---

> 13.

---

---

> 14.

---

