CRUD with SPRING BOOT

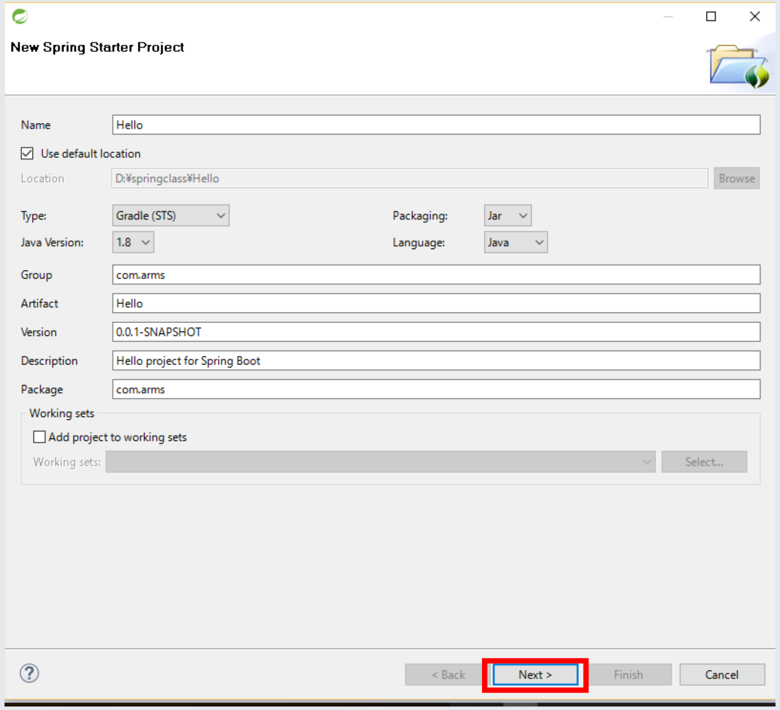
Presented by ARMS（THAILAND）Co., Ltd.

Index

1. Start creating a project

1-1. Start Project

- Open STS and File – New – Spring Starter Project


New Spring Starter Project

Name: Hello
☑ Use default location
Location: D:¥springclass¥Hello
Type: Gradle (STS) Packaging: Jar
Java Version: 1.8 Language: Java
Group: com.arms
Artifact: Hello
Version: 0.0.1-SNAPSHOT
Description: Hello project for Spring Boot
Package: com.arms
Working sets
☐ Add project to working sets
Working sets:

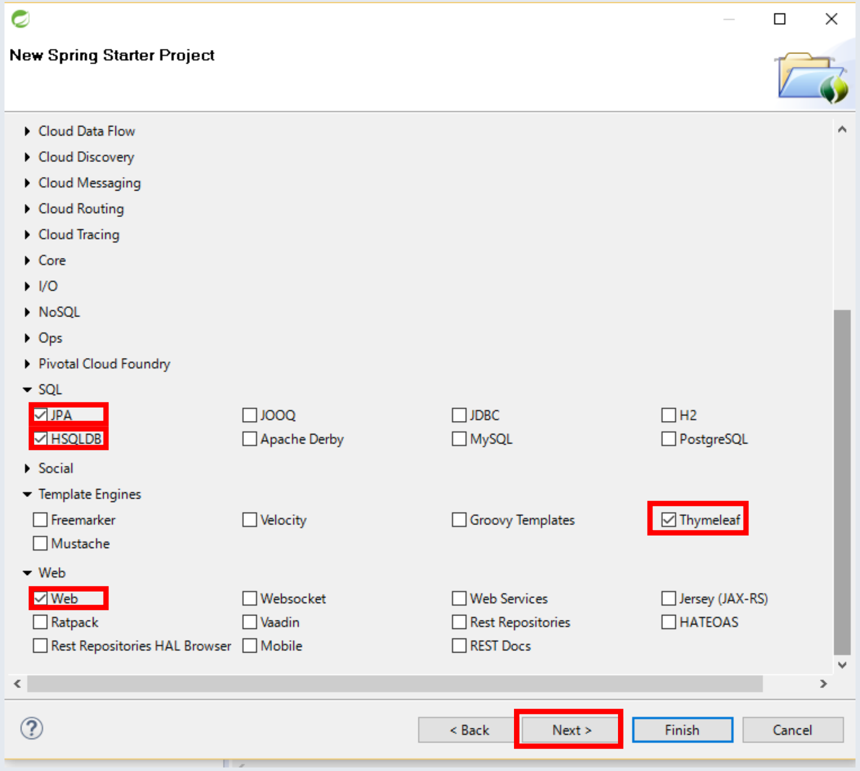
Name: Hello
Type: Gradle(STS)
Group: com.arms
Artifact: Hello
Package: com.arms

Press “Next>” button

1. Start creating a project

1-1. Start Project

- Open STS and File – New – Spring Starter Project



New Spring Starter Project

- Cloud Data Flow
- Cloud Discovery
- Cloud Messaging
- Cloud Routing
- Cloud Tracing
- Core
- I/O
- NoSQL
- Ops
- Pivotal Cloud Foundry
- SQL

☑ JPA	☐ JOOQ	☐ JDBC	☐ H2
☑ HSQLDB	☐ Apache Derby	☐ MySQL	☐ PostgreSQL

- Social
- Template Engines

☐ Freemarker	☐ Velocity	☐ Groovy Templates	☑ Thymeleaf
☐ Mustache			

- Web

☑ Web	☐ Websocket	☐ Web Services	☐ Jersey (JAX-RS)
☐ Ratpack	☐ Vaadin	☐ Rest Repositories	☐ HATEOAS
☐ Rest Repositories HAL Browser	☐ Mobile	☐ REST Docs	

Check the following checkbox,
JPA, HSQLDB under SQL

Thymeleaf under Template Engines.

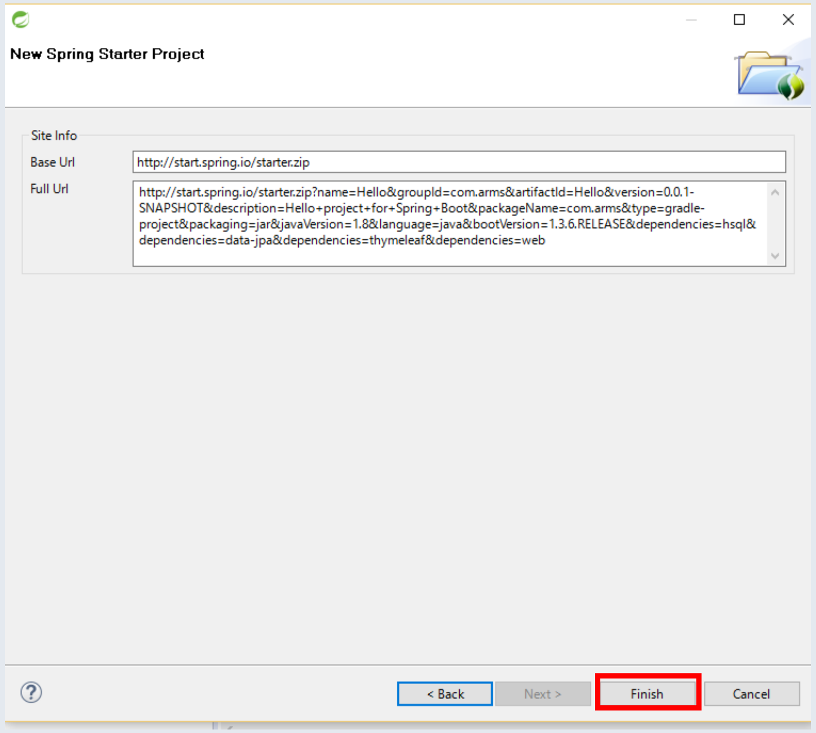
Web under Web

Press “Next>” button

1. Start creating a project

1-1. Start Project

- Open STS and File – New – Spring Starter Project


New Spring Starter Project

Site Info

Base Url: http://start.spring.io/starter.zip

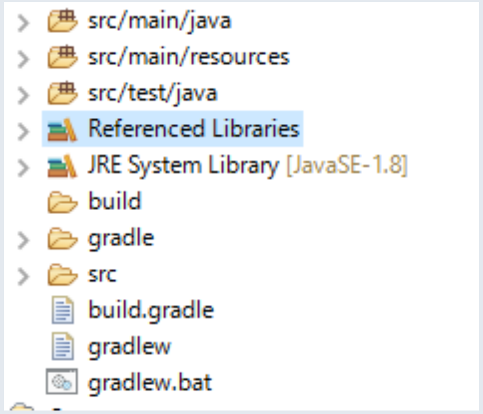
Full Url: http://start.spring.io/starter.zip?name=Hello&groupId=com.arms&artifactId=Hello&version=0.0.1-SNAPSHOT&description=Hello+project+for+Spring+Boot&packageName=com.arms&type=gradle-project&packaging=jar&javaVersion=1.8&language=java&bootVersion=1.3.6.RELEASE&dependencies=hsql&dependencies=data-jpa&dependencies=thymeleaf&dependencies=web

Press "Finish" button

1. Start creating a project

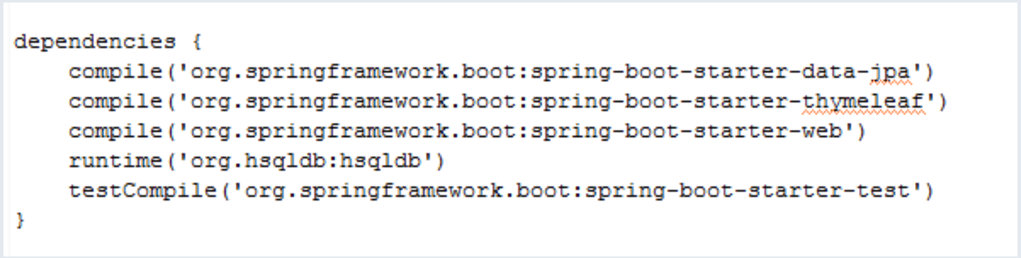
1-1. Start Project

- You will see the project structure like below



```
> src/main/java
> src/main/resources
> src/test/java
> Referenced Libraries
> JRE System Library [JavaSE-1.8]
  build
> gradle
> src
  build.gradle
  gradlew
  gradlew.bat
```

Make sure to have the version 1.8



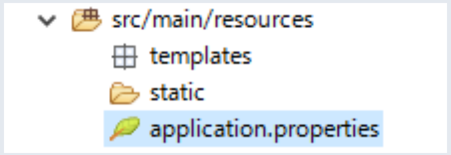
```
dependencies {
    compile('org.springframework.boot:spring-boot-starter-data-jpa')
    compile('org.springframework.boot:spring-boot-starter-thymeleaf')
    compile('org.springframework.boot:spring-boot-starter-web')
    runtime('org.hsqldb:hsqldb')
    testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

Open “build.gradle” and make sure to have “dependencies” you added during the spring starter project.

1. Start creating a project

1-1. Start Project

- Prepare to use thymeleaf



Open “application.properties”, and add the following lines

```
# Thymeleaf
spring.thymeleaf.mode=LEGACYHTML5
spring.thymeleaf.cache=false
spring.thymeleaf.encoding=UTF-8
```

☆spring.thymeleaf.mode=LEGACYHTML5
Without this setting, you have to follow the precise xhtml syntax.

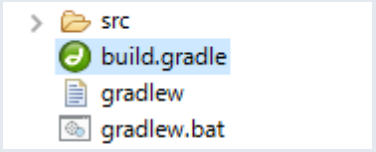
☆spring.thymeleaf.cache=false
Disabling thymeleaf cache for developing purpose.

application.properties is a configuration file for spring boot.

1. Start creating a project

1-2. Dependency management

- Prepare to use thymeleaf



Open “build.gradle” and add nekohtml

```
dependencies {
	compile('org.springframework.boot:spring-boot-starter-data-jpa')
	compile('org.springframework.boot:spring-boot-starter-thymeleaf')
	compile('org.springframework.boot:spring-boot-starter-web')
	runtime('org.hsqldb:hsqldb')
	testCompile('org.springframework.boot:spring-boot-starter-test')
	compile('net.sourceforge.nekohtml:nekohtml')
}
```

This setting is also needed to use legacyhtml5.

1. Start creating a project

1-2. Dependency management

- Dependency management

  One of the roles of build tool is compiling your source code into binary and packaging the binary.

  In order to make the above happen, Gradle needs to know the things which your project needs. These needs are called "dependencies"(of your project)

  If you add dependencies like below, Gradle will know the dependencies of your project and take care of finding these dependencies, and then make them available in your build and project.

```
dependencies {
            compile('org.springframework.boot:spring-boot-starter-data-jpa')
            compile('org.springframework.boot:spring-boot-starter-thymeleaf')
            compile('org.springframework.boot:spring-boot-starter-web')
            runtime('org.hsqldb:hsqldb')
            testCompile('org.springframework.boot:spring-boot-starter-test')
            compile('net.sourceforge.nekohtml:nekohtml')
}
```

Continue on next page

1. Start creating a project

1-2. Dependency management

- Dependency management

complie
The dependencies required to compile the production source of the project.
(To compile source of project, this dependency is needed)

runtime
The dependencies required by the production classes at runtime.
(To run the production class, this dependency is needed)

testCompile
The dependencies required to compile the test source of the project.

testRuntime
The dependencies required to run the tests.

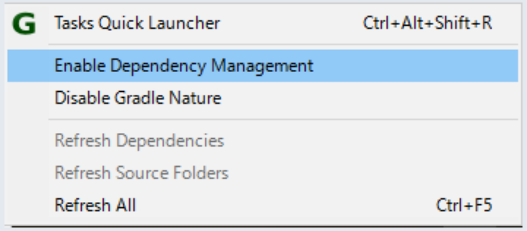
How Gradle resolves dependencies is that Gradle looks for them in a repository.
In build.gradle file, you will see the line like below.

```
repositories {
        mavenCentral()
}
```

1. Start creating a project

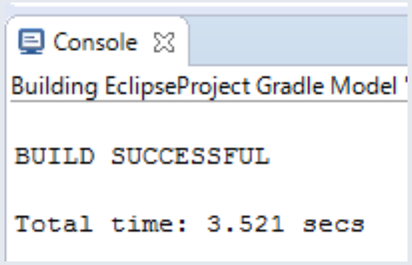
1-1. Dependency management

• Right click on your project, select Gradle(STS) – Enable Dependency Management from the context menu.



This will enable "Refresh Dependencies"

Select "Refresh Dependencies" to apply changes in the previous page.



```
Building EclipseProject Gradle Model

BUILD SUCCESSFUL

Total time: 3.521 secs
```

You will see "BUILD SUCCESSFUL" message in the Console screen.

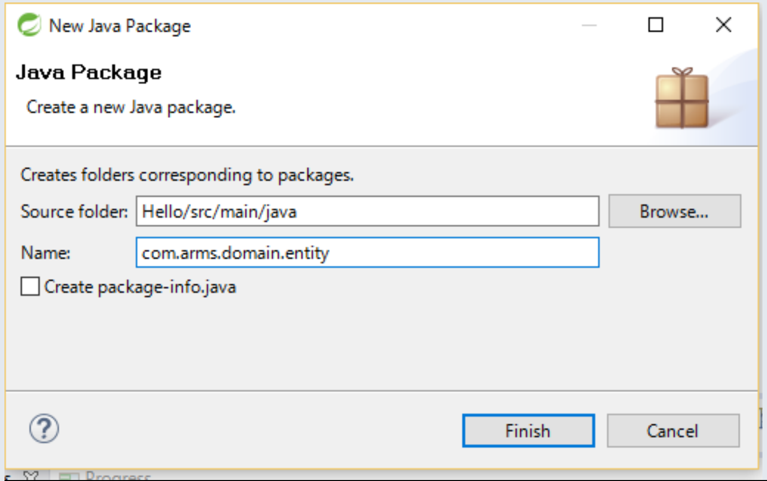
Referenced library is turned into Gradle dependencies.

Currently, Eclipse doesn't automatically update if build.gradle is updated.

2. Create Read and Create Users

2-1. Create user's list

- Right click on “com.arms”, new - package



Creates folders corresponding to packages.

Source folder: Hello/src/main/java

Name: com.arms.domain.entity

☐ Create package-info.java

Type “com.arms.domain.entity” in Name field

In the same way, add two more packages as below,

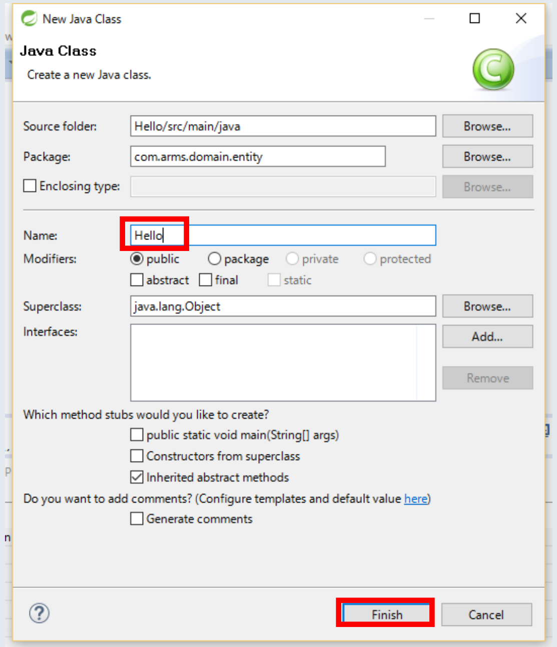
com.arms.app.hello

com.arms.domain.repository

2. Create and Read

2-1. Create user's list

- Right click on "com.arms.domain.entity", new – class to create a new class


New Java Class

Java Class
Create a new Java class.

Source folder: Hello/src/main/java
Package: com.arms.domain.entity
Enclosing type:
Name: Hello
Modifiers: [✓ public] package private protected
abstract final static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

Type "Hello" in the Name field, and click "Finish" button.

This will create an entity class under domain entity

2. Create and Read

2-1. Create user's list

- Double click on Hello.java and add the following code,

```
package com.arms.domain.entity;

public class Hello {

private int id;

private String name;

}
```

This is just a simple POJO….

Two things are needed to make this work as a JPA entity class…..

2. Create and Read

2-1. Create user's list

- The first thing is to add JPA @Entity annotation

```
package com.arms.domain.entity;

import org.hibernate.validator.constraints.NotEmpty;

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

@Entity
public class Hello {

   @Id
   @GeneratedValue
   private int id;

   @NotEmpty
   private String name;
}
```

JPA @Entity annotation tells that this class is mapped to a table, and each field of this class is mapped to the fields of the table.

@Id annotation tells that this field is a primary key field, and the value is automatically injected into the field of ObjectDB

@GeneratedValue annotation automatically allocates the primary key to the ObjectDB.

JPA: **J**ava **P**ersistence **A**PI makes it possible to persistently store data in a DB.
By using JPA, ORM(Object Relational Mapping) which maps DB records with Java Object can be used and deal with data as Object.

2. Create and Read

2-1. Create user's list

- The second thing is to add getters/setters for private fields.

```
package com.arms.domain.entity;

import lombok.Data;
import org.hibernate.validator.constraints.NotEmpty;

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

@Data
@Entity
public class Hello {

   @Id
   @GeneratedValue
   private int id;

   @NotEmpty
   private String name;
}
```

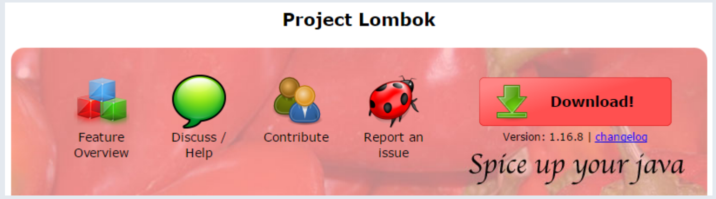
Since id and name fields are private, getters/setters should be created.

In our class, we use lombok to auto-create getters/setters.

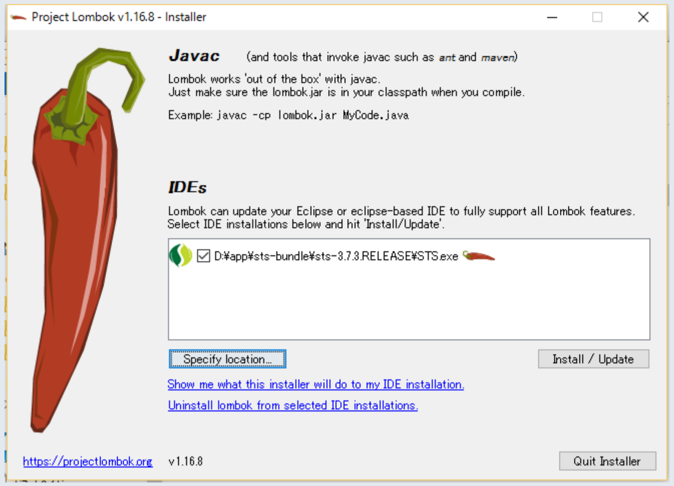
2. Create and Read

2-1. Create user's list

- To solve this lombok problem, go to https://projectlombok.org/ and click "Download!" button on the page



After downloading the lombok, double click on the lombok.jar



Project Lombok v1.16.8 - Installer

Javac (and tools that invoke javac such as *ant* and *maven*)

Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.

Example: `javac -cp lombok.jar MyCode.java`

IDEs

Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.

☑ D:¥app¥sts-bundle¥sts-3.7.3.RELEASE¥STS.exe

Specify location...
Install / Update
Show me what this installer will do to my IDE installation.
Uninstall lombok from selected IDE installations.
https://projectlombok.org
v1.16.8
Quit Installer

Lombok installer will automatically find IDEs installed on your PC.

Click "Install/Update" button

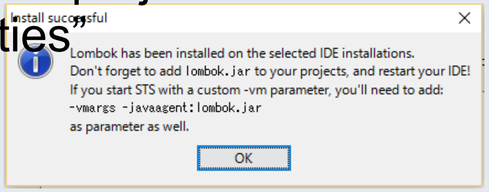
You will see "Install successful" message, and press "OK"

If you see "can't find IDE" message, please specify the location of your IDE"

2. Create and Read

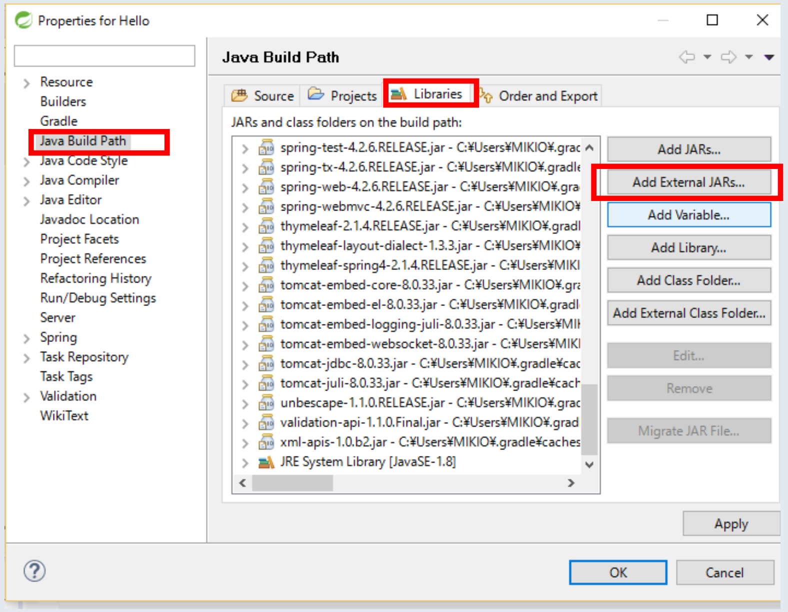
2-1. Create user's list

- As the message suggest, add lombok.jar to your project. Right-click on your project, and select "properties"



Install successful

Lombok has been installed on the selected IDE installations.
Don't forget to add `lombok.jar` to your projects, and restart your IDE!
If you start STS with a custom -vm parameter, you'll need to add:
`-vmargs -javaagent:lombok.jar`
as parameter as well.



Properties for Hello

Java Build Path

JARs and class folders on the build path:

```
spring-test-4.2.6.RELEASE.jar - C:¥Users¥MIKIO¥.grac
spring-tx-4.2.6.RELEASE.jar - C:¥Users¥MIKIO¥.gradle
spring-web-4.2.6.RELEASE.jar - C:¥Users¥MIKIO¥.gra
spring-webmvc-4.2.6.RELEASE.jar - C:¥Users¥MIKIO¥
thymeleaf-2.1.4.RELEASE.jar - C:¥Users¥MIKIO¥.gradl
thymeleaf-layout-dialect-1.3.3.jar - C:¥Users¥MIKIO¥
thymeleaf-spring4-2.1.4.RELEASE.jar - C:¥Users¥MIKI
tomcat-embed-core-8.0.33.jar - C:¥Users¥MIKIO¥.gra
tomcat-embed-el-8.0.33.jar - C:¥Users¥MIKIO¥.gradl
tomcat-embed-logging-juli-8.0.33.jar - C:¥Users¥MIK
tomcat-embed-websocket-8.0.33.jar - C:¥Users¥MIKI
tomcat-jdbc-8.0.33.jar - C:¥Users¥MIKIO¥.gradle¥cac
tomcat-juli-8.0.33.jar - C:¥Users¥MIKIO¥.gradle¥cach
unbescape-1.1.0.RELEASE.jar - C:¥Users¥MIKIO¥.grac
validation-api-1.1.0.Final.jar - C:¥Users¥MIKIO¥.grad
xml-apis-1.0.b2.jar - C:¥Users¥MIKIO¥.gradle¥caches
JRE System Library [JavaSE-1.8]
```

Select "Java Build Path" – "Libraries tab" – "Add External JARs…" button

You can find the lombok.jar from the location in the previous page.

Make sure to see lombok.jar in the Libraries.

2. Create and Read

2-1. Create user's list

- (**This is optional**) you can also add "dependencies" to "build.gradle"

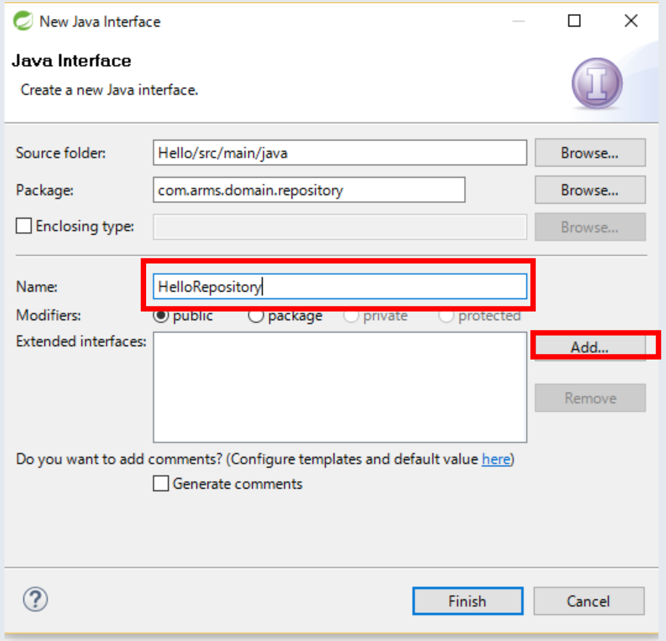
```
dependencies {
compile('org.springframework.boot:spring-boot-starter-data-jpa')
compile('org.springframework.boot:spring-boot-starter-thymeleaf')
compile('org.springframework.boot:spring-boot-starter-web')
runtime('org.hsqldb:hsqldb')
testCompile('org.springframework.boot:spring-boot-starter-test')
compile('net.sourceforge.nekohtml:nekohtml')
compile('org.projectlombok:lombok:1.16.6')
}
```

If you want to run this app on server and the previous lombok installation is out of reach from the virtual server, this dependency is necessary.

2. Create and Read

2-1. Create user's list

- Right click on "com.arms.domain.repository" New - Interface



New Java Interface

Java Interface

Create a new Java interface.

Source folder: Hello/src/main/java

Package: com.arms.domain.repository

☐ Enclosing type:

Name: HelloRepository

Modifiers: [✓ public] package private protected

Extended interfaces:

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

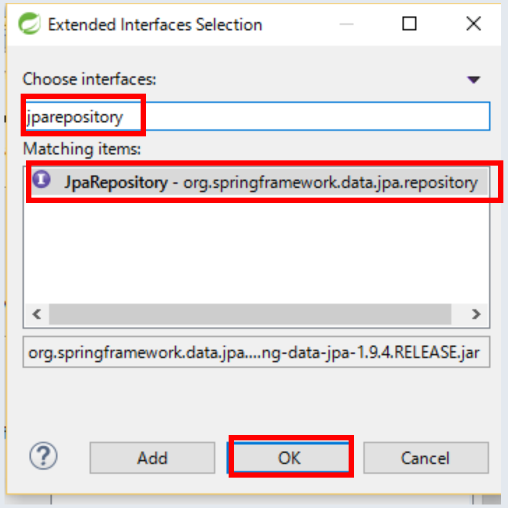
Type "HelloRepository" in the Name field

Click "Add" button to extend this interface.

2. Create and Read

2-1. Create user's list

- Extended Interfaces Selection dialog appears.



Choose interfaces:

jparepository

Matching items:

JpaRepository - org.springframework.data.jpa.repository

org.springframework.data.jpa....ng-data-jpa-1.9.4.RELEASE.jar

Type "jparepository" in the textbox, you will see JpaRepository in the Matching items.

Press "OK" to close this dialog, and click "Finish" button to create this interface.

2. Create and Read

2-1. Create user's list

- When you extend JpaRepository, you have to specify an entity class and its primary key field.

```
package com.arms.domain.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface HelloRepository extends JpaRepository<T, ID> {

}
```

Change <T, ID> to <Hello, Integer>

Add the following lines,
import com.arms.domain.entity.Hello;

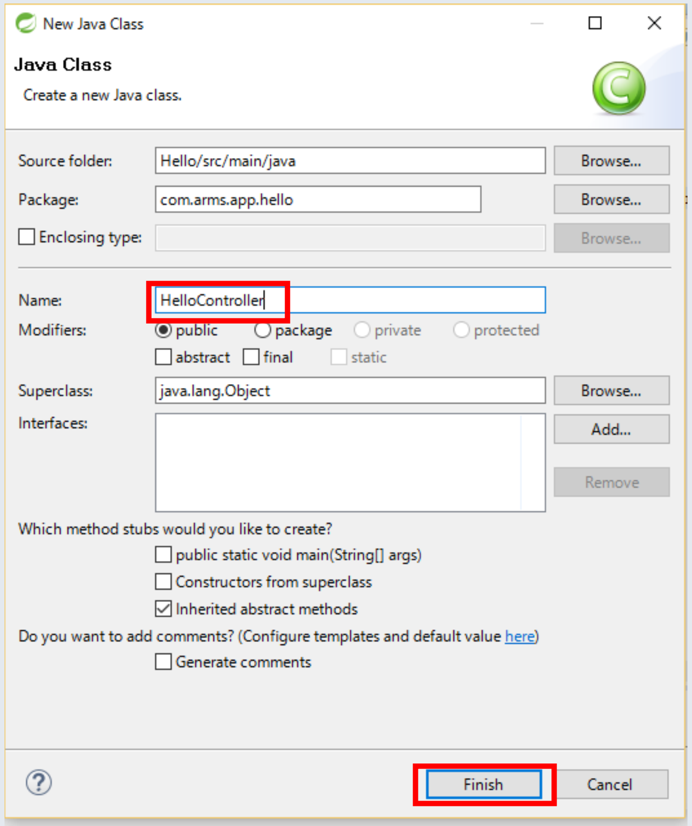
@Repository
import org.springframework.stereotype.Repository;
☆When you add @Repository, Spring recognizes this interface as Repository. You can do the basic DB control by JPA entitymanager

```
package com.arms.domain.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.arms.domain.entity.Hello;

@Repository
public interface HelloRepository extends JpaRepository<Hello, Integer> {

}
```

2. Create and Read

2-1. Create user's list

- Right-click on "com.arms.app.hello" new – class



New Java Class

Java Class

Create a new Java class.

Source folder: Hello/src/main/java

Package: com.arms.app.hello

☐ Enclosing type:

Name: HelloController

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Type "HelloController" in the Name field.

Press "Finish" button

2. Create and Read

2-1. Create user's list

- To create a controller class, you have to add the following line

```
package com.arms.app.hello;

import org.springframework.stereotype.Controller;

@Controller
public class HelloController {

}
```

1. You have to add @Controller annotation, and import.

```
package com.arms.app.hello;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/")
public class HelloController {

}
```

2. add @RequestMapping annotation, and import.

2. Create and Read

2-1. Create user's list

- Controllers in Spring works as entry points of view

```
package com.arms.app.hello;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/")
public class HelloController {

  @RequestMapping(value = "", method = RequestMethod.GET)
   public ModelAndView index(ModelAndView modelAndView) {
      modelAndView.addObject("list", helloRepository.findAll());
      modelAndView.setViewName("hello/list");
      return modelAndView;
   }

}
```

Add red lines to HelloController.java

2. Create and Read

2-1. Create user's list

- Important points

In Spring Boot, http requests are handled by a controller. **@Controller** annotation can make it easier to indentify these requests.

The **@RequestMapping** annotation will make sure that HTTP requests to "/" are mapped to the index() method. If you add @RequestMapping("/") to a class level, you can abbreviate when it is used in each method. In this case, @RequestMapping(value = "") for index() method means "/".

ModelAndView class

This class holds both model and view to make it possible for a controller to return both model and view **in a single return value**. You can also use addObject(), setViewName(), and etc… addObject() can store an object and give it to the view. setViewName() can render the corresponding view, ("hello/list") = hello folder, list.html

2. Create and Read

2-1. Create user's list

- Now you see "helloRepository.findAll() has a problem.

```
package com.arms.app.hello;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.beans.factory.annotation.Autowired;

@Controller
@RequestMapping("/")
public class HelloController {

   @Autowired
    HelloRepository helloRepository;
```

Add red lines to HelloController.java

2. Create and Read

2-1. Create user's list

- Spring Magic

You don't see any findAll() method in HelloRepository, but you can call findAll() method after adding @Autowired and HelloRepository type.

This is a magic you can use when you extend JpaRepositoy and add @Autowired annotation.

Other than findAll(), you can use findOne(), delete(), save() ,and etc… without creating methods.

With @Autowired, you can call the corresponding class without creating an instance by "new" operator.

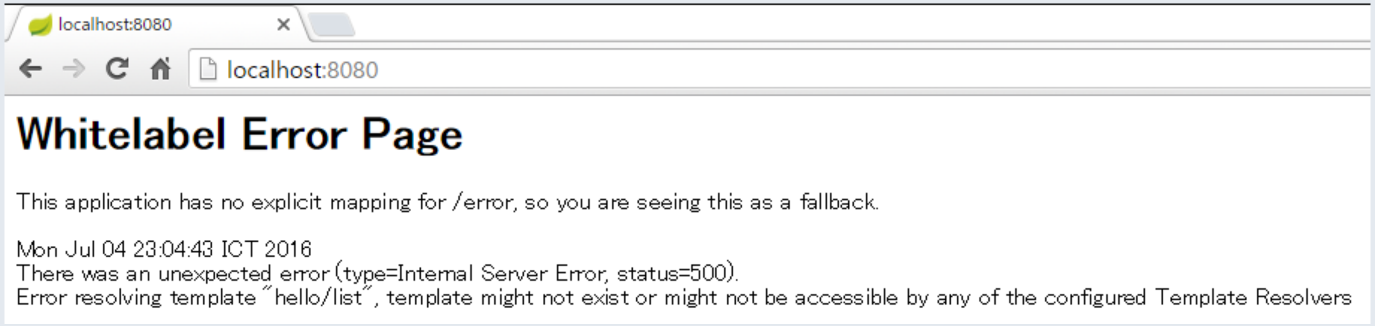
2. Create and Read

2-1. Create user's list

- Let's check if the code is properly working so far, Right-click on your project, Run As – Spring Boot App

Check "Console" screen if an error occurs.

If you don't see any errors, open your browser and type localhost:8080



Whitelabel Error Page

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

```
Mon Jul 04 23:04:43 ICT 2016
There was an unexpected error (type=Internal Server Error, status=500).
Error resolving template "hello/list", template might not exist or might not be accessible by any of the configured Template Resolvers
```

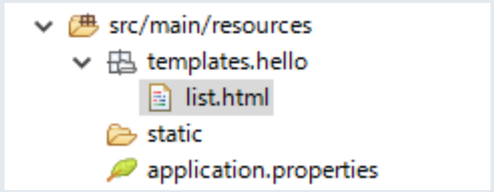
Controller tried to render list.html in hello folder. But list.html does not exsist

When you access @RequestMapping(value = "") = localhost:8080/, this tries to render modelAndView.setViewName("hello/list"); hello/list.html

2. Create and Read

2-1. Create user's list

- Now let's create hello folder and list.html. View files are normally created under **src/main/resources/templates**

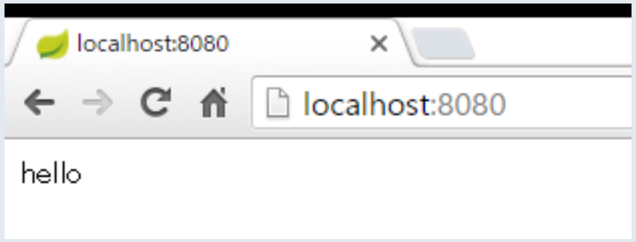


```
src/main/resources
  templates.hello
    list.html
  static
  application.properties
```

Create hello folder and list.html

Write something list.html
for example, hello

Restart your project and access localhost:8080 again, you will see "hello" messageC



```
hello
```

2. Create and Read

2-1. Create user's list

- Modify list.html as below

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Users List</title>
    <meta charset="utf-8">
    <link rel="stylesheet" media="screen" href="/stylesheets/main.css">
    <link rel="shortcut icon" type="image/png" href="/images/favicon.png">
    <script src="/javascripts/jquery-1.5.2.min.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
<h4>Users List</h4>
<ul th:each="hello : ${list}">
    <li>
        <span th:text="${hello.name}">AAA</span>
        <a href="#" th:href="'/update/' + ${hello.id} + '?form'" th:text="Edit">Edit</a>
        <a href="#" th:href="'/delete/' + ${hello.id}" th:onclick="'return confirm(\'Do you want to delete this user?\')'"
th:text="Delete">Delete</a>
    </li>
</ul>

<p><a href="/create?form">Create new user</a></p>
</body>
</html>
```

2. Create and Read

2-1. Create user's list

- Thymeleaf is a Java template engine for web environment and ideal for modern-day HTML5 JVM web development

Thymeleaf

http://www.thymeleaf.org/

With Thymeleaf, you can display application data on view.

To use Thymeleaf in Spring boot
1. Add the following in the gradle dependencies
compile('org.springframework.boot:spring-boot-starter-thymeleaf')

2. Declare xmlns(XML Namespece) in html
<html xmlns:th=*"http://www.thymeleaf.org"*>

2. Create and Read

2-1. Create user's list

- Thymeleaf Syntax

  Variable Expression ${…}

  Selection Variable Expression *{…}

  Message Expression #{…}

  URL Expression @{…}

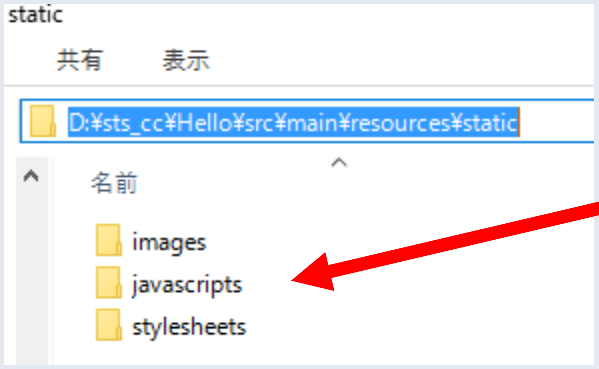
2. Create and Read

2-1. Create user’s list

- Add the following image, css, and js files

```
<link rel="stylesheet" media="screen" href="/stylesheets/main.css">
<link rel="shortcut icon" type="image/png" href="/images/favicon.png">
<script src="/javascripts/jquery-1.5.2.min.js" type="text/javascript" charset="utf-8"></script>
```

Normally, these files are added to **/src/main/resources/static**.
If you add those files under static folder, you only need **relative path** to the files.

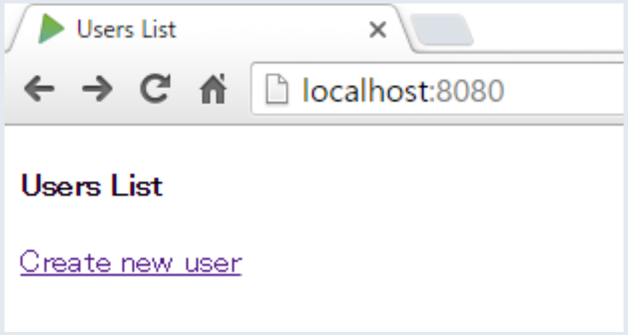


Go to static folder in your project, and copy folders and files.

2. Create and Read

2-1. Create user's list

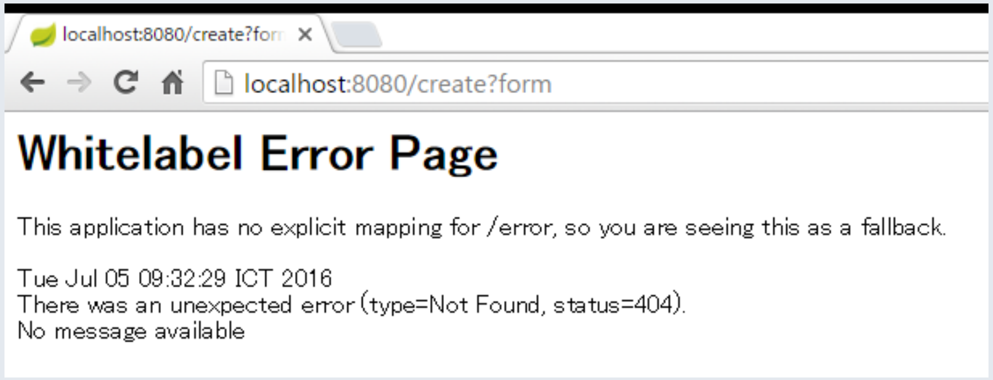
- Restart your project and access localhost:8080, you will see the screen as below,



Users List

Create new user

Click "Create new user"
But error occurs…



Whitelabel Error Page

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

Tue Jul 05 09:32:29 ICT 2016
There was an unexpected error (type=Not Found, status=404).
No message available

2. Create and Read

2-1. Create user's list

- Create a controller method and create.html.

```
@RequestMapping(value = "create", params = "form", method = RequestMethod.GET)
  public String create() {
     return "hello/create";
  }
```

Localhost:8080/create?form

Create "create.html" under hello folder.
return "hello/create";

hello
- create.html

2. Create and Read

2-1. Create user's list

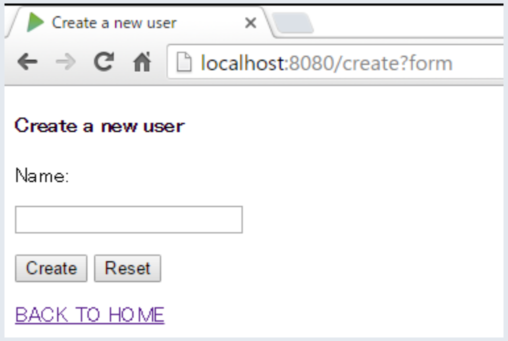
- Add the following codes to create.html.

```
<!DOCTYPE html>
<html>
<head>
    <title>Create a new user</title>
    <meta charset="utf-8">
    <link rel="stylesheet" media="screen" href="/stylesheets/main.css">
    <link rel="shortcut icon" type="image/png" href="/images/favicon.png">
    <script src="/javascripts/jquery-1.5.2.min.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
<h4>Create a new user</h4>
<form action="/create" method="POST" accept-charset="utf-8" enctype="application/x-www-form-urlencoded" >
    <label for="name">Name:</label>
    <p>
        <input id="name" type="text" name="name" />
    </p>
    <button type="submit" id="ok">Create</button>
    <button type="reset" id="reset">Reset</button>
</form>
<p>
    <a href="/" id="back" >BACK TO HOME</a>
</p>
</body>
</html>
```

2. Create and Read

2-1. Create user's list

- Restart your project and access localhost:8080, you will see the screen as below,

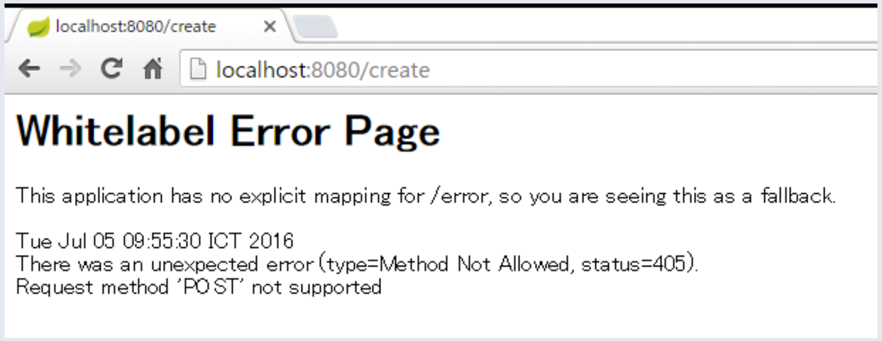


Create a new user

Name:

Type something in the Name textbox and press "Create" button

But error occurs…



Whitelabel Error Page

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

Tue Jul 05 09:55:30 ICT 2016
There was an unexpected error (type=Method Not Allowed, status=405).
Request method 'POST' not supported
```

2. Create and Read

2-1. Create user's list

- Create a controller method to respond to this form action.

```
<form action="/create" method="POST"……………
```

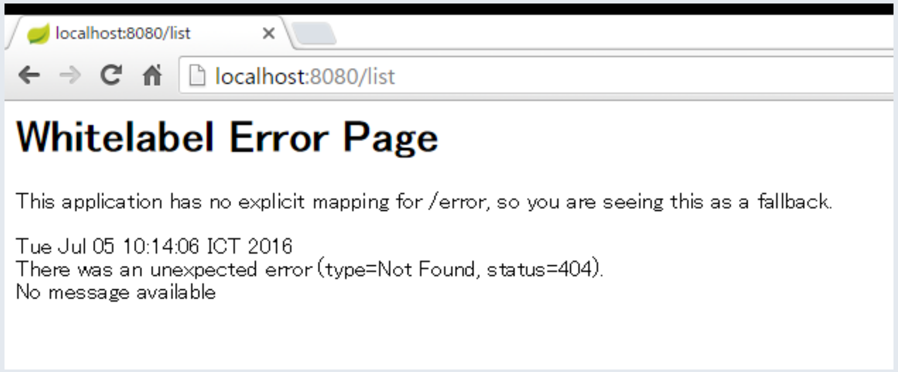
```
@RequestMapping(value = "create", method = RequestMethod.POST)
  public String doneCreate(@RequestParam("name") String name) {
     Hello hello = new Hello();
     hello.setName(name);
     helloRepository.save(hello);
     return "redirect:/list";
```

```
<form action="/create" method="POST"
↓
method = RequestMethod.POST

<input id="name" type="text" name="name" />
(@RequestParam("name") String name
hello.name = name;

return "redirect:/list";
↓
localhost:8080/list
```

2. Create and Read

2-1. Create user’s list

- Restart your project and access localhost:8080, type something in the textbox, and press “Create” button



localhost:8080/list

localhost:8080/list

```
Whitelabel Error Page

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

Tue Jul 05 10:14:06 ICT 2016
There was an unexpected error (type=Not Found, status=404).
No message available
```

But this time error message will change

2. Create and Read

2-1. Create user’s list

- Create a controller method to respond to this entry point.

```
http://localhost:8080/list
```

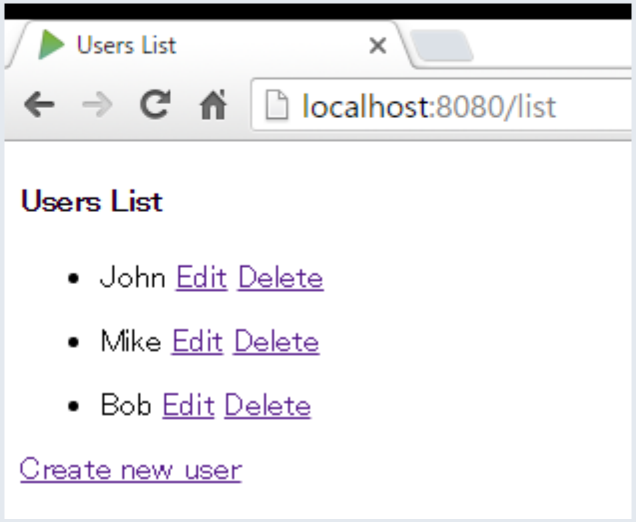
```
@RequestMapping(value = "list", method = RequestMethod.GET)
 public ModelAndView list(ModelAndView modelAndView) {
    modelAndView.addObject("list", helloRepository.findAll());
    modelAndView.setViewName("hello/list");
    return modelAndView;
 }
```

```
return "redirect:/list";
localhost:8080/list

@RequestMapping(value = "list“

modelAndView.setViewName("hello/list");
return modelAndView;
```

2. Create and Read

2-1. Create user's list

- Restart your project and access localhost:8080, now you can create many users



Users List

- John Edit Delete
- Mike Edit Delete
- Bob Edit Delete

Create new user

If you see the screen like the left figure, so far so good!!!

2. Create and Read

2-1. Create user's list

- Important points

```
@RequestMapping(value = "list", method = RequestMethod.GET)
 public ModelAndView list(ModelAndView modelAndView) {
    modelAndView.addObject("list", helloRepository.findAll());
    modelAndView.setViewName("hello/list");
    return modelAndView;
 }
```

```
<ul th:each="hello : ${list}">
   <li>
      <span th:text="${hello.name}">AAA</span>
      <a href="#" th:href="'/update/' + ${hello.id} + '?form'" th:text="Edit">Edit</a>
      <a href="#" th:href="'/delete/' + ${hello.id}" th:onclick="'return confirm(\'Do you want to delete this user?\')'" th:text="Delete">Delete</a>
   </li>
</ul>
```

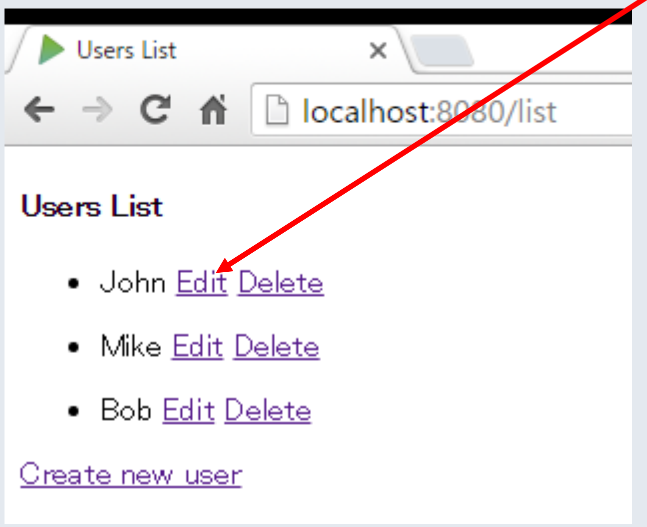
findAll() method fetches all data in database, and store them in "list" variable and give it to the view.
And then, th:each repeats ${list} as hello.

th:each, th:href, th:text and etc. are thymeleaf format.

3. Update and Delete

3-1. Create update

- Back to Users List and click “Edit” button

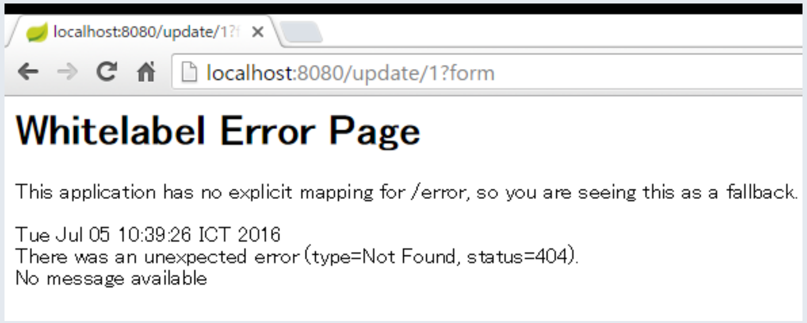


Users List

- John Edit Delete
- Mike Edit Delete
- Bob Edit Delete

Create new user

You will see an error message again.



Whitelabel Error Page

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

Tue Jul 05 10:39:26 ICT 2016
There was an unexpected error (type=Not Found, status=404).
No message available

3. Update and Delete

3-1. Create update

- Add a controller method to edit users and create the corresponding update.html

```
@RequestMapping(value = "update/{id}", params = "form", method = RequestMethod.GET)
  public ModelAndView update(@PathVariable("id") int id, ModelAndView modelAndView) {
     modelAndView.addObject("hello", helloRepository.findOne(id));
     modelAndView.setViewName("hello/update");
     return modelAndView;
  }
```

```
http://localhost:8080/update/1?form

@RequestMapping(value = "update/{id}", params = "form",
```

modelAndView.addObject("hello", helloRepository.findOne(id));
This time, you want to update a specific user's name, and have to get it by "id".

"@PathVariable("id") **int** id" can get a URL parameter.

modelAndView.setViewName("hello/update");
hello folder update.html

3. Update and Delete

3-1. Create update

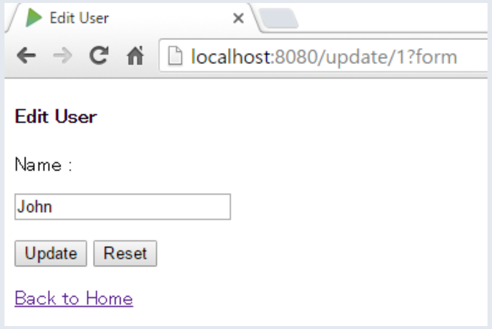
- Add the following code to update.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
   <title>Edit User</title>
   <meta charset="utf-8">
   <link rel="stylesheet" media="screen" href="/stylesheets/main.css">
   <link rel="shortcut icon" type="image/png" href="/images/favicon.png">
   <script src="/javascripts/jquery-1.5.2.min.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
   <h4>Edit User</h4>
   <form action="/update" method="POST" accept-charset="utf-8" enctype="application/x-www-form-urlencoded"
class="form-horizontal" >
      <input type="text" name="id" th:value="${hello.id}" hidden/>
      <div>
         <label for="name">Name :</label>
         <p>
            <input id="name" type="text" name="name" th:value="${hello.name}"/>
         </p>
      </div>
      <button type="submit" id="ok">Update</button>
      <button type="reset" id="reset">Reset</button>
      <p>
         <a href="/" id="back" >Back to Home</a>
      </p>
   </form>
</body>
</html>
```

3. Update and Delete

3-1. Create update

- Restart your project and go to a user's edit page



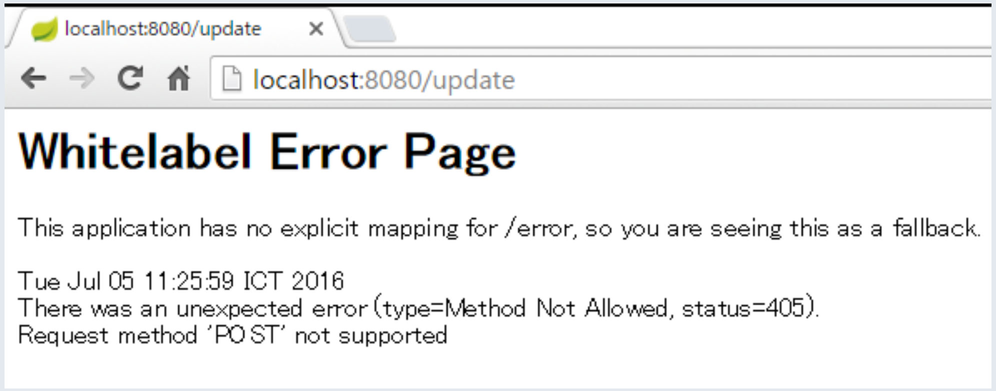
Edit User

Name :

John

Edit the user name as you like, and press "Update" button.

But error occurs….



Whitelabel Error Page

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

```
Tue Jul 05 11:25:59 ICT 2016
There was an unexpected error (type=Method Not Allowed, status=405).
Request method 'POST' not supported
```

3. Update and Delete

3-1. Create update

- Add a controller method to update users

```
@RequestMapping(value = "update", method = RequestMethod.POST)
  public String update(@RequestParam("id") int id, @RequestParam String name) {
     Hello hello = helloRepository.findOne(id);
     hello.setName(name);
     helloRepository.save(hello);
     return "redirect:/list";
  }
```

```
<form action="/update" method="POST"
@RequestMapping(value = "update", method = RequestMethod.POST)
```

helloRepository.save(hello);

save() method again for updating???? save() can automatically check data status and choose to create or update data.

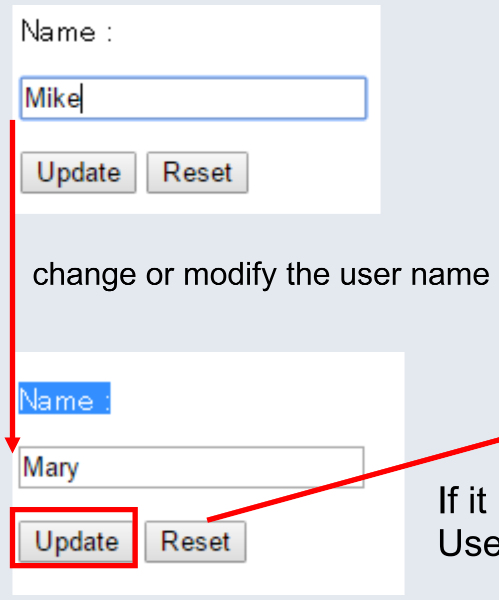
(@RequestParam("id") **int** id, @RequestParam String name)

@RequestParam receives values from the form.

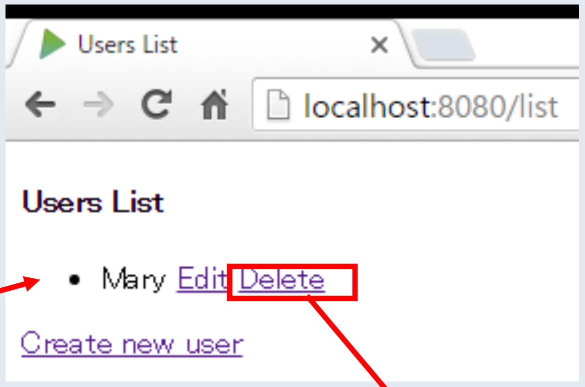
3. Update and Delete

3-1. Create update

- Restart your project and try to update a user.



change or modify the user name



If it is properly working so far, you will see Users List after updating.

But…. an error occurs if you click "Delete" button

3. Update and Delete

3-2. Create delete

- Add a controller method to delete users

```
@RequestMapping(value = "delete/{id}", method = RequestMethod.GET)
  public String delete(@PathVariable("id") int id) {
     helloRepository.delete(id);
     return "redirect:/list";
  }
```

```
<a href="#" th:href="'/delete/' + ${hello.id}" th:onclick="'return confirm(\'Do you want to
delete this user?\')'" th:text="Delete">Delete</a>
```

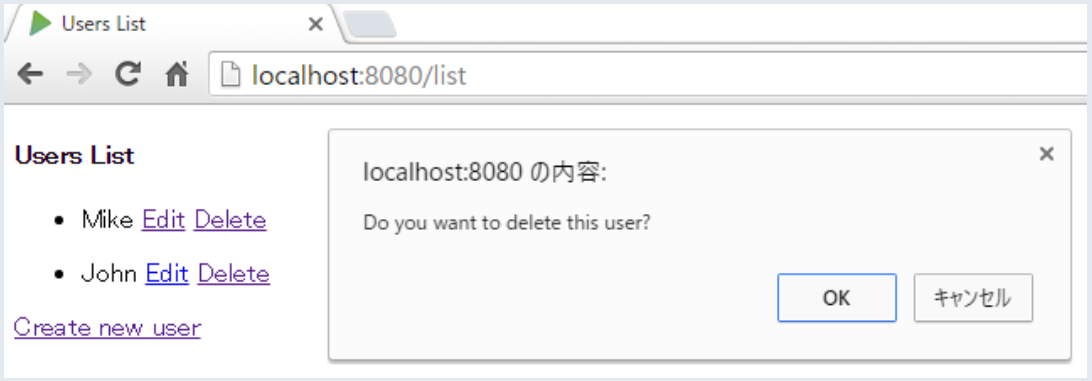
helloRepository.delete(id);
This time also has to delete a specific user by id.

delete/{id}
↓
(@PathVariable("id") **int** id
↓
helloRepository.delete(id);

3. Update and Delete

3-2. Create delete

- Restart your project and try to delete a user



Users List

localhost:8080/list

Users List

- Mike Edit Delete
- John Edit Delete

Create new user

localhost:8080 の内容:

Do you want to delete this user?

You will see the confirmation message and click “OK” to delete the user.

ARMS

Your Idea Leads Your Ideals

homepage: http://arms-asia.com/
facebook: https://www.facebook.com/arms.asia?fref=ts