Create to-do app called Indivilister with SPRING BOOT

Presented by ARMS（THAILAND）Co., Ltd.

Index

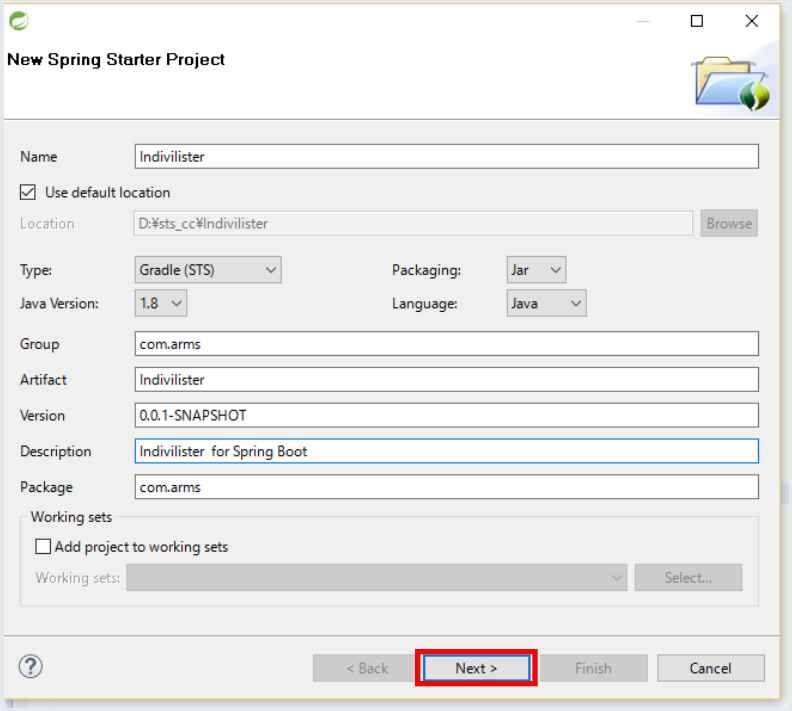
Note

- Indivilister means "Individual + lister“

1. Start creating a project

1-1. Start Project

- Open STS and File – New – Spring Starter Project



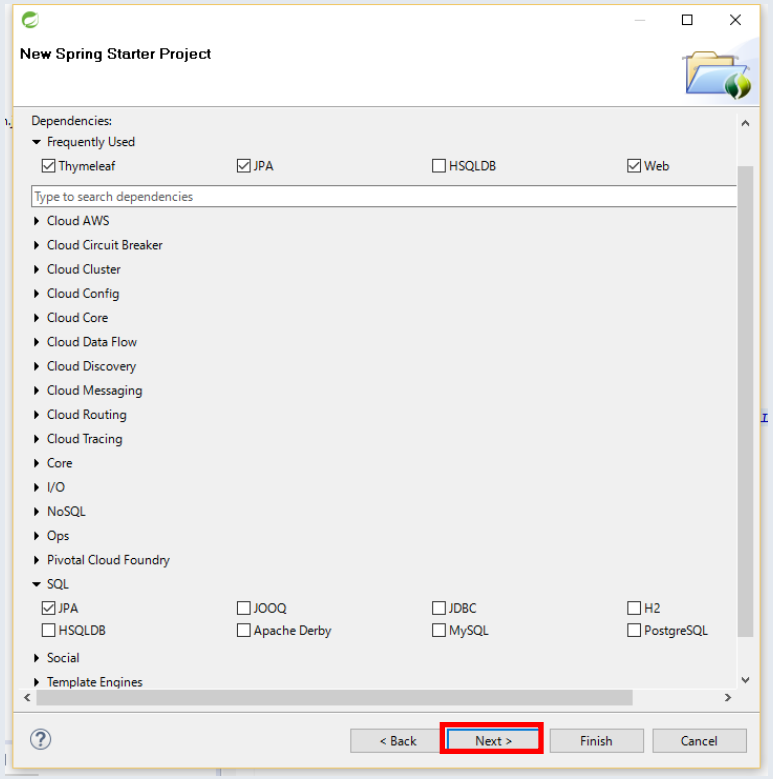
Name: Indivilister
Type: Gradle(STS)
Group: com.arms
Artifact : Indivilister
Version: 0.0.1-SNAPSHOT
Description:
Indivilister for Spring Boot
Package: com.arms

And press “Next>” button

1. Start creating a project

1-1. Start Project

- Open STS and File – New – Spring Starter Project



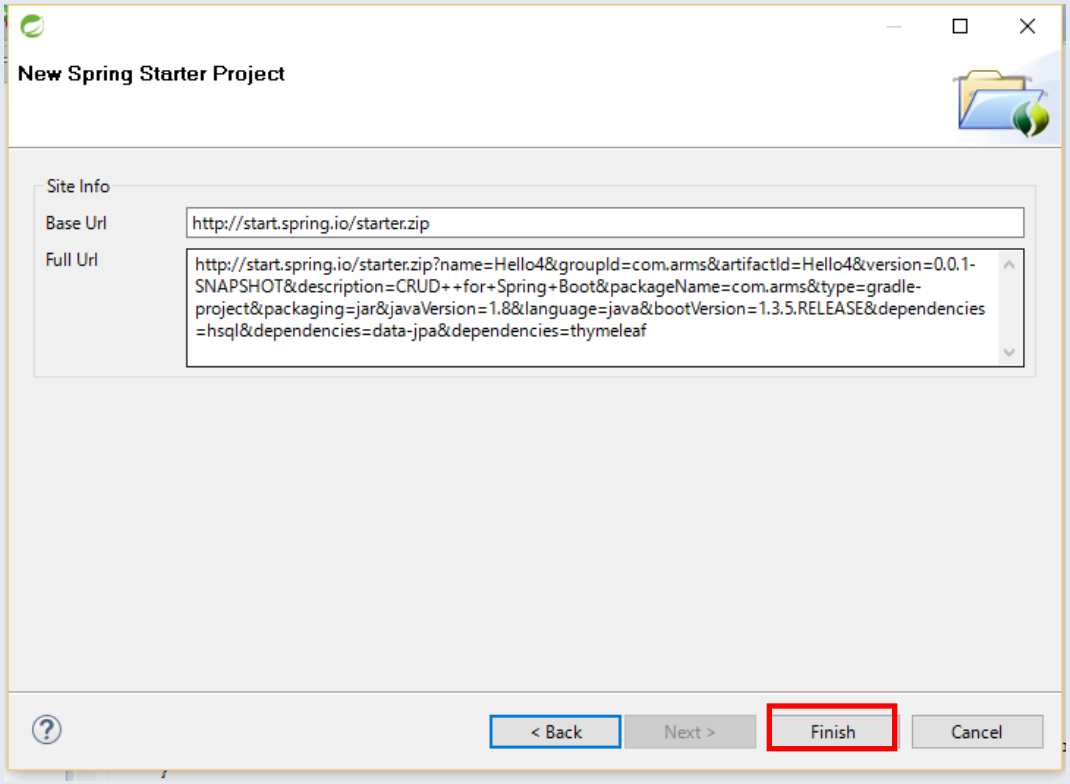
Check the following checkbox
Thymeleaf
JPA
Web

And press “Next>” button

1. Start creating a project

1-1. Start Project

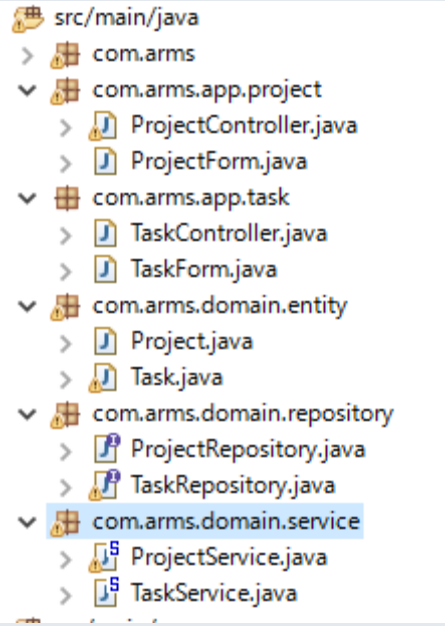
- Open STS and File – New – Spring Starter Project



1. Start creating a project

1-1. Start Project

- You will create the project structure as below



Create packages as below

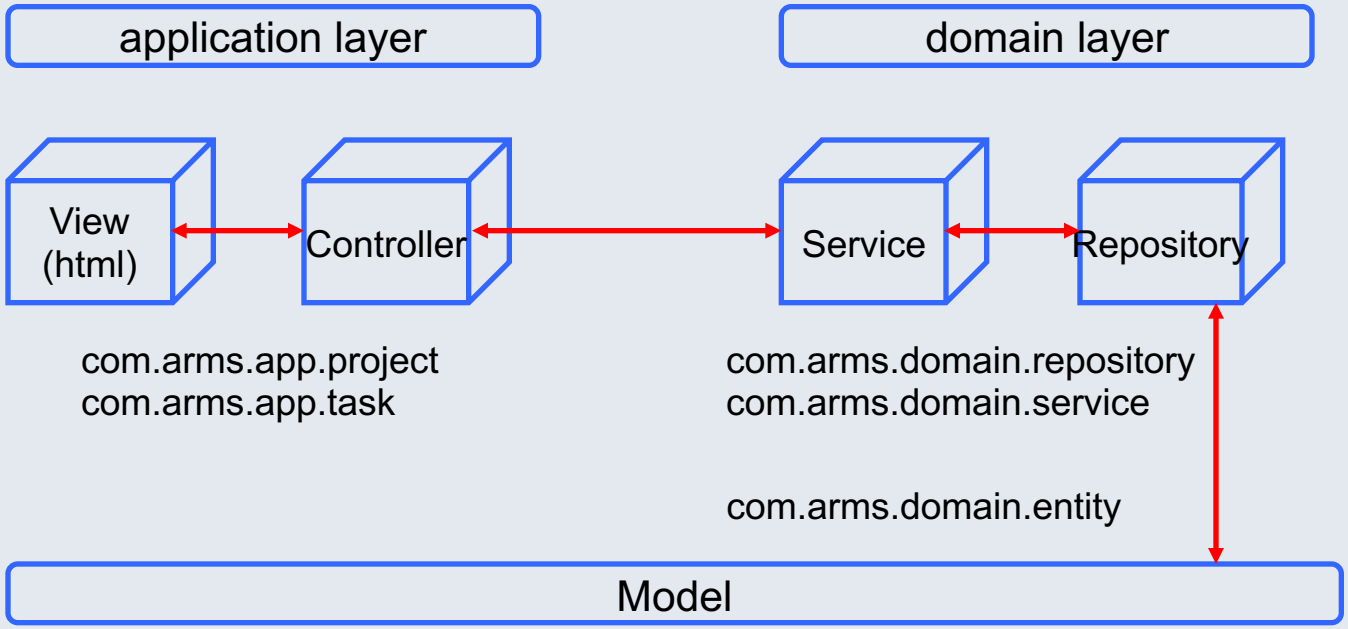
com.arms.app.project
com.arms.app.task
com.arms.domain.entity
com.arms.domain.repository
com.arms.domain.service

<u>Make sure to spell the package name right.</u>

1. Start creating a project

1-1. Start Project

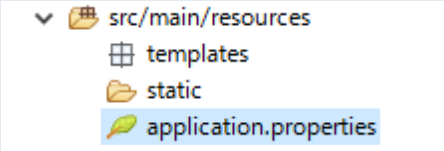
- You will create layered architecture as below



1. Start creating a project

1-1. Start Project

- Prepare to use Thymeleaf and Database



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

```
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/todo?useUnicode=true&characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=manager

spring.jpa.database=MYSQL
spring.jpa.show-sql=true

spring.jpa.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

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

1. Start creating a project

1-1. Start Project

- Connection to a production database

Red letters should be changed according to your environment.

```
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://hostname:port/Database Name?useUnicode=true&characterEncoding=UTF-8
spring.datasource.username=a user in database
spring.datasource.password=the above user's password

spring.jpa.database=MYSQL
spring.jpa.show-sql=true

spring.jpa.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
What dialect corresponds with MySQL. Hibernate has a dialect auto-detection mechanism
https://docs.jboss.org/hibernate/orm/3.5/api/  (other example)

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

1. Start creating a project

1-1. Start Project

- Modify build.gradle for Thymeleaf and Database connection

Red part should be added to dependencies.

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

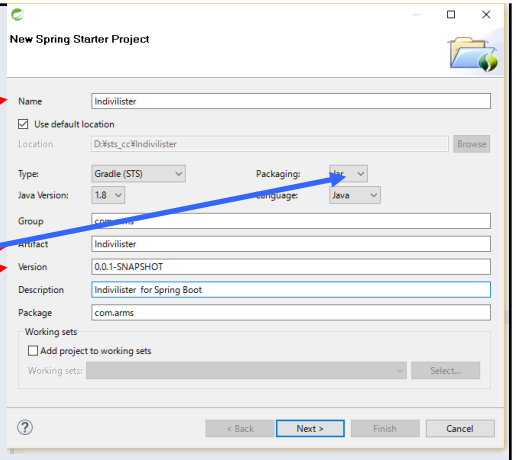
1. Start creating a project

1-1. Start Project

- Starter project parameters.

Name : Indivilister is project name

Artifact : Indivilister
Version: 0.0.1-SNAPSHOT
Packaging: Jar




```
build.gradle
jar {
        baseName = 'indivilister‘
        version = '0.0.1-SNAPSHOT‘
}
```

When gradle packaging your project into Java Archive(JAR)
baseName + version .jar
Indivilister-0.0.1-SNAPSHOT.jar

☆JAR file contains class files and related resources for your application. In general, archive enables you to bundle multiple files into a single file

Group is to identify the project, normally, package route of the project name is used.

1. Start creating a project

1-1. Start Project

- Create entity class for Indivilister's Project. Right-click on com.arms.domain.entity. New - Class

Type Project in the name textbox and click "finish" button

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

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

import javax.persistence.*;
import java.util.Date;
import java.util.List;

@Entity
@Data
@Table(name= "project")
public class Project {

    @GeneratedValue
    @Id
    private int id;

    @NotEmpty
    private String name;
```

```
    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
    private List<Task> taskList;

    private Date createdDate;

    private Date updatedDate;
}
```

1. Start creating a project

1-1. Start Project

- Create entity class for Indivilister's Task. Right-click on com.arms.domain.entity. New - Class

Type Task in the name textbox and click "finish" button

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

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

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.util.Date;
import java.util.List;

@Entity
@Data
@Table(name= "task")
public class Task {

    @GeneratedValue
    @Id
    private int id;
```

```
    @ManyToOne
    @JoinColumn(name = "project_id")
    private Project project;

    @NotEmpty
    private String name;

    @NotNull
    private boolean status;

    private Date createdDate;

    private Date updatedDate;
}
```

1. Start creating a project

1-1. Start Project

- Important points

@Table(name= "project")
specify a table name mapped to the entity. If **@Table** is not defined, the default values is used which is the entity class name.

@NotEmpty Null and “” will be an error.

One-To-Many Mapping

```
@OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
  private List<Task> taskList;
```

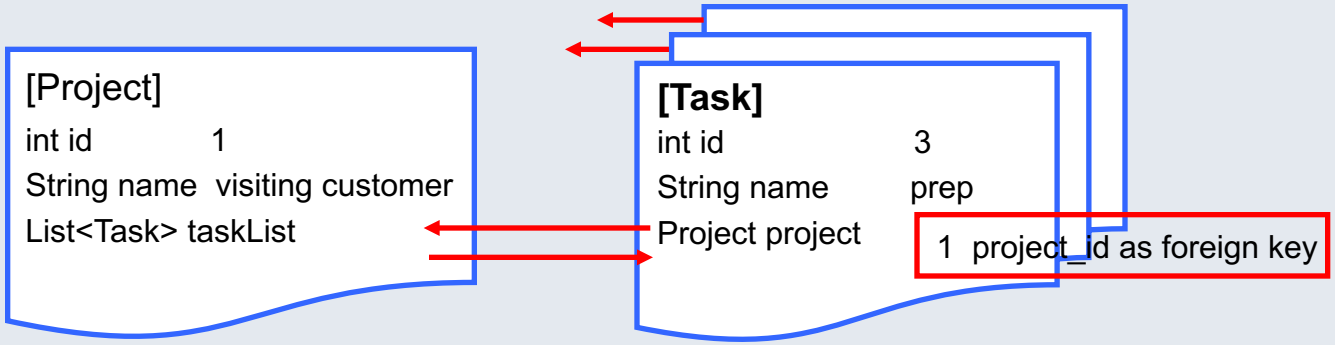
Project entity has a reference to List<Task>, as one project can have many tasks. Task entity has a reference to Project. In order to make Project to Task One-to-Many, put @OneToMany annotation over List<Task>. This signifies that one project can have many tasks. On Task side, @ManyToOne should be put over Project reference. This signifies that many tasks belong to one project. Since “project_id” has to be made foreign key in Task table. Put @JoinColumn annotation over Project property in Task entity and provide the name of the column which will hold project_id values in Task table as foreign key.

```
@ManyToOne
@JoinColumn(name = "project_id")
private Project project;
```

1. Start creating a project

1-1. Start Project

- Important point: mappedBy
  Each project can have zero to many tasks, and if each task has project information, you will know which project each task belongs to. To know where each task is coming from, add a foreign key to task table



One-To-Many Mapping

```
@OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
  private List<Task> taskList;

@ManyToOne
@JoinColumn(name = "project_id")
private Project project;
```

Unlike One-to-One and Many-to-Many, in One-to-Many/Many-to-One case, "mappedBy" has to be put on One-to-Many side and indicate Many side.

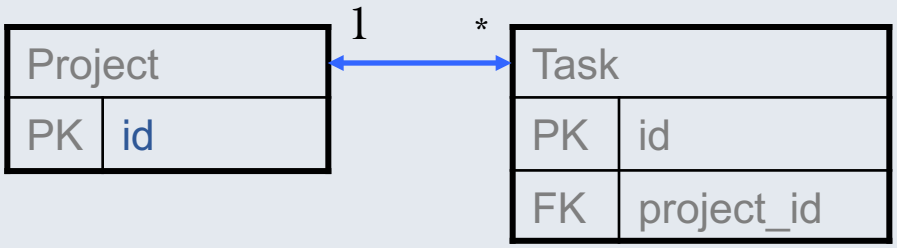
Continue on next page

1. Start creating a project

1-1. Start Project

- Important point: mappedBy attribute

In One-to-Many/Many-to-One relationship, mappedBy is used for non-owner entity class, and tells that the owner is in the other entity.



@ManyToOne should be defined on the owner entity class.
If @JoinColumn is not defined, the name is composed of the name of relationship property in the entity(in this case "project") + an underscore + the name of PK on the other field.
In this way, the name would be "project_id". @JoinColumn should be defined on the ManyToOne attribute side.

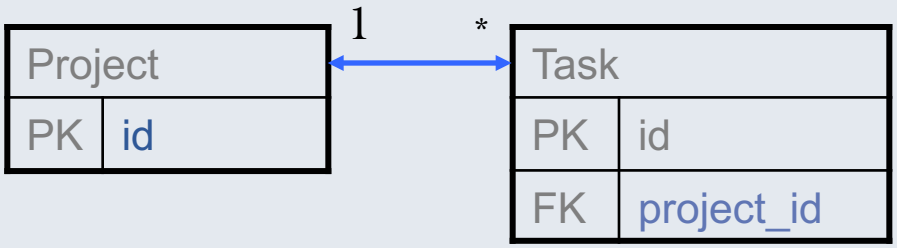
```
@ManyToOne
@JoinColumn(name = "project_id")
private Project project;
```

1. Start creating a project

1-1. Start Project

- Important point: cascade attribute

Common way to use cascade is Parent and Child association like a One-To-Many and a Many-To-One relationship in which the cascade should be used for the One-To-Many side.
** you have to carefully use cascade, (especially,when IDE auto-creates it.)



With cascade, parent operation is transmitted to its child. If a project id is removed, FK(project_id) is no longer needed.

```
@OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
  private List<Task> taskList;
```

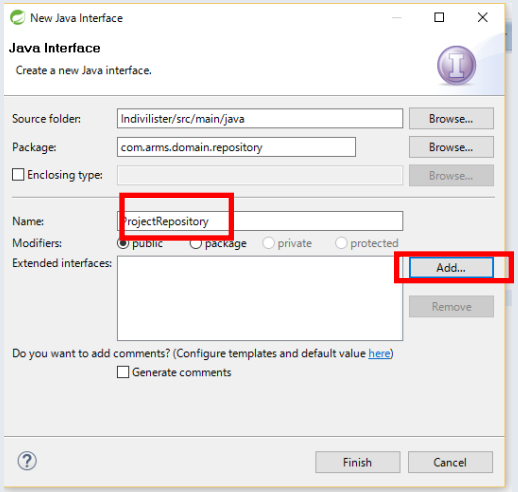
Cascade Types: MERGE, PERSIST, REFRESH, REMOVE and ALL. ALL is applied for all types.

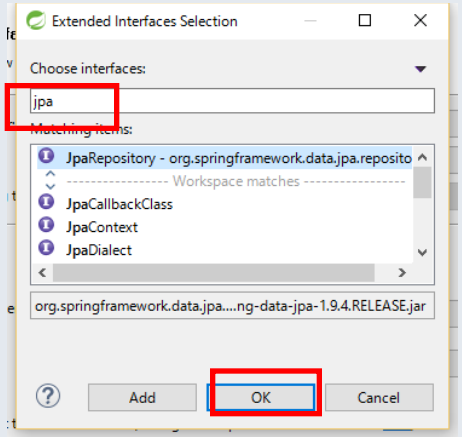
1. Start creating a project

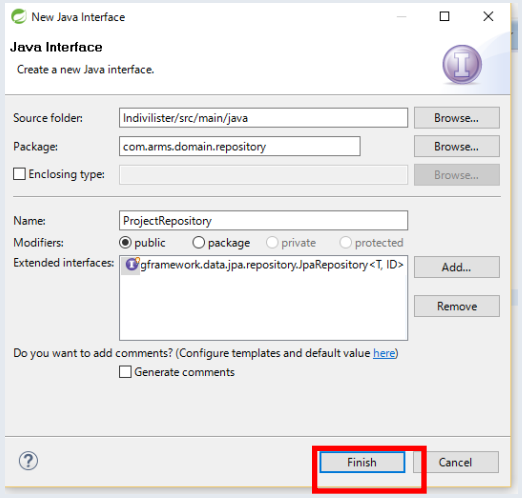
1-1. Start Project

- Create repository interface for Project. Right-click on com.arms.domain.repository. New - interface

  Type Project in the name textbox and click “add” button to choose interface to extend.







1. Start creating a project

1-1. Start Project

- Add @Repository annotation over interface, add Entity name and its primary key field. ProjectRepository.java

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

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

@Repository
public interface ProjectRepository extends JpaRepository<Project, Integer> {

}
```

1. Start creating a project

1-1. Start Project

- Create ProjectController under com.arms.app.project as below

```
package com.arms.app.project;

import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.arms.domain.entity.Project;

@Controller
@RequestMapping("/project")
public class ProjectController {

@RequestMapping(value = "", method = RequestMethod.GET)
    public String index(Model model) {
        return "project/list";
    }
}
```

@RequestMapping("/project") for class level

@RequestMapping(value = "", for method level

localhost:8080/project
↓
project/list.html

return "project/list";

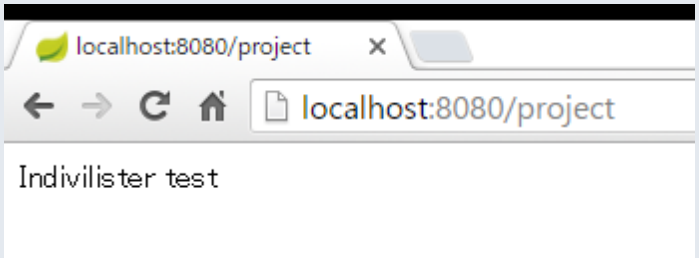
1. Start Creating a project

1-1. Start Project

- Create list.html under templates.project, write something in the list.html. Start your project and access server:8080/project

list.html
Indivilister test

You will see the contents of list.html



About accessing, if you are running on server

server:8080/project
↓
192.168.33.10:8080/project

if you are running on local

server:8080/project
↓
localhost:8080/project

2. Use thymeleaf to create view

2-1. Create view

- Modify list.html for Indivilister as below,

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>INDIVILISTER : Projects</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css" charset="utf-8">
   <link rel="stylesheet" type="text/css" href="/css/custom.css" charset="utf-8">
</head>
<body>
<main>
   <!-- START Navigation bar -->
   <nav th:include="common/nav :: nav"></nav>

   <!-- END Navigation bar -->
   <!-- START Header -->
   <div class="header-custom">
      <div class="container">
         <h2>Projects</h2>
         <p>Simple todo app by Spring boot and Bootstrap</p>
      </div>
   </div>
   <!-- END Header -->
```

Continue on next page

2. Use thymeleaf to create view

2-1. Create view

- Modify list.html for Indivilister as below,

```
<!-- START Content -->
  <div class="container">
    <div class="row">
      <div class="col-xs-12">
        <a th:href="@{/project/create?form}" class="btn btn-success">Let's create a new project</a>
      </div>
    </div>
    <div class="row">
      <div class="col-xs-12">
        <div class="col-lg-4" style="padding: 0;" th:each="project, stat : ${projectList}">
          <div class="materialCard success">
            <header>
              <a th:href="'/task/list/' + ${project.id}" class="btn btn-success">
                <span th:text="${project.name}">project0001</span>
                <span class="badge" th:text="${projectRemainingTaskMap.get(project.id)} + '/' +
${project.taskList.size()}">1/2</span>
              </a>
            </header>
            <a th:href="'/project/edit/' + ${project.id}">Edit</a>
            <a th:href="'/project/delete/' + ${project.id}" onclick="return confirm('Delete this project?');">Delete</a>
          </div>
        </div>
      </div>
    </div>
  </div>
  <!-- END Content -->
```

Continue on next page

2. Use thymeleaf to create view

2-1. Create view

- Modify list.html for Indivilister as below,

```
<div class="push"></div>
</main>

<!-- START Footer -->
<span th:include="common/footer :: footer"></span>
<!-- START Footer -->

<script type="text/javascript" language="javascript" charset="_charset" src="/js/jquery-1.12.2.min.js"></script>
<script type="text/javascript" language="javascript" charset="_charset" src="/js/bootstrap.min.js"></script>
</body>
</html>
```

2. Use thymeleaf to create view

2-1. Create view

- Create common part of the view. Navigation bar

```
list.html
<nav th:include="common/nav :: nav"></nav>
```

html path to load :: value set in th:fragment

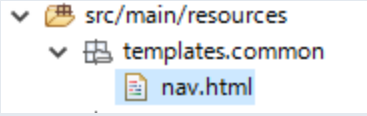
```
common/nav.html
<nav th:fragment="nav"> ….contents to load… </nav>
```

This means nav.html should be created under common folder in order to be loaded into list.html

2. Use thymeleaf to create view

2-1. Create view

- Create templates.common package and create nav.html as below




```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<body>
  <nav th:fragment="nav">
    <nav class="navbar navbar-default">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar-collapse">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" th:href="@{/project}">INDIVILISTER</a>
        </div>
        <div class="collapse navbar-collapse" id="navbar-collapse">
          <ul class="nav navbar-nav">
            <li class="dropdown">
```

Continue on next page

2. Use thymeleaf to create view

2-1. Create view

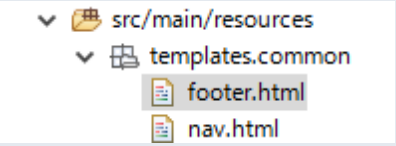
- Create templates.common package and nav.html as below

```
<a th:href="@{project/create#}" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-
expanded="false">Project
                        <span class="caret"></span>
                    </a>
                    <ul class="dropdown-menu">
                        <li>
                            <a th:href="@{/project}">List project</a>
                        </li>
                        <li>
                            <a th:href="'/project/create' + '?form'">Create a new project</a>
                        </li>
                    </ul>
                </li>
            </ul>
        </div>
    </div>
  </nav>
 </nav>
</body>
</html>
```

2. Use thymeleaf to create view

2-1. Create view

- You see the following code in list.html. Create footer.html under common folder.




```
list.hml
<span th:include="common/footer :: footer"></span>
```

```
common/footer.html
<span th:fragment="footer"> footer contents </span>
```

2. Use thymeleaf to create view

2-1. Create view

- Create footer.html as below

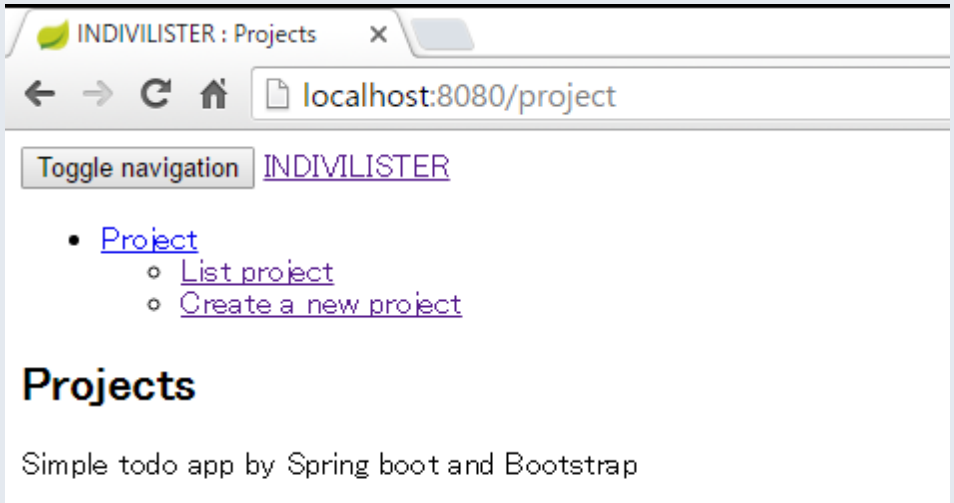
```
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:th="http://www.thymeleaf.org">
<body>
   <span th:fragment="footer">
      <footer>
         <span>Powered by ARMS with
            <img src="/img/spring_logo.png" height="30"> Spring boot and Bootstrap
         </span>
      </footer>
   </span>
</body>
</html>
```

2. Use thymeleaf to create view

2-1. Create view

- Restart your project and access server:8080/project

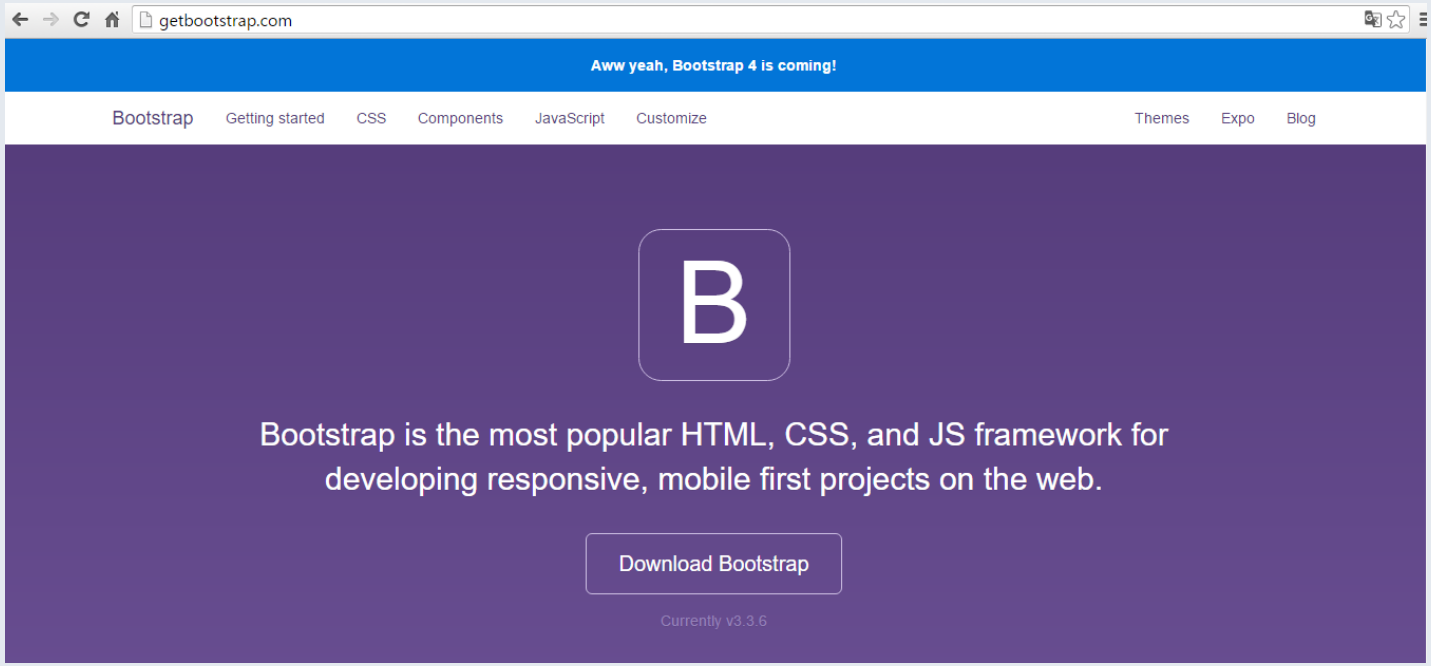
You only see the plain website
with no design….



2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

- Bootstrap http://getbootstrap.com/



2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

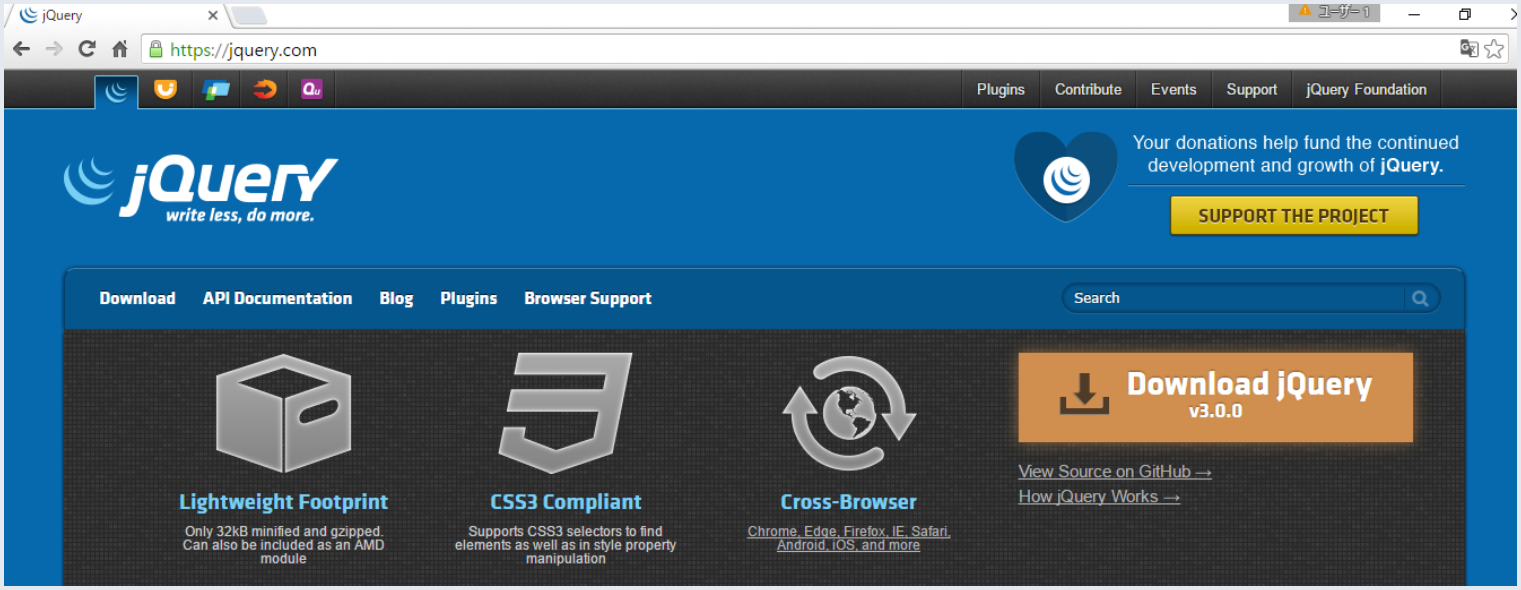
- Top 5 Reasons to use Bootstrap

1. Speed of Development
2. Responsiveness
3. Consistency
4. Customizable
5. Support

2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

- jQuery https://jquery.com/



2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

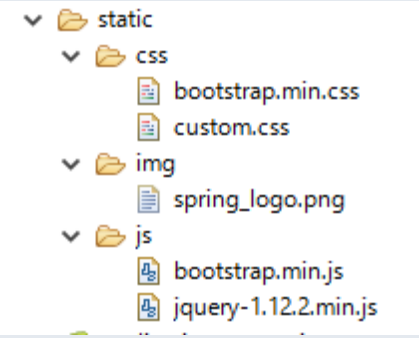
- jQuery Advantages

1. Easy to use
2. Large library
3. Opensource community
4. Documentation and tutorials
5. Ajax support

2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

- Place css and js files as below, normally if you put css and js files under static folder, you can use relative path to the files.



css
bootstrap.min.css
custom.css

img
spring_logo.png

js
bootstrap.min.js
jquery-1.12.2.min.js

list.html

```
<link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css" charset="utf-8">
```

2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

- This is how css, js and img are used in Indivilister.

list.html

```
<link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css" charset="utf-8">
<link rel="stylesheet" type="text/css" href="/css/custom.css" charset="utf-8">
<script type="text/javascript" language="javascript" charset="_charset" src="/js/jquery-1.12.2.min.js"></script>
<script type="text/javascript" language="javascript" charset="_charset" src="/js/bootstrap.min.js"></script>
```

footer.html

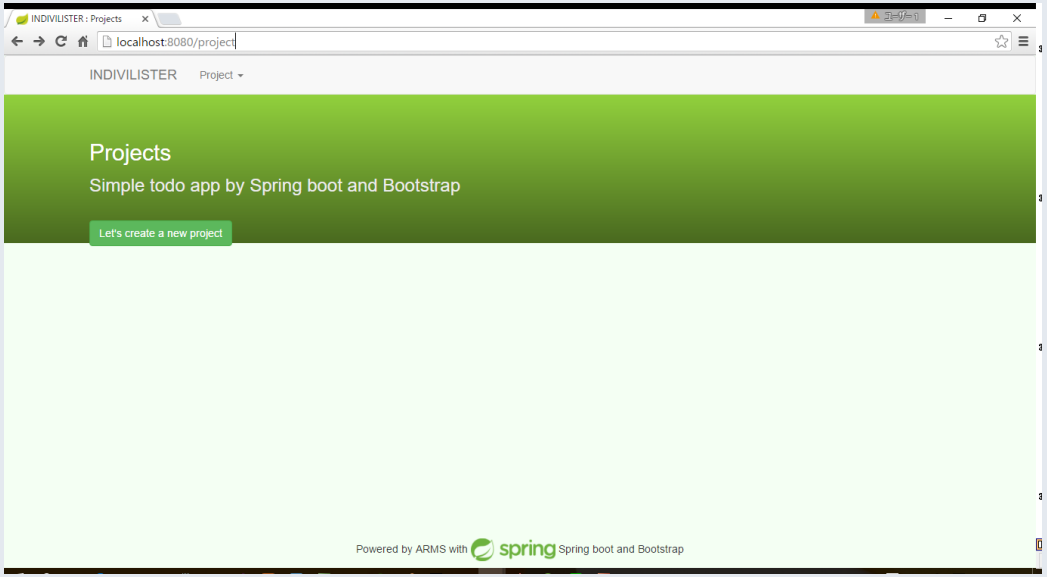
```
<img src="/img/spring_logo.png" height="30">
```

Make sure that all files are placed in each folder.

2. Use thymeleaf to create view

2-2. Apply Bootstrap and jQuery

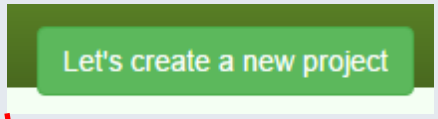
- Restart your project and access server:8080/project. If you can see the following design, so far so good!!



3. Create Indivilister's projects

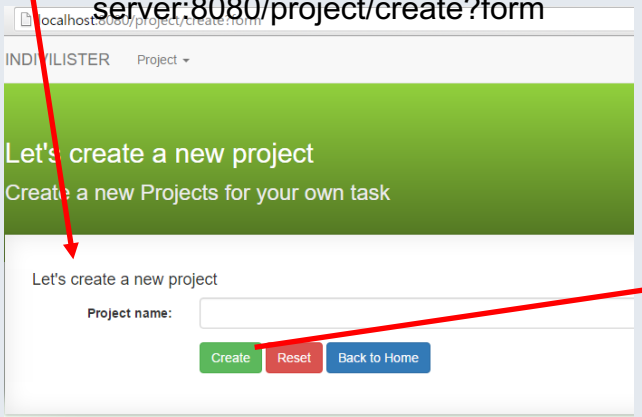
3-1. Create a project

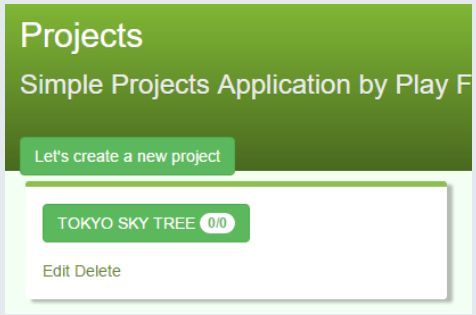
- Create this action and page as below.




```
<a th:href="@{/project/create?form}" class="btn btn-success">Let's create a new project</a>
```

server:8080/project/create?form

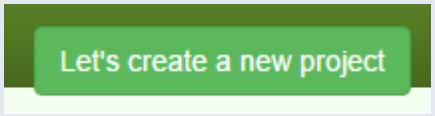




3. Create Indivilister's projects

3-1. Create a project

- Open ProjectController.java and create a method to respond to this action.




```
<a th:href="@{/project/create?form}" ……>Let's create a new project</a>
```

```
@RequestMapping(value = "create", params = "form", method = RequestMethod.GET)
    public String createForm(Model model) {
        model.addAttribute("projectForm", "Create a project");
        return "project/create";
    }
```

As you see the return value string "project/create", this means create.html under project folder.

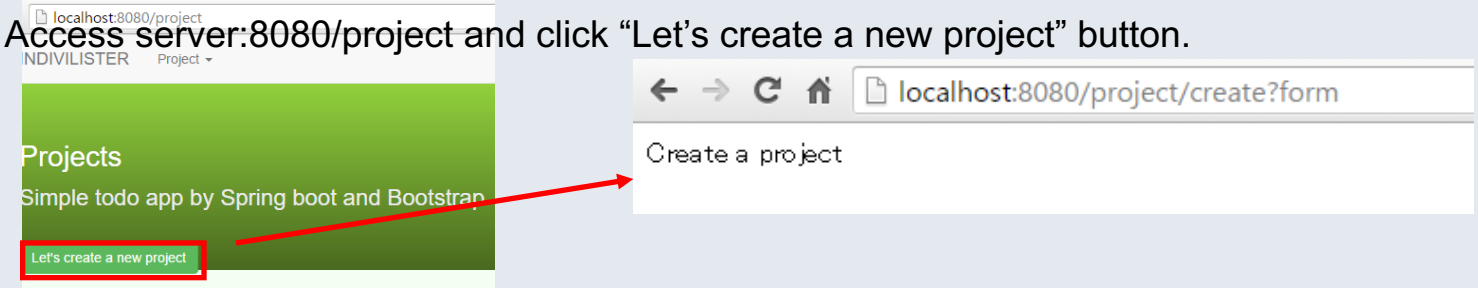
3. Create Indivilister's projects

3-1. Create a project

- Create create.html under project folder and add this text

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>INDIVILISTER : Let's create a new project</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css" charset="utf-8">
   <link rel="stylesheet" type="text/css" id="" href="/css/custom.css" charset="utf-8">
</head>
<body>
<p th:text=${projectForm}>text comes here</p>
</body>
</html>
```

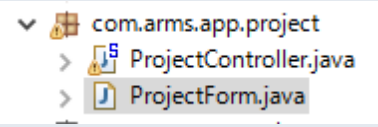
Access server:8080/project and click "Let's create a new project" button.



3. Create Indivilister's projects

3-1. Create a project

- Create a form object to receive data from text field. ProjectForm.java under com.arms.app.project




```
package com.arms.app.project;

import lombok.Data;

@Data
public class ProjectForm {
    private int id;
    private String name;

    public ProjectForm(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public ProjectForm() {}
}
```

3. Create Indivilister's projects

3-1. Create a project

- Modify createForm() method.

```
@RequestMapping(value = "create", params = "form", method = RequestMethod.GET)
  public String createForm(Model model) {
      model.addAttribute("projectForm", new ProjectForm());
      return "project/create";
  }
```

3. Create Indivilister's projects

3-1. Create a project

- Modify create.html as below

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>INDIVILISTER : Let's create a new project</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" type="text/css" href="/css/bootstrap.min.css" charset="utf-8">
   <link rel="stylesheet" type="text/css" href="/css/custom.css" charset="utf-8">
</head>
<body>
<main>
   <!-- START Navigation bar -->
   <nav th:include="common/nav :: nav"></nav>
   <!-- END Navigation bar -->

   <!-- START Header -->
   <div class="header-custom">
      <div class="container">
         <h2>Let's create a new project</h2>
         <p>Create a new project for your own task</p>
      </div>
   </div>
   <!-- END Header -->
```

Continue on next page

3. Create Indivilister's projects

3-1. Create a project

- Modify create.html as below

```
<!-- START Content -->
   <div class="container">
      <div class="row">
         <div class="col-lg-12">
            <div class="box effect1">
               <h4>Let's create a new project</h4>
               <form th:action="@{/project/create}" th:object="${projectForm}" method="POST" accept-charset="utf-8"
enctype="application/x-www-form-urlencoded" class="form-horizontal">
                  <input type="hidden" name="authenticityToken"
value="e739540dfb2b4389d499e26e8b6dc17f665de703">
                  <div class="form-group" th:classappend="${#fields.hasErrors('name')}? 'has-error'">
                     <label for="name" class="col-lg-2 control-label">Project name:</label>
                     <div class="col-lg-10">
                        <input class="form-control" th:field="*{name}" id="name" type="text" name="name" required="">
                        <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}" class="help-block">error!</span>
                     </div>
                  </div>
                  <div class="form-group">
                     <div class="col-lg-offset-2 col-lg-10">
                        <button type="submit" class="btn btn-success">Create</button>
                        <button type="reset" class="btn btn-danger">Reset</button>
                        <a th:href="@{/project}" class="btn btn-primary">Back to Home</a>
```

Continue on next page

3. Create Indivilister's projects

3-1. Create a project

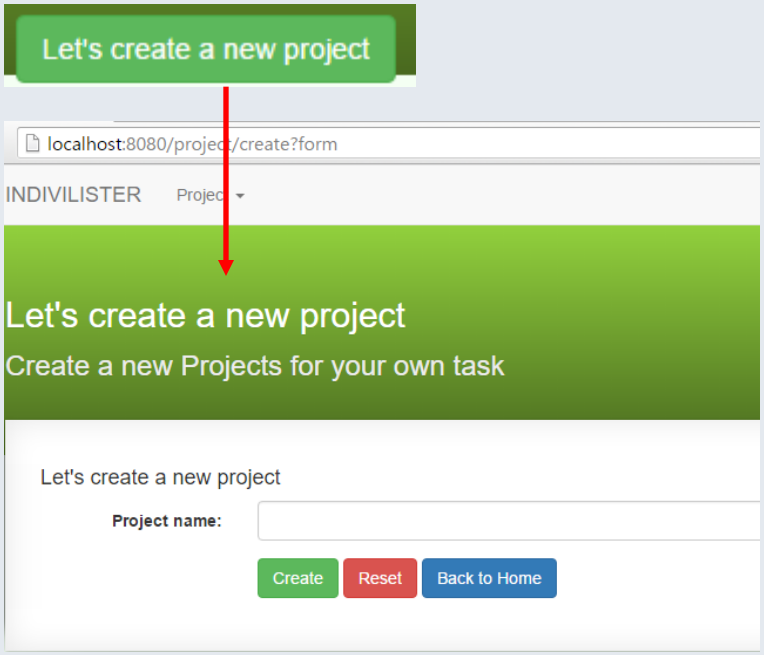
- Modify create.html as below

```
                    </div>
                  </div>
                </form>
              </div>
            </div>
          </div>
        </div>
        <!-- END Content -->
        <div class="push"></div>
    </main>

    <!-- START Footer -->
    <span th:include="common/footer :: footer"></span>
    <!-- START Footer -->

    <script type="text/javascript" language="javascript" charset="_charset" src="/js/jquery-1.12.2.min.js"></script>
    <script type="text/javascript" language="javascript" charset="_charset" src="/js/bootstrap.min.js"></script>

    </body>
    </html>
```

3. Create Indivilister's projects

3-1. Create a project

- Restart your project and access server:8080/project, then click "Let's create a new project" button.

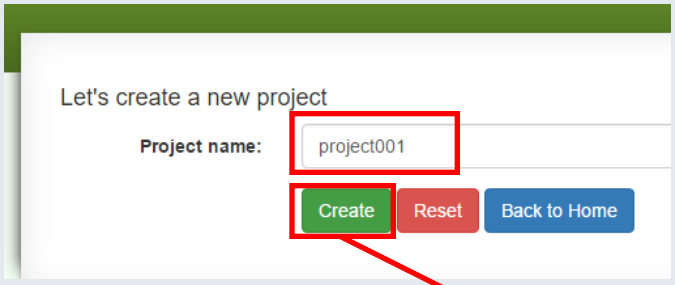


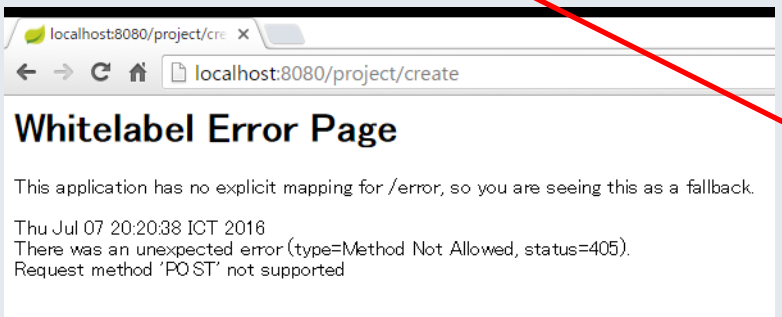
If you see the screen like the left figure, so far so good!!

3. Create Indivilister's projects

3-1. Create a project

- Type something in Project name textbox and click "Create" button



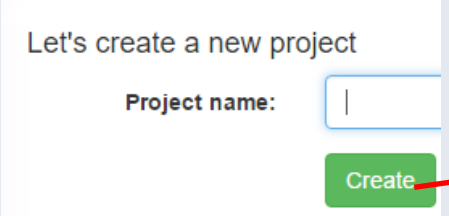


You will see the error like the left figure.

3. Create Indivilister's projects

3-1. Create a project

- Let's see the code for "Create" button




```
<form th:action="@{/project/create}" th:object="${projectForm}" method="POST"

<button type="submit" class="btn btn-success">Create</button>
```

You will need to set the controller as below

```
@RequestMapping("/project")
public class ProjectController {
……————————————————————..
@RequestMapping(value = "create", method = RequestMethod.POST)
……————————————————————..
```

3. Create Indivilister's projects

3-1. Create a project

- Add create() method into ProjectController.java

```
@RequestMapping(value = "create", method = RequestMethod.POST)
  public Object create(@ModelAttribute ProjectForm projectForm) {
      projectService.save(projectForm);
      return "redirect:/project";
  }
```

This method uses "projectService" variable, but no reference to ProjectService.

So Add ProjectService projectServicev and @Autowired with it.

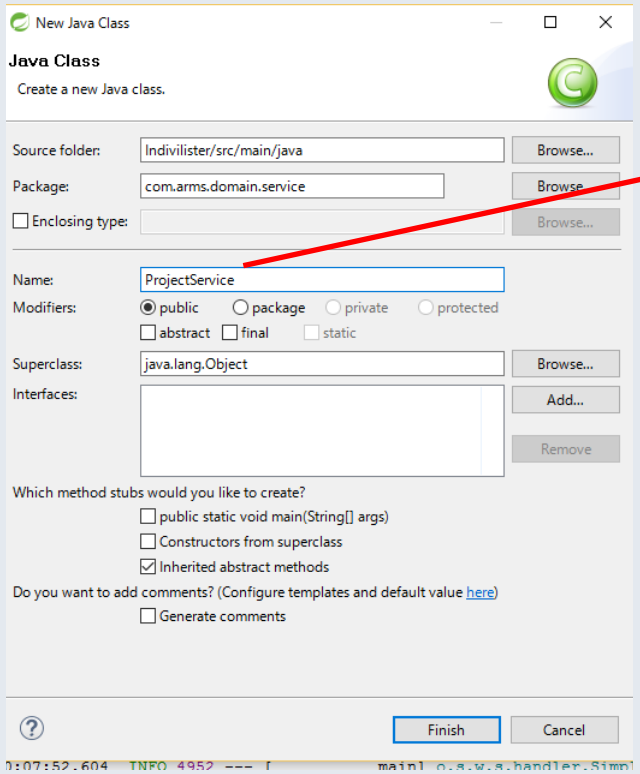
```
public class ProjectController {

   @Autowired
   ProjectService projectService;
```

Actually, we haven't created ProjectService class…..

3. Create Indivilister's projects

3-1. Create a project

- Right-click on com.arms.domain.service. New – Class



Type ProjectService in the Name field and press "Finish" button.

3. Create Indivilister's projects

3-1. Create a project

- Add the following code into ProjectService.java

```
package com.arms.domain.service;

import java.util.Calendar;
import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.arms.app.project.ProjectForm;
import com.arms.domain.entity.Project;
import com.arms.domain.repository.ProjectRepository;

@Service
@Transactional
public class ProjectService {
```

Continue on next page

3. Create Indivilister's projects

3-1. Create a project

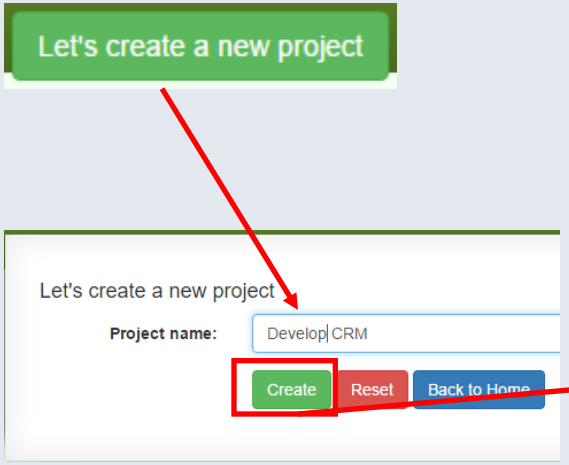
- Add the following code into ProjectService.java

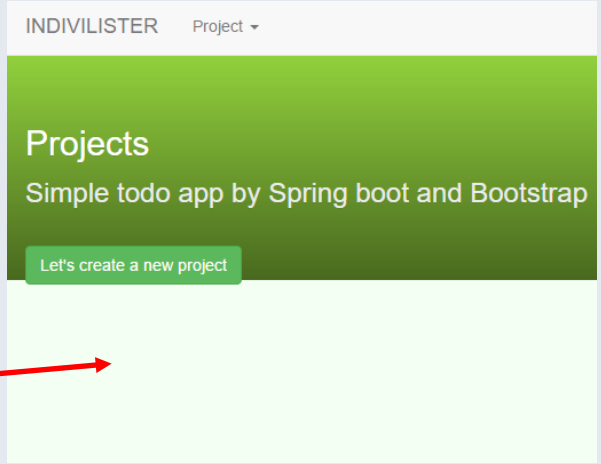
```
    @Autowired
    ProjectRepository projectRepository;

    public void save(ProjectForm projectForm) {
        Date date = Calendar.getInstance().getTime();
        Project project = new Project();
        project.setName(projectForm.getName());
        project.setCreatedDate(date);
        project.setUpdatedDate(date);
        projectRepository.save(project);
    }
}
```

3. Create Indivilister's projects

3-1. Create a project

- Restart your project and access server:8080/project, then create a project





You don't see errors. But can't see your project….

Projects should be listed….

3. Create Indivilister's projects

3-1. Create a project

- Modify index() method in ProjectController, and add findAllProject() method in ProjectService

ProjectController

```
@RequestMapping(value = "", method = RequestMethod.GET)
    public String index(Model model) {
        List<Project> projectList = projectService.findAllProject();
        model.addAttribute("projectList", projectList);
      return "project/list";
    }
```

ProjectService

```
public List<Project> findAllProject() {
      return projectRepository.findAll();
    }
```

3. Create Indivilister's projects

3-1. Create a project

- Comment out the following part in list.html

  <!-- -->

list.html

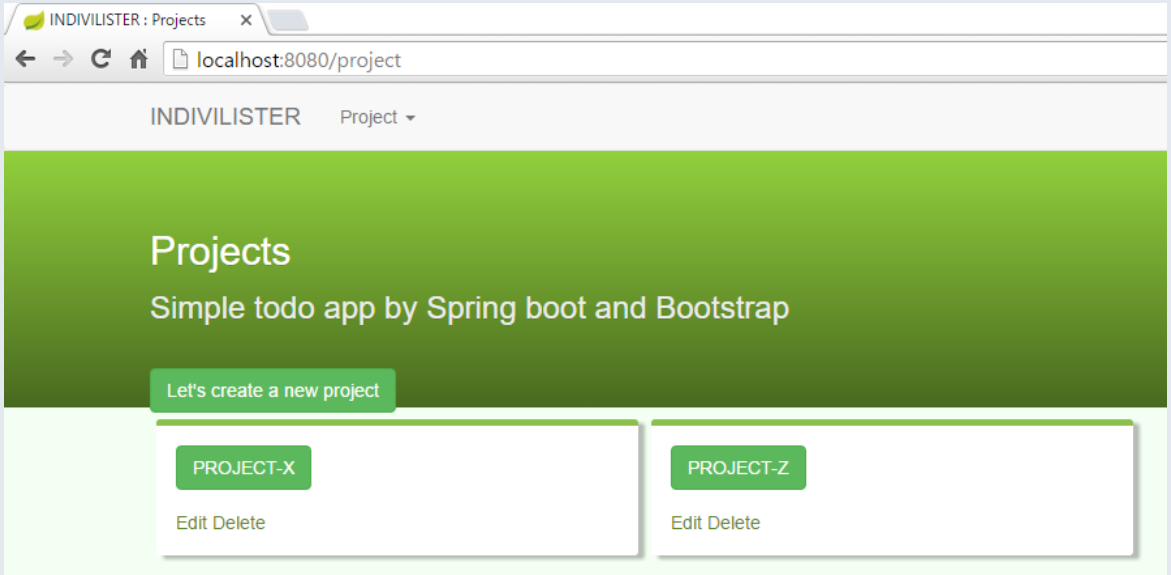
```
<!-- <span class="badge" th:text="${projectRemainingTaskMap.get(project.id)} + '/' + ${project.taskList.size()}">1/2</span> -->
```

3. Create Indivilister's projects

3-1. Create a project

- Restart your project and access localhost:8080/project

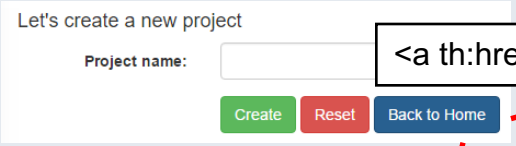
If you can see the page like below, so far so good!!



3. Create Indivilister's projects

3-1. Create a project

- Let's see "Back to Home" button




```
<a th:href="@{/project}" class="btn btn-primary">Back to Home</a>
```

When you click "Back to Home" button, the following index() method is called, and return "project/list"; project/list.html

```
@Controller
@RequestMapping("/project")
public class ProjectController {

@RequestMapping(value = "", method = RequestMethod.GET)
   public String index(Model model) {
        ...................................
        return "project/list";
   }
```

3. Create Indivilister's projects

3-1. Create a project

- Understand findAll()

ProjectRepository
findAll() ←Can't see findAll() but if you extend JpaRepository, you can use

ProjectService

```
public List<Project> findAllProject() {
        return projectRepository.findAll();
    }
```

←Return Project type List (Project data as a list)

ProjectController

```
 @RequestMapping(value = "", method = RequestMethod.GET)
   public String index(Model model) {
        List<Project> projectList = projectService.findAllProject();
        model.addAttribute("projectList", projectList);
       return "project/list";
   }
```

←Store projectList into projectList, and give it to list.html

list.html

```
<div class="col-lg-4" style="padding: 0;" th:each="project, stat : ${projectList}">
<span th:text="${project.name}">project0001</span>
```

←list.html receives projectList and repeat as project

${project.name} ${project.id}

3. Create Indivilister's projects

3-1. Create a project

- Understand the process of creating a project.

Let's create a new project

```
<a th:href="@{/project/create?form}"
```

list.html

ProjectController.java

```
@RequestMapping(value = "create", params = "form", method = RequestMethod.GET)
   public String createForm(Model model) {
      model.addAttribute("projectForm", new ProjectForm());
      return "project/create";
   }
```

createForm() method is called. ProjectForm object is created and store it in projectForm and this goes to the view(create.html)

create.html

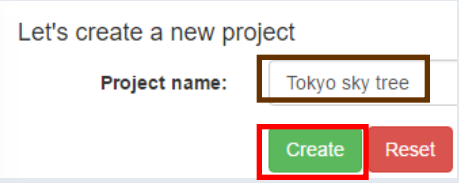
```
<form th:action="@{/project/create}" th:object="${projectForm}" method="POST"
<input class="form-control" th:field="*{name}" id="name" type="text" name="name" required="">
th:object="${projectForm}"  +  th:field="*{name}"     This *{name} = ${projectForm.name}
```

Continue on next page

3. Create Indivilister's projects

3-1. Create a project

- Understand the process of creating a project.



create.html

```
<form th:action="@{/project/create}" th:object="${projectForm}" method="POST"
<input class="form-control" th:field="*{name}" id="name" type="text" name="name"
```

Create button action calls create() method

ProjectController.java

```
@RequestMapping(value = "create", method = RequestMethod.POST)
  public Object create(@ModelAttribute ProjectForm projectForm) {
      projectService.save(projectForm);
      return "redirect:/project";
  }
```

Continue on next page

3. Create Indivilister's projects

3-1. Create a project

- Understand the process of creating a project.

ProjectController.java

```
@RequestMapping(value = "create", method = RequestMethod.POST)
  public Object create(@ModelAttribute ProjectForm projectForm) {
      projectService.save(projectForm);
      return "redirect:/project";
  }
```

ProjectService.java

```
public void save(ProjectForm projectForm) {
      Date date = Calendar.getInstance().getTime();
      Project project = new Project();
      project.setName(projectForm.getName());
      project.setCreatedDate(date);
      project.setUpdatedDate(date);
      projectRepository.save(project);
  }
```

3. Create Indivilister's projects

3-1. Create a project

- Understand the process of creating a project

ProjectRepository
save() ←Can't see save() but if you extend JpaRepository, you can use

ProjectService

```
public void save(ProjectForm projectForm) {
    Date date = Calendar.getInstance().getTime();
    Project project = new Project();
    project.setName(projectForm.getName());
    project.setCreatedDate(date);
    project.setUpdatedDate(date);
    projectRepository.save(project);
```

←project.name = projectForm.name;

ARMS

Your Idea Leads Your Ideals

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