Section 78: Spring Boot - Thymeleaf

0 / 3 | 26min

Spring Boot – ThymeLeaf:

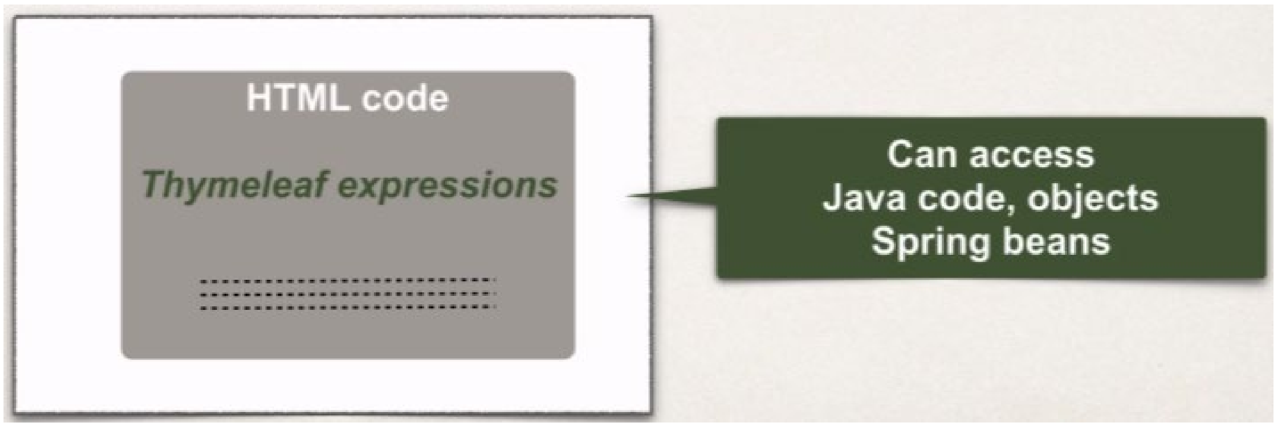
- Thymeleaf is a Java template engine.
- It's an opensource project and we can get more details about thymeleaf on https://www.thymeleaf.org/
- Commonly used to generate the HTML views for webapps
- However, it is a general-purpose templating engine
  - Can use Thymeleaf outside of web apps.

It's a separate project unrelated to spring.io, and we can create Java apps with Thymeleaf with no need of Spring.

What is thymeleaf?

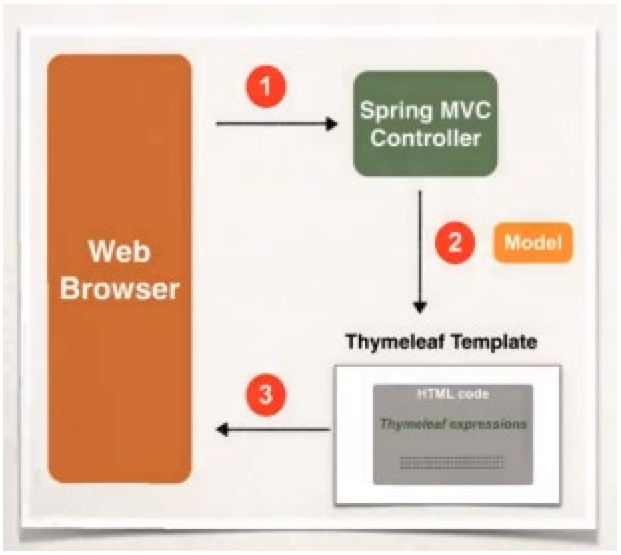
It Can be a HTML page with some Thymeleaf expressions.

Include dynamic content from Thymeleaf expressions, and they can access Java code, objects, Spring beans and so on.



Where is Thymeleaf template processed?

In web app, thymeleaf is processed on server. So, the results included in HTML returned to the browser.



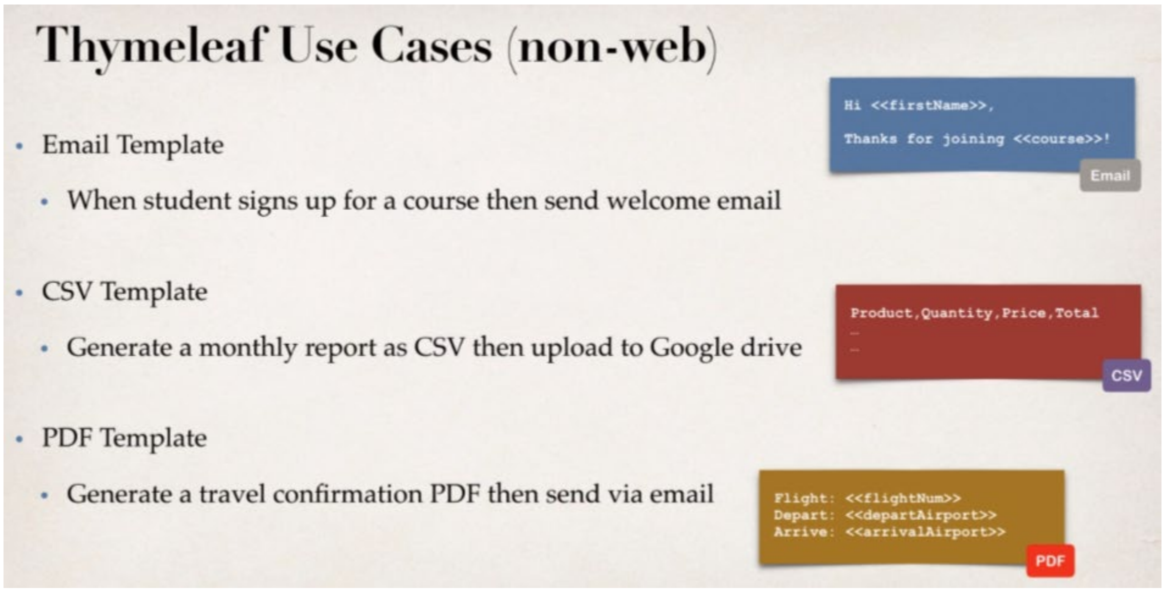
And It's very similar to the JSP.

Thymeleaf vs JSP

- **Yes,** Thymeleaf is similar to JSP.
  - Can be used for web view templates

One key difference

- JSP can only be used in a web environment.
- Thymeleaf can be used in web OR non-web environments.



Thymeleaf Use Cases (non-web)

- Email Template
  - When student signs up for a course then send welcome email
- CSV Template
  - Generate a monthly report as CSV then upload to Google drive
- PDF Template
  - Generate a travel confirmation PDF then send via email

```
Hi <<firstName>>,
Thanks for joining <<course>>!
```

```
Product,Quantity,Price,Total
```

```
Flight: <<flightNum>>
Depart: <<departAirport>>
Arrive: <<arrivalAirport>>
```

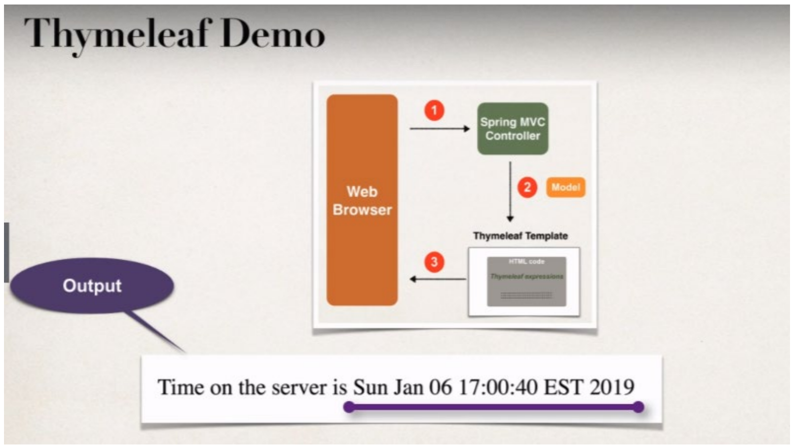
Email
CSV
PDF

FAQs: Should I use JSP or Thymeleaf?

- Depends on your project requirements.
- If we only need web views then we can use either one of them.
- But if we need a general-purpose template engine (non-web) use thymeleaf.

Development Process:

1. Add Thymeleaf to Maven POM file.
2. Develop Spring MVC Controller.
3. Create Thymeleaf template.

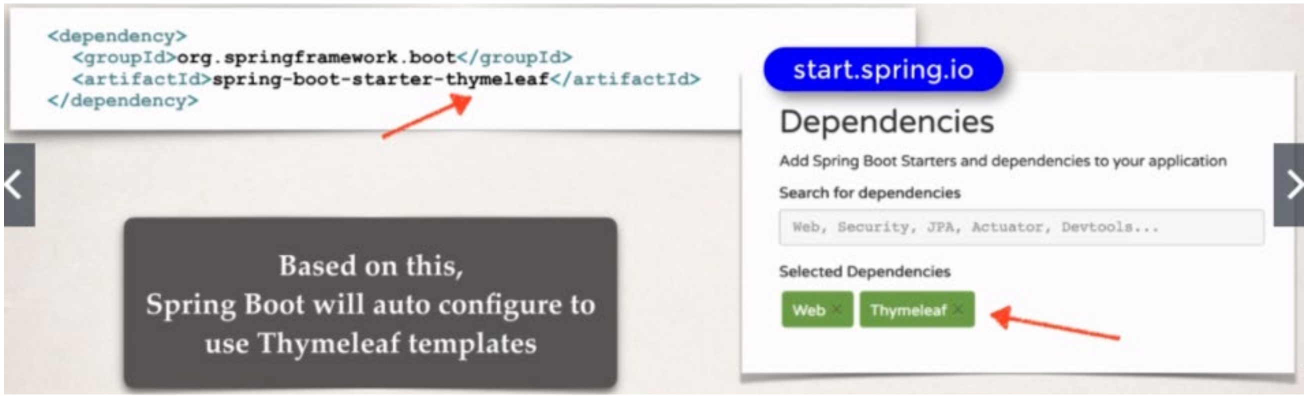


Thymeleaf Demo

Time on the server is Sun Jan 06 17:00:40 EST 2019

Java Project Link

Step 1:

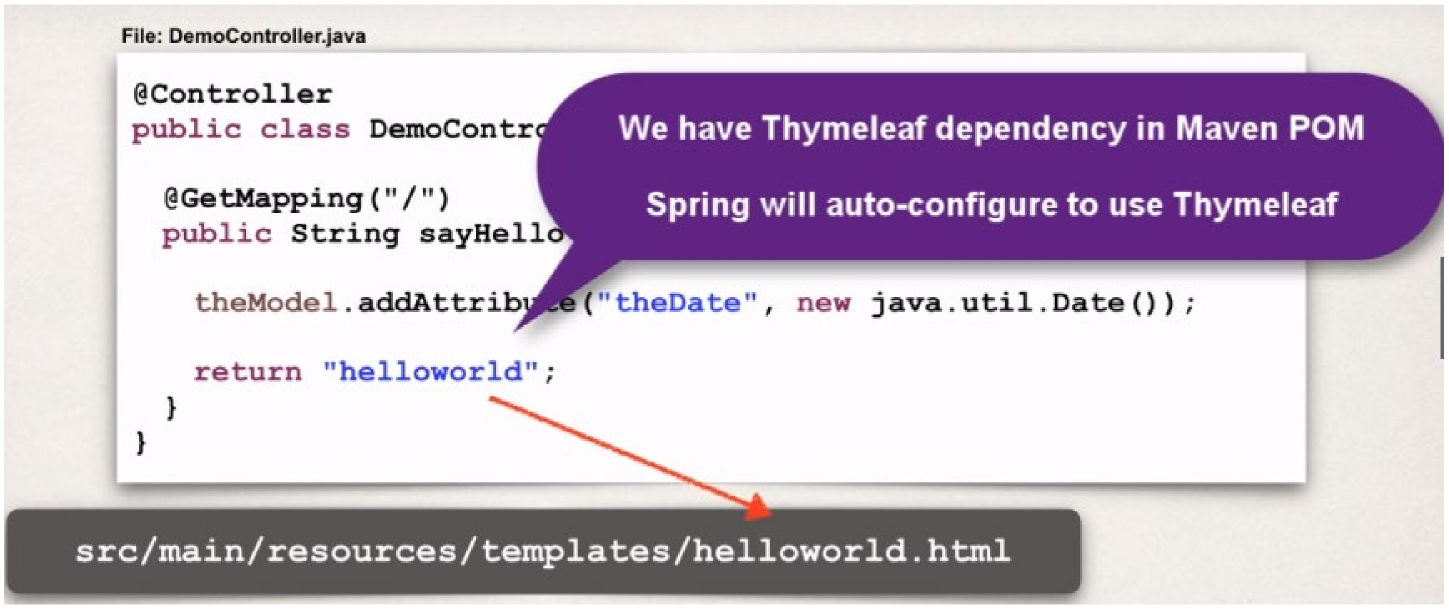


```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Based on this,
Spring Boot will auto configure to use Thymeleaf templates

start.spring.io
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Web
Thymeleaf

Step 2:



File: DemoController.java

```
@Controller
public class DemoContro

  @GetMapping("/")
  public String sayHello

    theModel.addAttribu e("theDate", new java.util.Date());

    return "helloworld";
  }
}
```

We have Thymeleaf dependency in Maven POM
Spring will auto-configure to use Thymeleaf

```
src/main/resources/templates/helloworld.html
```

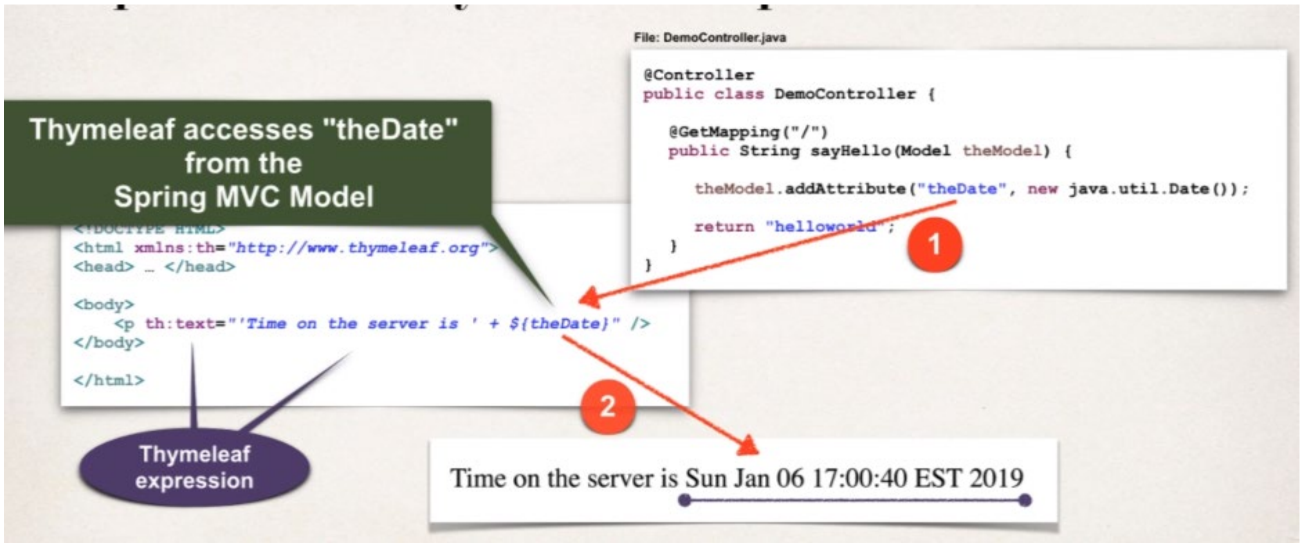
Where to Place Thymeleaf template?

- In spring boot, your thymeleaf template files go in

  `src/main/resources/templates`

- For web apps, thymeleaf templates have a **.html** extension.

Step 3:



File: DemoController.java

```
@Controller
public class DemoController {

  @GetMapping("/")
  public String sayHello(Model theModel) {

    theModel.addAttribute("theDate", new java.util.Date());

    return "helloworld";
  }
}
```

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head> … </head>

<body>
    <p th:text="'Time on the server is ' + ${theDate}" />
</body>

</html>
```

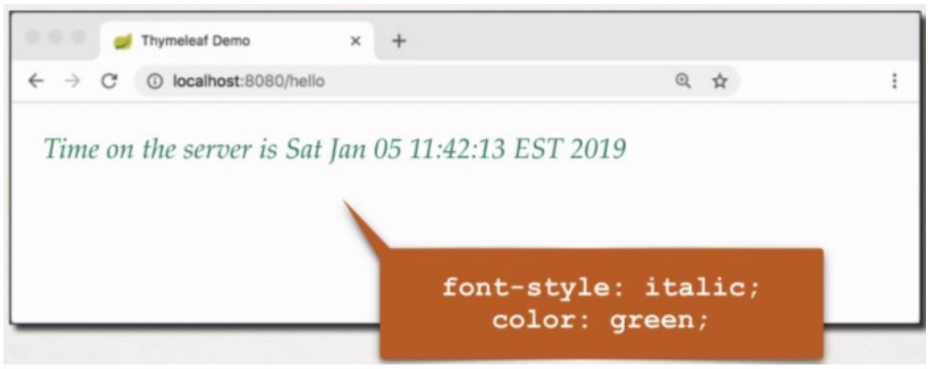
```
Time on the server is Sun Jan 06 17:00:40 EST 2019
```

Additional Features:

1. **Looping and conditionals**
2. **CSS and JavaScript integration**
3. **Template layouts and fragments**

CSS And Thymeleaf

Now, we want to add some styling to our view page.



Time on the server is Sat Jan 05 11:42:13 EST 2019

```
font-style: italic;
color: green;
```

Using CSS with Thymeleaf templates:

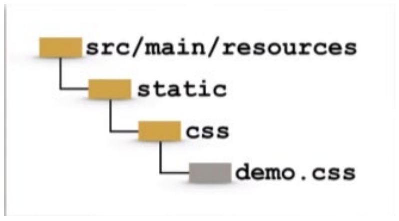
You have the option of using

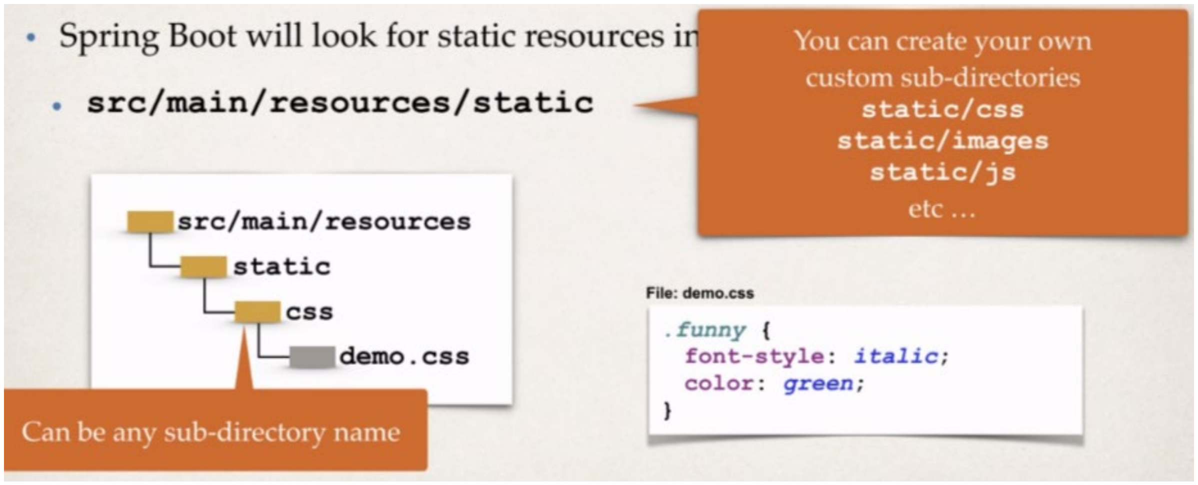
- Local CSS files as part of your project.
- Referencing remote CSS files

Development Process:

Step 1: Spring Boot will look for static resources in the directory.

`src/main/resources/static`





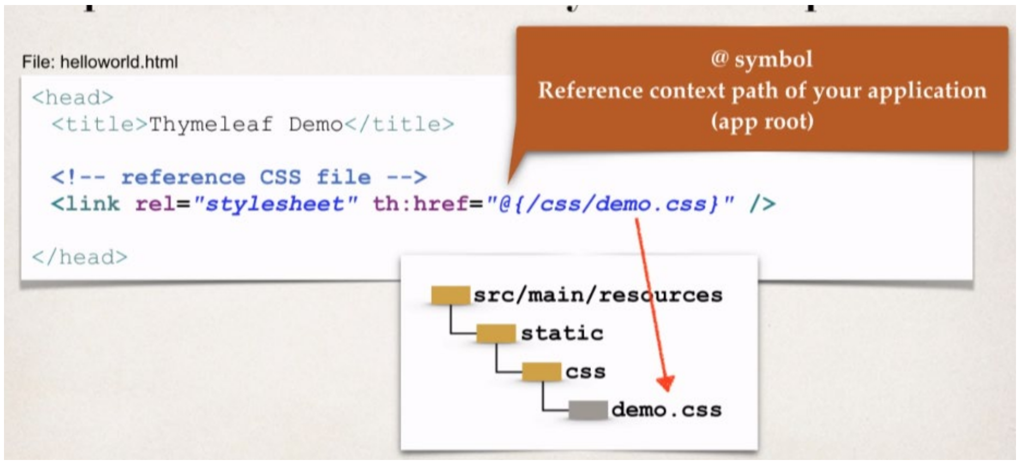
- Spring Boot will look for static resources in
  - src/main/resources/static

You can create your own custom sub-directories
static/css
static/images
static/js
etc ...

File: demo.css

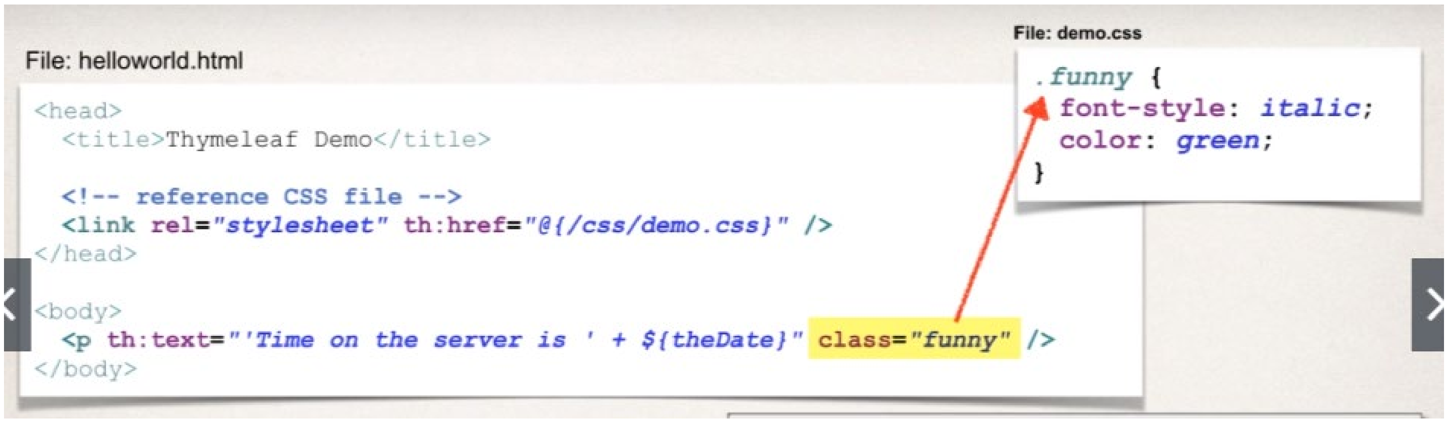
```
.funny {
  font-style: italic;
  color: green;
}
```

Step 2: Reference CSS in Thymeleaf template



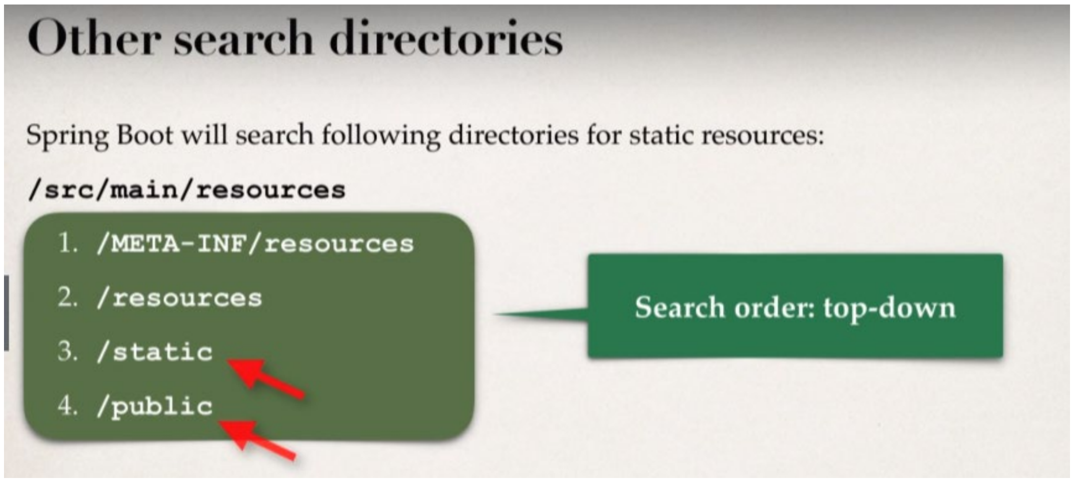
```
<head>
  <title>Thymeleaf Demo</title>

  <!-- reference CSS file -->
  <link rel="stylesheet" th:href="@{/css/demo.css}" />

</head>
```

Step 3: Apply CSS:



```
<head>
  <title>Thymeleaf Demo</title>

  <!-- reference CSS file -->
  <link rel="stylesheet" th:href="@{/css/demo.css}" />
</head>

<body>
  <p th:text="'Time on the server is ' + ${theDate}" class="funny" />
</body>
```

```
.funny {
  font-style: italic;
  color: green;
}
```



Other search directories

Spring Boot will search following directories for static resources:

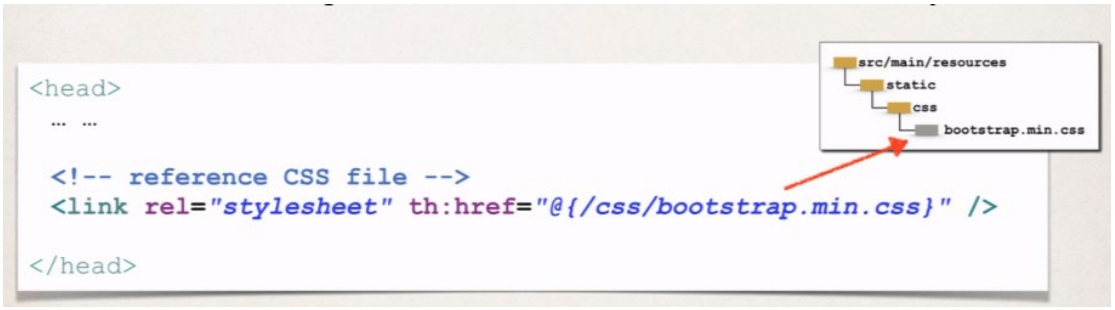
`/src/main/resources`

1. `/META-INF/resources`
2. `/resources`
3. `/static`
4. `/public`

Search order: top-down

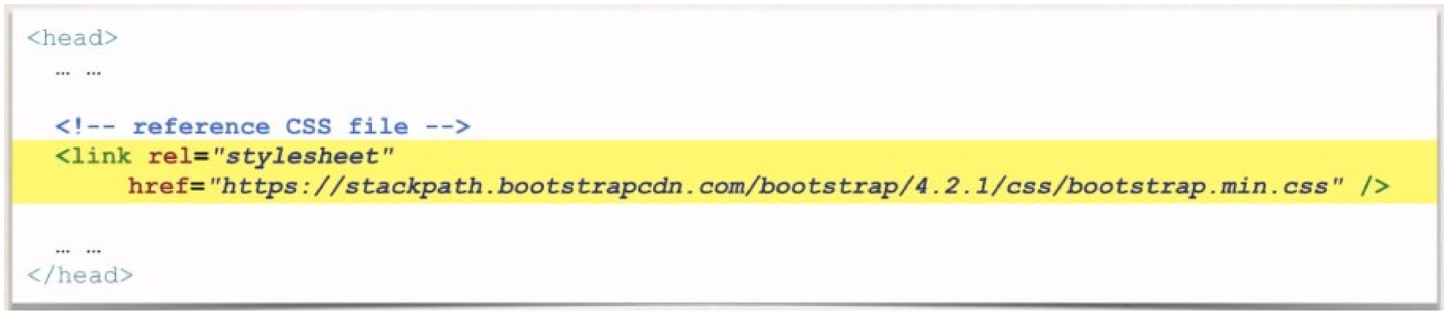
➢ **3rd party CSS Libraries – Bootstrap**

- Local Installation.
- Download Bootstrap file(s) and add to **`/static/css`** directory.



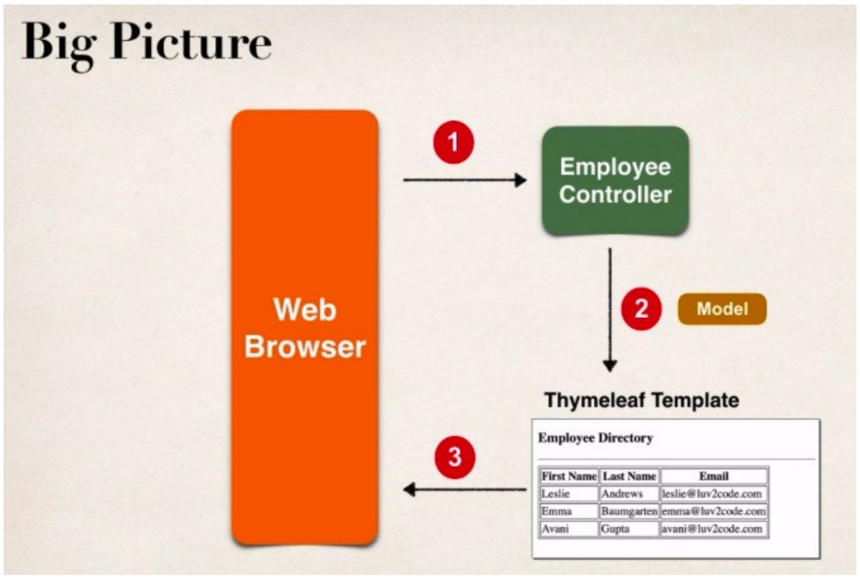
```
<head>
  … …

  <!-- reference CSS file -->
  <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" />

</head>
```

- We can access CSS files remotely onto the internet.
  So, we can simply give href location to the css file where it is stored on the internet.



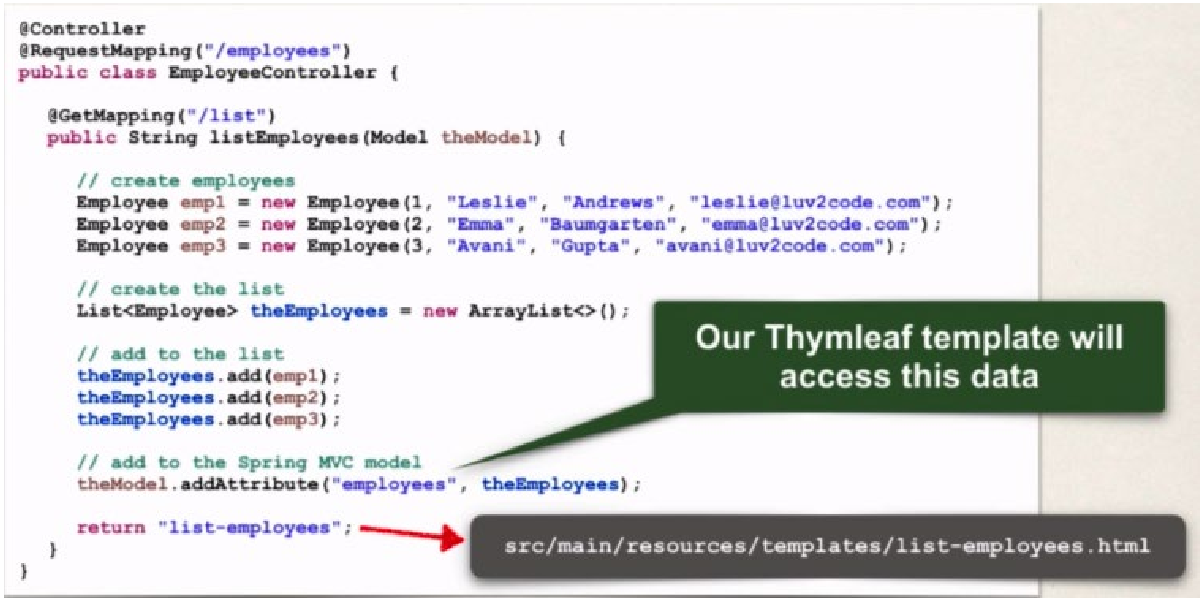
```
<head>
  … …

  <!-- reference CSS file -->
  <link rel="stylesheet"
       href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css" />

  … …
</head>
```

➢ **<u>Spring Boot thymeleaf-Build HTML table:</u>** Java Project Link


Big Picture
Web Browser
1
Employee Controller
2
Model
Thymeleaf Template
Employee Directory
First Name | Last Name | Email
Leslie | Andrews | leslie@luv2code.com
Emma | Baumgarten | emma@luv2code.com
Avani | Gupta | avani@luv2code.com
3

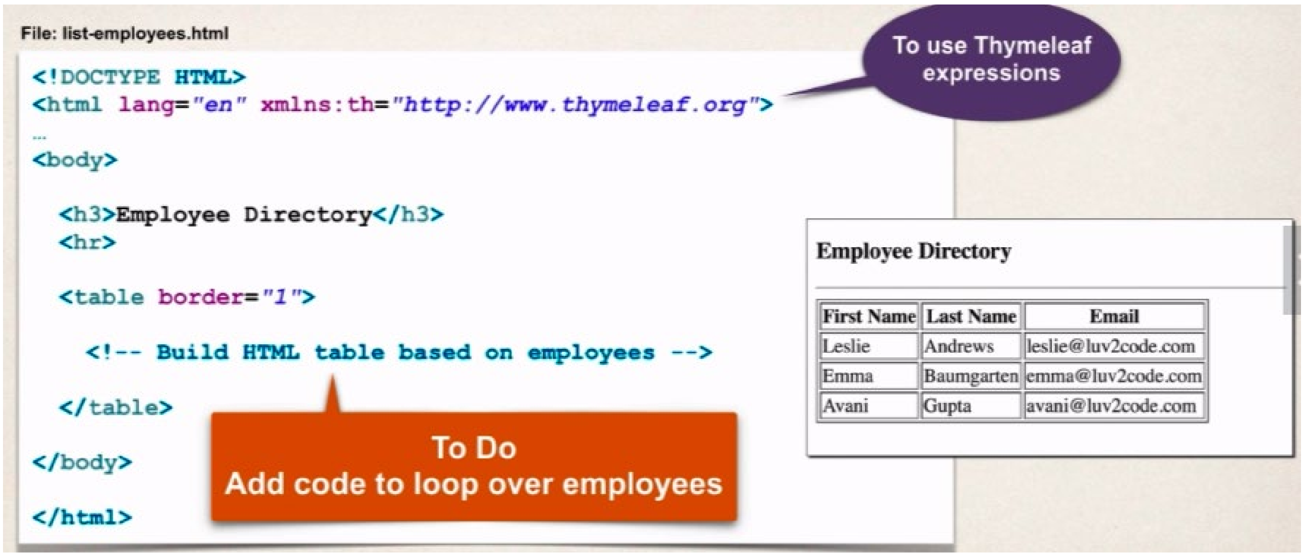
<u>Development Process:</u>

1. <u>Create Employee class:</u>
2. <u>Create Employee Controller</u>



```java
@Controller
@RequestMapping("/employees")
public class EmployeeController {

   @GetMapping("/list")
   public String listEmployees(Model theModel) {

      // create employees
      Employee emp1 = new Employee(1, "Leslie", "Andrews", "leslie@luv2code.com");
      Employee emp2 = new Employee(2, "Emma", "Baumgarten", "emma@luv2code.com");
      Employee emp3 = new Employee(3, "Avani", "Gupta", "avani@luv2code.com");

      // create the list
      List<Employee> theEmployees = new ArrayList<>();

      // add to the list
      theEmployees.add(emp1);
      theEmployees.add(emp2);
      theEmployees.add(emp3);

      // add to the Spring MVC model
      theModel.addAttribute("employees", theEmployees);

      return "list-employees";
   }
}
```

Our Thymeleaf template will access this data
src/main/resources/templates/list-employees.html



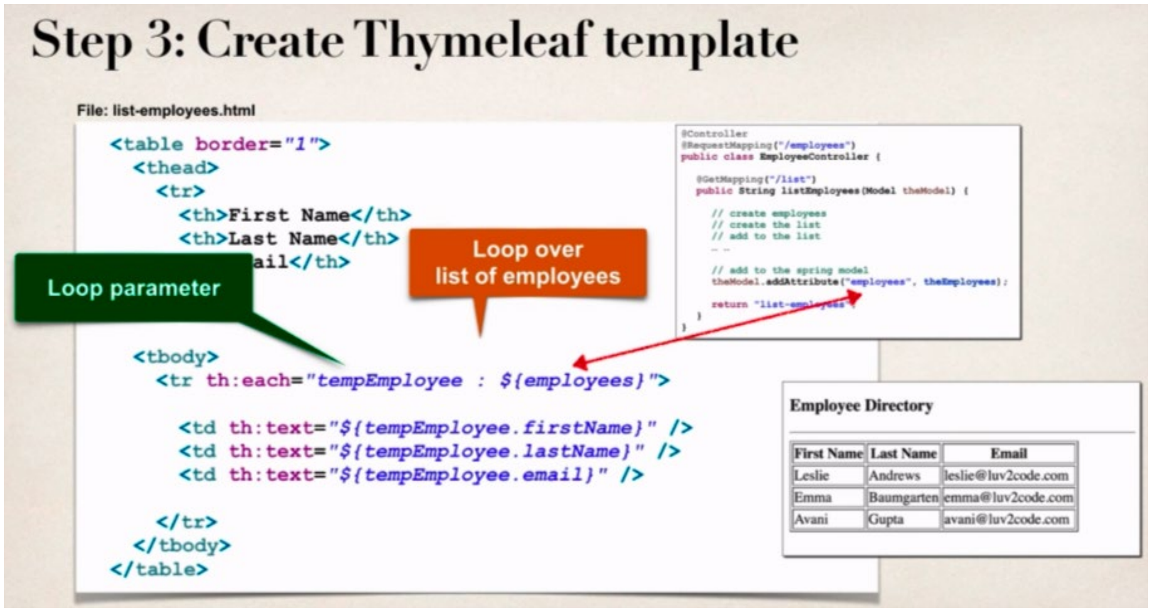
File: list-employees.html

```html
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
...
<body>

  <h3>Employee Directory</h3>
  <hr>

  <table border="1">

    <!-- Build HTML table based on employees -->

  </table>

</body>

</html>
```

To use Thymeleaf expressions
To Do
Add code to loop over employees

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@luv2code.com
Emma	Baumgarten	emma@luv2code.com
Avani	Gupta	avani@luv2code.com



Step 3: Create Thymeleaf template

File: list-employees.html

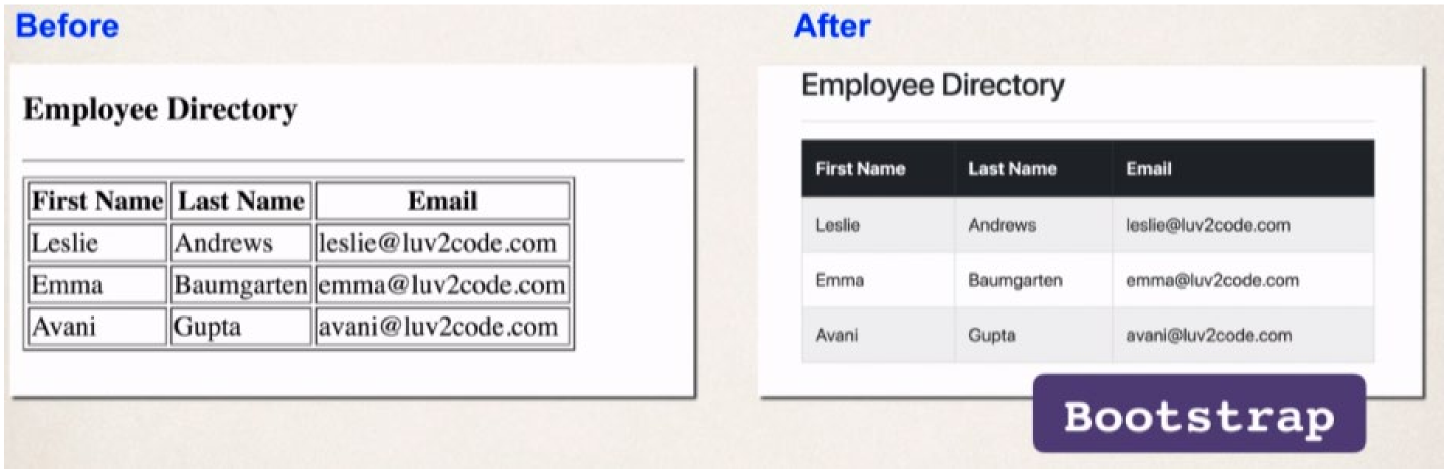
```html
<table border="1">
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
              ail</th>

  <tbody>
    <tr th:each="tempEmployee : ${employees}">

      <td th:text="${tempEmployee.firstName}" />
      <td th:text="${tempEmployee.lastName}" />
      <td th:text="${tempEmployee.email}" />

    </tr>
  </tbody>
</table>
```

```java
@Controller
@RequestMapping("/employees")
public class EmployeeController {

  @GetMapping("/list")
  public String listEmployees(Model theModel) {

    // create employees
    // create the list
    // add to the list
    -- --

    // add to the spring model
    theModel.addAttribute("employees", theEmployees);

    return "list-employees";
  }
}
```

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@luv2code.com
Emma	Baumgarten	emma@luv2code.com
Avani	Gupta	avani@luv2code.com

Let's add the CSS:



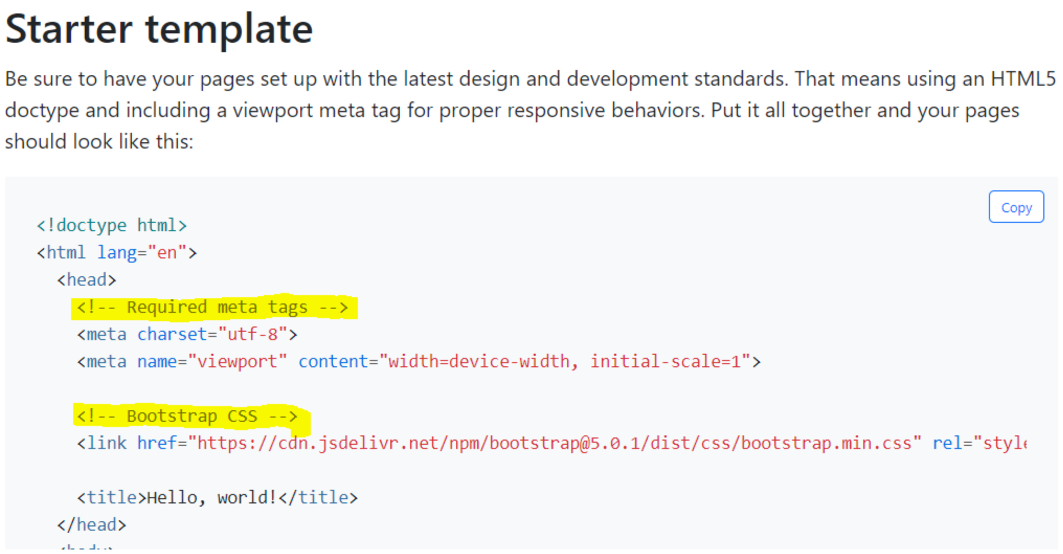
Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@luv2code.com
Emma	Baumgarten	emma@luv2code.com
Avani	Gupta	avani@luv2code.com

Development process:

1. Get links for remote Bootstrap files.

- Visit Bootstrap website: **www.getbootstrap.com**
- Website has instructions on how to **Get Started**



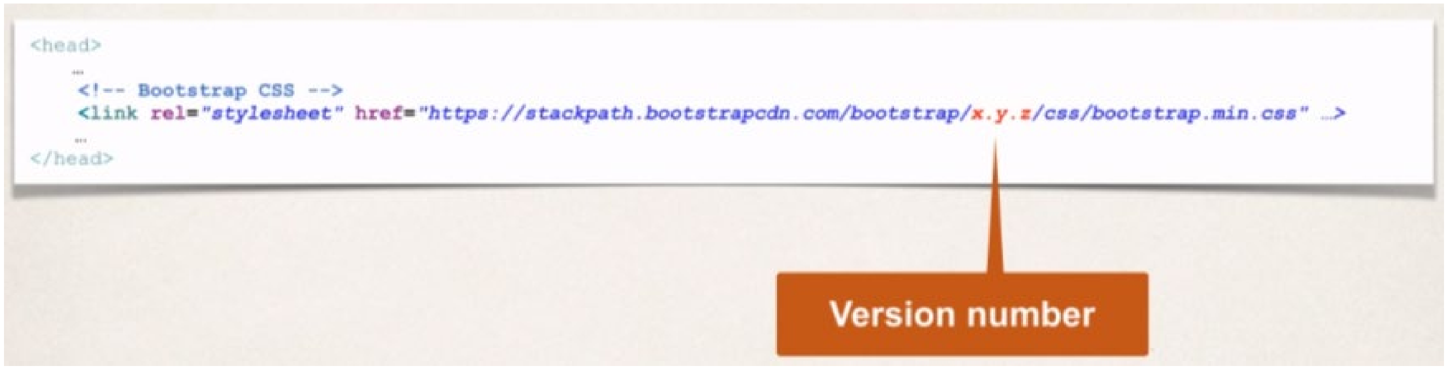
Starter template

Be sure to have your pages set up with the latest design and development standards. That means using an HTML5 doctype and including a viewport meta tag for proper responsive behaviors. Put it all together and your pages should look like this:

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="styl

    <title>Hello, world!</title>
  </head>
```

Copy-Paste the highlighted texts into the html header section.

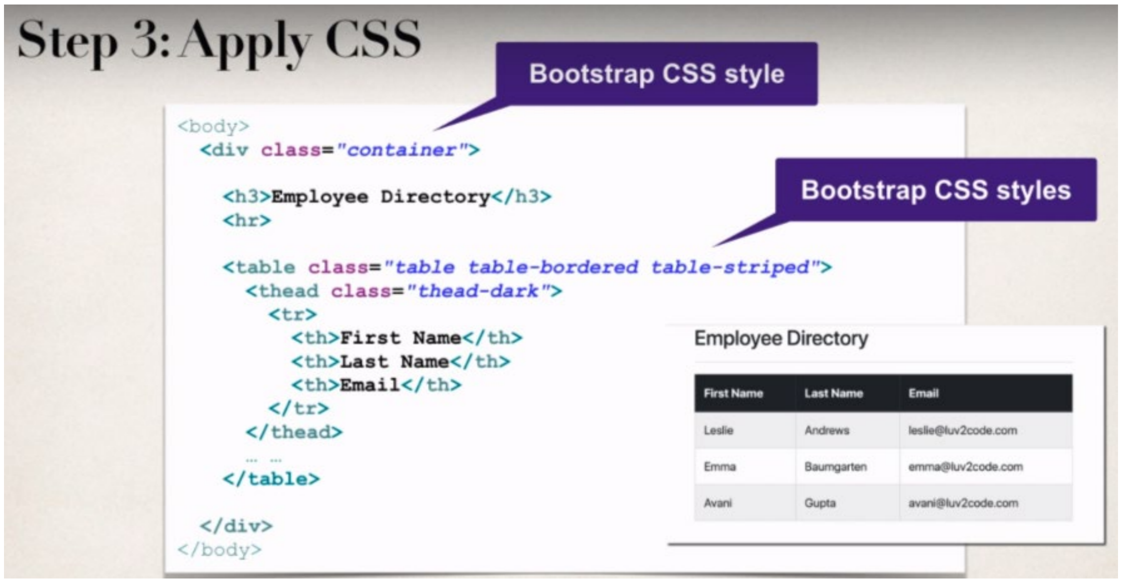
2. Add links in Thymeleaf template



```
<head>
    ...
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/x.y.z/css/bootstrap.min.css" ...>
    ...
</head>
```

Version no. will vary depending upon the current bootstrap release.

3. Apply Bootstrap CSS styles.

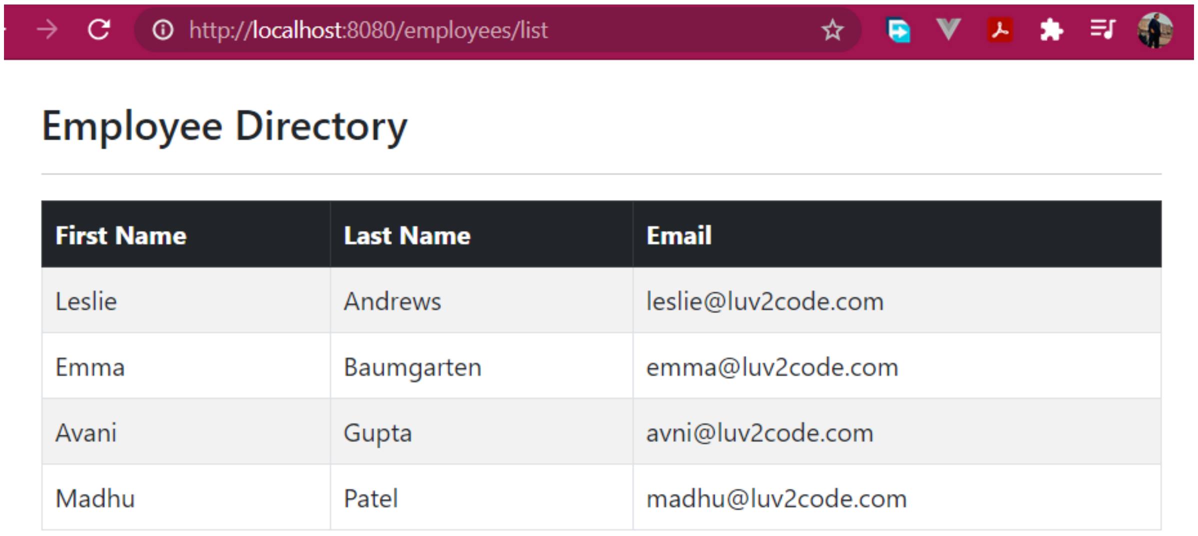


Step 3: Apply CSS

```
<body>
  <div class="container">

    <h3>Employee Directory</h3>
    <hr>

    <table class="table table-bordered table-striped">
      <thead class="thead-dark">
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Email</th>
        </tr>
      </thead>
      ... ...
    </table>

  </div>
</body>
```

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@luv2code.com
Emma	Baumgarten	emma@luv2code.com
Avani	Gupta	avani@luv2code.com

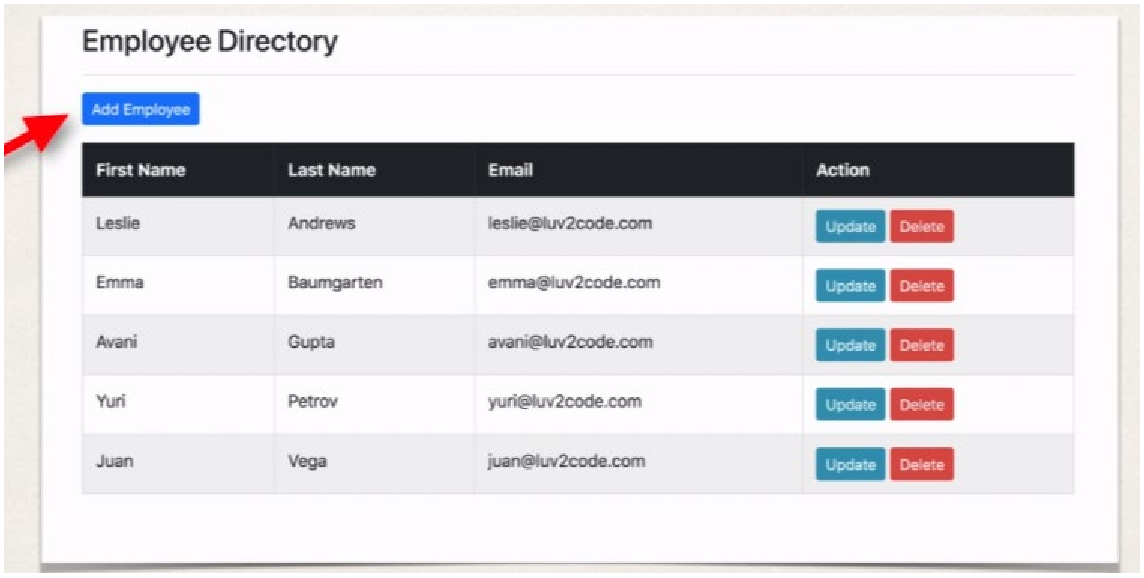


http://localhost:8080/employees/list

Employee Directory

First Name	Last Name	Email
Leslie	Andrews	leslie@luv2code.com
Emma	Baumgarten	emma@luv2code.com
Avani	Gupta	avni@luv2code.com
Madhu	Patel	madhu@luv2code.com

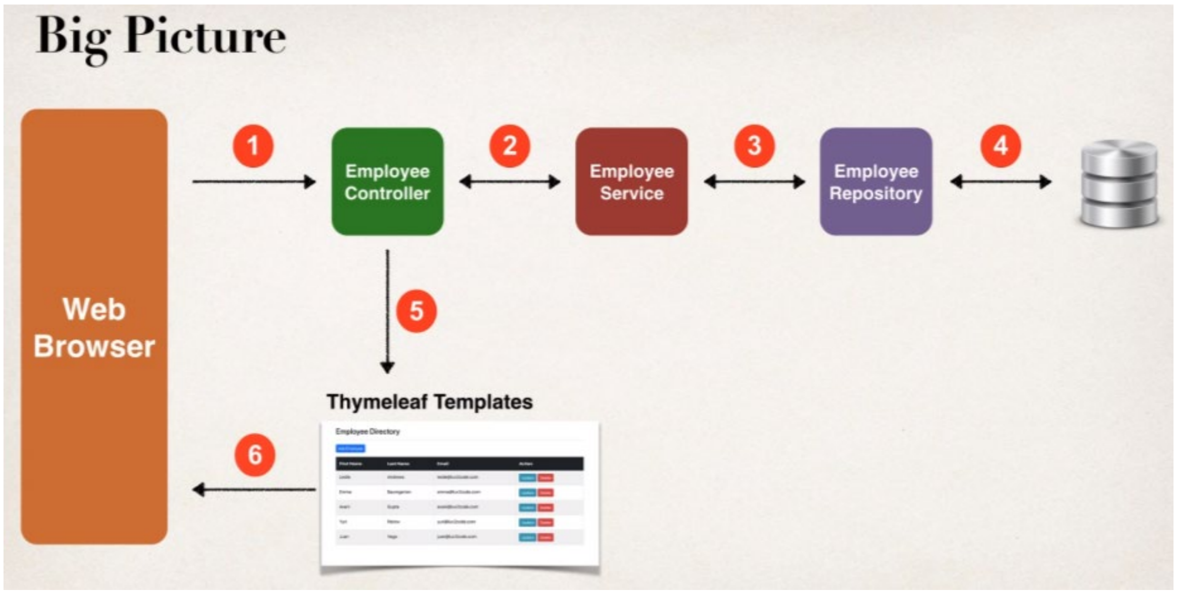
➢ **Real-time Project** Java Project Link



Employee Directory

Add Employee

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update Delete
Emma	Baumgarten	emma@luv2code.com	Update Delete
Avani	Gupta	avani@luv2code.com	Update Delete
Yuri	Petrov	yuri@luv2code.com	Update Delete
Juan	Vega	juan@luv2code.com	Update Delete



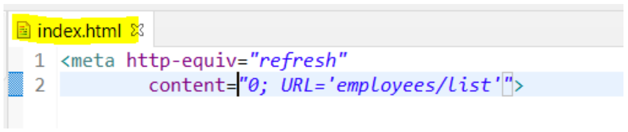
On hitting the URL: http://localhost:8080/ we want to redirect to http://localhost:8080/employees/list

For that, we will create `index.html` page inside our `src/main/resources/static/`

And following code will be written inside the index.html page to redirect to http://localhost:8080/employees/list

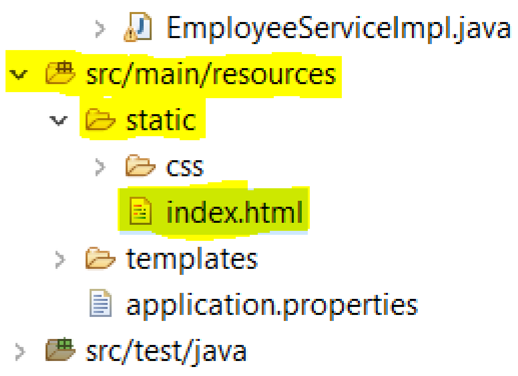
File: `src/main/resources/static/index.html`

```
<meta http-equiv="refresh"  content="0;
URL='employees/list'">
```



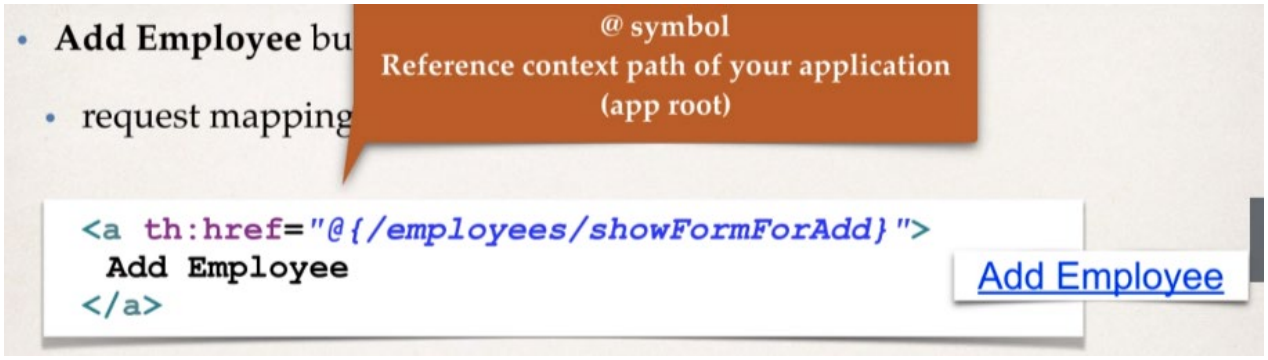
index.html

```
<meta http-equiv="refresh"
      content="0; URL='employees/list'">
```



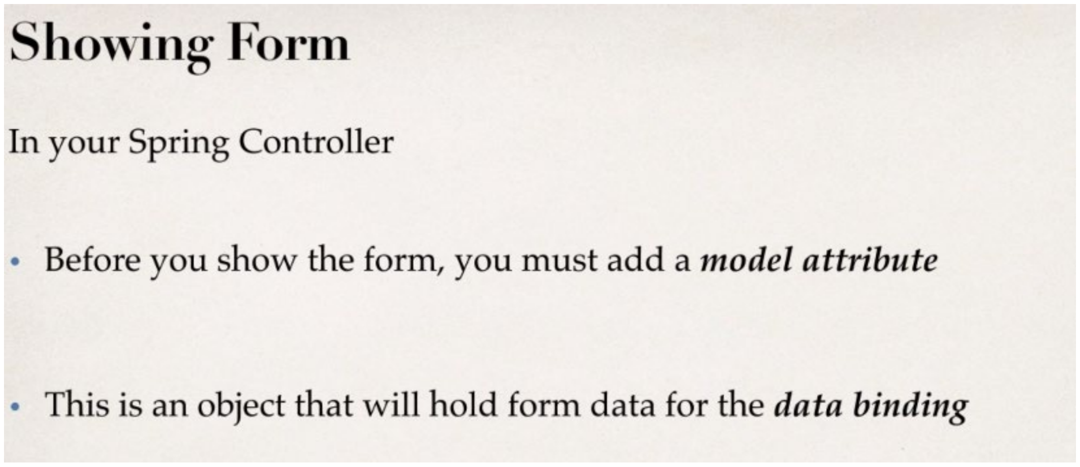
➢ **<u>Add Employee</u>**

1. <u>New Add Employee button for list-employees.html</u>



- Add Employee bu
  - request mapping

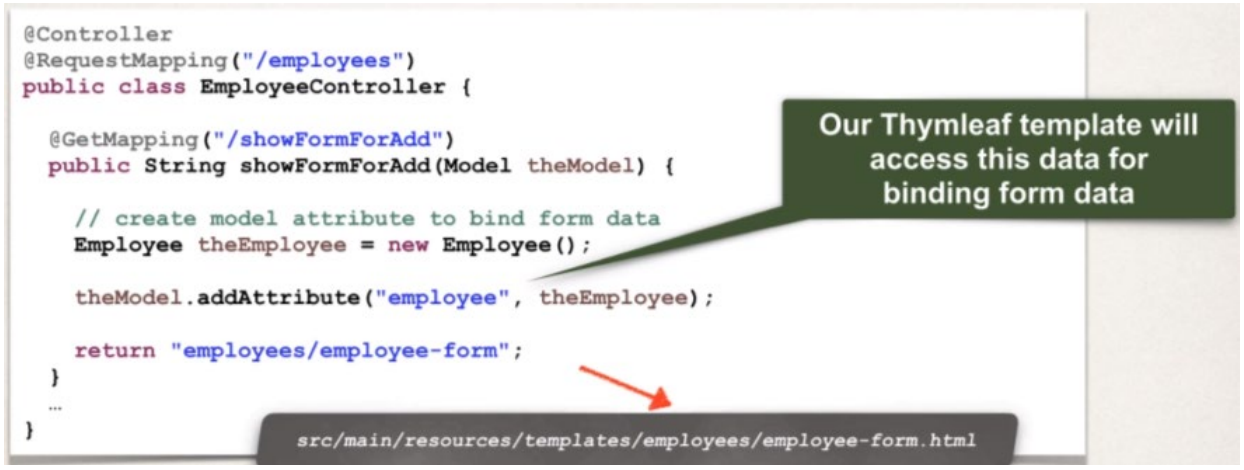
```
<a th:href="@{/employees/showFormForAdd}">
  Add Employee
</a>
```



Showing Form

In your Spring Controller

- Before you show the form, you must add a ***model attribute***
- This is an object that will hold form data for the ***data binding***



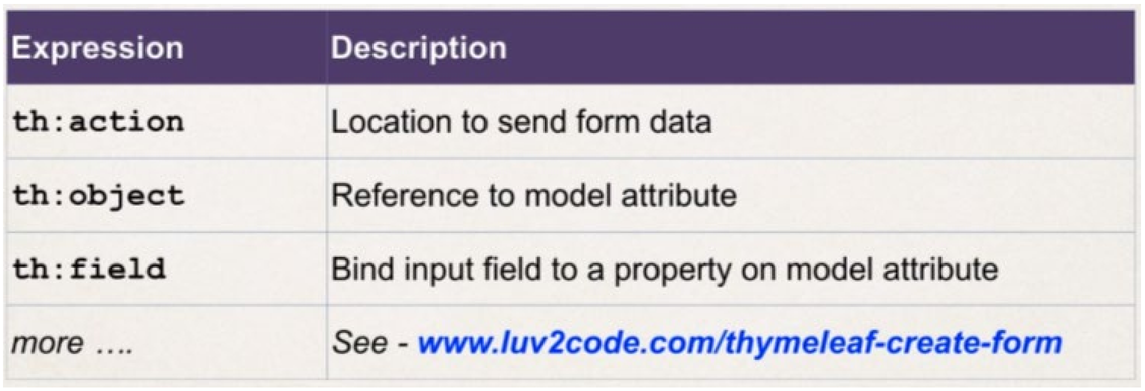
```
@Controller
@RequestMapping("/employees")
public class EmployeeController {

  @GetMapping("/showFormForAdd")
  public String showFormForAdd(Model theModel) {

    // create model attribute to bind form data
    Employee theEmployee = new Employee();

    theModel.addAttribute("employee", theEmployee);

    return "employees/employee-form";
  }
  ...
}
```

Our Thymleaf template will access this data for binding form data
src/main/resources/templates/employees/employee-form.html

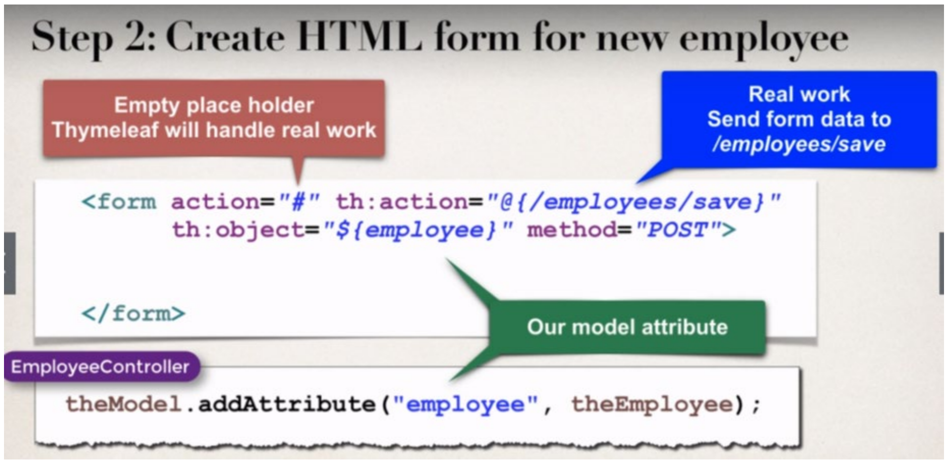
2. <u>Create HTML form for new Employee.</u>

Thymeleaf and Spring MVC DATA Binding:

- Thymeleaf has special expressions for binding Spring MVC form data
- Automatically setting/retrieving data from a Java Object.
- So, these thymeleaf expressions can help you to build the HTML form.



Expression	Description
`th:action`	Location to send form data
`th:object`	Reference to model attribute
`th:field`	Bind input field to a property on model attribute
more	*See - www.luv2code.com/thymeleaf-create-form*



Step 2: Create HTML form for new employee

Empty place holder
Thymeleaf will handle real work

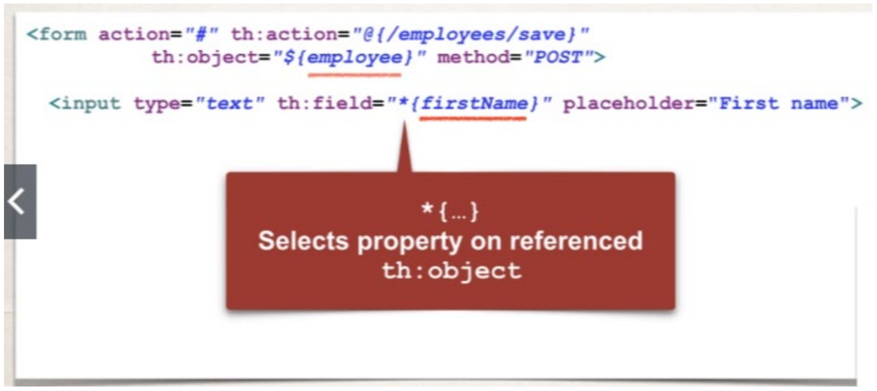
Real work
Send form data to
/employees/save

```
<form action="#" th:action="@{/employees/save}"
      th:object="${employee}" method="POST">


</form>
```

Our model attribute

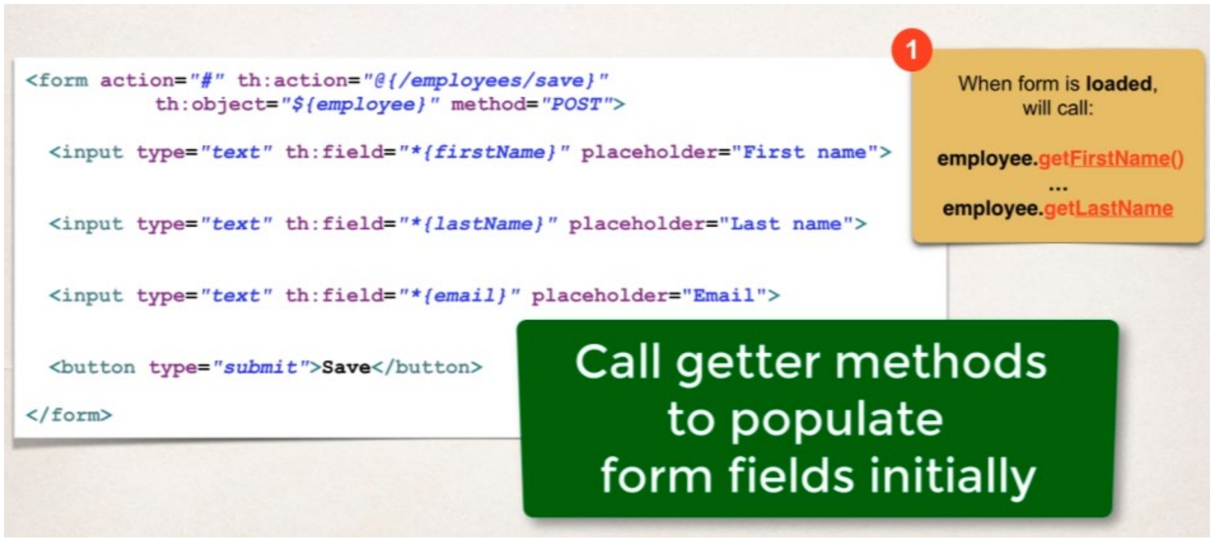
EmployeeController

```
theModel.addAttribute("employee", theEmployee);
```



```
<form action="#" th:action="@{/employees/save}"
          th:object="${employee}" method="POST">

  <input type="text" th:field="*{firstName}" placeholder="First name">
```

*{...}
Selects property on referenced
th:object



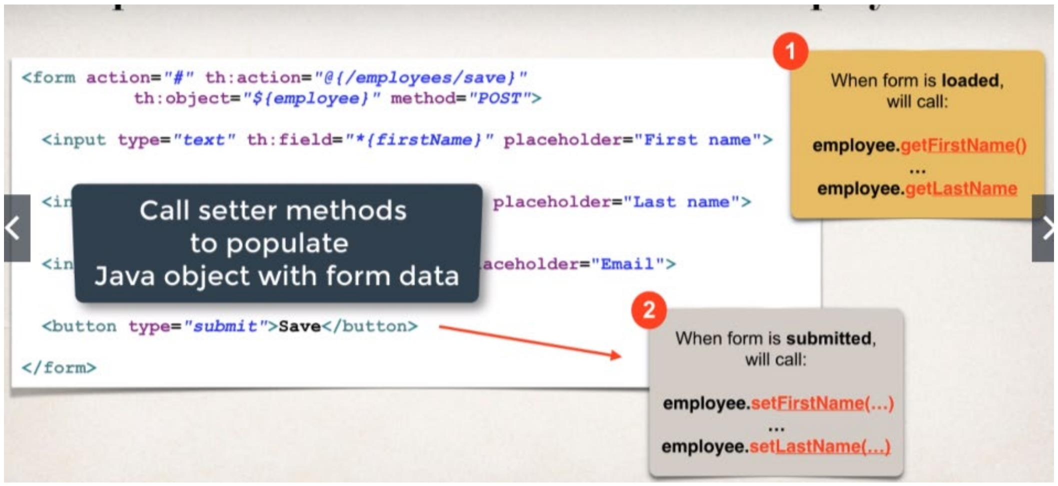
```
<form action="#" th:action="@{/employees/save}"
          th:object="${employee}" method="POST">

  <input type="text" th:field="*{firstName}" placeholder="First name">


  <input type="text" th:field="*{lastName}" placeholder="Last name">


  <input type="text" th:field="*{email}" placeholder="Email">


  <button type="submit">Save</button>

</form>
```

1

When form is **loaded**,
will call:

employee.getFirstName()
...
employee.getLastName

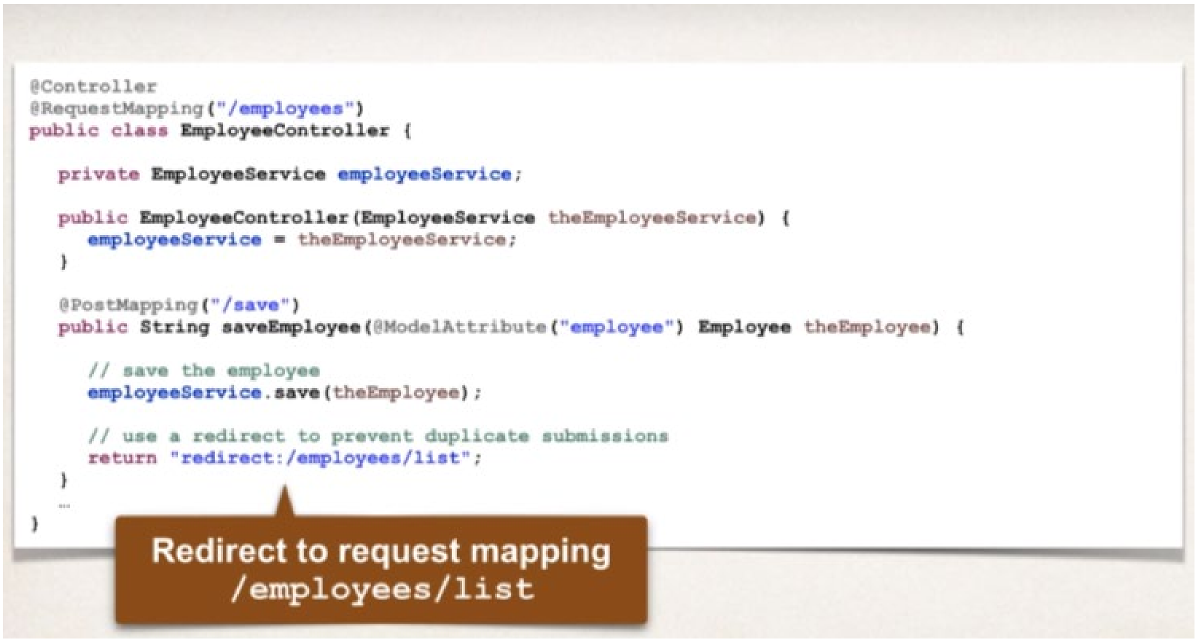
Call getter methods
to populate
form fields initially



```
<form action="#" th:action="@{/employees/save}"
          th:object="${employee}" method="POST">

  <input type="text" th:field="*{firstName}" placeholder="First name">

  <in                                        placeholder="Last name">

  <in                                       aceholder="Email">

  <button type="submit">Save</button>

</form>
```

Call setter methods to populate Java object with form data
1
When form is **loaded**, will call:
employee.getFirstName()
...
employee.getLastName
2
When form is **submitted**, will call:
employee.setFirstName(...)
...
employee.setLastName(...)

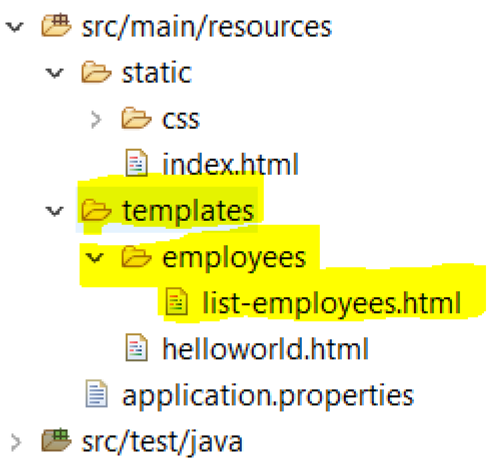
3. Process form data to save employee



```
@Controller
@RequestMapping("/employees")
public class EmployeeController {

   private EmployeeService employeeService;

   public EmployeeController(EmployeeService theEmployeeService) {
      employeeService = theEmployeeService;
   }

   @PostMapping("/save")
   public String saveEmployee(@ModelAttribute("employee") Employee theEmployee) {

      // save the employee
      employeeService.save(theEmployee);

      // use a redirect to prevent duplicate submissions
      return "redirect:/employees/list";
   }
   ...
}
```

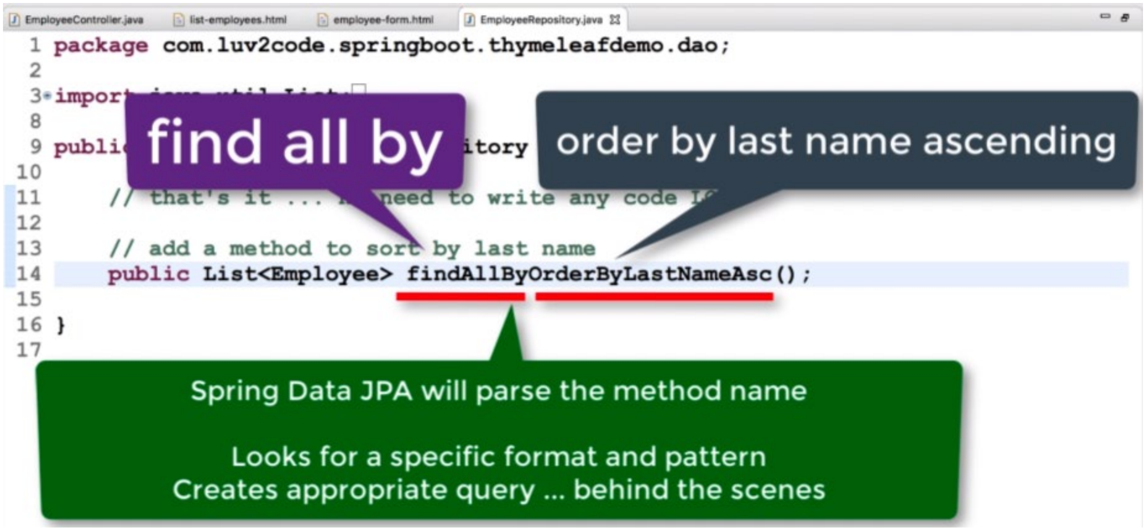
Note: We will keep all the HTML(s) related to employee into one folder.
So, accordingly we need to update in our controller class so that it can find the appropriate view page.

```
//add mapping for "/list"
@GetMapping("/list")
public String getEmployeesList(Model m) {

    List<Employee> theEmployees = employeeService.findAll();

    //add to the spring model
    m.addAttribute("employees",theEmployees);
    return "employees/list-employees";
}
```



```
src/main/resources
  static
    css
    index.html
  templates
    employees
      list-employees.html
    helloworld.html
  application.properties
src/test/java
```

We want to get the list of employees from the database which must be sorted on to the basis of Last Name then we will add an abstract method into the **EmployeeRepository** Interface:



```java
package com.luv2code.springboot.thymeleafdemo.dao;

import 

publi                                   itory

    // that's it ... need to write any code 

    // add a method to sort by last name
    public List<Employee> findAllByOrderByLastNameAsc();

}
```

find all by

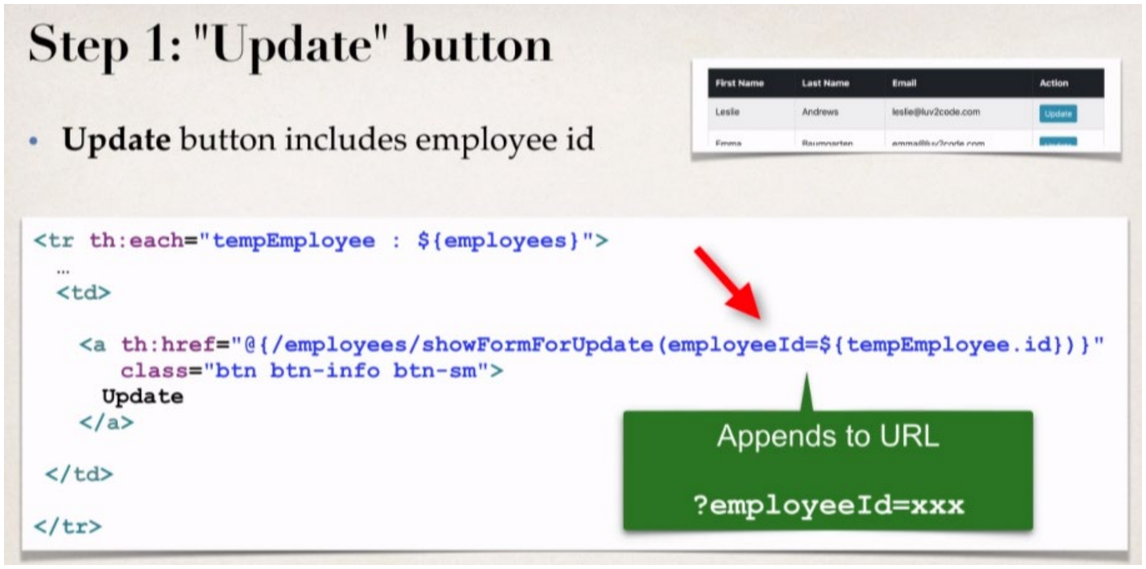
order by last name ascending

Spring Data JPA will parse the method name

Looks for a specific format and pattern
Creates appropriate query ... behind the scenes

Update the Employee

1. “Update” button.



Step 1: "Update" button

- **Update** button includes employee id

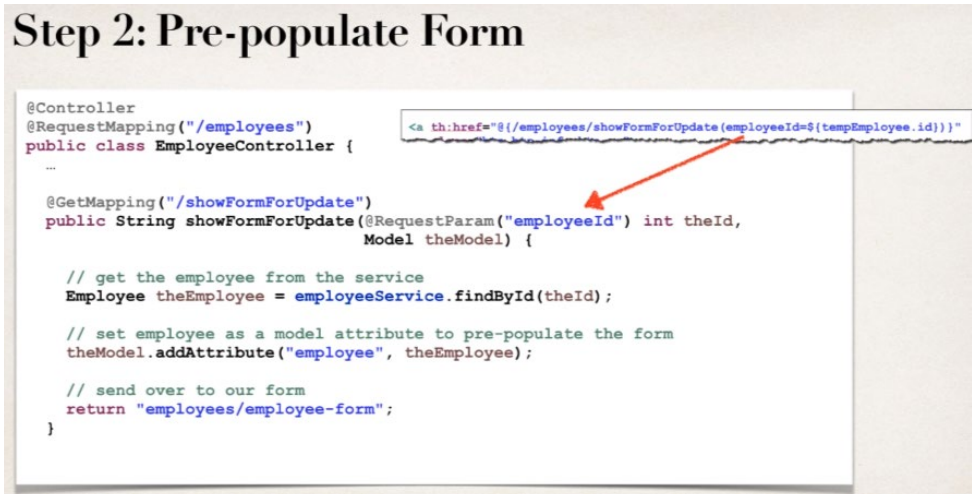
First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update
Emma	Baumgarten	emma@luv2code.com	Update

```html
<tr th:each="tempEmployee : ${employees}">
  ...
  <td>

    <a th:href="@{/employees/showFormForUpdate(employeeId=${tempEmployee.id})}"
       class="btn btn-info btn-sm">
      Update
    </a>

  </td>

</tr>
```

Appends to URL

?employeeId=xxx

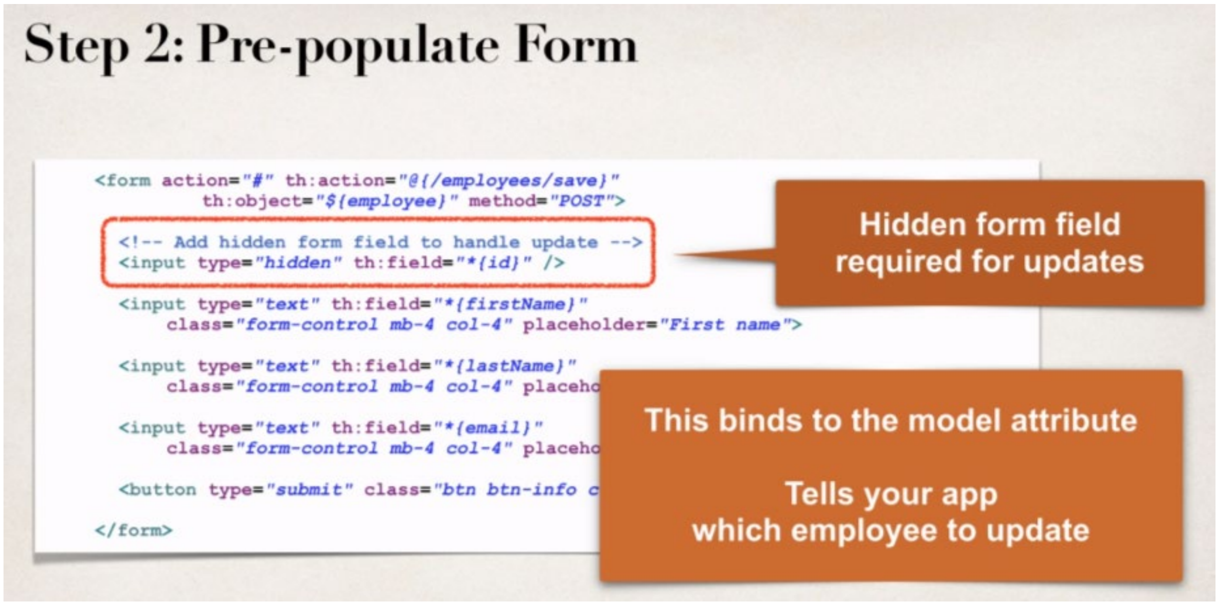
2. <u>Pre-populate the form.</u>



Step 2: Pre-populate Form

```
<a th:href="@{/employees/showFormForUpdate(employeeId=${tempEmployee.id})}"
```

```
@Controller
@RequestMapping("/employees")
public class EmployeeController {
  …

  @GetMapping("/showFormForUpdate")
  public String showFormForUpdate(@RequestParam("employeeId") int theId,
                                  Model theModel) {

    // get the employee from the service
    Employee theEmployee = employeeService.findById(theId);

    // set employee as a model attribute to pre-populate the form
    theModel.addAttribute("employee", theEmployee);

    // send over to our form
    return "employees/employee-form";
  }
```



Step 2: Pre-populate Form

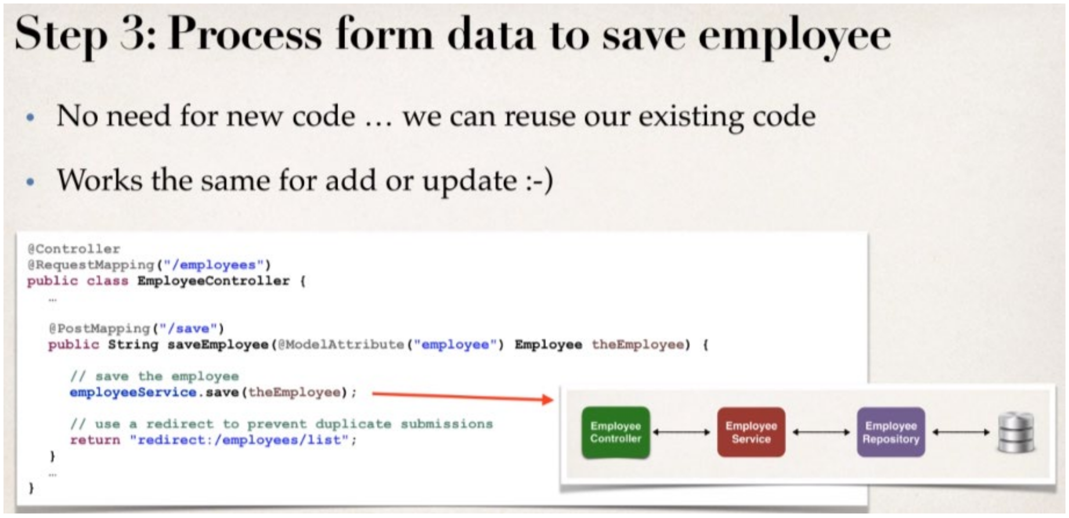
```
<form action="#" th:action="@{/employees/save}"
         th:object="${employee}" method="POST">

  <!-- Add hidden form field to handle update -->
  <input type="hidden" th:field="*{id}" />

  <input type="text" th:field="*{firstName}"
      class="form-control mb-4 col-4" placeholder="First name">

  <input type="text" th:field="*{lastName}"
      class="form-control mb-4 col-4" placeho

  <input type="text" th:field="*{email}"
      class="form-control mb-4 col-4" placeho

  <button type="submit" class="btn btn-info c

</form>
```

Hidden form field required for updates

This binds to the model attribute

Tells your app which employee to update

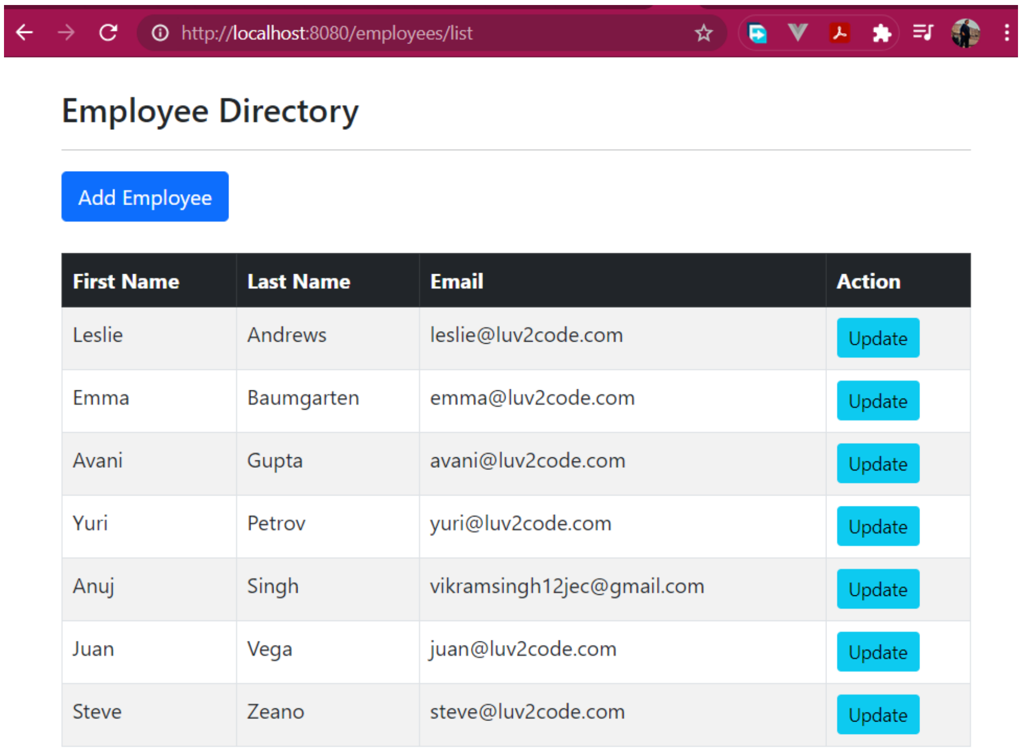
3. <u>Process form data.</u>



Step 3: Process form data to save employee

- No need for new code … we can reuse our existing code
- Works the same for add or update :-)

```
@Controller
@RequestMapping("/employees")
public class EmployeeController {
  …

  @PostMapping("/save")
  public String saveEmployee(@ModelAttribute("employee") Employee theEmployee) {

    // save the employee
    employeeService.save(theEmployee);

    // use a redirect to prevent duplicate submissions
    return "redirect:/employees/list";
  }
  …
}
```



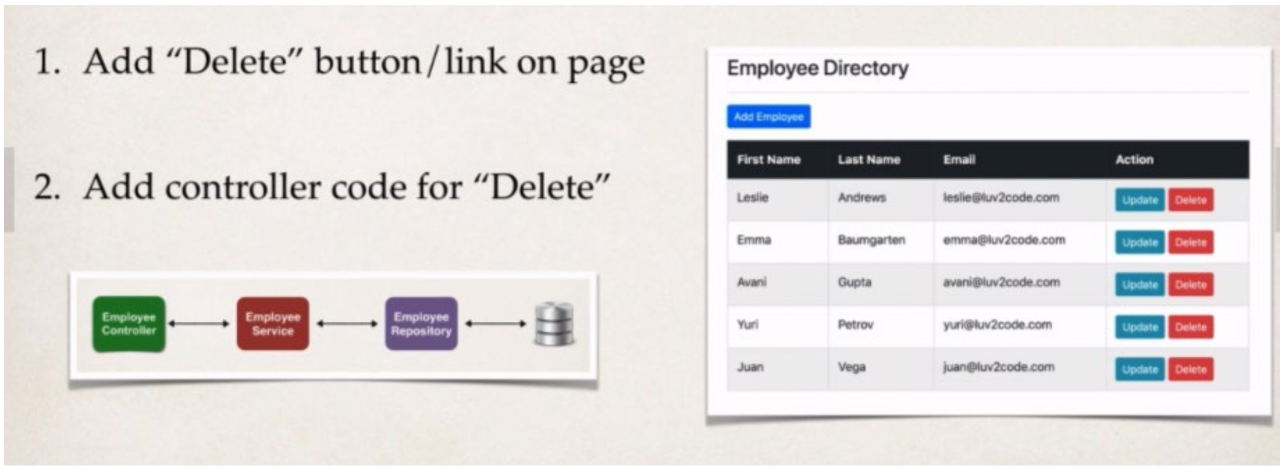
http://localhost:8080/employees/list

Employee Directory

Add Employee

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update
Emma	Baumgarten	emma@luv2code.com	Update
Avani	Gupta	avani@luv2code.com	Update
Yuri	Petrov	yuri@luv2code.com	Update
Anuj	Singh	vikramsingh12jec@gmail.com	Update
Juan	Vega	juan@luv2code.com	Update
Steve	Zeano	steve@luv2code.com	Update

Delete Employee

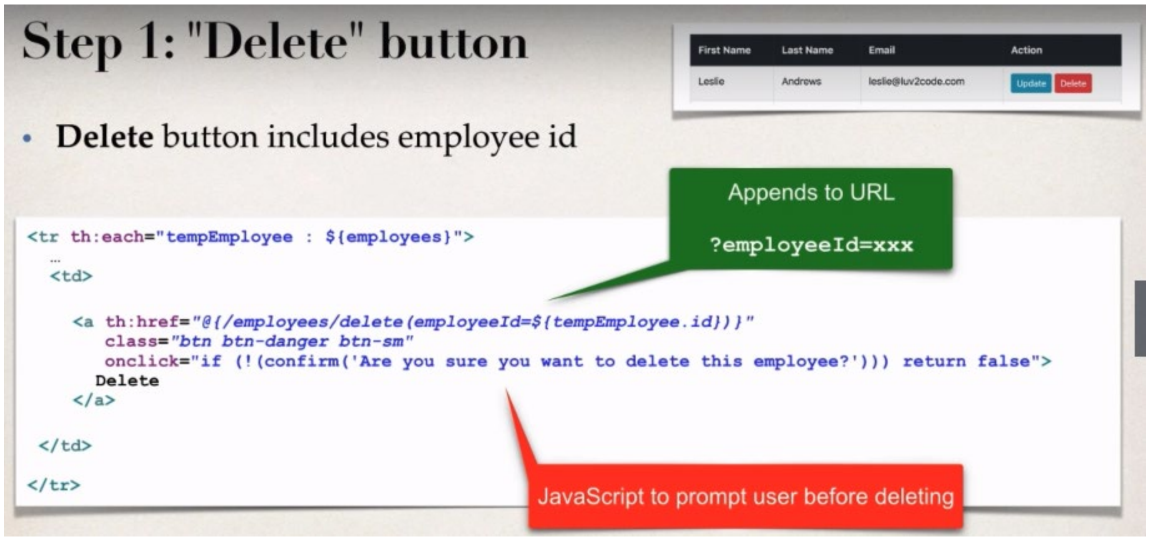


1. Add "Delete" button / link on page
2. Add controller code for "Delete"

Employee Directory

Add Employee

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update Delete
Emma	Baumgarten	emma@luv2code.com	Update Delete
Avani	Gupta	avani@luv2code.com	Update Delete
Yuri	Petrov	yuri@luv2code.com	Update Delete
Juan	Vega	juan@luv2code.com	Update Delete

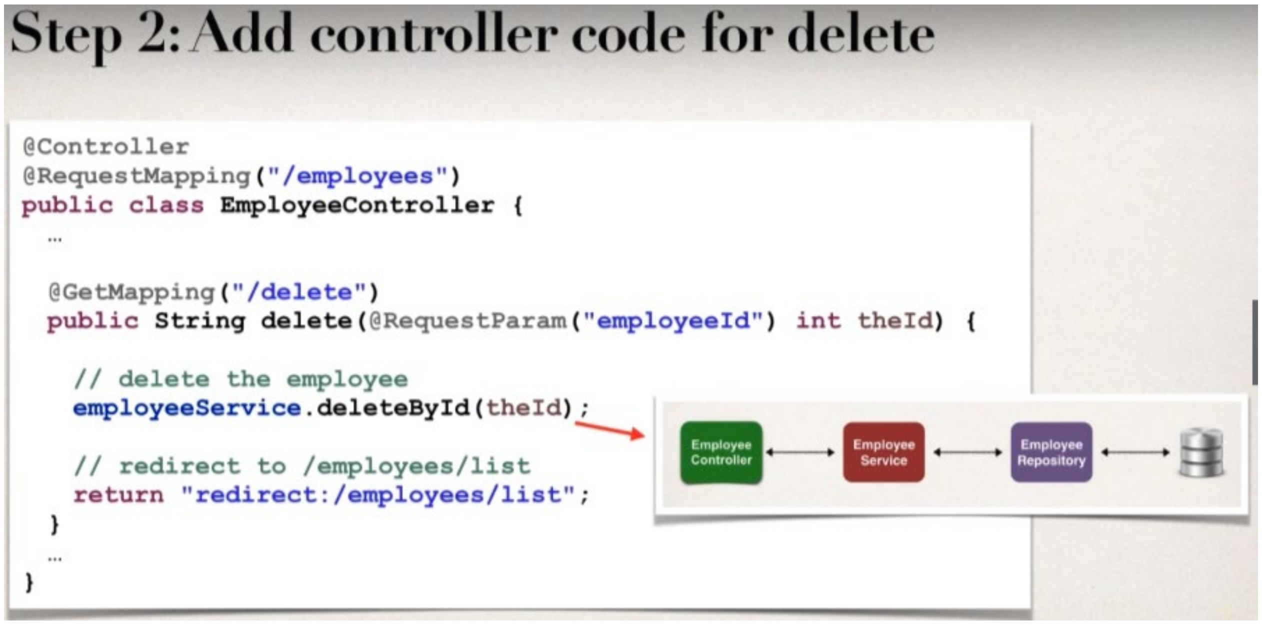


Step 1: "Delete" button

- **Delete** button includes employee id

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update Delete

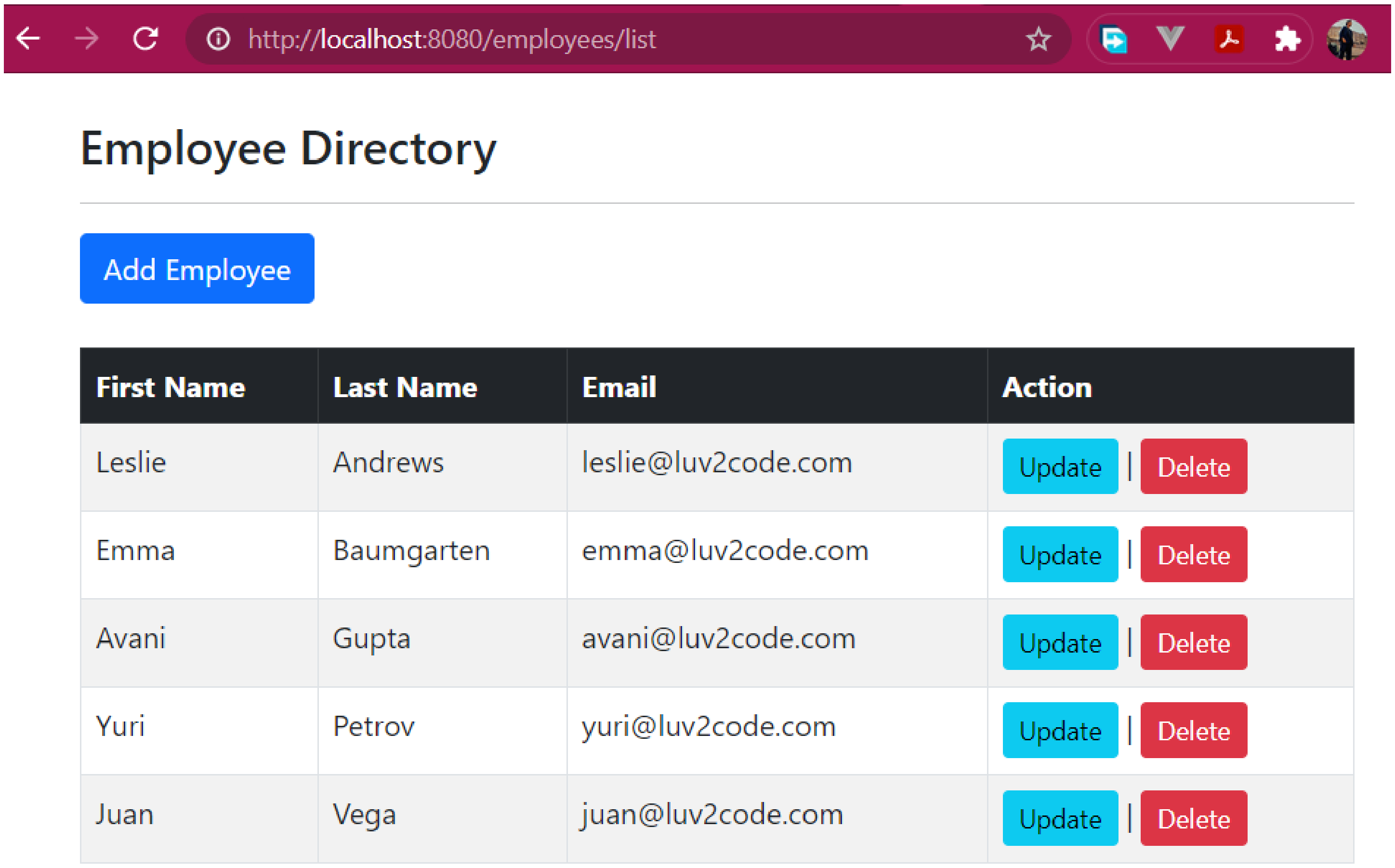
```
<tr th:each="tempEmployee : ${employees}">
  ...
  <td>

    <a th:href="@{/employees/delete(employeeId=${tempEmployee.id})}"
       class="btn btn-danger btn-sm"
       onclick="if (!(confirm('Are you sure you want to delete this employee?'))) return false">
      Delete
    </a>

  </td>

</tr>
```



Step 2: Add controller code for delete

```
@Controller
@RequestMapping("/employees")
public class EmployeeController {
  ...

  @GetMapping("/delete")
  public String delete(@RequestParam("employeeId") int theId) {

    // delete the employee
    employeeService.deleteById(theId);

    // redirect to /employees/list
    return "redirect:/employees/list";
  }
  ...
}
```

Employee Controller
Employee Service
Employee Repository



http://localhost:8080/employees/list

Employee Directory

Add Employee

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update | Delete
Emma	Baumgarten	emma@luv2code.com	Update | Delete
Avani	Gupta	avani@luv2code.com	Update | Delete
Yuri	Petrov	yuri@luv2code.com	Update | Delete
Juan	Vega	juan@luv2code.com	Update | Delete

Link to this project having registration page as well: Link

Link to this project having login and logout support (in-memory): Link

Link to this project having login and logout support

(JDBC authentication with encrypted password): Link

FAQ: How to Configure Multiple Datasources in Spring and Spring Boot

Answer

Here are two examples, one using regular Spring and another using Spring Boot

1. Multiple Data Sources in Spring
This project shows how to configure multiple datasources in Spring. The project makes use of all Java configuration with Spring. The project is based on Maven.

In this project, we connect to two different databases: web_customer_tracker and employee_directory

1. SQL Scripts

The SQL scripts are located in the directory:

sql-scripts

- customer.sql: creates the database schema "web_customer_tracker", also adds sample data

- employee.sql: creates the database schema "employee_directory", also adds sample data

You will need to run these scripts accordingly.

2. Data source configuration

The project includes two configuration files to the data source configuration. The files are in the directory:

src/main/resources

- customer-persistence-mysql.properties

- employee-persistence-mysql.properties

3. Spring All Java Configuration

Directory: src/main/java/com/luv2code/demo/datasources/config

View the file: DemoAppConfig.java

This file defines two datasources using the @Bean annotation. One datasource for customerDataSource and another for employeeDataSource. The datasources also need their respective session factories and transaction managers

4. Java DAO code

The project includes DAOs for Customer and Employee. Make note of the @Autowired for the respective session factory. Also make note of the use of @Transactional with the name the of appropriate bean.

5. Controller code

The controller makes use of the customer and employee DAOs. The data is placed in the model.

6. View page

File: display-results.jsp

This JSP page displays the results. It has an HTML table for Employee data and another HTML table for Customer data.

----

2. Multiple Data Sources in Spring Boot
Creating a custom data source using @Configuration and DataSourceBuilder

- Create configuration class. Specify the prefix of your properties

```java
@Configuration
public class DemoConfiguration {

  @Bean
  @ConfigurationProperties("app.datasource")
  public DataSource dataSource() {
    return DataSourceBuilder.create().build();
  }
}
```

1. Add these to application.properties. Note the prefix of properties. Also note: "jdbc-url" … not just “url"

```
#
# JDBC properties
#
app.datasource.jdbc-url=jdbc:mysql://localhost:3306/employee_directory?useSSL=false&serverTimezone=UTC
app.datasource.username=springstudent
app.datasource.password=springstudent
```

Now if you want more datasources … just use more @Beans

```
@Bean
@ConfigurationProperties(“alpha.datasource")
public DataSource alphaDataSource() {
  return DataSourceBuilder.create().build();
}
```

```
@Bean
@ConfigurationProperties(“bravo.datasource")
public DataSource betaDataSource() {
  return DataSourceBuilder.create().build();
}
```

In application.properties

```
alpha.datasource.jdbc-url=jdbc:mysql://localhost:3306/employee_directory?useSSL=false&serverTimezone=UTC
alpha.datasource.username=springstudent
alpha.datasource.password=springstudent

beta.datasource.jdbc-url=jdbc:mysql://localhost:3306/demo?useSSL=false&serverTimezone=UTC
beta.datasource.username=foo
beta.datasource.password=bar
```

Resources for this lecture: LINK

FAQ: Spring Student Questions

Congrats for finishing the course.
A frequently asked question is "Where to go from here?" A lot of developers want to further their knowledge by learning advanced Spring topics and practicing projects.

I've compiled a list of resources that you can use to get more information on Spring advanced features. Enjoy!

Spring Boot and Angular
- https://github.com/dsyer/spring-boot-angular

Spring MVC and File Upload
- https://spring.io/guides/gs/uploading-files/

Spring RESTful web services
- https://spring.io/guides/gs/rest-service/

Spring Security for Web Apps
- https://spring.io/guides/gs/securing-web/

Spring and Facebook
- https://spring.io/guides/gs/accessing-facebook/

Spring and Twitter
- https://spring.io/guides/gs/accessing-twitter/

--- **Build a Basic CRUD App with Angular and Spring Boot**
https://developer.okta.com/blog/2017/12/04/basic-crud-angular-and-spring-boot
=====

FAQ: I would like to see examples of real-world projects that use Spring
Answer:
Here are some sample Spring projects you can look at.

They are of moderate size complexity

Project Sagan
This is a real-world app that powers the Spring.io website. It is in production and used by thousands of users each day.

You can get information about the project and get source code here: - https://github.com/spring-io/sagan/wiki
---

Spring Petstore Example
This is an example project for the classic PetClinic / PetStore example. https://github.com/spring-projects/spring-petclinic
---

E-Commerce Product - Broadleaf
https://www.broadleafcommerce.com/
The Broadleaf product is based on Spring and Hibernate. You can get details on their framework and source code at the link below

https://www.broadleafcommerce.com/framework

OpenSource Projects Using Spring
Access real-world projects that make use of Spring code
- http://www.programcreek.com/2012/08/open-source-projects-that-use-spring-framework/

Finally there are some other instructors here on Udemy that created courses on Spring ecommerce, angular etc. Be sure to check the reviews
and perform your own research on those courses. I am not involved in any of those other courses. I just wanted to pass information along :-)
=====

FAQ: How to Host my Java apps Online?

Here's a free guide that walks you through the steps:

The Ultimate Guide to Hosting a Java Web App with Amazon Web Services (AWS) http://coderscampus.com/ultimate-guide-hosting-java-web-app-amazon-web-services-aws/
=====

Student Question I want a solution for hiding customer id in URL. Maybe,change request from GET to POST?

Solution
In the files below, look for modified code. Simply search for the text "luv2code: UPDATES".

Here's the basic approach.

For the links, change the table to use forms. There is a form for each row of data. The form would be setup to POST the data. Each row would have a unique button with the ID embedded. Apply special CSS to make the Submit button look like a hypertext link.

The controller request mappings will now support @PostMapping. This is for /showFormForUpdate and /delete

---

Notes about the solution.
Using the POST method does not add security. It simply "hides" the request data. But any web user can still easily see the data. All they have to do is use Chrome Dev Tools or FireFox Firebug. So, using POST is only giving you "security by obscurity" which is weak if you have highly sensitive data.

If you have highly sensitive data then you should use SSL encryption on your server and make use of Spring Security to protect sensitve web URLs in your app.

Having the customer ID in the URL is not a problem. The GET approach is a standard practice that is used in the industry. However the solution was provided based on student's request.

In our example, the ID is for customers ... but this could easily be a product ID. The important thing is the customer ID is not sensitive data.

If you check the major ecommerce sites like Amazon or Best Buy, the product ID is heavily used in the URL.

The URLs below are live URLs on production ecommerce systems that use product ID in the URL.

Amazon
- https://www.amazon.com/dp/B01DFKC2SO
BestBuy
- http://www.bestbuy.com/site/amazon-echo-dot/5578864.p?skuId=5578864

Solution Source Code
- https://gist.github.com/darbyluv2code/df856411a3e0c926a4654660045acda4
======

FAQ: Which more secure? GET or POST?
Note, simply using the POST method does not add secure encryption. The data is still sent in the clear without any protection or encryption.

See this link:

Is either GET or POST more secure than the other?
https://stackoverflow.com/questions/198462/is-either-get-or-post-more-secure-than-the-other
---

You can use SSL for enterprise-grade network security and encryption.

Here's a tutorial on Tomcat SSL: https://www.mulesoft.com/tcat/tomcat-ssl

---

Deploying Spring Boot WAR file with JSP to Tomcat

Deploy Spring Boot apps with JSP to Tomcat
You can deploy a Spring Boot application using JSP to Tomcat. In this scenario, we will create a WAR file and deploy the WAR to the Tomcat server running externally. This is known as a traditional deployment.

High-level steps
1. Update main Spring Boot application

2. Update Maven POM file

3. Update application.properties

4. Move JSP view files to WEB-INF/view

5. Create WAR file

6. Deploy to Tomcat

Spring Boot Reference Manual
For full details on this process, see the Spring Boot Reference Manual: Section 92.1 Creating a Deployable WAR file

Working Example
I have a full working project. You can download this app and perform test deployments to Tomcat

Download: deploy-spring-boot-and-jsp-on-tomcat.zip

This app is a very simple helloworld example that exposes a "/test" request mapping

```
package org.demo.bootjsp.controller;

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

@Controller
```

```
public class HelloWorldController {

    @RequestMapping("/test")
    public String sayHello() {
        return "hello";
    }

}
```

and a simple JSP page: hello.jsp

```
    <html><body>

    <p>
    Hello World! Time is <%= new java.util.Date() %>
    </p>

    <p>
    We are running on  <%= application.getServerInfo() %>!!!
    </p>

    </body></html>
```

Detailed steps
1. Update main Spring Boot application
In your main Spring Boot application, you need to

a. extend the SpringBootServletInitializer

b. override the configure(...) method

Your code should look like this

```
package org.demo.bootjsp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class DemowebApplication extends SpringBootServletInitializer {

  @Override
  protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
          return application.sources(DemowebApplication.class);
  }

  public static void main(String[] args) {
          SpringApplication.run(DemowebApplication.class, args);
  }

}
```

2. Update Maven POM file

Update your POM.xml to use WAR packaging

`<packaging>war</packaging>`

In POM.xml, add dependency to be able to compile JSPs

```
<dependency>
   <groupId>org.apache.tomcat.embed</groupId>
   <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Make sure the Tomcat embedded does not interfere with external Tomcat server

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-tomcat</artifactId>
   <scope>provided</scope>
</dependency>
```

3. Update application.properties

In your application.properties file, you should have

```
spring.mvc.view.prefix=/WEB-INF/view/
spring.mvc.view.suffix=.jsp
```

4. Move JSP view files to WEB-INF/view

Move your JSP view pages should to `src/main/webapp/WEB-INF/view`

5. Create WAR file

Create the WAR file with the command: `mvn clean package`

This will generate a WAR file in your project directory: **target/bootjspdemo.war**

6. In Eclipse, stop all servers you may have running

7. Outside of Eclipse, run your Tomcat server

8. Copy your WAR file to the **<<tomcat-install-dir>>/webapps** directory

Wait for about 15-30 seconds for Tomcat to deploy your app. You will know your app is deployed when you see a new folder created based on your WAR file name. In our example, you will see a new directory named: **bootjspdemo**

9. In a web browser, access your app at: `http://localhost:8080/bootjspdemo/test`

Replace <<bootjspdemo>> with the name of your WAR file if you are using a different app

If everything is successful, you will see your application's web page.

Congratulations! You deployed a Spring Boot WAR file with JSP on to a Tomcat server :-)