Thymeleaf with Spring Boot

What is Thymeleaf?

Thymeleaf

www.thymeleaf.org

- Thymeleaf is a Java templating engine
- Commonly used to generate the HTML views for web apps
- However, it is a general purpose templating engine
  - Can use Thymeleaf outside of web apps (more on this later)

What is Thymeleaf?

Thymeleaf

www.thymeleaf.org

- Thymeleaf is a Java templating engine
- Commonly used to generate the HTML views for
- However, it is a general purpose templating engine

Separate project
Unrelated to spring.io

You can create Java apps with Thymeleaf

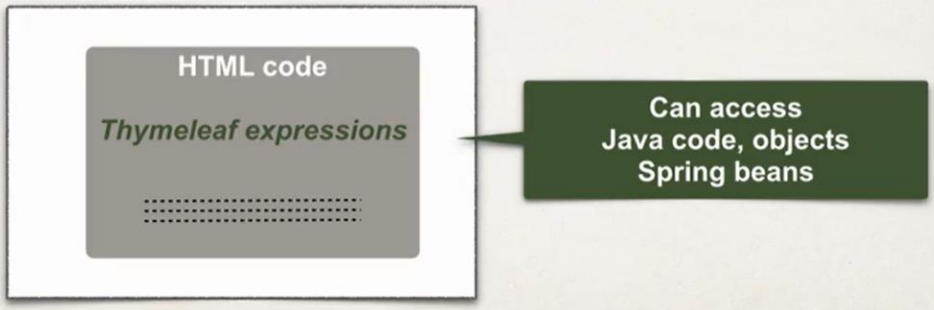
No need for Spring

web ap

But there is a lot of synergy between the two projects

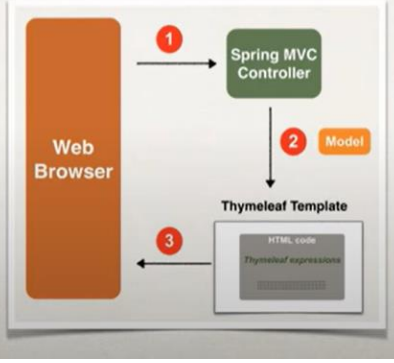
What is a Thymeleaf template?

- Can be an HTML page with some Thymeleaf expressions
- Include dynamic content from Thymeleaf expressions



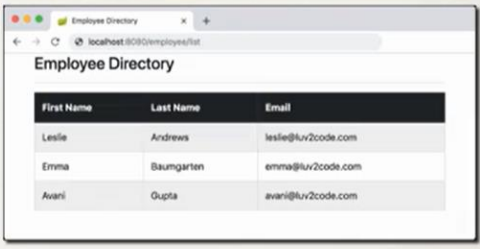
Where is the Thymeleaf template processed?

- In a web app, Thymeleaf is processed on the server
- Results included in HTML returned to browser



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

```
Hi <<firstName>>,

Thanks for joining <<course>>!
```

Email

- CSV Template
  - Generate a monthly report as CSV then upload to Google drive

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

CSV

- PDF Template
  - Generate a travel confirmation PDF then send via email

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

PDF

FAQ: Should I use JSP or Thymeleaf?

- Depends on your project requirements
- If you only need web views you can go either way
- If you need a general purpose template engine (non-web) use Thymeleaf

Development Process

Step-By-Step

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

Step 1: Add Thymeleaf to Maven pom file

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

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 ×

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

Step 2: Develop Spring MVC Controller

File: DemoController.java

```
@Controller
public class DemoContro

  @GetMapping("/")
  public String sayHello

    theModel.addAttribute("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: Create Thymeleaf template

File: DemoController.java

```java
@Controller
public class DemoController {

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

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

      return "helloworld";
   }
}
```

1

Thymeleaf accesses "theDate" from the Spring MVC Model

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

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

</html>
```

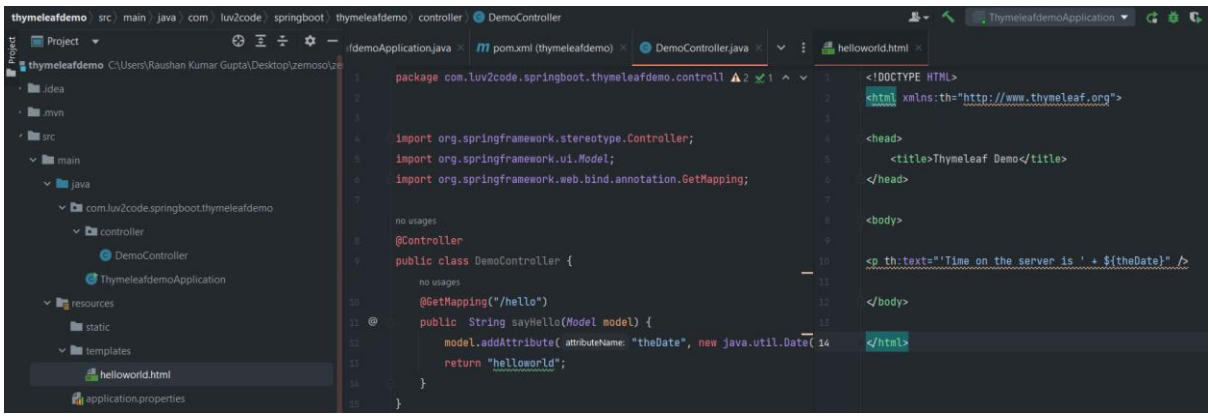
Thymeleaf expression

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

Additional Features

- Looping and conditionals
- CSS and JavaScript integration
- Template layouts and fragments

www.thymeleaf.org



CSS and Thymeleaf

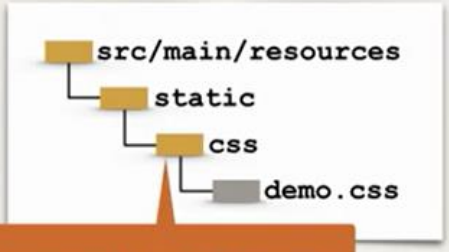
Using CSS with Thymleaf Templates

- You have the option of using
  - Local CSS files as part of your project
  - Referencing remote CSS files
- We'll cover both options in this video

Step 1: Create CSS file

- 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 …



Can be any sub-directory name

File: demo.css

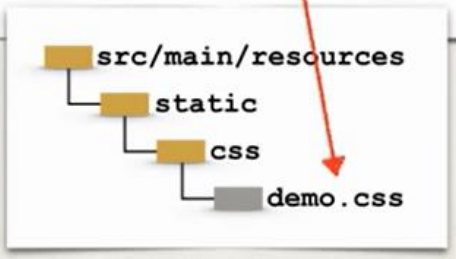
```
.funny {
  font-style: italic;
  color: green;
}
```

Step 2: Reference CSS in Thymeleaf template

File: helloworld.html

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

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

</head>
```

@ symbol
Reference context path of your applicatio
(app root)



Step 3: Apply CSS

File: helloworld.html

```
<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>
```

File: demo.css

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

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

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>
```

3rd Party CSS Libraries - Bootstrap

- Remote Files

```
<head>
  … …

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

  … …
</head>
```

Create HTML Tables with Thymeleaf

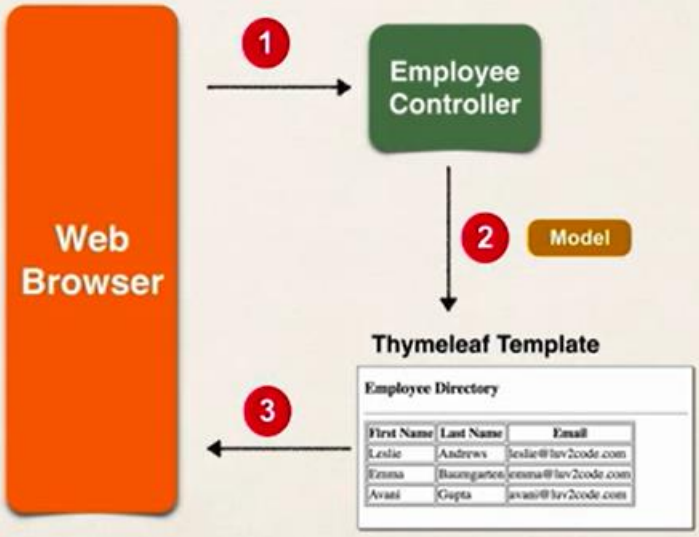
HTML Tables

Start with plain table
Will add CSS in later videos

Employee Directory

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

Big Picture



Development Process

Step-By-Step

1. Create Employee class
2. Create Employee Controller
3. Create Thymeleaf template

Step 3: Create Thymeleaf template

File: list-employees.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

```
<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>
```

Loop parameter

Loop over
list of employees

```
@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

Thymeleaf CRUD - Real Time Project

Application Requirements

From the Boss

Create a Web UI for the Employee Directory

Users should be able to
- Get a list of employees
- Add a new employee
- Update an employee
- Delete an employee

Thymeleaf + Spring Boot

Real-Time Project

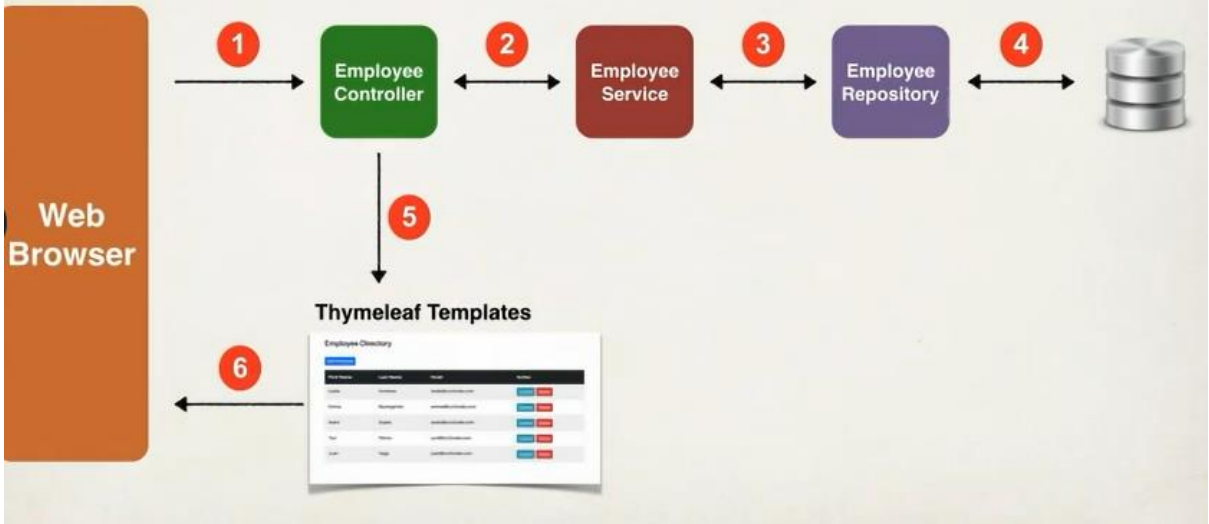
Thymeleaf + Spring Boot

Employee Directory

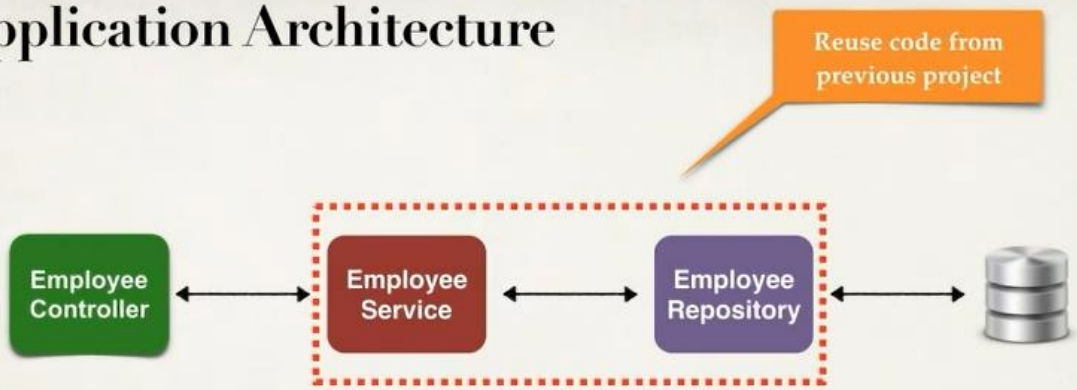
Add Employee

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

Big Picture



Application Architecture



Project Set Up

- We will extend our existing Employee project and add DB integration
- Add **`EmployeeService`**, **`EmployeeRepository`** and **`Employee`** entity
  - Available in one of our previous projects
  - We created all of this code already from scratch … so we'll just copy / paste it
- Allows us to focus on creating **`EmployeeController`** and Thymeleaf templates

Development Process - Big Picture

Step-By-Step

1. Get list of employees
2. Add a new employee
3. Update an existing employee
4. Delete an existing employee

ioc aop,security, rest,autowired, dto with mvc

Add Employee - DEMO

Employee Directory

Add Employee

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

Save Employee

localhost:8080/employees/showFormForAdd

Employee Directory

Save Employee

Michael

Ze

Email

Save

Back to Employees List

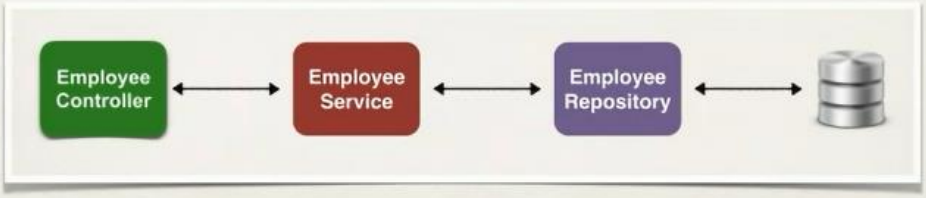
Add Employee

Step-By-Step

1. New **Add Employee** button for list-employees.html

2. Create HTML form for new employee

3. Process form data to save employee



Step 1: New "Add Employee" button

- **Add Employee** bu
  - request mapping

@ symbol
Reference context path of your application
(app root)

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

Add Employee

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 code to show form

```
@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

Thymeleaf and Spring MVC Data Binding

- Thymeleaf has special expressions for binding Spring MVC form data
- Automatically setting / retrieving data from a Java object

Thymeleaf Expressions

- Thymeleaf expressions can help you 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);
```

Step 2: Create HTML form for new employee

```
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

Save Employee

First name

Last name

Email

Save

luv2code

www.luv2code.com

Step 2: Create HTML form for new employee

```
<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

2

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

employee.setFirstName(...)
...
employee.setLastName(...)

Step 2: Create HTML form for new employee

```
<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

2

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

employee.setFirstName(...)
...
employee.setLastName(...)

Example of Data Binding

Step 3: Process form data to save employee

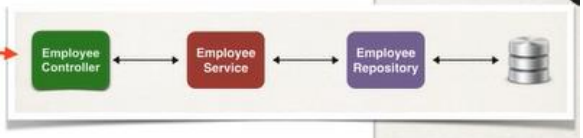
```
@Controller
@RequestMapping("/employees")
public class EmployeeController {

   private EmployeeService emp

   public EmployeeController(E
      employeeService = theEmp
   }

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

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

Form data is passed in

Step 3: Process form data to save employee

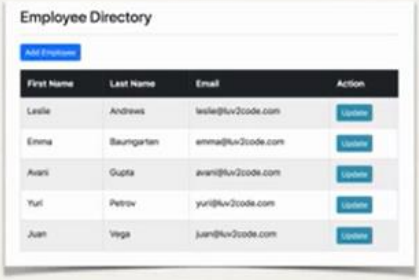
```
@Controller
@RequestMapping("/employees")
public class EmployeeController {

   private EmployeeService employeeService;

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

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

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



Update Employee

Step-By-Step

1. "Update" button
2. Pre-populate the form
3. Process form data



Step 1: "Update" Button

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
Juan	Vega	juan@luv2code.com	Update

Each row has an **Update** link

- current employee id embedded in lin

When **clicked**

- will load the employee from databas
- prepopulate the form

Step 1: "Update" button

- **Update** button includes employee id

```
<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

Step 2: Pre-populate Form

```
@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";
  }

}
```

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

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

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";
  }
  ...
}
```

Eclipse File Edit Source Navigate Search Project Run Window Help luv2code

dev-spring-boot - 36-thymeleafdemo-employees-update/src/main/resources/templates/employees/list-employees.html - Eclipse IDE

*list-employees.html

```
36 >
37 ·
38
39
40 th:each="tempEmployee : ${employees}">
41
42 <td th:text="${tempEmployee.firstName}" />
43 <td th:text="${tempEmployee.lastName}" />
44 <td th:text="${tempEmployee.email}" />
45
46 <!-- Add update button/link -->
47 <td>
48     <a th:href="@{/employees/showFormForUpdate(employeeId=${tempEmployee.id})}"
49         Update
50     </a>
51 </td>
52
53
54 >
55 ·
```

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

Appends to URL

?employeeId=xxx