

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

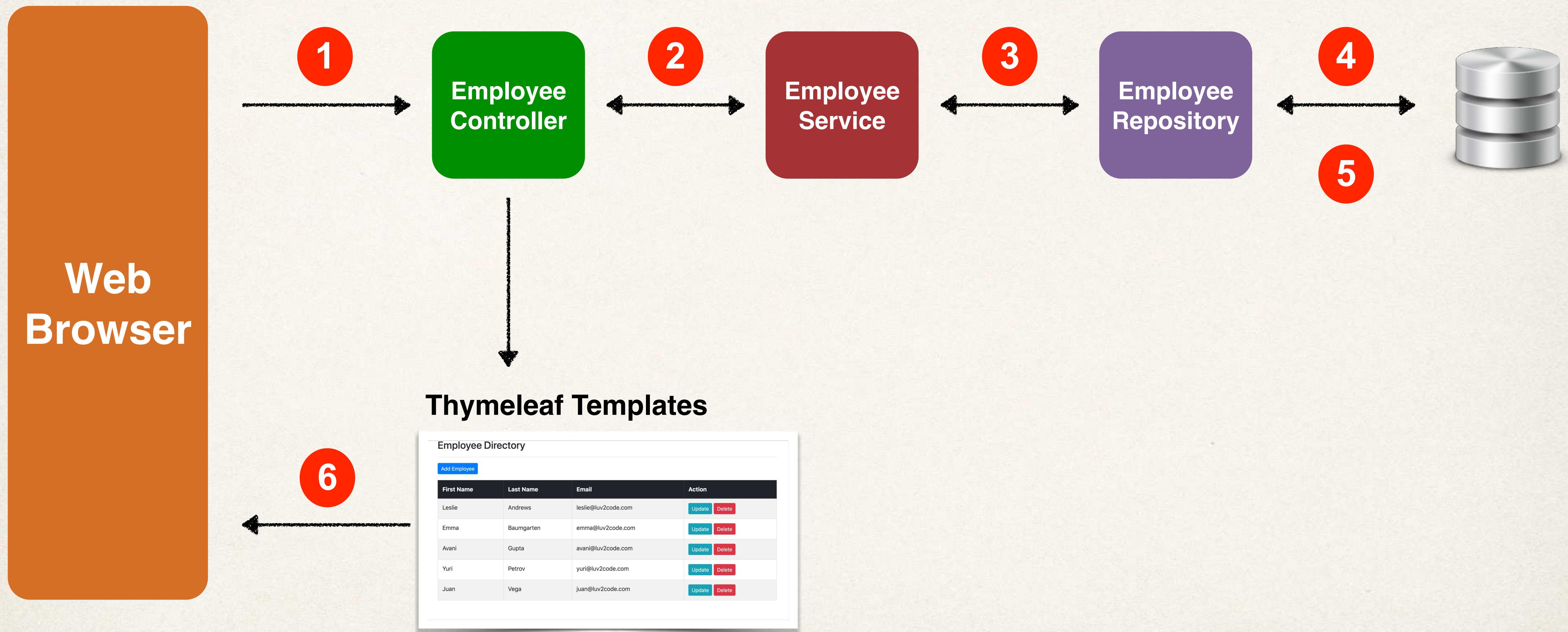
Thymeleaf + Spring Boot

Employee Directory

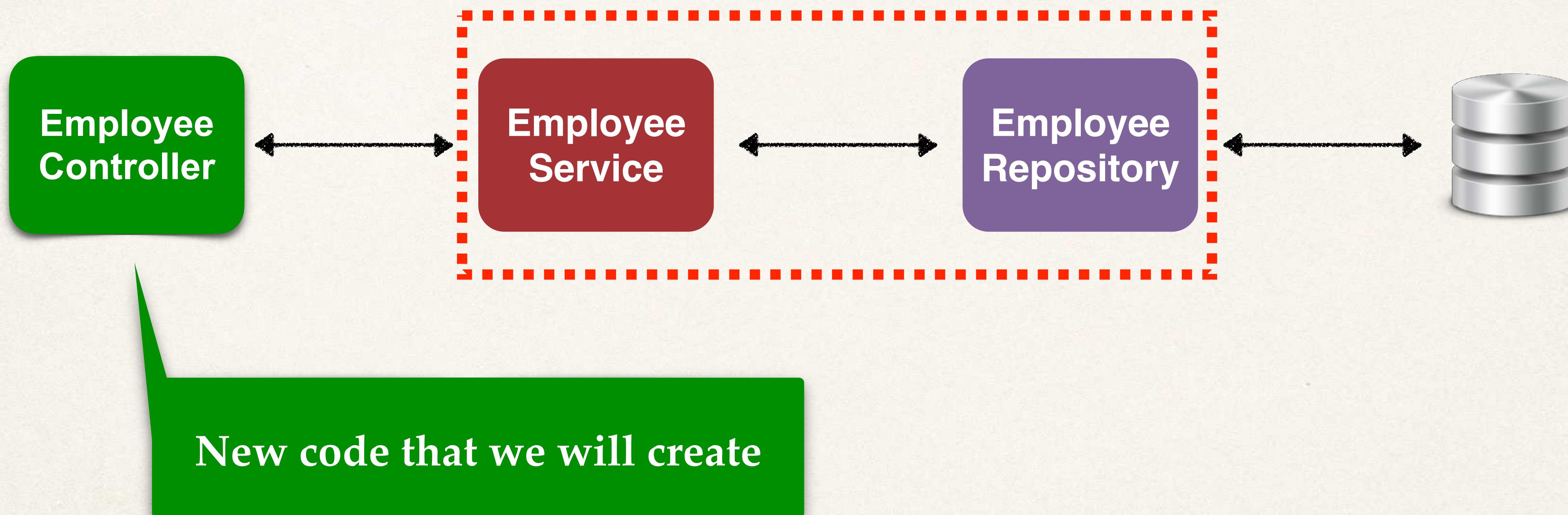
Add Employee

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

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

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

Thymeleaf - Add Employee



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

Add Employee

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

Step-By-Step

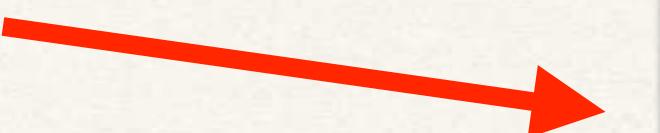
The image shows a screenshot of a web application. At the top left, there is a blue button labeled "Add Employee". To the right of this button is a table titled "Employee Directory". The table has three columns: "First Name", "Last Name", and "Email". The data in the table is as follows:

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

Add Employee

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

2. Create HTML form for new employee



Save Employee

First name

Last name

Email

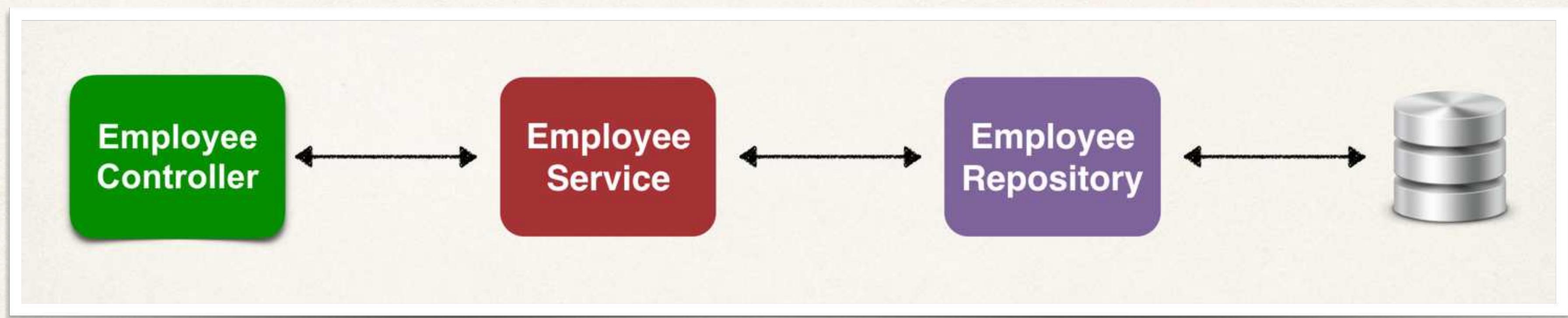
Save

[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 button will href link to
- request mapping **/employees/showFormForAdd**

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

Add Employee

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

Step 1: New "Add Employee" button

- Add Employee button will href link to
- request mapping **/employees/showFormForAdd**

```
<a th:href="@{/employees/showFormForAdd}"  
    class="btn btn-primary btn-sm mb-3">  
    Add Employee  
</a>
```



Apply Bootstrap styles

Docs on Bootstrap styles: www.getbootstrap.com

Button
Button Primary
Button Small
Margin Bottom, 3 pixels

Step 1: New "Add Employee" button

- Add Employee button will href link to
 - request mapping **/employees/showFormForAdd**

```
<a th:href="@{/employees/showFormForAdd}"  
    class="btn btn-primary btn-sm mb-3">  
    Add Employee  
</a>
```



TODO:

Add controller request mapping for
/employees/showFormForAdd

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";  
    }  
...  
}
```

src/main/resources/templates/employees/employee-form.html

Our Thymleaf template will
access this data for
binding form data

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
<i>more</i>	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

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

Step 2: Create HTML form for new employee

Save Employee

First name

Last name

Email

Save

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>
```

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

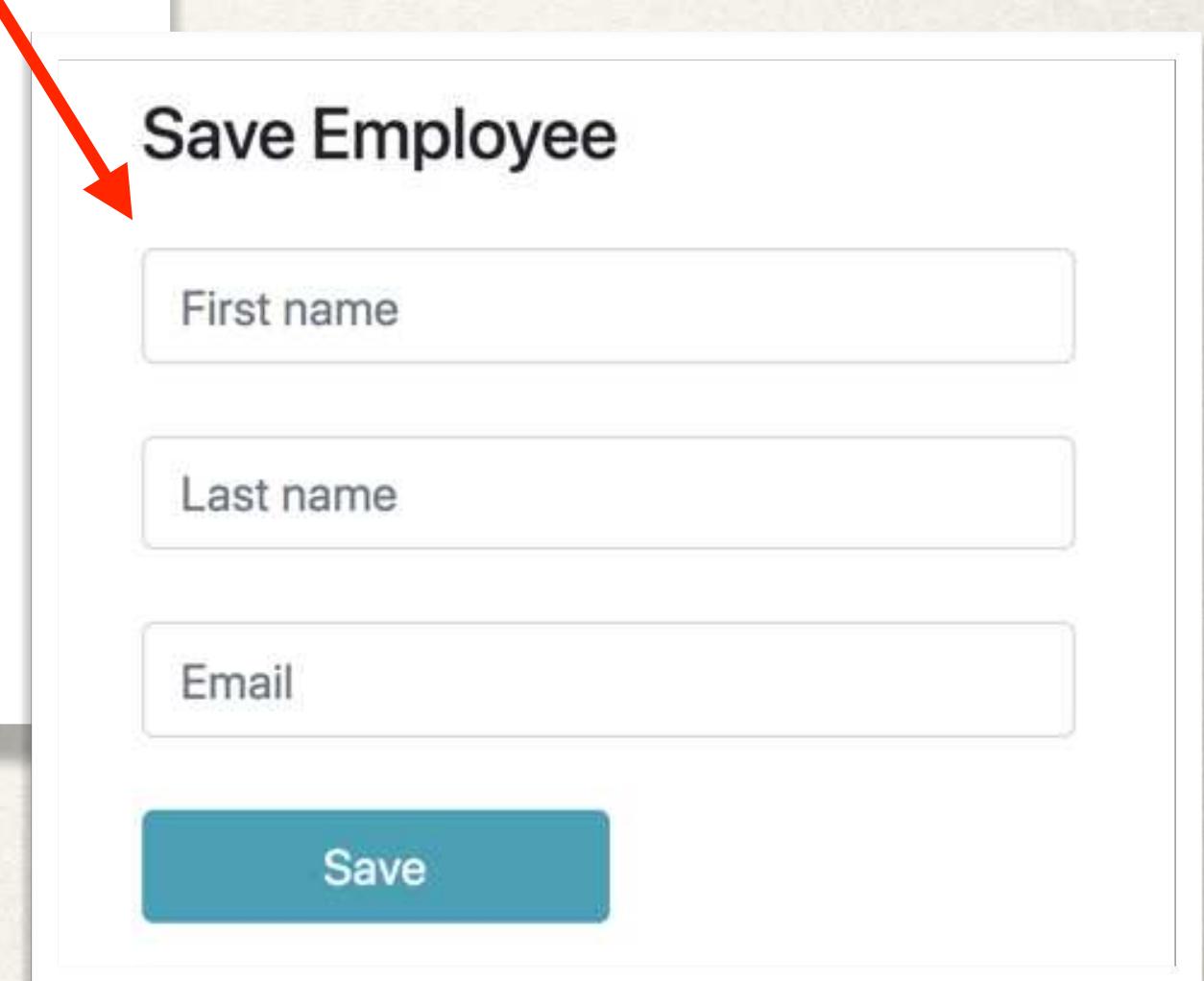
Save Employee

First name

Last name

Email

Save



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"  
       class="form-control mb-4 w-25">
```

Apply Bootstrap styles

Form control
Margin Bottom: 4 pixels
Width: 25%

Save Employee

First name

Last name

Email

Save

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"  
       class="form-control mb-4 w-25">  
  
<input type="text" th:field="*{lastName}" placeholder="Last name"  
       class="form-control mb-4 w-25">  
  
<input type="text" th:field="*{email}" placeholder="Email"  
       class="form-control mb-4 w-25">  
  
<button type="submit" class="btn btn-info col-2">Save</button>  
  
</form>
```

Apply Bootstrap styles

Button
Button Info
Column Span 2

Save Employee

First name

Last name

Email

Save

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"  
      class="form-control mb-4 w-25">
```

```
<input type="text" th:field="*{lastName}" placeholder="Last name"  
      class="form-control mb-4 w-25">
```

```
<input type="text" th:field="*{email}" placeholder="Email"  
      class="form-control mb-4 w-25">
```

```
<button type="submit" class="btn btn-info col-2">Save</button>
```

```
</form>
```

TODO:

Add controller request mapping for
/employees/save

Save Employee

First name

Last name

Email

Save

Step 3: Process form data to save employee

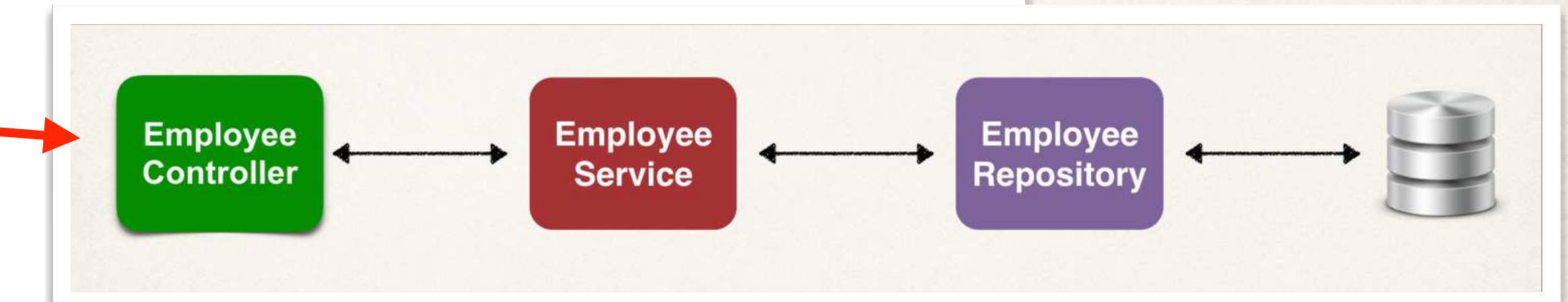
```
@Controller  
@RequestMapping("/e  
public class Employ  
  
    private EmployeeService employeeService;  
  
    public EmployeeController(EmployeeService theEmployeeService) {  
        employeeService = theEmployeeService;  
    }  
  
    @PostMapping("/save")  
    public String saveEmployee(@ModelAttribute("emp")  
  
        // save the employee  
        employeeService.save(theEmployee);  
  
        // use a redirect to prevent duplicate submissions  
        return "redirect:/employees/list";  
    }  
    ...  
}
```

Since only one constructor
@Autowired is optional

Constructor injection

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) {  
  
        // save the employee  
        employeeService.save(theEmployee);  
  
        // use a redirect to prevent duplicate submissions  
        return "redirect:/employees/list";  
    }  
    ...  
}
```



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

Redirect to request mapping
`/employees/list`

"Post/Redirect/Get" pattern

For more info see
www.luv2code.com/post-redirect-get

Thymeleaf - Update Employee



Update Employee - Demo

Employee Directory			
First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	<button>Update</button>
Emma	Baumgarten	emma@luv2code.com	<button>Update</button>
Avani	Gupta	avani@luv2code.com	<button>Update</button>
Yuri	Petrov	yuri@luv2code.com	<button>Update</button>
Juan	Vega	juan@luv2code.com	<button>Update</button>

Update Employee

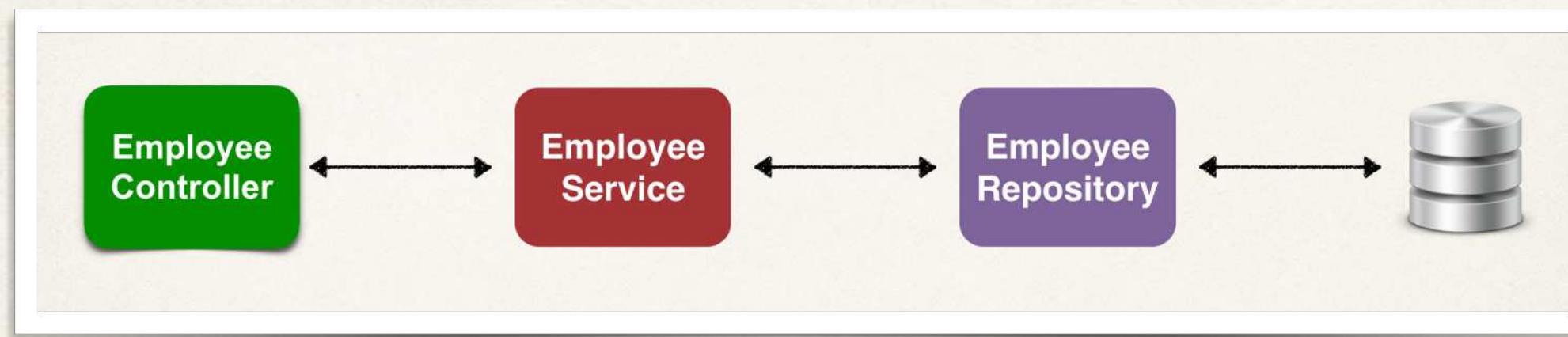
1. "Update" button

2. Pre-populate the form

3. Process form data

Step-By-Step

Employee Directory			
First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	<button>Update</button>
Emma	Baumgarten	emma@luv2code.com	<button>Update</button>
Avani	Gupta	avani@luv2code.com	<button>Update</button>
Yuri	Petrov	yuri@luv2code.com	<button>Update</button>
Juan	Vega	juan@luv2code.com	<button>Update</button>



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 link

When **clicked**

- will load the employee from database
- prepopulate the form

Step 1: "Update" button

- Update button includes employee id

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	<button>Update</button>
Emma	Raumgarten	emma@luv2code.com	<button>Update</button>

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

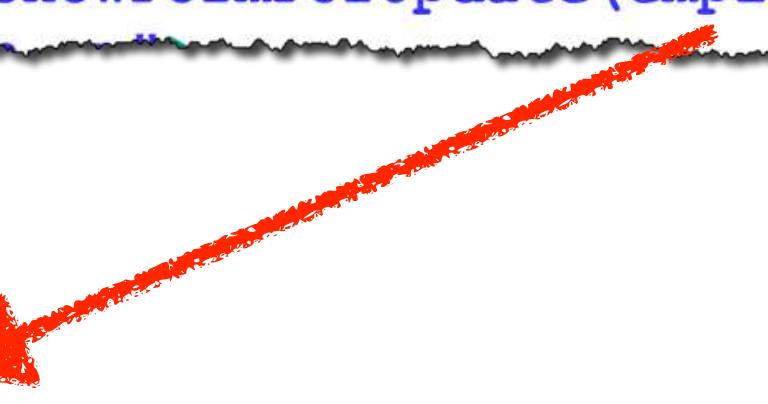
Appends to URL

?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

1

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

employee.getFirstName()

...

employee.getLastName

```
<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 w-25" placeholder="First name">  
  
    <input type="text" th:field="*{lastName}"  
          class="form-control mb-4 w-25" placeholder="Last name">  
  
    <input type="text" th:field="*{email}"  
          class="form-control mb-4 w-25" placeholder="Email">  
  
    <button type="submit" class="btn btn-info col-2">Save</button>  
  
</form>
```

This is how form is
pre-populated
Thanks to calls to getters

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 w-25" placeholder="First name">

    <input type="text" th:field="*{lastName}"
           class="form-control mb-4 w-25" placeholder="Last name">

    <input type="text" th:field="*{email}"
           class="form-control mb-4 w-25" placeholder="Email">

    <button type="submit" class="btn btn-info col-2">Save</button>

</form>
```

Hidden form field
required for updates

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 w-25" placeholder="First name">

    <input type="text" th:field="*{lastName}"
           class="form-control mb-4 w-25" placeholder="Last name">

    <input type="text" th:field="*{email}"
           class="form-control mb-4 w-25" placeholder="Email">

    <button type="submit" class="btn btn-info col-2">Save</button>

</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 w-25" placeholder="First name">

    <input type="text" th:field="*{lastName}"
           class="form-control mb-4 w-25" placeholder="Last name">

    <input type="text" th:field="*{email}"
           class="form-control mb-4 w-25" placeholder="Email address">

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

</form>
```

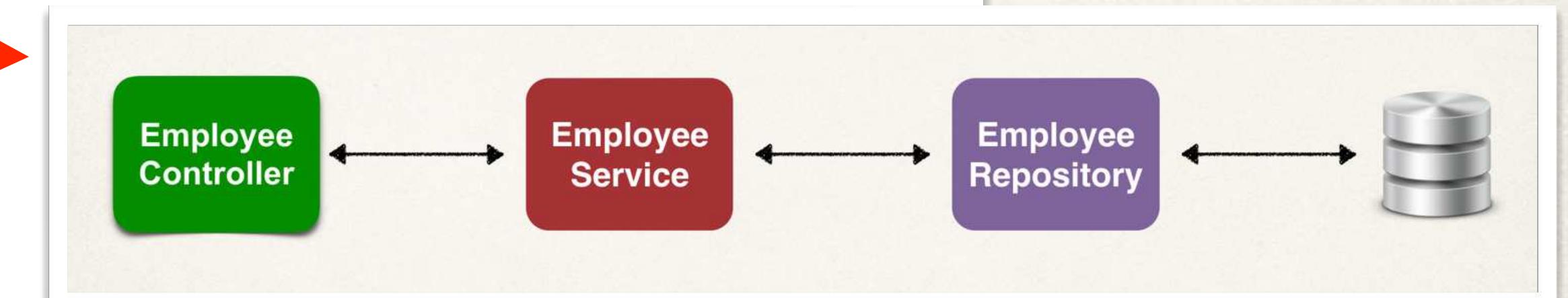
This binds to the model attribute

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";  
    }  
    ...  
}
```



Thymeleaf - Delete Employee



Delete Employee - DEMO

Employee Directory

[Add Employee](#)

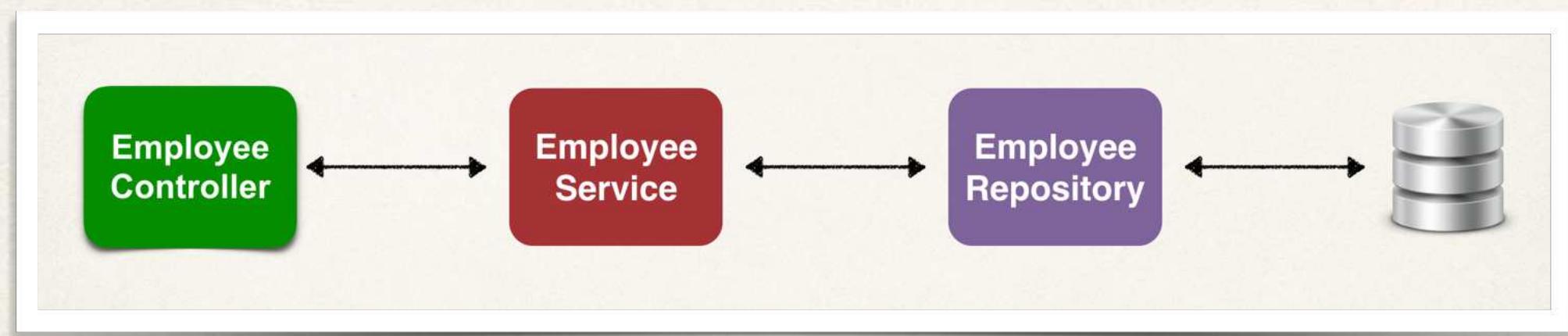
First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	Update Delete
Emma	Baumgarten	emma@luv2code.com	Update Delete
Avani	Gupta	avani@luv2code.com	Update Delete
Yuri	Petrov	yuri@luv2code.com	Update Delete
Juan	Vega	juan@luv2code.com	Update Delete

Delete Employee

Step-By-Step

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

Employee Directory

Add Employee

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

Each row has a **Delete** button/link

- current employee id embedded in link

When **clicked**

- prompt user
- will delete the employee from database

Step 1: "Delete" button

- Delete button includes employee id

First Name	Last Name	Email	Action
Leslie	Andrews	leslie@luv2code.com	<button>Update</button> <button>Delete</button>

```
<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>
```

Appends to URL

?employeeId=xxx

JavaScript to prompt user before deleting

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";  
    }  
    ...  
}
```

