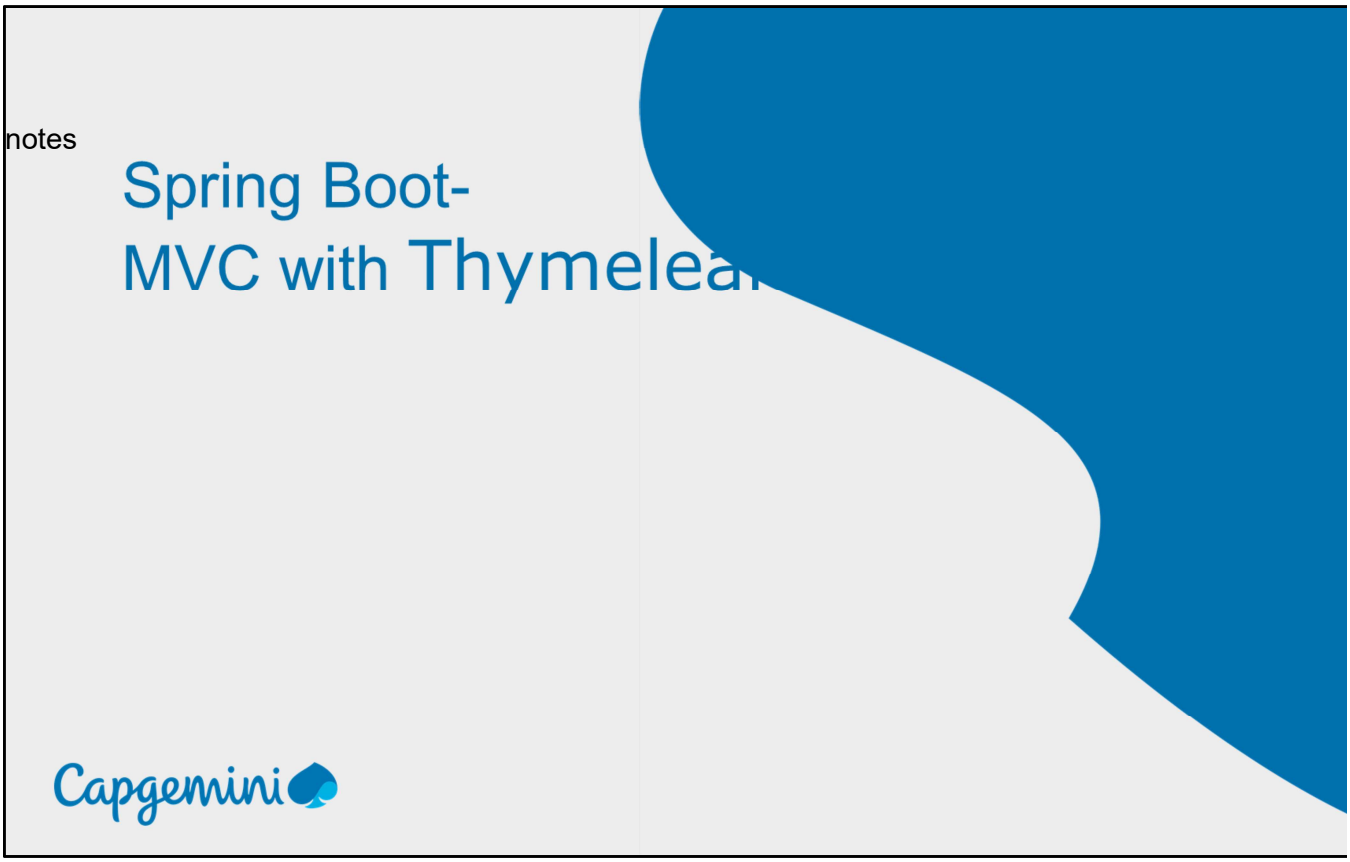
Add instructor notes here.



Objective

- Thymeleaf
- Spring Boot MVC with Thymeleaf

Thymeleaf

- Thymeleaf is a modern server-side Java template engine for both web and standalone environments.
- Thymeleaf's main goal is to bring elegant natural templates to our development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.
- With modules for Spring Framework, a host of integrations with our favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development — although there is much more it can do.
- Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text

What kind of templates can Thymeleaf process?

Thymeleaf allows you to process six kinds of templates, each of which is called a **Template Mode**:

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

Thymeleaf vs. JSP

- Both of them are view layers of Spring MVC. Firstly, the very basic difference is the file extensions. (.jsp & .html)
- Thymeleaf is more like an HTML-ish view when you compare it with JSP views.
- Since it is more HTML-ish code, thymeleaf codes are more readable
- Standard Dialect (The expression language) is much more powerful than JSP Expression Language.
- This looks much more HTML-ish than the JSP version – no strange tags, just some meaningful attributes.
- Variable expressions (${…}) are Spring EL and execute on model attributes, asterisk expressions (*{…}) execute on the form backing bean, hash expressions (#{…}) are for internationalization and link expressions (@{…}) rewrite URLs.

The Template Engine

- Template Engine objects are implementations of the org.thymeleaf.ITemplateEngine interface. One of these implementations is offered by the Thymeleaf core: org.thymeleaf.TemplateEngine
- spring-boot-starter-web: Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container.
- spring-boot-starter-thymeleaf: Starter for building MVC web applications using Thymeleaf views
- spring-boot-starter-test: Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito

Standard Expression Syntax

Simple expressions:
Variable Expressions: ${...}
Selection Variable Expressions: *{...}
Message Expressions: #{...}
Link URL Expressions: @{...}
Fragment Expressions: ~{...}
Literals
Text literals: 'one text', 'Another one!',…
Number literals: 0, 34, 3.0, 12.3,…
Boolean literals: true, false
Null literal: null
Literal tokens: one, sometext, main,…
Text operations:
String concatenation: +
Literal substitutions: |The name is ${name}|

We will take a small break in the development of our grocery virtual store to learn about one of the most important parts of the Thymeleaf Standard Dialect: the Thymeleaf Standard Expression syntax.
We have already seen two types of valid attribute values expressed in this syntax: message and variable expressions:
Welcome to our grocery store! Today is: 13 february 2011

Standard Expression Syntax

Arithmetic operations:
Binary operators: +, -, *, /, %
Minus sign (unary operator): -
Boolean operations:
Binary operators: and, or
Boolean negation (unary operator): !, not
Comparisons and equality:
Comparators: >, <, >=, <= (gt, lt, ge, le)
Equality operators: ==, != (eq, ne)
Conditional operators:
If-then: (if) ? (then)
If-then-else: (if) ? (then) : (else)
Default: (value) ?: (defaultvalue)
Special tokens:
No-Operation: _

Standard Expression Syntax

Thymeleaf Standard Dialect: the Thymeleaf Standard Expression syntax.

We have already seen two types of valid attribute values expressed in this syntax: message and variable expressions:

```
<p th:utext="#{home.welcome}">Welcome to our grocery
store!</p>

<p>Today is: <span th:text="${today}">13 february
2011</span></p>
```

Thymeleaf form

Command object is the name Spring MVC gives to form-backing beans, this is, to objects that model a form's fields and provide getter and setter methods that will be used by the framework for establishing and obtaining the values input by the user at the browser side

Thymeleaf requires you to specify the command object by using a th:object attribute in our <form> tag:

```
<form action="#" th:action="@{/seedstartermng}"
    th:object="${seedStarter}" method="post">
                    ...
</form>
```

Checkbox fields & *Inputs*

- The th:field attribute behaves differently depending on whether it is attached to an <input>, <select> or <textarea>

```
<input type="text" id="datePlanted" name="datePlanted"
       th:value="*{datePlanted}" />
```

- th:field also allows us to define checkbox inputs

```
<div>
  <label th:for="${#ids.next('covered')}"
th:text="#{seedstarter.covered}">Covered</label>
  <input type="checkbox" th:field="*{covered}" />
</div>
```

Radio Button fields

- Radio button fields are specified in a similar way to non-boolean (multi-valued) checkboxes — except that they are not ultivalued, of course

```
<ul>
<li th:each="ty : ${allTypes}">
<input type="radio" th:field="*{type}" th:value="${ty}" />
<label th:for="${#ids.prev('type')}" th:text="#{${'seedstarter.type.' +
ty}}">Wireframe</label>
</li>
</ul>
```

Dropdown/List selectors

- Select fields have two parts: the <select> tag and its nested <option> tags. When creating this kind of field, only the <select> tag has to include a th:field attribute, but the th:value attributes in the nested <option> tags will be very important

```
<select th:field="*{type}">
<option th:each="type : ${allTypes}"
        th:value="${type}"
    th:text="#{${'seedstarter.type.' +
type}}">Wireframe</option>
</select>
```

Demo

SpringBootMVC

Lab 4