Understanding Thymeleaf

Overview

Thymeleaf is a modern server-side Java-based template engine for web and standalone environments. It allows for creating dynamic, maintainable, and easily testable templates for web applications. Thymeleaf templates are typically used in web frameworks like Spring MVC.

Here's an overview of the key aspects of Thymeleaf:

1. **Template Engine:**
   Thymeleaf is a template engine that processes templates and produces HTML (or other formats) as output. It allows you to create templates with embedded expressions and provides mechanisms to evaluate those expressions at runtime to generate dynamic content.

2. **Natural Templates:**
   Thymeleaf templates look similar to standard HTML, making them easy to read and understand. It uses attribute-based syntax, primarily the **`th:`** prefix to denote expressions and actions.

3. **Expression Language:**
   Thymeleaf uses a powerful expression language (Thymeleaf Standard Expression Language - Thymeleaf/Standard) to evaluate expressions within the templates. These expressions can be used for variables, conditions, iterations, etc.

4. **HTML5 Support:**
   Thymeleaf fully supports HTML5, making it suitable for modern web application development. It handles HTML5 features seamlessly, allowing you to use new elements, attributes, and validations.

5. **Integration with Java Frameworks:**
   Thymeleaf integrates well with Java-based frameworks like Spring, Spring Boot, and Java EE. It is widely used in Spring MVC applications, allowing for the easy creation of dynamic HTML templates.

6. **Template Resolvers:**

Thymeleaf supports different template resolvers, allowing you to use various template sources such as files, classpaths, databases, etc., to load templates.

In summary, Thymeleaf is a versatile and feature-rich template engine that simplifies the creation of dynamic and maintainable templates for Java-based web applications. Its ease of use, support for modern web standards, and seamless integration with Java frameworks make it a popular choice among developers.

How to create a form using thymeleaf:

- First we are going to add the dependency for thymeleaf in our ***pom.xml*** file.

  Maven Repository Link:- https://mvnrepository.com/artifact/org.thymeleaf/thymeleaf

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

- Now, let's suppose we are building a login form for an employee management system that has only two attributes on the form ***username*** and ***password***.

- In Order to do so we first need to define the entity and dto classes for the employee Management System. Given below is a sample code for the entity and the sample class.

```
// Employee Dto Class

public class EmployeeDto {

    private String username;
    private String password;
    private String role;
}
```

```
// Employee User Class
import java.util.List;
import javax.persistence.*;


@Entity
```

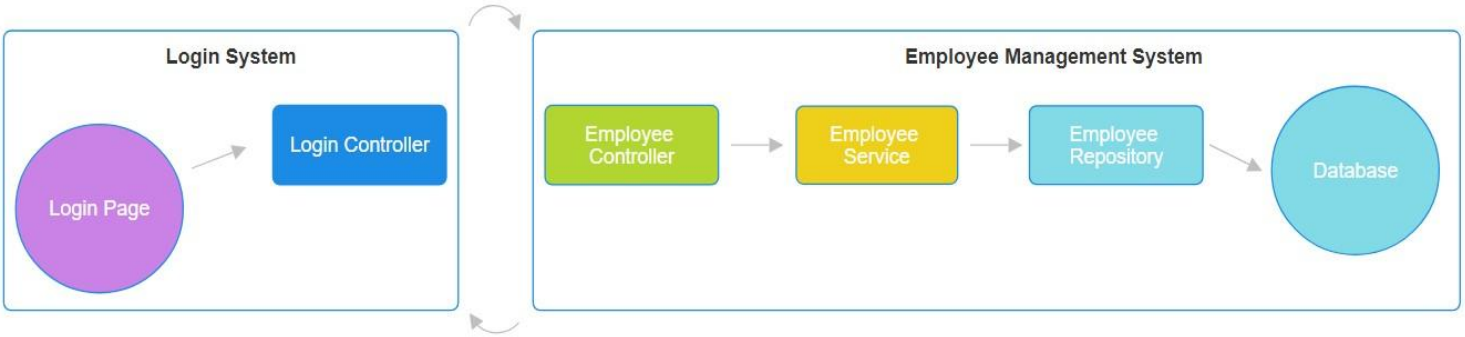
```
public class Employee {

    @Id
    private Long id;
    private String username;
    private String password;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    Private List<Role>;
}
```

```
// Role Class
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Role {

    @Id
    private Long roleId;
    private String roleName;
}
```

- Also, we need to define a LoginController so that the HTML page can be displayed when hitting the **"/login"** api.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class LoginController {
      @GetMapping("/login")
      public String login()
      {
            return "login";
      }
}
```

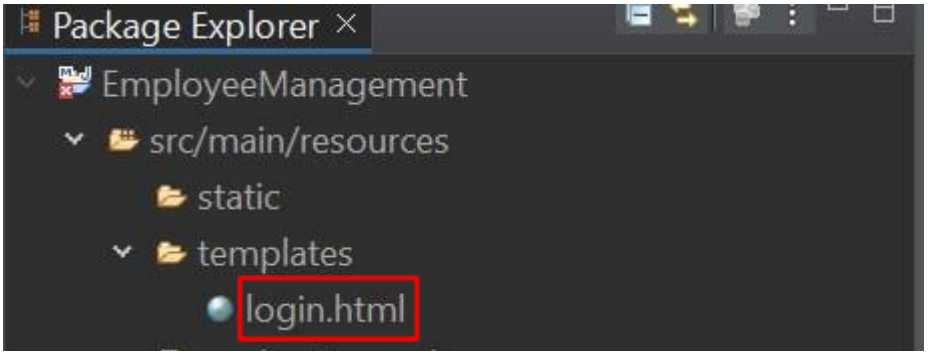
- Now we have understood what the different entities and dto's are. similarly, we can now comprehend the ***Employee Controller*** and ***SecurityConfig*** classes accordingly which will have the feature to ***register/ add/ update/ delete*** an employee based on their roles, Subsequently, we can create a *service* and the *repository*.



From the above diagram, it's clear that we need an HTML form to take inputs for the user and subsequently register the user into the database whenever the required API is called.

Steps to create a Login form using thymeleaf:

1. Create an HTML file (**login.html**) in the ***src/main/resources/templates*** directory



2. Now, let's build a thymeleaf login form.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
</body>
</html>
```

- The above code is an HTML (HyperText Markup Language) template, specifically written in HTML5, which is the fifth and latest version of the HTML standard.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta  name="viewport"  content="width=device-width,  initial-scale=1.0,
minimum-scale=1.0">
    <title>Login</title>
</head>
<body>
</body>
</html>
```

- In the line **<html xmlns:th="http://www.thymeleaf.org">**, we're giving a special name (namespace) to "**th**" which is linked to a specific technology called Thymeleaf. Think of a namespace as a unique nickname.

  • ***xmlns*** is a way to declare a namespace in XML/HTML.
  • ***th*** is the chosen nickname for the Thymeleaf namespace.
  • "***http://www.thymeleaf.org***" is the location or identifier of the Thymeleaf namespace.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta  name="viewport"  content="width=device-width,  initial-scale=1.0,
minimum-scale=1.0">
    <title>Login</title>
</head>
<body>
<div class="container-fluid text-center">
  <form th:action="@{/login}" method="post" style="max-width: 350px; margin: 0
auto;">
    </form>
</div>
</body>
</html>
```

- The above HTML `**<form>**` element is utilising Thymeleaf, a server-side Java-based templating engine. Let's break down the code step by step:

  • **<form>**: This is the HTML form element used to create a form on a web page. Forms are used to collect and submit data.

• **th:action="@{/login}**: This is a Thymeleaf attribute called `th: action`. Thymeleaf uses `th:action` to specify the action (i.e., the URL where the form will be submitted) dynamically. In this case, it's set to `@{/login}`, which is a Thymeleaf syntax to specify the URL for the form submission.

• **@{/login}**: This Thymeleaf expression generates the correct URL for the form submission based on the application's configuration. It's often used in Spring Boot applications.

• **method="post"**: This is a standard HTML attribute for the form element, specifying that the HTTP method for submitting the form is "POST." This is a common method used to send form data to the server.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <meta  name="viewport"  content="width=device-width,  initial-scale=1.0,
minimum-scale=1.0">
    <title>Login</title>
</head>
<body>
<div class="container-fluid text-center">
  <form th:action="@{/login}" method="post" style="max-width: 350px; margin: 0
auto;">
<div class="border border-secondary p-3 rounded">
            <p>Enter Employee Details</p>
            <p>
<input type="text" name="username" class="form-control" placeholder="Username"
required autofocus/>
            </p>
            <p>
           <input    type="password"    name="password"    class="form-control"
placeholder="Password" required />
   <input type="submit" value="Login" class="btn btn-primary" />
            </p>
            <div>
   <h2><a th:href="@{/oauth2/authorization/okta}">Login with Okta</a></h2>
            </div>

        </div>
    </form>
</div>
</body>
</html>
```

- Finally, we have added the following properties to the form:

  • Input fields for **username** and **password**.
  • **login** button.
  • **CSS-**based space formatting.

3. After running the spring-boot application, we must hit **localhost:active_portno/login** from our browser. To view the login form below, it is necessary to create a login controller; we will learn this in the upcoming lectures.

4. Likewise, you can create many forms based on thyme leaf; below are some brainstormed ideas for you to try:

   - User Registration Form

```
<!DOCTYPE html>
<html lang="en"
    xmlns:th="http://www.thymeleaf.org"
>
<head>
    <meta charset="UTF-8">
    <title>Registration and Login System</title>
                                                                    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
          rel="stylesheet"

integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
          crossorigin="anonymous">
</head>
```

```
<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a class="navbar-brand" th:href="@{/index}">Registration and Login System</a>
              <button  class="navbar-toggler"  type="button"  data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"         aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                <li class="nav-item">
                                        <a  class="nav-link  active"  aria-current="page"
th:href="@{/register}">Register</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
<br /><br />
<div class="container">
    <div class="row">
        <h1 class="text-center"> Registration and Login System </h1>
    </div>
</div>
</body>
</html>
```

- Survey Form

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Survey Form</title>
</head>
<body>
    <h1>Survey Form</h1>
    <form action="#" method="post">
        <label for="question1">Question 1: What is your favorite color?</label>
        <select id="question1" name="question1">
            <option value="red">Red</option>
            <option value="blue">Blue</option>
            <option value="green">Green</option>
```

```
            <option value="other">Other</option>
        </select>

        <label for="question2">Question 2: Do you like programming?</label>
        <input type="radio" id="yes" name="question2" value="yes">
        <label for="yes">Yes</label>
        <input type="radio" id="no" name="question2" value="no">
        <label for="no">No</label>

        <label for="question3">Question 3: How often do you code?</label>
        <input type="radio" id="daily" name="question3" value="daily">
        <label for="daily">Daily</label>
        <input type="radio" id="weekly" name="question3" value="weekly">
        <label for="weekly">Weekly</label>
        <input type="radio" id="monthly" name="question3" value="monthly">
        <label for="monthly">Monthly</label>

        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

5. Likewise, you can create any form using Thymeleaf.

References:

1. Official website
2. Spring
3. Baeldung