# **✅ What is Thymeleaf?**

**Thymeleaf** is a modern server-side Java template engine for both web and standalone environments. It’s widely used with **Spring Boot** to render HTML views.

**🔍 Why It Came:**

* Traditional JSP lacked support for modern HTML5 and needed servlet containers.
* Developers needed natural templating with full HTML rendering during design-time.
* Thymeleaf allowed for **natural templates**—i.e., templates viewable in browsers without backend.

**✅ Thymeleaf Basics**

* Works with HTML/XML templates.
* Uses th:\* attributes (e.g., th:text, th:each) to dynamically bind data.
* Supports loops, conditionals, text manipulation, etc.

**✅ Installing Thymeleaf**

You don’t "install" Thymeleaf directly—Spring Boot auto-configures it when the dependency is added.

**✅ Core Syntax & Expressions**

| **Syntax** | **Description** |
| --- | --- |
| ${} | Variable expressions (Model attributes) |
| \*{} | Selection expressions (used within th:object) |
| #{} | Message expressions (i18n properties) |
| @{} | URL expressions (links, resources) |
| th:if, th:each, th:text | Logical attributes |

Example:

<span th:text="${user.name}">Name</span>

**✅ Integration with Spring Boot**

Spring Boot auto-configures Thymeleaf:

* Templates in: src/main/resources/templates/
* Static files in: src/main/resources/static/

Controller Example:

@GetMapping("/user")

public String user(Model model) {

model.addAttribute("name", "Alice");

return "user"; // returns user.html

}

**✅ Templating & Layouts**

Thymeleaf allows reusable fragments via:

* th:insert
* th:replace
* th:include

Example:

<!-- Main page -->

<div th:replace="fragments/header :: headerFragment"></div>

**✅ Template Structure**

Default locations:

* HTML templates → src/main/resources/templates
* Fragments → under same folder or fragments/
* Static assets (CSS, JS, images) → src/main/resources/static

**✅ Advanced Features**

* **Conditional Rendering**: th:if, th:unless
* **Switch-case**: th:switch, th:case
* **Date formatting**: th:text="${#dates.format(now, 'dd-MM-yyyy')}"
* **String operations**: #strings.toUpperCase(str)
* **URL resolution**: @{/profile(id=${user.id})}

**✅ Working with Static Resources**

Spring Boot maps /static, /public, and /resources to serve files.

Example:

<link rel="stylesheet" th:href="@{/css/style.css}">

**✅ Creating Forms with Thymeleaf**

1. Bind model with th:object
2. Bind fields with th:field
3. Submit to Spring Controller

Example:

<form th:action="@{/register}" th:object="${user}" method="post">

<input type="text" th:field="\*{name}" />

<input type="email" th:field="\*{email}" />

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

</form>

Controller:

@PostMapping("/register")

public String register(@ModelAttribute User user) {

userService.save(user);

return "success";

}

**✅ Error Handling & Validation**

**With Spring Validator:**

1. Add validation annotations in the model:

@NotEmpty

@Email

private String email;

1. Update controller:

@PostMapping("/register")

public String submit(@Valid @ModelAttribute("user") User user, BindingResult result) {

if (result.hasErrors()) {

return "form";

}

return "success";

}

1. Display errors in Thymeleaf:

<div th:if="${#fields.hasErrors('email')}" th:errors="\*{email}"></div>

**✅ Real-World Examples**

**1. User Registration Form**

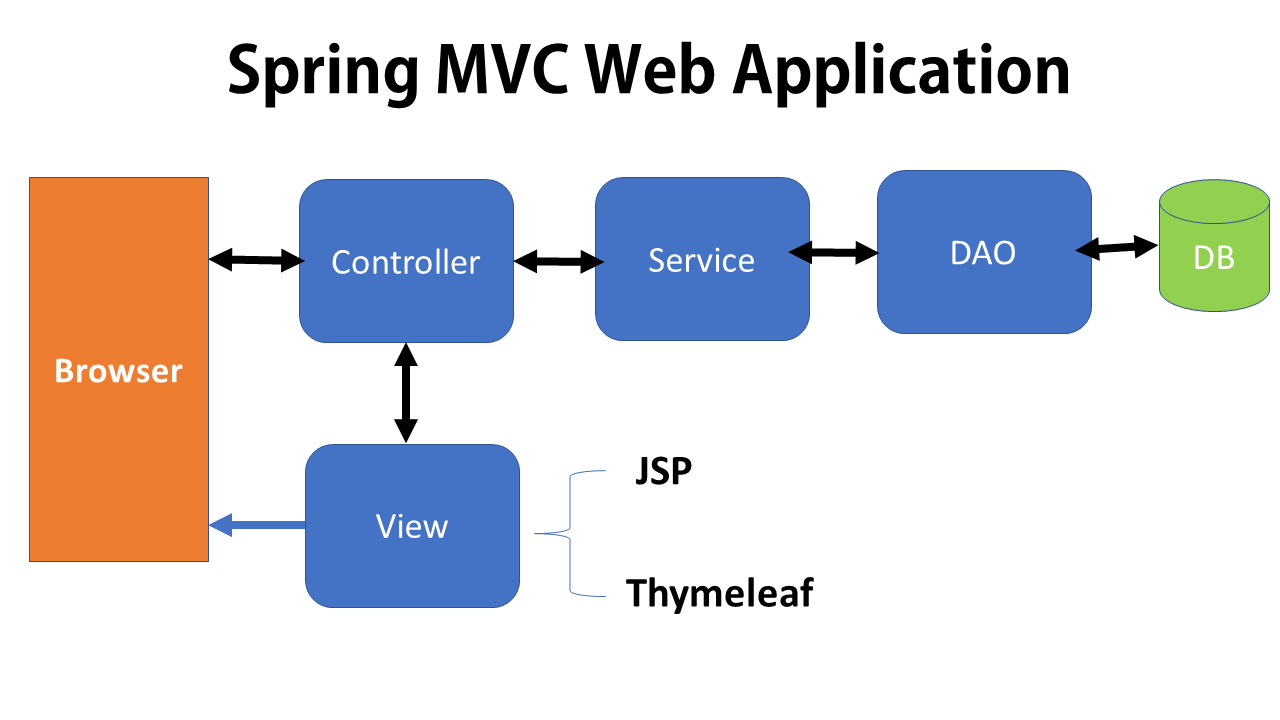
* Collect name, email, password.
* Validates using JSR-303.
* Displays field-level errors using th:errors.

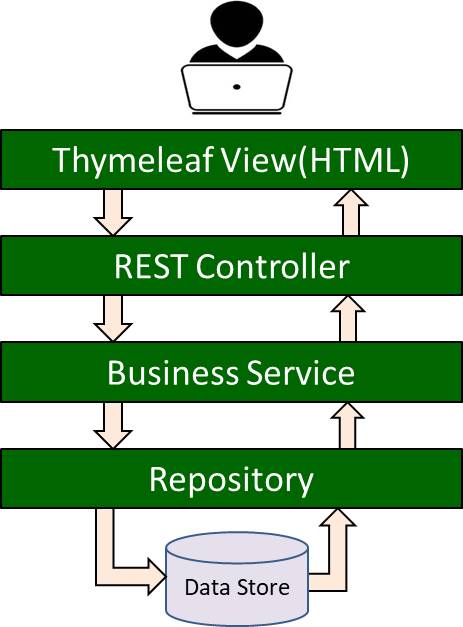
**2. Employee Directory**

* List employees using th:each.
* Show links like /employee/{id} using @{}.
* Reusable layouts for table headers using fragments.

**3. Blog with Commenting System**

* Renders blog post dynamically using model attributes.
* Submits comments through Thymeleaf form.
* Validates empty comment fields and displays messages.





**3. Maven Dependency**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

* **Why It Came**  
  Bundles Thymeleaf core, Spring integration, and default settings in one starter.
* **Rules & Tips**
  1. Use the **Spring Boot starter** for auto-configuration.
  2. If using vanilla Spring, add thymeleaf-spring5 and thymeleaf separately.
  3. Ensure template files are UTF-8 encoded (spring.thymeleaf.encoding=UTF-8).
* **When & Where to Use**
  1. Always in **Spring Boot** web applications.
  2. **Exclude** if you need a custom version of Thymeleaf or non-Spring integration.
* **Real-World Examples**
  1. **Monolithic Web App:** Add starter and immediately write controllers returning view names.
  2. **Micro-frontend:** Only static UI service needs Thymeleaf for navigation templates.
  3. **Hybrid App:** Both REST controllers and Thymeleaf controllers coexist—starter covers both.

**4. Integration with Spring Boot**

* **Why It Came**  
  Reduce manual view resolver and message resolver configuration.
* **Rules & Behavior**
  1. **Auto-Configuration:** Spring Boot registers ThymeleafViewResolver pointing at /templates/\*.html.
  2. **Properties:** Customize via application.properties:

spring.thymeleaf.cache=true

spring.thymeleaf.prefix=classpath:/custom-templates/

spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML

* 1. **Model Binding:** Controllers return a view name string; model attributes become available in templates.
* **When & Where to Use**
  1. **Standard Boot web projects**: no extra Java config needed.
  2. **Multi-module apps**: share a common starter that includes Thymeleaf.
* **Real-World Examples**
  1. **Dashboard Service:** @Controller methods return "dashboard" to resolve /templates/dashboard.html.
  2. **Internationalized Site:** Spring Boot + Thymeleaf message bundles (messages.properties) pick up #{welcome.msg}.
  3. **Themed Templates:** Switch prefixes via profile (spring.profiles.active=dark uses /templates/dark/).

**5. Template Structure**

src/main/resources/

└─ templates/

├─ fragments/

│ ├─ header.html

│ └─ footer.html

├─ users/

│ ├─ list.html

│ └─ edit.html

└─ index.html

* **Why It Came**  
  Encourage **modular**, **reusable** templates using fragments and layouts.
* **Rules & Best Practices**
  1. **Keep fragments** (header, footer, nav) in a fragments/ folder.
  2. **Organize by feature** (users/, orders/, products/).
  3. **Use .html suffix** consistently—configurable but recommended.
* **When & Where to Use**
  1. **Large apps**: organize templates to avoid collisions.
  2. **Multi-tenant UI**: separate tenant templates by folder.
  3. **Reusable Libraries**: package Thymeleaf fragments in a JAR for reuse.
* **Real-World Examples**
  1. **Corporate Portal:** Shared header/footer in fragments/, pages in departments/, employees/.
  2. **SaaS App:** Tenant-specific overrides in templates/tenantA/... and default in templates/default/....
  3. **Themeable Blog:** Templates under templates/light/, templates/dark/, switched by property.

**6. Thymeleaf Syntax**

* **Core Attributes**

| **Prefix** | **Attribute** | **Purpose** |
| --- | --- | --- |
| th: | th:text="${user.name}" | Set element body text |
| th: | th:utext="${html}" | Unescaped HTML |
| th: | th:each="item : ${items}" | Loop over a collection |
| th: | th:if="${cond}" | Conditional inclusion |
| th: | th:href="@{/users/{id}(id=${u.id})}" | URL building |
| th: | th:replace="fragments/header :: headerFrag" | Include fragment |

* **Why It Came**  
  Provide a **concise**, **HTML-valid** way to inject dynamic data and structure flows.
* **Rules & Tips**
  1. **Use th:if instead of th:unless** for clarity.
  2. **Prefer th:replace over th:include** to avoid leftover tags.
  3. **Avoid mixing ${} and \*{}**—use selection expressions in forms (\*{field}).
* **When & Where to Use**
  1. **Data binding** in views.
  2. **URL generation** to avoid hard-coded links.
  3. **Fragment composition** for layouts.
* **Real-World Examples**
  1. **User List Page:**

<tr th:each="u : ${users}">

<td th:text="${u.id}"></td>

<td th:text="${u.name}"></td>

<td><a th:href="@{/users/{id}(id=${u.id})}">View</a></td>

</tr>

* 1. **Conditional Banner:**

<div th:if="${session.user != null}">

Welcome, <span th:text="${session.user.name}">User</span>

</div>

* 1. **Fragment Layout:**

<html>

<head th:replace="fragments/header :: headFrag"></head>

<body>

<div th:replace="fragments/nav :: navFrag"></div>

<div th:insert="${content}"></div>

<footer th:replace="fragments/footer :: footerFrag"></footer>

</body>

</html>

**7. Creating Forms with Thymeleaf**

* **Why It Came**  
  Simplify binding HTML forms to Spring MVC @ModelAttribute objects without JavaScript.
* **Rules & Key Attributes**
  1. **Use Spring’s form namespace:**

xmlns:th="http://www.thymeleaf.org"

xmlns:form="http://www.thymeleaf.org"

* 1. **th:object** at <form> level binds a model attribute.
  2. **th:field="\*{fieldName}"** on inputs auto-generates name, id, value.
  3. **Error display:** <div th:if="${#fields.hasErrors('name')}" th:errors="\*{name}"/>.
* **When & Where to Use**
  1. **CRUD forms:** create/edit pages for entities.
  2. **Validation:** integrate JSR-303 annotations (@Valid) and display errors.
  3. **Multi-step wizards:** use hidden fields and fragments.
* **Real-World Examples**
  1. **User Registration Form:**

<form th:action="@{/users}" th:object="${user}" method="post">

<input th:field="\*{email}" placeholder="Email"/>

<div th:if="${#fields.hasErrors('email')}" th:errors="\*{email}"/>

<input th:field="\*{password}" type="password" placeholder="Password"/>

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

</form>

* 1. **Product Edit Form with Select:**

<select th:field="\*{category}">

<option th:each="c : ${categories}" th:value="${c}" th:text="${c}"/>

</select>

* 1. **Order Checkout with Checkboxes:**

<div th:each="item : ${order.items}">

<input type="checkbox" th:field="\*{itemsSelected}" th:value="${item.id}"/>

<span th:text="${item.name}"/>

</div>

## **Quiz Questions on Spring Boot & Thymeleaf**

1. **Which Maven dependency adds Thymeleaf support in a Spring Boot project?**  
   A. spring-boot-starter-web  
   B. spring-boot-starter-thymeleaf  
   C. thymeleaf-spring5  
   D. spring-boot-thymeleaf  
     
   **Answer: B**
2. **By default, Spring Boot Thymeleaf templates are located in:**  
   A. src/main/resources/views  
   B. src/main/resources/templates  
   C. src/main/webapp/WEB-INF  
   D. templates/ at project root  
     
   **Answer: B**
3. **Which attribute prints a model variable user.name into an element’s body?**  
   A. th:text="${user.name}"  
   B. th:utext="${user.name}"  
   C. th:value="${user.name}"  
   D. th:insert="${user.name}"  
     
   **Answer: A**
4. **How do you iterate a list items in a Thymeleaf template?**  
   A. <th:each="item : items">  
   B. <tr th:each="item : ${items}">  
   C. <div th:for="item in items">  
   D. <li th:repeat="item : items">  
     
   **Answer: B**
5. **Which expression builds a URL to /products/{id} with id=5?**  
   A. @{/products/{id}(5)}  
   B. @{/products/{id}(id=5)}  
   C. @{/products/5}  
   D. @{/products/{id}(id=${5})}  
     
   **Answer: B**
6. **To include a fragment headerFrag from fragments/header.html, you use:**  
   A. th:insert="fragments/header :: headerFrag"  
   B. th:replace="fragments/header :: headerFrag"  
   C. Both A and B (slightly different behavior)  
   D. th:include="fragments/header :: headerFrag"  
     
   **Answer: C**
7. **What does th:if do in Thymeleaf?**  
   A. Always removes the element  
   B. Includes the element only if the condition is true  
   C. Comments out the element  
   D. Iterates the element based on the condition  
     
   **Answer: B**
8. **Which namespace must be declared for Thymeleaf attributes?**  
   A. xmlns:th="http://www.thymeleaf.org"  
   B. xmlns:thyme="http://thymeleaf.org"  
   C. xmlns:tl="http://www.thymeleaf.org"  
   D. xmlns="http://www.thymeleaf.org"  
     
   **Answer: A**
9. **To bind a form to a model attribute user, you set:**  
   A. <form th:object="${user}">  
   B. <form th:bind="\*{user}">  
   C. <form th:model="user">  
   D. <form th:form="user">  
     
   **Answer: A**
10. **Which attribute binds an input to user.email?**  
    A. th:value="${user.email}"  
    B. th:field="\*{email}"  
    C. th:bind="user.email"  
    D. th:name="user.email"  
      
    **Answer: B**
11. **How do you print unescaped HTML content stored in htmlContent?**  
    A. th:text="${htmlContent}"  
    B. <div th:utext="${htmlContent}"></div>  
    C. <div th:unescaped="${htmlContent}"></div>  
    D. <div th:html="${htmlContent}"></div>  
      
    **Answer: B**
12. **Which property disables Thymeleaf template caching in Spring Boot?**  
    A. spring.thymeleaf.cache=false  
    B. thymeleaf.cache=false  
    C. spring.cache.templates=false  
    D. spring.thymeleaf.enabled-cache=false  
      
    **Answer: A**
13. **To resolve messages in messages.properties, you use:**  
    A. ${msg.welcome}  
    B. #{welcome.message}  
    C. @{welcome.message}  
    D. \*{welcome.message}  
      
    **Answer: B**
14. **Which template mode renders plain text rather than HTML?**  
    A. HTML  
    B. XML  
    C. TEXT  
    D. LEGACYHTML5  
      
    **Answer: C**
15. **What does th:replace do differently from th:insert?**  
    A. th:replace replaces the entire host tag, th:insert places inside it  
    B. th:replace only replaces tag body, th:insert replaces tag itself  
    C. Both behave the same  
    D. th:replace uses AJAX, th:insert does not  
      
    **Answer: A**

# **Assignment: Student Management Web App using Spring Boot & Thymeleaf**

Build a small web application to **manage students**—listing, adding, editing, and deleting—using Spring Boot on the backend and Thymeleaf for server-side HTML rendering. Below are the requirements, sample inputs, expected outputs, and acceptance criteria. *Do not provide code—focus on behavior, pages, and data.*

**🔖 Scenario**

You are creating an admin interface for a university to manage its students. Each student has:

* **ID** (auto-generated)
* **Name** (non-blank)
* **Email** (must be valid format, unique)
* **Major** (e.g. “Computer Science”, “Mathematics”)

Administrators must be able to:

1. **View** all students
2. **Add** a new student
3. **Edit** existing student details
4. **Delete** a student
5. **Handle validation errors** and “not found” cases gracefully

All pages are rendered with Thymeleaf templates.

**📋 Requirements**

1. **List Students**
   * **URL:** GET /students
   * **Behavior:** Query all students; display in an HTML table. If none, show “No students found.”
   * **Sample Seed Data:**

| **ID** | **Name** | **Email** | **Major** |
| --- | --- | --- | --- |
| 1 | Alice Lee | alice@example.com | Computer Science |
| 2 | Bob Patel | bob.patel@uni.edu | Physics |
| 3 | Carol Chen | carol.chen@uni.edu | Mathematics |

* + **Expected Output:**
    - A page titled **“Student List”**
    - Table rows for Alice, Bob, Carol
    - “Add New Student” button linking to the form

1. **Add Student Form**
   * **URL:** GET /students/new
   * **Behavior:** Display a form with fields **Name**, **Email**, **Major**.
   * **Sample Input (user types into form):**
     + Name: David Kumar
     + Email: david.kumar@uni.edu
     + Major: Chemistry
   * **On Submit:** POST /students
     + If valid, save and redirect to **/students**.
     + If invalid (blank name or bad email, or duplicate email), re-display the form with inline error messages under each field.
   * **Expected Output (success):**
     + Redirect to Student List, now including David as ID 4.
   * **Expected Output (validation error):**
     + Same form page, with e.g. under Email: “Please enter a valid email.”
2. **Edit Student**
   * **URL:** GET /students/{id}/edit
   * **Behavior:**
     + Load student by {id}.
     + Pre-populate the same form used for “Add.”
     + On submit (POST /students/{id}), update and redirect back to list.
   * **Sample Input:** Edit Bob (ID 2), change Major to Astronomy.
   * **Expected Output:** Student List shows Bob’s major updated.
   * **Not Found:** If ID doesn’t exist (e.g. /students/99/edit), display a **404 page** with message “Student not found.”
3. **Delete Student**
   * **URL:** GET /students/{id}/delete
   * **Behavior:** Delete the record and redirect to **/students**.
   * **Sample Action:** Click “Delete” on Carol (ID 3).
   * **Expected Output:** Student List no longer shows Carol; a flash message “Student Carol Chen deleted successfully.”
4. **Template Structure & Layout**
   * Use a **common header/footer** fragment for all pages (fragments/header.html, fragments/footer.html).
   * Place templates in src/main/resources/templates.
   * Static assets (CSS) under src/main/resources/static/css/style.css, referenced as <link th:href="@{/css/style.css}">.
5. **Error Handling**
   * **Validation errors** show inline next to the offending field.
   * **404 Not Found** for edit/delete on non-existing ID uses a Thymeleaf error template (error/404.html).

**🎯 Acceptance Criteria**

* All four CRUD operations work via standard Thymeleaf-backed HTML forms and links.
* Validation prevents blank names, invalid emails, and duplicate emails; errors display correctly.
* Templates use Thymeleaf’s th:each, th:if, th:field, th:errors, and th:replace for fragments.
* Static CSS is correctly loaded and applied to pages.
* “Student not found” uses a custom 404 view, not a server stack-trace.
* On success operations, the user is redirected back to /students and sees an updated table plus a success message.