# Desarrollo web en entorno servidor

## UT1.2 – Programación web

7 – Thymeleaf

IES Clara del Rey – Madrid

# Plantillas

Una plantilla es un documento diseñado para su reutilización.

En desarrollo web, las plantillas contienen contenido estático (HTML, CSS) con marcadores para incrustar contenido dinámico.

En tiempo de ejecución, estos marcadores son evaluados y reemplazados por el contenido, por los datos. Ejemplo de un fragmento de plantilla:

```
<html>
    <head><title>Página</title></head>
    <body>
        <h1>¡Bienvenido, ${username}!</h1>
    </body>
</html>
```

${username} se sustituye por contenido cuando se procesa la plantilla.

# Motor de plantillas

Herramienta que procesa las plantillas para generar contenido dinámico.

Combina datos con plantillas para generar documentos completos en tiempo de ejecución.

El sistema de plantillas + motor de plantillas permite:

- Separación de la lógica y la presentación de la aplicación.
- Un código más limpio y fácil de mantener.

Ejemplos de motores: Thymeleaf, Freemarker, Velocity, JSP, Twig, Razor.

Los motores de plantillas no son exclusivos del desarrollo web, se pueden usar en todo tipo de aplicaciones, pero se integran especialmente bien con tecnologías como HTML o CSS.

# Thymeleaf

Thymeleaf es un motor de plantillas desarrollado en Java que se puede usar en entornos web, pero también en otro tipo de aplicaciones.

Open source, con licencia Apache 2.0. Gratis para uso y modificación.

Se centra en favorecer el llamado "natural templating", que facilita la colaboración en equipos multidisciplinares, con diferentes perfiles, como maquetadores, diseñadores o programadores.

Permite visualizar las plantillas correctamente en un navegador, sin necesidad de que el motor las procese. Esto facilita el trabajo de estilo y maquetación.

Permite generar HTML (HTML5, XHTML 1.0/1.1, HTML 4), XML, JS, CSS o texto plano.

# Thymeleaf en Spring

Thymeleaf dispone de módulos específicos para el desarrollo en Spring, con un dialecto especial para el lenguaje de plantillas, y con configuración específica para este framework.

Además, Spring Boot simplifica aún más el desarrollo con un starter:

*spring-boot-starter-thymeleaf del grupo org.springframework.boot*

Este starter facilita la configuración de Thymeleaf en Spring, ahorrando configuración XML (beans y otras características), y la creación de cierto código necesario para inicializar el motor de plantillas.

En el contexto de Spring MVC, el motor de Thymeleaf es el encargado de generar las vistas, a partir de los datos que recibe del controlador.

# Thymeleaf en Spring – Configuración

En Spring Boot, Thymeleaf está listo para usar al incluir la dependencia "spring-boot-starter-thymeleaf", con una configuración por defecto:

- Ubicación de las plantillas: src/main/resources/templates
- Sufijo de archivos: Thymeleaf esperará que los ficheros de plantilla tengan la extensión ".html"
- Codificación de los ficheros de plantilla: UTF-8.
- Cacheo de plantillas: activado, para mejorar rendimiento.
- Modo de trabajo: HTML5
- Contenido del modelo: Spring Boot agrega automáticamente al modelo datos comunes como ServletContext y ServletRequest, entre otros.

# Thymeleaf en Spring – Configuración

Se puede sobrescribir / modificar la configuración por defecto, usando propiedades en el fichero application.properties. Algunos ejemplos:

- Cambiar la ubicación de las plantillas:
  *spring.thymeleaf.prefix=classpath:/custom-templates/*
  Cambia la carpeta "templates" por "custom-templates"
- Cambiar el sufijo (extensión) de los archivos de plantilla
  spring.thymeleaf.suffix=.xhtml
- Desactivar el cache en desarrollo
  *spring.thymeleaf.cache=false*
- # Cambiar la codificación de los ficheros de plantilla
  *spring.thymeleaf.encoding=UTF-16*

# Thymeleaf en Spring – Lenguaje y dialectos

Thymeleaf proporciona un lenguaje para poder evaluar expresiones dentro de las plantillas, y así poder acceder a los datos.

Este lenguaje se llama Thymeleaf Standard Dialect.

De este dialecto base derivan otros dialectos para uso específico.

Los dialectos añaden funcionalidades, personalizan el comportamiento o simplifican el desarrollo en contextos concretos.

Thymeleaf permite, incluso, crear dialectos personalizados, a la medida del proyecto en el que se utilice.

El dialecto específico para Spring es el dialecto SpringStandard.

# Thymeleaf – Natural templating

El natural templating se basa en:

- Mantener la plantilla HTML lo más limpia y correcta posible, ajustándose al estándar (HTML5, HTML4)
- Permitir que la plantilla se pueda visualizar normalmente, sin necesidad de lanzar la ejecución de la aplicación, por lo que se pueden validar con sólo un navegador web.

Thymeleaf ha optado por un funcionamiento basado en atributos. No existen elementos específicos para Thymeleaf, sólo hay atributos.

Hay que usar el namespace xmlns:th=http://www.thymeleaf.org, para que no se interpreten los atributos como errores. En algunos editores puede funcionar perfectamente si el namespace, pero es recomendable.

# Thymeleaf – Recepción de datos en plantilla

La vista recibe del controlador los datos a través de distintos mecanismos, siendo los más habituales:

- Objeto Model
- Objeto ModelMap o Map
- Objeto ModelAndView

En todos los casos, la forma de desarrollar la plantilla es la misma.

La sintaxis para acceder a los datos será la misma en cualquiera de los casos. Todos estos objetos están formados por atributos a los que se puede acceder desde la plantilla.

Además, desde la plantilla se puede acceder a otros recursos.

# Thymeleaf – Natural templating

El natural templating se basa en:

- Mantener la plantilla HTML lo más limpia y correcta posible, ajustándose al estándar (HTML5, HTML4)
- Permitir que la plantilla se pueda visualizar normalmente, sin necesidad de lanzar la ejecución de la aplicación, por lo que se pueden validar con sólo un navegador web.

Thymeleaf ha optado por un funcionamiento basado en atributos. No existen elementos específicos para Thymeleaf, sólo hay atributos.

Hay que usar el namespace xmlns:th=http://www.thymeleaf.org, para que no se interpreten los atributos como errores. En algunos editores puede funcionar perfectamente si el namespace, pero es recomendable.

# Thymeleaf – Atributos (atributos "th:")

Son la base de las plantillas y permiten agregar lógica a las vistas.

Amplían la funcionalidad del HTML, permitiendo generar contenido dinámico, basado en datos proporcionados por el controlador.

Se integran en el HTML sin romper su estructura. Los navegadores ignoran estos atributos si no están procesados por Thymeleaf, permitiendo que las plantillas sean legibles sin un servidor.

Reemplazan o complementan contenido o atributos nativos de HTML, según la lógica de la aplicación.

Thymeleaf procesa los atributos "th:" en el servidor, antes de enviar la respuesta al cliente. Utiliza los datos proporcionados por el controlador dentro del modelo para generar contenido dinámico.

# Thymeleaf – Atributos (atributos "th:")

En el siguiente ejemplo:

```
<h1 th:text="${titulo}">Esto es el título por defecto</h1>
```

Pasan dos cosas

- th:text reemplaza el texto del h1 con el atributo "titulo" del modelo.
- Si se visualiza la plantilla en un navegador, el atributo th:text se ignora (no es HTML válido), y el título será "Esto es un título por defecto"

Si se quiere realizar validación de las plantillas según estándares HTML, se puede usar "data-th" en lugar de "th:". Ejemplo:

```
<h1 data-th-text="${titulo}">Esto es el título por defecto</h1>
```

Ambas notaciones son equivalentes, y no suponen ninguna diferencia.

# Thymeleaf – Expresiones

En Thymeleaf, dentro de los atributos (th:text, th:href, th:attr, etc.) se usan expresiones, que pueden ser de distintos tipos:

- ${...} – Variables. Se utilizan para acceder a los datos del modelo.
- *{...} – De selección. Se utilizan para acceder a los datos de un objeto que se ha seleccionado previamente, usando "th:object".
- #{...} – Mensajes. Se utilizan para acceder a mensajes en el contexto de internacionalización y localización (traducción) de aplicaciones. Dedicaremos un monográfico a esto más adelante.
- @{...} – URL. URL dinámicas en función de mappins y parámetros.
- Literales y números. No se anteponen de ningún carácter.

# Thymeleaf – Expresiones

- Expresiones lógicas. Específicamente en atributos condicionales como "th:if" o "th:unless"
- Expresiones aritméticas y concatenación de strings. Operaciones entre valores del modelo y/o literales. +, -, *, /, %, o el operador ternario:
  codición ? valor-si verdadero : valor-si falso
- {#utilidad} – Utilidades. Ojo, la almohadilla está dentro de las llaves, no fuera, como en mensajes. Proporciona variables y funciones para hacer ciertas operaciones. Algunas de estas utilidades: #locale, #dates, #numbers, #strings, #lists. Hay bastantes más.
- ~{...} – Fragmentos. Inclusión de fragmentos reutilizables. Similar a include/requiere en PHP. Los veremos en detalle más adelante.

# Thymeleaf – Generar texto y atributos

**th:text –** Reemplaza el texto de un elemento con un valor.

```
<p th:text="${mensaje}">Texto por defecto</p>
```

Si mensaje es "¡Hola!", el resultado es:

```
<p>¡Hola!</p>
```

Escapa el texto al insertarlo, para evitar XSS, de forma que si el mensaje fuera "¡Hola!" el resultado sería:

```
<p>&lt;span&gt;¡Hola!&lt;/span&gt;</p>
```

Para no escapar el texto, se usa "**th:utext**", en cuyo caso se obtendría:

```
<p><span>¡Hola!</span></p>
```

# Thymeleaf – Generar texto y atributos

**th:attr –** Define dinámicamente atributos en el elemento. Si el atributo ya existe, se reemplaza con el valor de la expresión.

```
<p th:attr="title=${titulo},class=${clase}">Texto</p>
```

Si titulo es "El título" y clase es "destacado", el resultado es:

```
<p title="El título" class="destacado">Texto</p>
```

Como es muy habitual usarlo con solo un atributo, hay muchos atajos, para los atributos HTML más utilizados: **th:href**, **th:src**, **th:class**, etc.

Para añadir sin reemplazar, sin perder el valor "estático", se puede usar "**th:attrappend**" (añade al final) y "**th:attrprepend**" (añade al principio).

Útil para añadir clases CSS. Atajos: **classappend** / **styleappend**.

# Thymeleaf – Condicionales

**th:if –** Genera el elemento si la condición se cumple.

**th:unless –** Genera el elemento si la condición NO se cumple. Es equivalente a usar "not" o "!" en la condición de un if

```
<p th:if="${user.loggedIn}">¡Bienvenido de vuelta!</p>

<p th:unless="${user.loggedIn}">Identifícate para acceder</p>
```

Si el usuario está logado (identificado) en el sistema, se mostrará el mensaje de bienvenida. Si no lo está se mostrará la indicación de que debe acceder. El unless se podría reescribir como:

```
<p th:if="!${user.loggedIn}">Identifícate para acceder</p>
<p th:if="not ${user.loggedIn}">Identifícate para acceder</p>
```

# Thymeleaf – Condicionales

**th:switch –** Define el comienzo de una estructura switch / case.

**th:case –** Cada caso dentro de una estructura switch/case.

Ejemplo con un switch sobre String:

```
<div th:switch="${tipo}">
    <p th:case="'admin'">Administrador</p>
    <p th:case="'manager'">Gerente</p>
    <p th:case="'power-user'">Usuario avanzado</p></div>
    <p th:case="*">Usuario</p>
</div>
```

th:case="*" es el equivalente a "default:" en Java. Se cumple cuando no se cumple ninguno de los anteriores.

# Thymeleaf – Repetitivas

**th:each –** Genera el elemento tantas veces como elementos haya en una colección. Funciona con listas, mapas, o arrays, entre otros.

Es el elemento más utilizado para iterar colecciones y repetir elementos

```
<ul>
    <li th:each="item : ${items}" th:text="${item}">Aquí el texto</li>
</ul>
```

Con una lista de Strings con los valores "Plátano" y "Melón" genera:

```
<ul>
    <li>Plátano</li>
    <li>Melón</li>
</ul>
```

# Thymeleaf – Repetitivas

Dentro de un th:each se puede acceder información sobre el proceso de iteración. Para usarlo se usa la sintaxis th:each="item, itemStat : ${col}"

itemStat.index – Índice actual de la iteración. Comienza en cero.

itemStat.count – Número actual de iteraciones. Comienza en uno.

itemStat.size,  – Tamaño de la colección.

itemStat.current – Elemento actual de la colección

itemStat.even – True si la iteración actual es par (0, 2, 4, 6, ...)

itemStat.odd – True si la iteración actual es impar (1, 3, 5, 7, ...)

itemStat.first – True si es el primer elemento de la iteración.

itemStat.last – True si es el último elemento de la iteración.